BEA WebLogic Personalization Server

Guide to Building Personalized Applications

BEA WebLogic Personalization Server 3.5
Document Edition 3.5
March 2002

Guide to Building Personalized Applications

Document Edition	Date	Software Version
3.5.2	March 2002	BEA WebLogic Commerce Server 3.5 BEA Weblogic Personalization Server 3.5

Contents

12. Personalization Server JSP Tag Library Reference

About This Document

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

This document includes the following topics:

- Chapter 1, "Overview of Personalization Development," provides developer components and utilities that enable developers to create personalized applications. The pieces documented in this guide include the Advisor, Foundation classes and utilities, and JSP tag reference.
- Chapter 2, "Creating Personalized Applications with the Advisor," recommends content and performs several important functions in creating a personalized application, including searching for content, tying the other core personalization services together, and matching content to user profiles.
- Chapter 3, "Introducing the Rules Manager," 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, "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 6, "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 7, “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 8, “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 9, “Working with Ad Placeholders,” shows how ad placeholders display documents that advertise products or services (ads) and record customer reactions to them.
- Chapter 10, “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 11, “The WebLogic Personalization Server Database Schema,” documents the database schema for the WebLogic Personalization Server.
- Chapter 12, “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 Personalization Server documentation Home page on the e-docs Web site (and also on the documentation CD). You can open the PDF in Adobe Acrobat Reader and print the entire document (or a portion of it) in book format. To access the PDFs, open the WebLogic Personalization Server documentation Home page, click the PDF files button and select the document you want to print.

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

Contact Us!

Your feedback on the BEA WebLogic Personalization Server documentation is important to us. Send us e-mail at **docsupport@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 3.5.

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.
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. *Example*s: LPT1 SIGNON OR

Convention	Item
`{ }`	Indicates a set of choices in a syntax line. The braces themselves should never be typed.
`[ ]`	Indicates optional items in a syntax line. The brackets themselves should never be typed. *Example*: `buildobjclient [-v] [-o name ] [-f file-list]... [-l file-list]...`
`|`	Separates mutually exclusive choices in a syntax line. The symbol itself should never be typed.
`...`	Indicates one of the following in a command line: ■ That an argument can be repeated several times in a command line ■ That the statement omits additional optional arguments ■ That you can enter additional parameters, values, or other information The ellipsis itself should never be typed. *Example*: `buildobjclient [-v] [-o name ] [-f file-list]... [-l file-list]...`
. . .	Indicates the omission of items from a code example or from a syntax line. The vertical ellipsis itself should never be typed.

CHAPTER

1 Overview of Personalization Development

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.

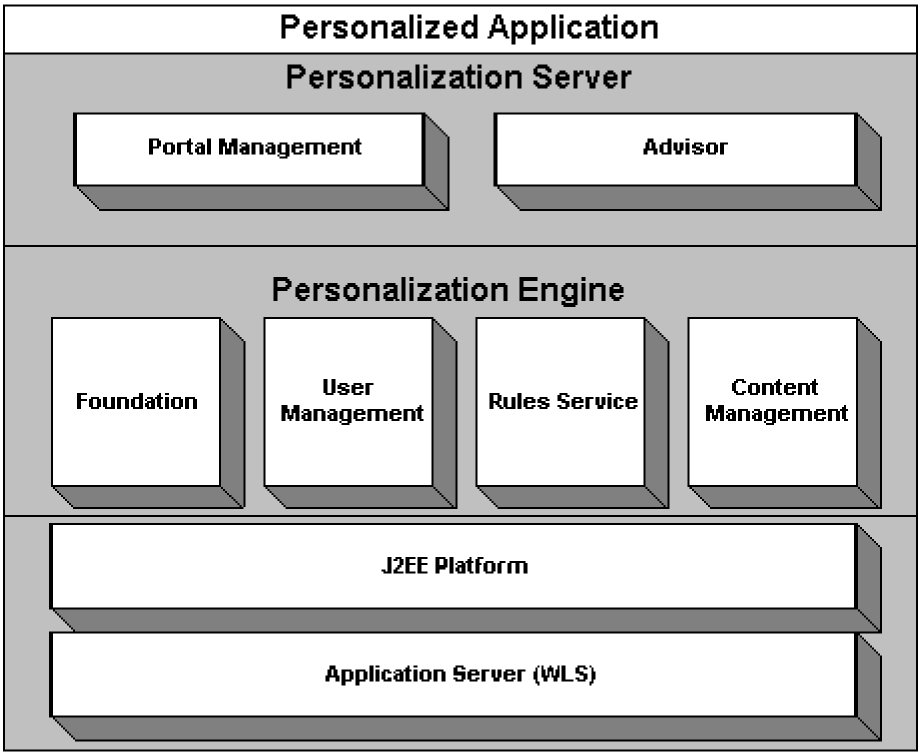
This topic includes the following sections:

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

Personalization Server Run-Time Architecture

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

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



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

Advisor

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

Portal Management

The Portal Management component provides tags and EJB objects to support creating a framework of portals and portlets. It is configured using the Portal Administration Tools and has embedded JSP fragments built by the developer. For additional information about Portal Management, see the *Guide to Creating Portals and Portlets*.

User Management

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

Content Management

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

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.

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

Library	Tag	Description
Content Management	`<cm:getProperty>`	Retrives 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.
Flow Manager	`<fm:getApplicationURI>`	Gets the Flow Manager.
	`<fm:getCachedAttribute>`	Gets an attribute out of the session/global cache.
	`<fm:setCachedAttribute>`	Sets an attribute in the session/global cache.
	`<fm:removeCachedAttribute>`	Removes an attribute from the session/global cache.
	`<fm:getSessionAttribute>`	Gets an attribute out of the HttpSession.
	`<fm:setSessionAttribute>`	Sets an attribute in the HttpSession.
	`<fm:removeSessionAttribute>`	Removes an attribute from the HttpSession.
Internationalization	`<i18n:localize>`	Defines the language, country, variant, and base bundle name to be used throughout a page when accessing resource bundles via the `<i18n:getmessage>` tag. Also allows a character encoding and content type to be specified for a JSP.

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

Library	Tag	Description
	`<i18n:getMessage>`	Used in conjunction with the `<i18:localize>` tag to retrieve localized static text or messages from a JspMessageBundle.
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.
User Management (Profile)	`<um:getProfile>`	Retrieves the Unified User Profile object.
	`<um:getProperty>`	Gets the value for the specified property from the current user profile in the session.
	`<um:getPropertyAsString>`	Works exactly like the `<um:getProperty>` tag above, but ensures that the retrieved property value is a `String`.
	`<um:removeProperty>`	Removes the property from the current user profile in the session.
	`<um:setProperty>`	Sets a new value for the specified property for the current user profile in the session.

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

Library	Tag	Description
(Group-User Management)	`<um:addGroupToGroup>`	Adds the group corresponding to the provided `childGroupName` to the group corresponding to the provided `parentGroupName`.
	`<um:addUserToGroup>`	Adds the user corresponding to the provided `userName` to the group corresponding to the provided `parentGroupName`.
	`<um:changeGroupName>`	Adds the user corresponding to the provided `userName` to the group corresponding to the provided `parentGroupName`.
	`<um:createGroup>`	Creates a new `com.beasys.commerce.axiom.contact.Group` object.
	`<um:createUser>`	Creates a new persisted User object with the specified username and password.
	`<um:getChildGroupNames>`	Returns the names of any groups that are children of the given group.
	`<um:getChildGroups>`	Retrieves an array of `com.beasys.commerce.axiom.contact.Group` objects that are children of the Group corresponding to the provided `groupName`.
	`<um:getGroupNamesForUser>`	Retrieves a `String` array that contains the group names matching the provided search expression and corresponding to groups to which the provided user belongs.
	`<um:getParentGroupName>`	Retrieves the name of the parent of the `com.beasys.commerce.axiom.contact.Group` object associated with the provided `groupName`.

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

Library	Tag	Description
	`<um:getTopLevelGroups>`	Retrieves an array of `com.beasys.commerce.axiom.contact.Group` objects, each of which has no parent group.
	`<um:getUsernames>`	Retrieves a String array that contains the usernames matching the provided search expression.
	`<um:getUsernamesForGroup>`	Retrieves a `String` array that contains the usernames matching the provided search expression and correspond to members of the provided group.
	`<um:removeGroup>`	Removes the `com.beasys.commerce.axiom.contact.Group` object corresponding to the provided `groupName`.
	`<um:removeGroupFromGroup>`	Removes a child group from a parent group.
	`<um:removeUser>`	Removes the `com.beasys.commerce.axiom.contact.User` object corresponding to the provided `userName`.
	`<um:removeUserFromGroup>`	Removes a user from a group.
(Security)	`<um:login>`	Authenticates a user/password combination.
	`<um:logout>`	Ends the current user's WebLogic Server session. This is independent of the FlowManager's user session tracking, and should be used in combination with the `<um:login>` tag.
	`<um:setPassword>`	Updates the password for the user corresponding to the provided username.

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

Library	Tag	Description
Personalization Utilities	`<es:counter>`	Creates a `for loop` construct.
	`<es:date>`	Gets a date and time formatted string based on the user's time zone preference.
	`<es:forEachInArray>`	Iterates over an array.
	`<es:isNull>`	Checks to see if a value is null. If the value type is a `String`, also checks to see if the `String` is empty.
	`<es:monitorSession>`	Disallows access to a page if the session is not valid or if the user is not logged in.
	`<es:notNull>`	Checks to see if a value is not `null`. If the value type is a `String`, also checks to see if the `String` is not empty.
	`<es:simpleReport>`	Creates a two-dimensional array out of a simple query.
	`<es:transposeArray>`	Transposes a standard [row][column] array to a [column][row] array.
	`<es:uriContent>`	Pulls content from a URL.
WebLogic Utilities	`<wl:process>`	Provides a attribute-based flow control construct.
	`<wl:repeat>`	Used to iterate over a variety of Java objects, as specified in the set attribute.

Integration of External Components

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

Table 1-2 Useful External Components the Personalization Server

External Component	Out-of-the-Box Support	Methods and Notes
DBMS	Integrated and tested with Cloudscape, Oracle 8.1.6, 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:

  - 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
- Foundation 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 adivslets 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 Personalization EJB and its features via the public Advisor bean interface. (For more information, see the API documentation in theWebLogic Personalization Server *Javadoc*.) Think of the Advisor as sitting on top of the core services to provide a unified personalization API.

The Advisor recommends document content for the following items:

- Web content included or excluded as determined by a user's classification using rules-based matching against user profile information. For more information about classifying users, see "Classifying Users with the JSP <pz:div> Tag" on page 2-10 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 a personalized application, the Advisor can also provide information about user classifications. For example, an application can ask the Advisor if, based on predefined rules, the current user is classified as a *Premier*

Customer or an *Aggressive Investor*, and take action accordingly. The Advisor accomplishes this classification by gathering relevant user profile information, submitting it to the Rules 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 most useful to use the JSP tags when building typical pages. The tags provide ways to switch content on and off based on user classification, return content based on a static query, and match content to users based on rules that execute a content query. The JSP tags that perform these tasks are: `<pz:div>` , `<pz:contentSelector>,` and `<pz:contentQuery>`.
- **Using the Advisor session bean**. The page or application developer may use the Advisor session bean directly in place of the tags, if desired. The Advisor session 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

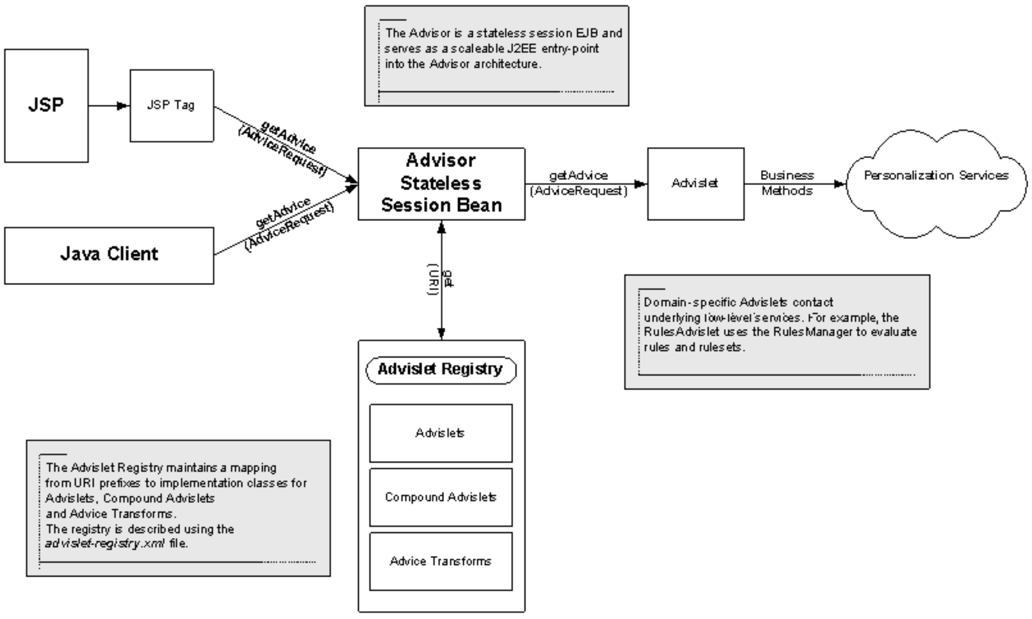
The Advisor is a stateless session EJB and has a simple interface with a `getAdvice` method on it. The `getAdvice` method returns `Advice` objects that contain the detailed result information that was returned from the personalization services.

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 `HttpSession` 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)
Advisor Stateless Session Bean
getAdvice (AdviceRequest)
Advislet
Business Methods
Personalization Services
Java Client
getAdvice (AdviceRequest)
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 Advislet uses the Advislet Registry to determine which Advislets to invoke in the processing of a request.

The WebLogic Commerce 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 versioning information as specified in the Advislet Registry.

Note: Unless otherwise indicated, all classes referenced here reside in the `com.bea.commerce.platform.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://` prefix, all of the work is done in one class—`com.bea.commerce.platform.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` which can be found in the WLCS root directory. 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.

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

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

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

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

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

  Note: For more information about how WebLogic Personalization Server manages content, see Chapter 8, "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 "Retrieving Documents with Content Selectors" in*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 topic "Creating a New Customer Segment" in the chapter "Using Customer Segments to Target High-Value Markets" in*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 in a Content Manager. It returns an array of `Content` objects that a developer can handle in numerous ways.

Note: For information about using the `<pz:contentQuery>` JSP tag, see "<pz:contentQuery>" on page 12-31. 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:

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

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

Note: For more information about these JSP tags, see "<cm:printProperty>" on page 12-11 and "<es:forEachInArray>" on page 12-76.

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 12-34.

For information about creating classifier rules, see the chapter "Using Customer Segments to Target High-Value Markets" in*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.

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

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

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: Refer to the API documentation in the *Javadoc* 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 to recommend content.

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 `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 sytem: 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 2-1 shows the logic the Advisor uses to determine how to map a recommendation request to an Advislet.

Table 2-1 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. ■ Selects content based on the content query obtained.
`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 (refer to the *Javadoc* for API details).

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

1. Look up an instance of the Advisor session bean.
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_SESSION` – the session object (retrieved from `P13NJspHelper.createP13NSession(HttpServletRequest)`).
   - `USER` – the user object (retrieved from `P13NJspHelper.createP13NProfile(HttpServletRequest)`).
   - `HTTP_REQUEST` – the request object (retrieved from `P13NJspHelper.createP13NRequest(HttpServletRequest)`).
   - `NOW` – a `java.sql.Timestamp` object representing *now.*
   - `RULES_RULESET_NAME` – (optional) the name of the segmentation rule to fire. (For more information about customer segment rules, see Chapter 3, "Introducing the Rules Manager," in this guide.)
4. Call the `getAdvise` method on the Advisor.
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 `not true`.

Note: If the optional Advise Request parameter `RULES_RULESET_NAME` 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 (refer to the *Javadoc* for API details).

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

1. Look up an instance of the Advisor session bean.
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.
5. The Advisor returns an instance of `Advice`. The `getResult` method is called to obtain the array of `Content` objects representing the recommendation.

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 (refer to the *Javadoc* for API details).

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

1. Look up an instance of the Advisor session bean.
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_SESSION` – the session object (retrieved from `P13NJspHelper.createP13NSession`).
   - `USER` – the user object (retrieved from `P13NJspHelper.createP13NProfile`).
   - `HTTP_REQUEST` – the request object (retrieved from `P13NJspHelper.createP13NRequest`).
   - `NOW` – a `java.sql.Timestamp` object representing *now.*
   - `RULES_RULESET_NAME` – the name of the segmentation rule to fire. (For more information about customer segments, see the chapter "Introducing the Rules Manager" 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.
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 Manager

Rules Management forms a key part of the personalization process by prescribing custom content to fit individual user profiles. 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?
  - Well-known Objects
  - How the Rules Engine Works
  - What Are Rule Sets?
  - Classifier Rules
  - Content Selector Rules
  - Debugging Rule Sets
- Configuring the Rules Framework
  - The RulesManager Deployment Descriptor
  - The rules-common.properties file

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.

The Rules Manager component of 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 tool with a graphical user interface (GUI) that allows Business Analysts to define classification and content selection rules, and a run-time service that matches users with content based on these rules.

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."

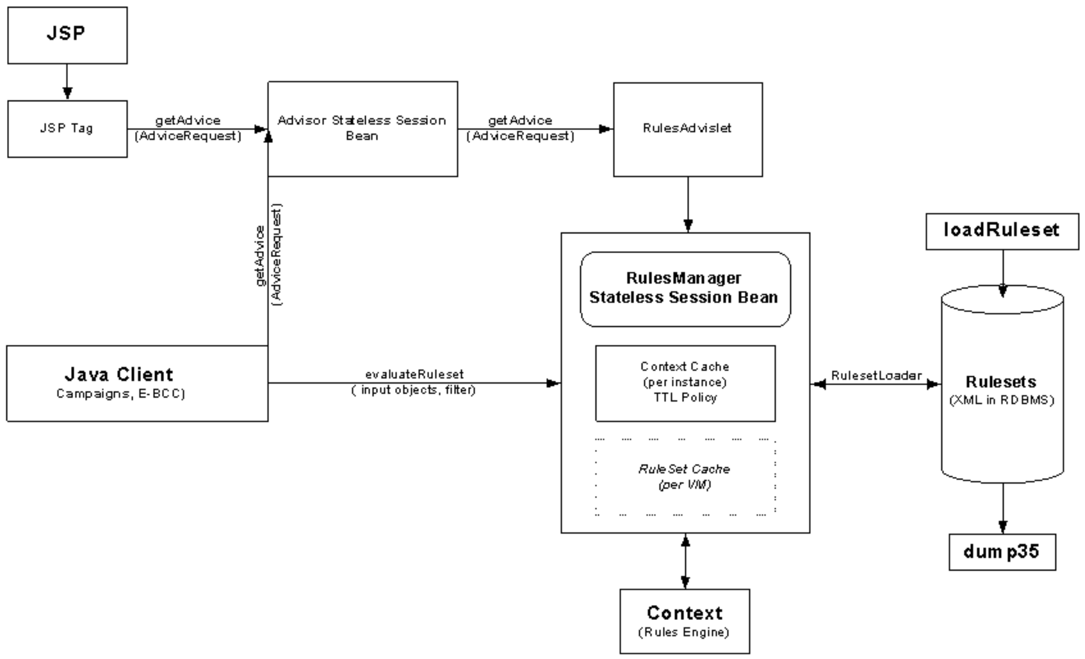
Well-known Objects

The Rules Management component uses several well-known objects:

- `ContentQuery`: This object describes the parameters of a query that is executed as a result of firing a content selector rule.
- `Now`: A well-known object in the rule editor, of type `(com.bea.commerce.platform.xml.schema)TimeInstant` that corresponds to the instant of a user request.
- `User`: For each call to the rules component, 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 expression 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. (For more information, see "Default Request Property Set" on page 5-10 in the "Foundation Classes and Utilities" chapter 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. (For more information, see the "Default Session Property Set" on page 5-12 in the "Foundation Classes and Utilities" chapter of this guide.)

Figure 3-2 The Rules Framework


JSP
JSP Tag
getAdvice (AdviceRequest)
Advisor Stateless Session Bean
getAdvice (AdviceRequest)
RulesAdvislet
getAdvice (AdviceRequest)
RulesManager Stateless Session Bean
Context Cache (per instance) TTL Policy
RuleSet Cache (per VM)
loadRuleset
Rulesets (XML in RDBMS)
RulesetLoader
Java Client Campaigns, E-BCC)
evaluateRuleset (input objects, filter)
dump35
Context (Rules Engine)

How the Rules Engine Works

The Rules Engine functions with a set of rules operating on objects in working memory. This working memory is first populated with input from the calling objects, and contains the cached user profile bean, among other things. 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 assert new objects into the working memory. For example, if our Classifier rule tests for `USER.age > 45`, then we might assert 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 the optional Object Filter. The filter can selectively ignore objects or mutate them.

The resultant objects, if any, are then returned to the Advisor.

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.

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 chapter “Using Customer Segments to Target High-Value Markets” in*Using the E-Business Control Center*.

Classifier rules dynamically categorize users into groups (user segments) using Boolean logic. 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 illustrate the logic involved in processing a classifier rule (note the implicit *and* between the rule phrases):

```
Classifier MiddleAgeMan
If User has the following characteristics:
    User.age > 35 AND User.age < 65
    and User.gender == "M" OR "male"
```

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

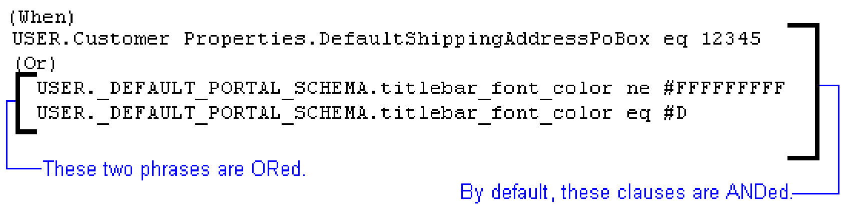
Classifier rules are the building blocks of more complicated rules. Content selector rules can use classifier rules as they select personalized content to match a user or group profile. (See “Content Selector Rules” below.)

Use the `<pz:div>` JSP tag to include a classifier rule in a JSP page. For a complete listing, see “<pz:div>” on page 12-39 in the “JSP Tag Library Reference” chapter of this guide.

The AND and OR operators

Figure 3-3 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.

Figure 3-3 AND and OR Example



```
(When)
USER.Customer Properties.DefaultShippingAddressPoBox eq 12345
(Or)
  USER._DEFAULT_PORTAL_SCHEMA.titlebar_font_color ne #FFFFFFFF
  USER._DEFAULT_PORTAL_SCHEMA.titlebar_font_color eq #D
```

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 chapter “Retrieving Documents with Content Selectors” in *Using the E-Business Control Center*.

Content selector rules construct queries on the fly and return content based on the user profile. This type of rule adds time and content components to the basic classifier rule and 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 include content selector rules in JSP pages. (See `<pz:contentSelector>` in the chapter “JSP Tag Library Reference” in this guide.)

For an in-depth look at using content selectors, refer to Chapter 4, “Working with Content Selectors,” in this guide.

Debugging Rule Sets

Note: The underlying structure of the Rules Engine has been greatly enhanced for WebLogic Personalization Server release 3.5. If you have created rules and rule sets in previous versions of this product, please refer to 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 functioning 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 set of components that use business rules to match users and groups with appropriate content are known collectively as the rules framework. Two files configure the properties of the rules framework:

- The RulesManager Deployment Descriptor
- The rules-common.properties file

The RulesManager Deployment Descriptor

Within the file `$WL_COMMERCE_HOME/config/wlscDomain/applications/wlscApp/rules.jar` is a file named `ejb-jar.xml` which specifies how WebLogic Personalization Server deploys the rules EJB.

The following <env-entry> element in the `ejb-jar.xml` deployment descriptor prevents differently deployed RulesManager beans from seeing each others' deployed rulesets.

```
<env-entry>
<env-entry-name>namespace</env-entry-name>
<env-entry-type>java.lang.String</env-entry-type>
<env-entry-value>default</env-entry-value>
</env-entry>
```

Rulesets are stored in the `RULESET` database table. Table 3-1 illustrates the contents of the `NAME` and `DOCUMENT` columns in the `RULESET` database table.

Table 3-1 Columns in the RULESET Database Table

NAME	DOCUMENT
default/MyRuleSetName	XML data for the RuleSet (as a CLOB)
default/MySecondRuleSet	XML data

By default, the namespace for Rule Sets is `default`, which is prepended to the RuleSet name. By modifying the value in the `<env-entry-value>` element that is described in this section, you can change the composite name that a RulesManager EJB stores in the database.

The rules-common.properties file

The rules-common.properties file is located in `$WL_COMMERCE_HOME/classes`. It configures the following properties:

- Rules Framework Debugging
- Rule Set TTL (time-to-live expiration policy)
- Rules Engine Listeners
- Rules Engine Expression Caching Optimizations
- Rules Engine Error Handling and Reporting
- JSP Tag Properties
- Rules Manager Properties
- Expression Evaluation Settings

Rules Framework Debugging

This property should normally be set to `false`.

```
# Rules framework debug flag:
#
# Set this property to true for rules framework debugging.  Defaults
# to false.
##

rules.framework.debug=false
```

Rule Set TTL

This property determines the time-to-live (in milliseconds) of the cached rule sets. In a single node deployment, this parameter should be set to -1 (no timeout); however, if the rules manager is deployed in a clustered environment, this parameter will determine the maximum elapsed time between updates of a rule set (via the E-Business Control Center) and propagation of the changes throughout the cluster.

```
##
# Rule set expiration TTL (in milliseconds):
#
# Set this property to -1 for infinite TTL.  Defaults to -1.
##

rule.set.expiration.ttl=-1
```

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=com.bea.commerce.platform.rules.i
nternal.engine.RulesEngineStatisticsListener
```

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 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.exception` parameter is set to `false`, no exceptions will be propagated, and any expression condition that generates an exception will evaluate to false. Otherwise, all exceptions, with the exception of those listed by 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
```

JSP Tag Properties

These are internal properties, and should not be modified.

```
##
# JSP Tag settings
##
# Filter class for ContentSelectorTag

contentselector.filter.class=com.bea.commerce.platform.content.ad
vislets.ContentQueryAdvice
# Filter class for DivTag
```

```
divtag.filter.class=com.bea.commerce.platform.user.Classification
```

Rules Manager Properties

These are internal properties, and should not be modified.

```
#
# RulesManager settings
#
# RulesManager JNDI name.

rules.manager.home.jndi=com.bea.commerce.platform.rules.manager.R
ulesManager

# RulesManager delegate class name.
#delegate.class.name=com.bea.commerce.qa.platform.rules.manager.i
nternal.RulesEvaluationDelegateStubImpl
```

Expression Evaluation Settings

The `rules.framework.comparator.nullcheck` property determines if an implicit null check is added to all expression comparisons. It should remain `true`.

The `rules.framework.comparator.epsilon` property determines the epsilon value for numeric equality and inequality comparisons. The *epsilon value* is an absolute value (rather than a percentage, etc.) and may be adjusted in accordance to equality precision requirements.

The `rules.framework.introspector.method.array.cache` property and `rules.framework.introspector.method.cache` property are internal properties and should not be modified.

```
##
# 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.
##

rules.framework.comparator.nullcheck=true
```

```
##
# Expression Comparator equality epsilon.
#
# The following property determines the epsilon value for
# numeric equality comparisons.
#
# Defaults to 0.
##

rules.framework.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.
##

rules.framework.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.
##

rules.framework.introspector.method.cache=true
```

CHAPTER

4 Working with Content Selectors

A content selector is one of several mechanisms that WebLogic Personalization Server 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 Personalization Server 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 BA 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 Commerce Business Engineer (CBE) creates content selector JSP tags and a set of other JSP tags that display the content the content selector retrieves in JSPs.

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, refer to "Methods for Retrieving and Displaying Documents" on page 8-4.

What Are Content Selectors?

Content selectors specify conditions under which they query the content management system for documents. They 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 an event that occurs on the page or has occurred previously on some other page, or the current date/time. For a complete list of conditions, refer to “Conditions That Activate Content-Selector Queries,” under “Retrieving Documents with Content Selectors” in *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 CBEs can use to tune the performance of the content selection process. CBEs create 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, CBEs can add attributes to the content selector JSP tag that cause WebLogic Personalization Server to store the array in a cache. Then, CBEs 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 CBE must use the `<es:forEachInArray>` tag. Depending on the scope of the cache, a `<es:forEachInArray>` can access a content-selector cache that WebLogic Personalization Server created for another page and for another user.

Using Content-Selector Tags and Associated JSP Tags

To use the content selector features on a given JSP, a CBE 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

Attributes of the <pz:contentSelector> Tag

While BAs use the E-Business Control Center to configure the dynamic properties of a content selector, a CBE 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, refer to "<pz:contentSelector>" on page 12-34 in the "Personalization Server JSP Tag Library Reference" chapter of 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 CBE 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.beasys.commerce.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 to the array, 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, define 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 CBE 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, refer to "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.beasys.commerce.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="textDocs"/>
<es:forEachInArray array="<%=textDocs%>" id="aTextDoc"
type="com.beasys.commerce.axiom.content.Content">

  <%  "<P>" + 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"))
{
  <%  "<P>" + 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 Personalization Server deploys the modifications. If you specified a page-check rate for your Web application, WebLogic Personalization Server waits for the page-check interval to expire before deploying any changes. For more information on setting the page-check interval, refer to the *Performance Tuning Guide*.

To Retrieve and Display Image-Type Documents

1. Determine the name of the attribute that your content management system uses to uniquely identify documents. This procedure assumes that your content management system uses an attributed named `docID`.

2. Open a JSP in a text editor.

3. 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.beasys.commerce.content.ContentHelper"%>
<%@ taglib uri="es.tld" prefix="es" %>
<%@ taglib uri="pz.tld" prefix="pz" %>
<%@ taglib uri="um.tld" prefix="um" %>
<%@ taglib uri="cm.tld" prefix="cm" %>
```

4. 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.

5. 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.beasys.commerce.axiom.content.Content">

     <img src="<cm:printProperty id="anImageDoc" rule="docID"
     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="<cm:printProperty id="anImageDoc" rule="docID"
     encode="URL">" >
   }
   %>
   ```

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

   WebLogic Personalization Server deploys the modifications. If you specified a page-check rate for your Web application, WebLogic Personalization Server waits for the page-check interval to expire before deploying any changes. For more information on setting the page-check interval, refer to 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.beasys.commerce.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" <pz:contentSelector
   rule="SpringSailing"
   contentHome="<%=ContentHelper.DEF_DOCUMENT_MANAGER_HOME %>"
   id="docs"/>

   <ul>
      <es:forEachInArray array="<%=docs%>" id="aDoc"
      type="com.beasys.commerce.axiom.content.Content">

      <li>The document title is: <cm:printProperty id="aDoc"
      rule="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 Personalization Server deploys the modifications. If you specified a page-check rate for your Web application, WebLogic Personalization Server waits for the page-check interval to expire before deploying any changes. For more information on setting the page-check interval, refer to 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 Personalization Server 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, refer to “Create and Configure the Cache to Improve Performance” on page 4-5.

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. Add tags to retrieve the data from the cache. For example, the following lines print a list of documents that are in the cache:

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

   <ul>
      <es:forEachInArray array="<%=SpringSailingDocs%>" id="aDoc"
      type="com.beasys.commerce.axiom.content.Content">

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

      </es:forEachInArray>

   </ul>
   ```

7. 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.

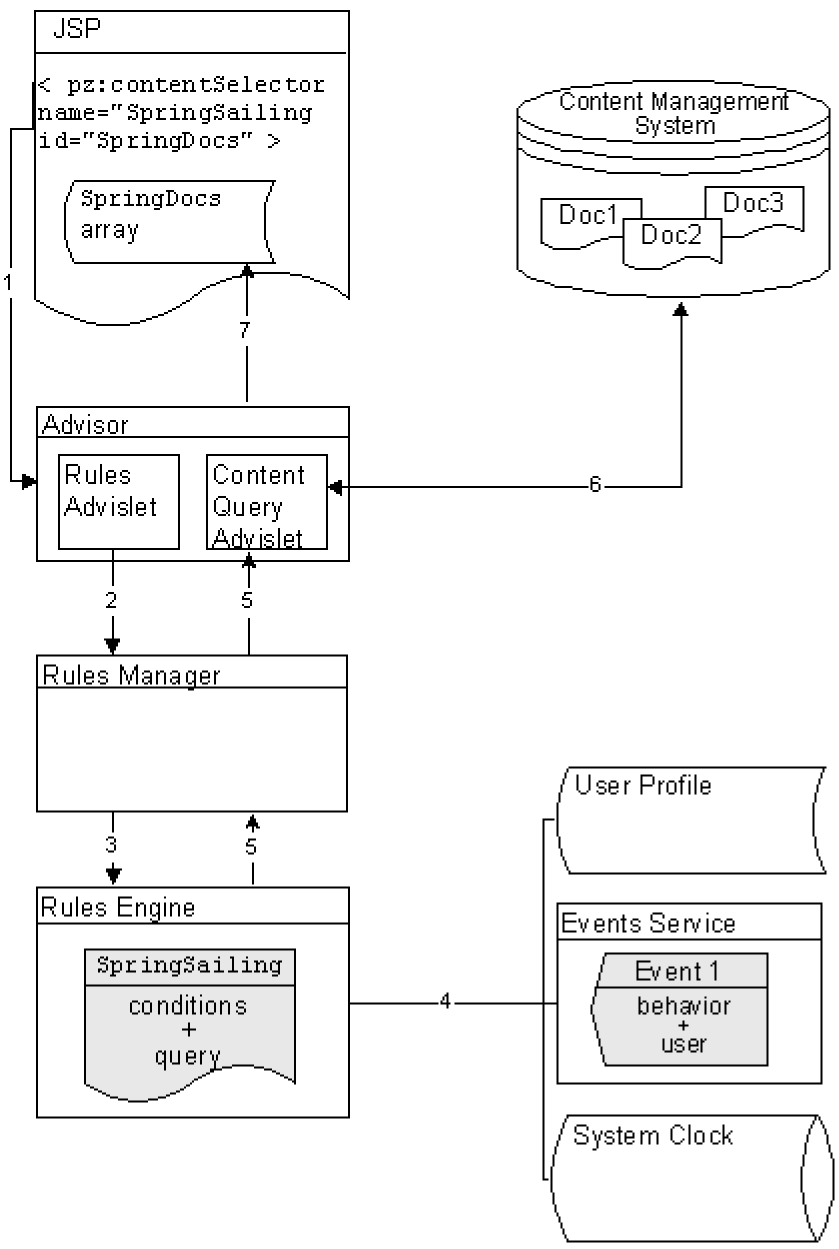
Note: For information about the Advisor, see Chapter 2, "Creating Personalized Applications with the Advisor."
For information about the Rules Engine, see Chapter 3, "Introducing the Rules Manager."

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 Events Service to determine if any events that conditions in the content selector specify have occurred.
   - 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. For more information, see "Create and Configure the Cache to Improve Performance" on page 4-5.

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

Figure 4-4 How Content Selectors Select Documents



For more information about using this tag, refer to “Using Content-Selector Tags and Associated JSP Tags” on page 4-3.

CHAPTER

5 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:

  - JspBase
  - P13NJspBase
  - ContentHelper
  - CommercePropertiesHelper
- Utilities in commerce.util Package
  - ExpressionHelper
  - TypesHelper

Flow Manager

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

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

Dynamic Flow Determination and Handling

The Flow Manager allows the determination of page routing to be centralized on the server based on an application's needs. To define properties of your unique application, you will create a property set of type `APPLICATION_INIT`. (See “Property Set Usage” on page 5-5.) There are three required values:

- destinationdeterminer – an implementation of the `com.beasys.commerce.foundation.flow.DestinationDeterminer` interface.

- destinationhandler – an implementation of the `com.beasys.commerce.foundation.flow.DestinationHandler` interface.
- ttl – how long (in milliseconds) before reloading the application init property set.

How the FlowManager Works

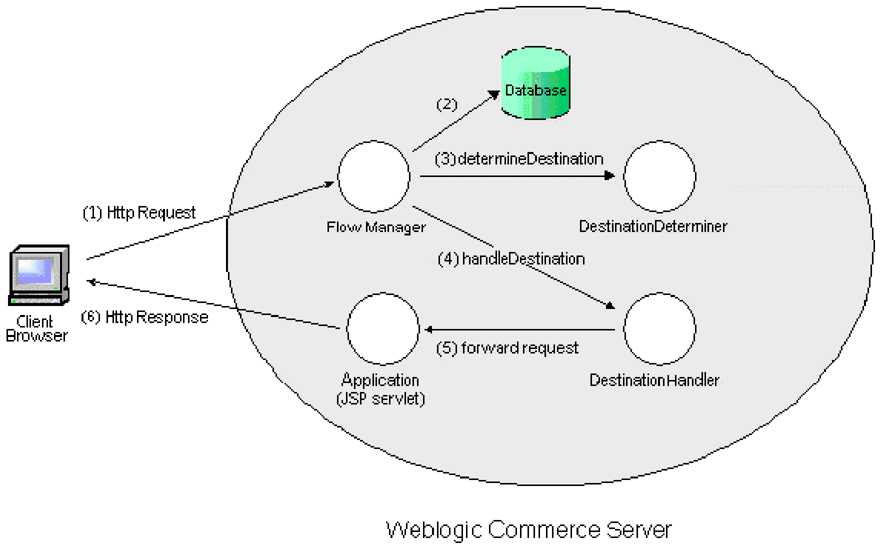
When WebLogic Personalization Server is installed, the Flow Manager servlet is registered with the WebLogic server in the `web.xml` file:

```
<servlet>
  <servlet-name>application</servlet-name>
  <servlet-class>com.beasys.commerce.foundation.flow.
  FlowManager</servlet-class>
</servlet>
```

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

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

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



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

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

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

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

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

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

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

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

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

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

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

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

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

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

Property Set Usage

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

The following three properties support the Flow Manager:

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

destinationdeterminer Property

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

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

destinatationhandler Property

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

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

ttl (time-to-live) Property

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

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

Creating a New Property Set

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

Set Parameters for Your Portal or Application

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

Webflow

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

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

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

Accessing Your Application via the Flow Manager

The URL for Acme Demo Portal accessed as a Web application is:
http://localhost:7501/exampleportal

The `exampleportal` portion of the URL is the context name for the Web application, as defined through the WebLogic console in the `config.xml` file.

```
<WebAppComponent Name="exampleportal" Targets="wlcsServer"
URI="exampleportal" ServletReloadCheckSecs="300" />
```

Within the Web application, the `web-inf/web.xml` file includes `<servlet>` and `<servlet-mapping>` entries for the Flow Manager, associating all URL accesses starting with `“application/*”` with the Flow Manager’s class.

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

Repository

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

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

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

HTTP Handling

Both the `<pz:div>` and `<pz:contentselector>` tag implementations send `HttpRequest` and `Session` information to the 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 Personalization Rules Service which uses these objects to execute classifier and content selector rules.

Personalization Request Object

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

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

Default Request Property Set

For Rules developers to write rules for classifier rules that contain conditions based on an `HttpRequest`, there must be a property set defined for the `HttpRequest`. By default, WebLogic Personalization Server ships with a default request property set for the standard `HttpRequest` properties. Developers adding properties to the request programatically will need to add those properties to the default property set in order for them to be available to the 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.beasys.commerce.axiom.p13n.http.Session`) before they can be set on the appropriate `AdviceRequest` (see the *Javadoc* API documentation). While the `HttpSession` object can be wrapped by directly calling the Personalization `Session` constructor, using the `createP13NSession` helper method on `P13NJspBase` (`com.beasys.commerce.axiom.p13n.jsp.P13NJspBase`)is recommended. See the *Javadoc* API documentation for more information.

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

Default Session Property Set

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

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

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

1. It first looks in its own cloned HTTP Session properties.
2. If it does not find the property, it locates the schema for the Personalization Session for the `scopeName` method parameter.

3. If it still does not find the property, it uses the *`scopeName`* parameter to find a schema entity for the `Session` schema group name and, if the schema is found, uses the default value in the schema.
4. If it still does not find the property, it uses the default value passed into the `getProperty` method call.

Utilities

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

JspHelper

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

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

JspBase

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

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

P13NJSPHelper

`P13NJspHelper` provides convenience methods to developers writing JSP pages (including but not limited to portals and portlets) that include personalized content. It provides methods for wrapping HTTP `Request` and `Session` objects into their personalization surrogates, and a method for retrieving the current Profile (User, Group, and so on) for an application.

P13NJspBase

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

ContentHelper

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

CommercePropertiesHelper

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

Utilities in commerce.util Package

ExpressionHelper

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

TypesHelper

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

CHAPTER

6 Creating and Managing Property Sets

Property sets are the schemas for personalization attributes. Using the Property Set Management tool, you can create property sets and define the properties that make up these property sets.

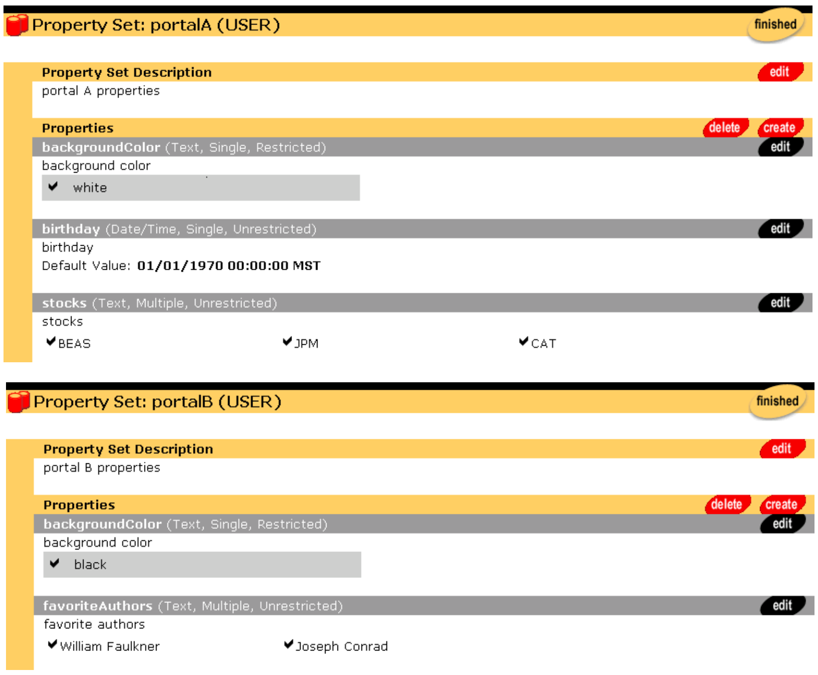
This chapter includes the following topics:

Overview of Property Sets

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 instance, Web site developers might want users to be able to specify different background colors for each of their portals by requiring the property "backgroundColor" for a user. By creating "portalA" and "portalB" property sets, the property "backgroundColor" can exist for both portalA and portalB. While the two "backgroundColor" properties have the same name, they could have the same or different definitions. Figure 6-5 shows two property sets with redundant property names, corresponding to unique definitions.

Figure 6-5 Property Sets Serving as Namespaces



Property Set: portalA (USER)

Property Set Description
portal A properties

Properties
backgroundColor (Text, Single, Restricted)
background color
✓ white

birthday (Date/Time, Single, Unrestricted)
birthday
Default Value: **01/01/1970 00:00:00 MST**

stocks (Text, Multiple, Unrestricted)
stocks
✓BEAS ✓JPM ✓CAT

Property Set: portalB (USER)

Property Set Description
portal B properties

Properties
backgroundColor (Text, Single, Restricted)
background color
✓ black

favoriteAuthors (Text, Multiple, Unrestricted)
favorite authors
✓William Faulkner ✓Joseph Conrad

A property definition includes the following information:

- Property Value Type: The data type of the property value, for example, Text, Integer, Float, or Date/Time. A property called `age` might be an Integer type, while `lastName` would be Text.
- Plurality: Whether the property can contain a single value, or multiple values. A property called `firstName` might be a single-valued property, while `childrenNames` would most likely be multivalued.
- Restriction: Whether the allowable values for a property are restricted. A property called `favoriteDayOfTheWeek` would only have seven possible values, while `email` would most likely be unrestricted.
- Default Property Value: Default values provided by the property set corresponding to the property. A property called `favoriteDayOfTheWeek` might have a default value of "Saturday." A property called `daysOff` might have the defaults "Saturday" and "Sunday."

For Personalization Server purposes, property sets are applied to six major areas.

1. **User and Group Profiles**

   The User/Group 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 portalA. Subsequent property retrieval for a particular user or group can then be scoped with this property set name to retrieve the user's background color for the portal. See Chapter 7, "Creating and Managing Users," for an in-depth discussion of how property retrieval works for users and groups.

2. **HTTP Sessions**

   The Session property set type is used for defining the property sets and properties that apply to HTTP sessions. Like the User/Group property set type, a "Session" property set type might be called "portalA." Properties available through this property set can then be accessed via the Advisor.

3. **HTTP Requests**

   The Request property set type is used for defining the property sets and properties that apply to HTTP requests. Again, like the "User/Group" property set type, a "Request" property set type might be called "portalA." Properties available through this property set can then be accessed via the Advisor.

4. **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. For more complete information on this subject, see Chapter 8, "Creating and Managing Content," in this guide.

5. **Application Initialization**

   The Application Init property set type uses default values to define application initialization parameters. These are the property sets used by the Flow Manager in support of portal (`DEFAULT_PORTAL_INIT`) and non-portal (`DEFAULT_APP_INIT`) based personalized applications. For more information about the Flow Manager, see Chapter 5, "Foundation Classes and Utilities," in this guide.

6. **Catalog Custom Attributes**

   You can define a property set that establishes custom attributes for a product item in the WebLogic Commerce Server catalog. For a given product item, a custom attribute that you define can be used in addition to the default attributes provided by WebLogic Commerce Server in the catalog database tables. For more information, see "Catalog Administration Tasks" in the *Guide to Building a Product Catalog*.

Creating a property set is a simple task via the Property Set Management tools. A name for the set must be provided as well as description. Properties can be copied from an existing property set if a pre-existing property set defines similar properties. Expanding the previous example, if portalA's properties have been defined and portalB is going to have the same (or similar) properties, then you can copy the properties from portalA's property set when creating portalB's property set. Finally, the type of property set ("User/Group", "Session", or "Request") must be chosen.

When defining a property, specify the following:

- Property name – the name of the property, such as `backgroundColor`.
- Description – a textual description of the property, perhaps describing the purpose of the property.
- Type – the data type of the property value. Data types supported by the administration tools are Text, Integer (equivalent to Long in Java),

Floating-Point number (equivalent to Double in Java), Boolean, and Date/Time (equivalent to java.sql.Timestamp).

- Selection option – determines whether the property is single-valued or multi-valued.
- Creation category – determines whether the possible values are restricted. Restricted property values are restricted to values listed in the property definition. Unrestricted property values have no such limitation.

The following table lists the property definition attribute and value.

Property Definition Attribute	**Attribute Value**
Name	Text (100 character length maximum)
Description	Text (255 character length maximum)
Type	Text, Integer (equivalent to Long in Java), Floating-Point Number (equivalent to Double in Java), Boolean, or Date/Time
Selection Option	Single-valued or multi-valued
Creation Category	Restricted or unrestricted
Default Value	Up to the user—can be `null`

Once created, User/Group 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.

Property Value Retrieval via ConfigurableEntity

Property Sets created with the administration tools are stored as `com.beasys.commerce.foundation.property.Schema` components. The component that acts as an "owner" of properties associated with Property Sets is the `com.beasys.commerce.foundation.ConfigurableEntity`. During inspection of the *Javadoc* for `Schema` and `ConfigurableEntity`, the reader may see the words "schema" and "scope" used interchangeably with "Property Set." Figure 6-6 shows a simplified representation of property value retrieval through a ConfigurableEntity. For the `ConfigurableEntity`, the value of backgroundColor for portalB has been overridden. The value of backgroundColor for portalA has not. Therefore, when backgroundColor is requested for the portalB property set, the overridden value, red, will be returned. When backgroundColor is requested for the portalA property set, the property set default value, white, will be returned.

Figure 6-6 backgroundColor Property Retrieval

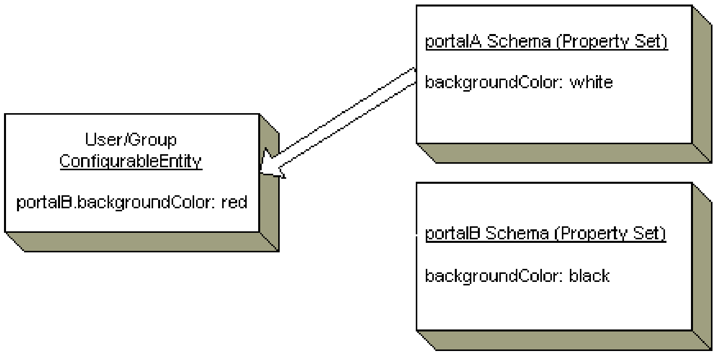

portalA Schema (Property Set)
backgroundColor: white
User/Group
ConfigurableEntity
portalB.backgroundColor: red
portalB Schema (Property Set)
backgroundColor: black

Figure 6-7 shows another simple example of backgroundColor property retrieval to demonstrate the notion of an explicit successor. A second ConfigurableEntity can be specified in the `ConfigurableEntity getProperty()` API that acts as a "backup" place to look for a particular property value. This second `ConfigurableEntity` is considered an explicit property successor. In this example, a particular group is used as an explicit successor, and the value for portalA's background color, green, is "inherited" from this successor.

Figure 6-7 Explicit Successor backgroundColor Property Retrieval

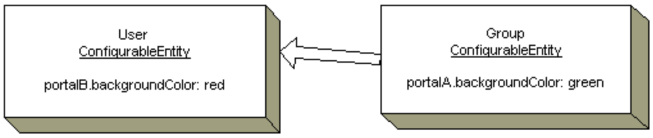
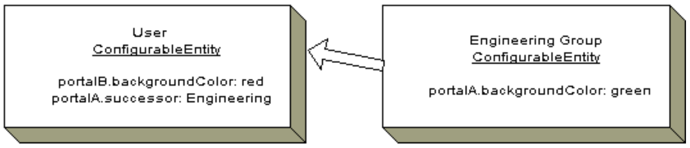


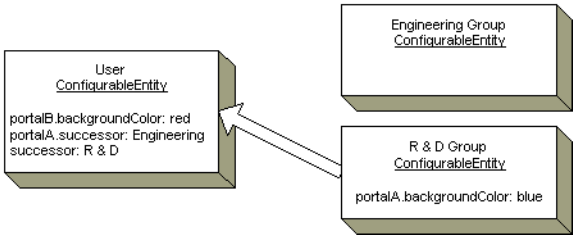
Figure 6-8 provides an example of an implicit successor. An implicit successor is a successor tied to a particular Property Set. In this case, the user does not have a value for portalA.backgroundColor, and no explicit successor is provided in the `getProperty()` call. However, the group has already been associated with the user as its successor for the portalA Property Set. Again, the user "inherits" the property value, green, from the group.

Figure 6-8 Property Inheritance Through Property Set-related Successor


User
ConfigurableEntity
portalB.backgroundColor: red
portalA.successor: Engineering
Engineering Group
ConfigurableEntity
portalA.backgroundColor: green

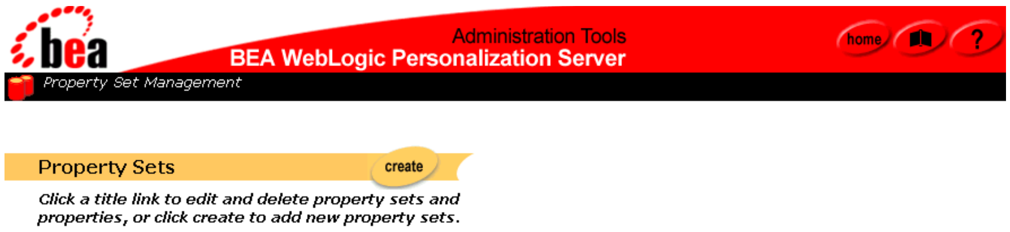
There also exists the notion of a default successor, which can be searched after an explicit successor and a Property Set-related successor have failed to return a value for the property. Figure 6-9 shows such a case. In this example, the Property Set-related successor cannot produce the necessary property value for backgroundColor in portalA, so the value must be retrieved from the default successor.

Figure 6-9 Property Inheritance Through a Default Successor



Keep in mind that these examples have been considerably simplified for brevity and to easily explain relevant concepts. More details of `ConfigurableEntity` property inheritance are available in the topic "Users and Groups" in the chapter *Creating and Managing Users*.

Using the Property Set Management Tool


bea
Administration Tools
BEA WebLogic Personalization Server
home
?
Property Set Management
Property Sets
create
Click a title link to edit and delete property sets and properties, or click create to add new property sets.

The Property Set Management tools allow you to create and manage sets of typed properties. Property Sets may be defined to describe user and group, session, request, and content properties.

Creating Property Sets

To create a property set:

1. On the Administration Tools Home page, click the Property Set Management icon. The Property Set Management Home page appears.
2. Click Create in the Property Sets banner. The Create Property Set page appears.

   To enter a new property set:

   a. Enter the name of the new property set in the Name field.

   b. Enter a description of the new property set in the Description field.

   c. Leave the Copy Properties From default as *Don't copy properties*.

   d. From the Property Set Type drop-down list, select a property set type.

   To copy properties from an existing property set into the new one:

   a. Enter the name of the new property set in the Name field.

   b. Enter a description of the new property set in the Description field.

c. From the Copy Properties From drop-down list, select the property set containing the properties you want copied.

3. Click Create to create the property set.

4. Click Back to return to the Property Set Management Home page.

Note: At any time, you can click Back to return to the Property Set Management Home page without saving the property set.

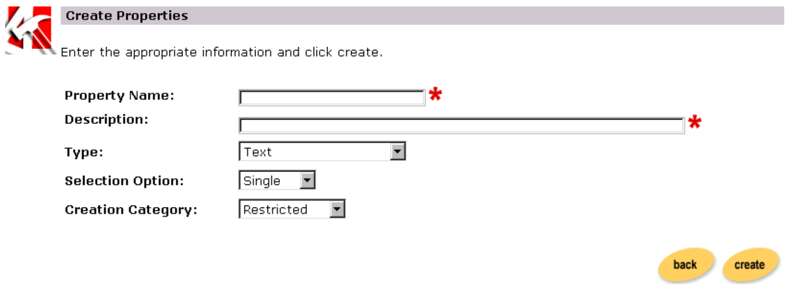
Creating Properties Within a Property Set

To create properties within a property set:

1. On the Administration Tools Home page, click the Property Set Management icon. The Property Set Management Home page appears.

2. From the Property Set list, click the title link for the property set to which you will add a property. The Property Set view page appears.

3. Click Create on the Properties bar. The Create Properties page appears.



Create Properties

Enter the appropriate information and click create.

Property Name:	*
Description:	*
Type:	Text
Selection Option:	Single
Creation Category:	Restricted

a. Enter the property name in the Property Name field.

b. Enter a description of the new property in the Description field.

c. Select the type from the Type drop-down list box.

d. Select option (single, multiple) from the Selection Option drop-down list box.

Note: The single option refers to those properties having only one option (for example, Property: Color, Attribute: red). The multiple option refers to those properties having multiple options (for example, Property: Colors, Attributes: red, green, blue, and so on).

e. Select the creation of category (Restricted, Unrestricted) from the Creation Category drop-down box.

Note: Restricted categories refer to values that are selected via a list, radio buttons, check boxes, and so on. Unrestricted categories refer to instances in which users populate a form field.

4. Click Create.
5. Click Back to return to the Property Set view.

Setting Up the Property Default Value

Notes: Different steps are required for setting up default values, given your option/category selection.

To set up the property default value for *single/restricted* categories:

1. From Property Set view, click Edit on the appropriate Property Description bar.
2. Click Edit on the Properties Values bar.
3. Enter a new value to the property in the New Value field.
4. Click Create. The new value appears in the Values matrix at the bottom of the page.
5. Indicate the default value(s) by selecting the appropriate radio button.
6. Click Create.

To set up the property default value for *single/unrestricted* categories:

1. From Property Set view, click Edit on the appropriate Property Set Description bar.
2. Click Edit on the Properties Values bar.
3. Enter a new value to the property in the New Value field.
4. Click Create.

To set up the property default value for *multiple/restricted* categories*:*

1. From Property Set view, click Edit on the appropriate Property Set Description bar.
2. Click Edit on the Properties Values bar.
3. Enter a new value to the property in the New Value field.
4. Click Create. The new value appears in the Values matrix at the bottom of the page.
5. Indicate the default value(s) by selecting the appropriate radio button(s).
6. Click Create.

To set up the property default value for *multiple/unrestricted* categories:

1. From Property Set view, click Edit on the appropriate Property Set Description bar.
2. Click Edit on the Properties Values bar.
3. Enter a new value to the property in the New Value field.
4. Click Save.

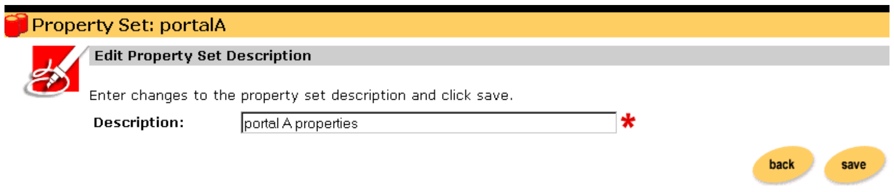
Editing Property Sets

To edit a property set:

1. On the Administration Tools Home page, click the Property Set Management icon. The Property Set Management Home page appears.
2. Click the appropriate title link from the Property Sets list. The Property Set view page appears.

To edit the Property Set Description:

1. Click Edit on the Property Set Description bar. The Edit Property Set page appears.



Property Set: portalA

Edit Property Set Description

Enter changes to the property set description and click save.

Description: portal A properties *

2. Enter the new description in the Description field.
3. Click Save to save changes. The general Property Set view appears with the new information. Alternately, click Back to return to Property Set view page without saving your changes.

Editing Properties Within a Property Set

To edit properties within a property set:

1. On the Administration Tools Home page, click the Property Set Management icon. The Property Set Management Home page appears.
2. Click the appropriate title link from the Property Sets list. The Property Set view page appears.
3. Click Edit on the appropriate property bar. The specific Property view page appears, containing information specific to the property you wish to edit.
4. Click Edit on the appropriate Description or Property Values bar. The Edit Property page appears.
5. Enter changes in the field(s) provided.
6. Click Save. The specific Property view returns. Alternatively, click Back. The specific Property view appears and your changes are not saved.

Deleting Property Sets

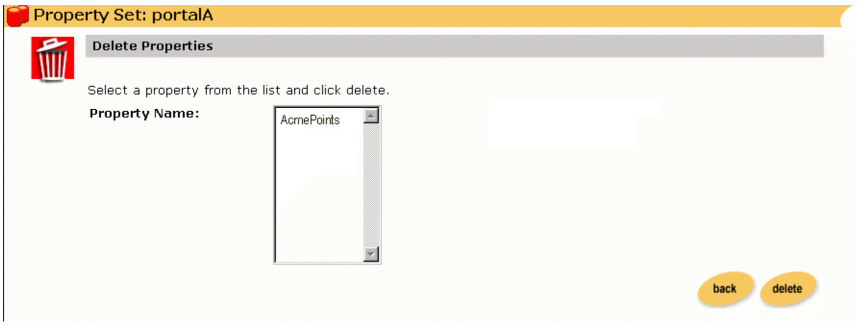
To delete a property set:

1. On the Administration Tools Home page, click the Property Set Management icon. The Property Set Management Home page appears.
2. Click the X to the right of the appropriate title link from the Property Sets list.
3. Click OK to confirm the deletion.

Deleting Properties

To delete properties:

1. On the Administration Tools Home page, click the Property Set Management icon. The Property Set Management Home page appears.
2. Select the appropriate title link from the Property Sets list. The general Property Set view appears.
3. Click Delete on the Properties bar. The Delete Properties page appears.



Property Set: portalA

Delete Properties

Select a property from the list and click delete.

Property Name: AcmePoints

4. Select a property from the Property Name list.
5. Click Delete.
6. Click OK to confirm the deletion. The specific Property view returns. Alternatively, click Back. The Property view appears and the property is not deleted.

CHAPTER

7 Creating and Managing Users

This chapter discusses how User Management combines enterprise data about users with profile data that is used to personalize the users' view of the application.

This topic includes the following sections:

- Overview of User Management
- Users and Groups
- Unified User Profiles
- Using WebLogic Realms
- Anonymous User Profiles
- Privacy Statement
- User Manager
- Using the User Management Tool
- Using the LDAP Realm
- Using Other Realms

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

Overview of User Management

The User Management system is a set of JSP tags, EJBs, and tools that facilitate the creation and persistence of user and group profile properties. It provides access to user profile information within a larger personalization server solution. In addition, the User Management system provides user-authentication mechanisms and user-to-group associations.

The User Management system responsibilities include:

- **User Authentication—**the user management system is used to authenticate a user against a persistent set of authentication information (typically a username-password combination).
- **User/Group Association Management—**the association of a user with one or more groups can play an essential role in determining user profile information pertinent to the user's session. The User Management system can either provide a default schema for user-group information persistence, or interface with existing user databases via standardized interfaces (for example, LDAP) or customized connectors.
- **User Profile Management**—the User Management system constructs the user profile from persisted user and group attributes. User attributes 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. The User Management system facilitates the creation and persistence of user profile properties.

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

Users and Groups

The two primary components employed by the WebLogic Personalization Server's User Management system are the User and Group, which extend ConfigurableEntity. It is from these components that User, and Group, and Unified User Profile functionality stems. User and Group components are also referred to as "user profiles" and "group profiles."

The fully qualified name of each object is as follows:

- User: `com.beasys.commerce.axiom.contact.User`
- Group: `com.beasys.commerce.axiom.contact.Group`
- ConfigurableEntity:
  `com.beasys.commerce.foundation.ConfigurableEntity`

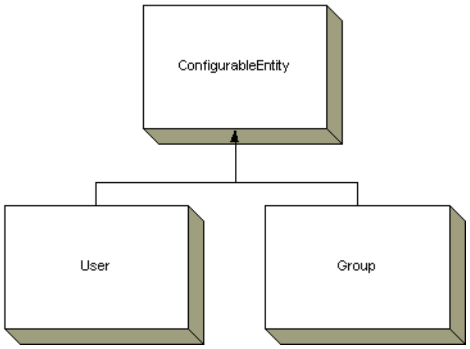
The User Management system works in conjunction with the WebLogic Server's security realm. In this arrangement, the security realm provides a list of users and groups, group membership information, and authentication. The User Management system uses the security realm to authenticate users and to know which users and groups exist and are valid, and which users are in a group. With this information from the security realm, it is possible for the User Management system to accomplish its primary duties: creating, retrieving, and managing user and group profiles complete with property data. A default security realm (User Management RDBMSRealm) is provided by the WebLogic Personalization Server as part of its "out-of-the-box" configuration.

Property data can be anything that is relevant to a user or group profile in the context of your personalized application. Things like age, gender, and favorite genres of music could all be property data. Things like department, position, and office location could also be property data. Much more is explained later about the actual and possible implementation details of handling property data in user and group profiles.

Group hierarchies permit property inheritance. For example, if a user profile does not yet have a "backgroundColor" property value, then the backgroundColor property value might be inherited from an "engineering" group. Groups may have only one or no parent group. As will be discussed later in this chapter, even if a realm for a third-party data store (for example, LDAP server) is used to access users and groups, any arbitrary group hierarchy may be configured for personalization purposes (property inheritance) via the User Management tools.

Profile functionality for both the User and Group components is inherited from the ConfigurableEntity implementation. Figure 7-10 shows a simplified representation of the User-Group-ConfigurableEntity relationship.

Figure 7-10 The User-Group-ConfigurableEntity Relationship



Unified User Profiles

In the BEA WebLogic Personalization Server, system users are represented by user profiles. A user profile provides an ID for a user and access to the properties of a user, such as age or e-mail address. Property values can be single-valued or multi-valued, and are requested via a `getProperty()` method which takes a property name as a key.

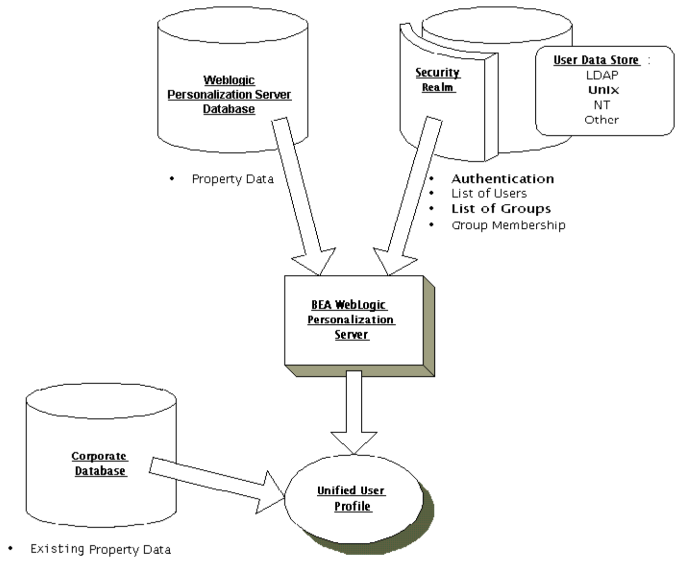
An advantage of the user profile is that it can be extended and customized to retrieve user information from an existing data source. For example, the user profile that ships with the WebLogic Personalization Server can combine user properties from the Personalization Server database with user properties from an LDAP server into a single user profile for use within an application. Developers and system users need not worry about the different underlying data sources. To them there is just one place to go for user information—the user profile.

The Unified User Profile (UUP) is the name used to describe this aggregation of properties from an existing data source and the WebLogic Personalization Server database tables into a single, customized user profile. More specifically, a UUP marries existing user/customer data by extending BEA's User component. By installing the WebLogic Personalization Server's database tables into the existing database instance and extending the provided `com.beasys.commerce.axiom.contact.User` implementation, developers can quickly create a customized UUP that retrieves and stores properties from/to the existing database. This powerful flexibility is desirable because it allows access to existing data without requiring data migration or disrupting existing applications that also use the data. Conversely, if it is more desirable to migrate existing data into a separate WebLogic Personalization Server database instance, this is also possible.

Configuration 1

Users and groups exist in some type of data store already, such as an LDAP directory. Existing user property data must be incorporated into the Unified User Profile as shown in Figure 7-11.

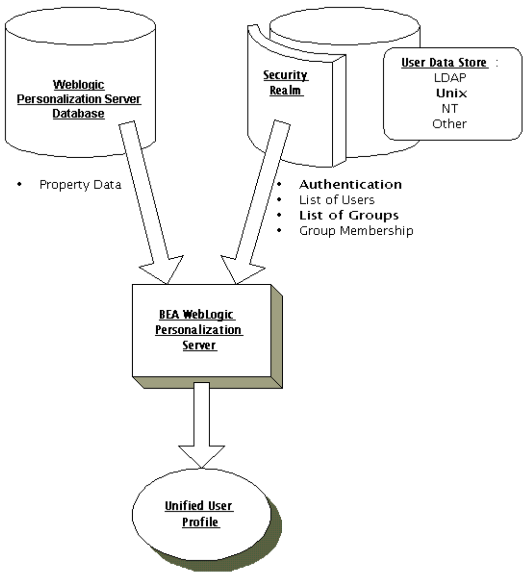
Figure 7-11 Configuration 1



Configuration 2

Users and groups already exist in a data store such as an LDAP directory. No existing user or group data must be incorporated into the Unified User Profile. All user and group property data is stored in the WebLogic Personalization Server's database tables as shown in Figure 7-12.

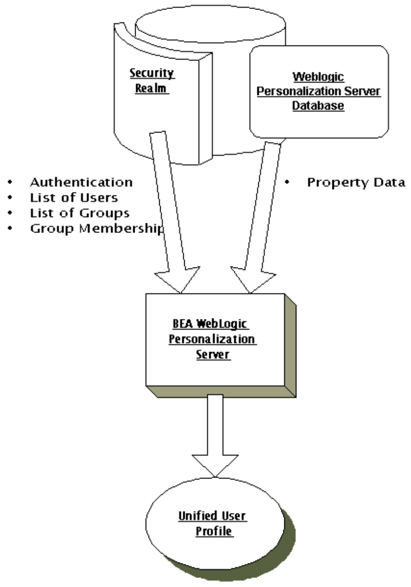
Figure 7-12 Configuration 2



Configuration 3

There is no existing store of users and groups. The WebLogic Personalization Server's database tables contain all user and group data as shown in Figure 7-13.

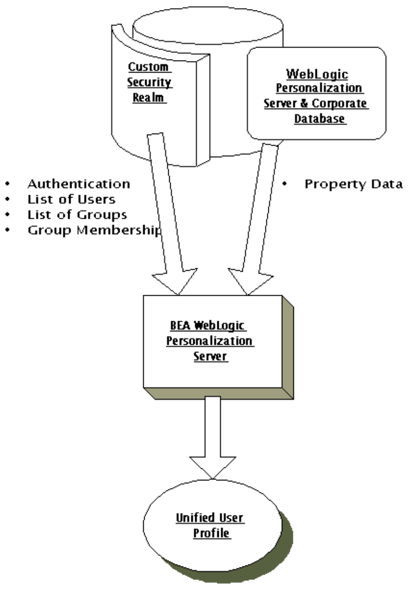
Figure 7-13 Possible Configuration 3



Configuration 4

User, group, and property data are in an existing database. Existing user property data must be incorporated into the Unified User Profile. A custom realm must be created in order to use the existing users and groups with the WebLogic Personalization Server as shown in Figure 7-14.

Figure 7-14 Possible Configuration 4



The UnifiedUser example, found at
`<install_dir>/config/wlcsDomain/applications/wlcsApp/defaultWebApp/examples/unifieduserprofile/index.html`
demonstrates a fictitious company's use of the UUP to take advantage of existing customer data. The UnifiedUser extends `com.beasys.commerce.axiom.contact.User` and retrieves data from a pre-existing database. If you have existing user information that you wish to leverage in your application, it is recommended that you study this example. The UnifiedUser shows how, with relative ease, you can create a customized UUP that suits your application's persistence needs.

Table 7-1 explains exactly what must be extended in order to create your own custom UUP.

Table 7-1 UUP Extensions

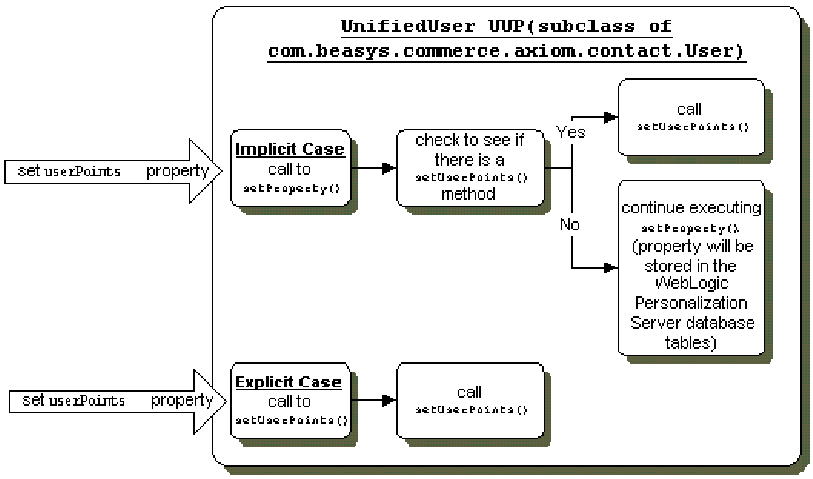
Object	Must Extend
UUP Primary Key	`com.beasys.commerce.axiom.contact.UserPk--with no key fields added.`
UUP EJB Interface	`com.beasys.commerce.axiom.contact.User`
UUP EJB Implementation	`com.beasys.commerce.axiom.contact.UserImpl`

Setting Properties Explicitly or Implicitly

The fact that UUPs are ConfigurableEntities means that user profiles have the notion of setting and getting a property explicitly or implicitly. Explicitly setting a property means calling a setter method for a property directly. Implicitly setting a property means setting a property via the `setProperty()` method where no explicit setter method is available. For example, if a UUP contains a "userPoints" property, calling `setUserPoints()` directly would explicitly set the `userPoints` property, while calling `setProperty()` with the "userPoints" key would implicitly set the `userPoints` property. When it is called, `setProperty()` will first look for a `setUserPoints()` setter method to call in the user profile. If such a setter method exists, this method is called and is responsible for setting the property and doing whatever else is necessary regarding that property's change in value. Ultimately it is the UUP implementation's responsibility to persist explicitly-set property values—even if they are implicitly called via `setProperty()`. ConfigurableEntity only handles persisting implicitly set properties where no explicit setter method exists.

Figure 7-15 diagrams both an explicit and implicit call to `setUserPoints()`. In both cases, it is the UUP bean's responsibility to handle storing the `userPoints` value. If no `setUserPoints()` method had existed in the UUP bean, the ConfigurableEntity implementation would have handled storing the `userPoints` value.

Figure 7-15 Implicit and Explicit Calls to Set the userPoints Property


UnifiedUser UUP(subclass of com.beasys.commerce.axiom.contact.User)
set userPoints property
Implicit Case call to setProperty()
check to see if there is a setUserPoints() method
Yes
No
call setUserPoints()
continue executing setProperty() (property will be stored in the WebLogic Personalization Server database tables)
set userPoints property
Explicit Case call to setUserPoints()
call setUserPoints()

This notion of implicitly and explicitly setting properties allows for additional flexibility in UUP implementation. If any special logic needs to happen during the setting or getting of a property, such as the recalculation of some other value, it can conveniently be done in a setter or getter method for that property. Functionality external to the UUP can always count on having a `setProperty()` method and a `getProperty()` method for access to properties, eliminating any need to know whether a property has its own setter or getter. For example, the `<um:getProperty>` JSP tag can always retrieve the `userPoints` property value even if a `getUserPoints()` method is the only way provided by the UUP to retrieve `userPoints`. This is because the UUP's `getProperty()` method will first check to see if it has a `getUserPoints()` method before checking elsewhere. Properties that have an explicit `set<PropertyName>()` and `get<PropertyName>()` method are referred to as "explicit properties," while properties that can only be set through a call to `setProperty()` are referred to as "implicit properties."

When implementing a custom UUP EJB, you only need to worry about implementing explicit getter and setter methods for the explicit properties you want the UUP to have. The implementations of these setters and getters then do whatever is necessary to set and retrieve the property values in the existing datastore.

There are a few important things to be aware of when creating a custom UUP. The `get<PropertyName>()`, `set<PropertyName>()` convention must be followed for all explicit property setting and getting in a UUP. This means if you have a UUP with an explicit `userPoints` property, you must provide an explicit `getUserPoints()`

method—`retrieveUserPoints()` would not work. Similarly, setting `userPoints` must be done with a `setUserPoints()` method. This is because the `getProperty()` and `setProperty()` methods look for getters and setters that follow this convention when getting and setting properties via implicit calls. Overriding `setProperty()` or `getProperty()` is not permitted—all getting and setting of explicit properties must be done through getter and setter methods. Explicit getters and setters must take and return objects—primitives such as long and float must be wrapped in java.lang.Long and java.lang.Float objects to be compatible with ConfigurableEntity's `getProperty()` and `setProperty()` methods.

Also, if you provide a getter method, it is a good idea to also provide a setter method and vice versa. This is because you can never predict when someone will try to set or get a property. For example, let's say you provide a getter that retrieves a property from a database table but no corresponding setter. If `setProperty()` is called for that property it will be stored in a WebLogic Personalization Server table. This is messy because you have the value being retrieved from one place and set in another. The next time the property is retrieved, it would have its original value—not the value that was set. If you want to provide a read-only property, you should implement an empty setter method.

The definition of ConfigurableEntity's `getProperty()` method is as follows:

```
public Object getProperty(String propertySet,
                          String propertyName,
                          ConfigurableEntity explicitSuccessor,
                          Object defaultValue);
```

The `getProperty()` method searches for properties in different places in a specific order which is important to understand. For example, if a property is not found for a User, perhaps a Group should be queried for the value. In this case the User would inherit the property value from a Group. In ConfigurableEntity terms, the Group would be the User's "successor." If a property is not found in a ConfigurableEntity, then the ConfigurableEntity's successor is queried for the value. This way ConfigurableEntities can inherit and override values from a parent entity. Successors can be implicit or explicit. An implicit successor is a ConfigurableEntity's default successor or a successor that is set for a specific Property Set. An explicit successor is a ConfigurableEntity that is passed as a parameter to the `getProperty()` method. Following is the order of the `getProperty()` property search as it exists in ConfigurableEntity, and hence the User and Group objects as well as any UUP objects:

1. Look for an explicit `getter` method for that property.

2. Look in the entity for the property for the specified Property Set.

3. Look in the entity for the property in the default (null) Property Set.
4. Look in the entity for the property in the Reserved Property Set (for properties from LDAP if using the LDAPRealm).

   Note: Properties to be retrieved from LDAP must be registered as LDAP attributes. See "Registering User Attributes for Retrieval from LDAP" on page 7-46.
5. Look for the property in the entity's explicit successor (if specified).
6. Look for the property in the entity's successor for the specified Property Set.
7. Look for the property in the entity's default successor.
8. Look for a default value as defined in the Property Set if the Property Set is specified (not null).
9. Return the deflectable passed into the `getProperty()` method.

The definition of ConfigurableEntity's `setProperty()` method is as follows:

```
public Object setProperty(String propertySet,
                          String propertyName,
                          Object value);
```

This method has a few details that are also important to understand. If `setProperty()` is used to set a property for a Property Set that is inconsistent with the property set's definition, an exception is thrown. For example, suppose we have defined a "UnifiedUserExample" Property Set that has a `userPoints` property of type Integer. If someone tries to set the `userPoints` property for the "UnifiedUserExample" Property Set to be "foo," an exception would be thrown because `userPoints` is defined as being of type Integer and "foo" is text. Similarly, setting a Boolean property value to "bar" would result in an exception because Boolean values are restricted to Boolean objects.

If `setProperty()` is called and `null` is passed for the Property Set, the property value is set in the `null` Property Set—referred to as the default Property Set. As described previously in the search order of `getProperty()`, the default property set is searched before looking for the property value in the "Reserved" Property Set and then a successor.

The "Reserved" Property Set is a read-only Property Set that is used to hold property values from an external datastore. The only time the "Reserved" Property Set is currently used in the WebLogic Personalization Server is when properties are retrieved from an LDAP directory. Attempting to set a property in the “Reserved” Property Set will result in an exception being thrown. Properties in the “Reserved” Property Set and the Reserved Property Set itself are not editable via the User Management tools. The User Management tools allow the specification of attributes to be retrieved from an LDAP server for users and groups.Only these attributes will be retrieved at run-time.

Properties can be set via `setProperty()` with a Property Set specified that does not exist. This is allowed, but strongly discouraged. When this is done, a Property Set is not created “on-the-fly” for the specified Property Set name. Rather, the specified Property Set name serves only as a namespace for the property. Similarly, it is allowed but strongly discouraged to set a property via `setProperty()` for an existing Property Set specifying a property that does not exist for that Property Set. Properties set in either of these ways are not editable through the User Management tools, but properties in the “null” (“default”) property set are editable from the tools.

A couple of additional points about `getProperty()` and `setProperty()` that are worth mentioning are as follow:

- `getProperty()` returns a java.lang.Long object if `setProperty()` is called passing a java.lang.Integer object value. Code retrieving such a property should be written as follows:

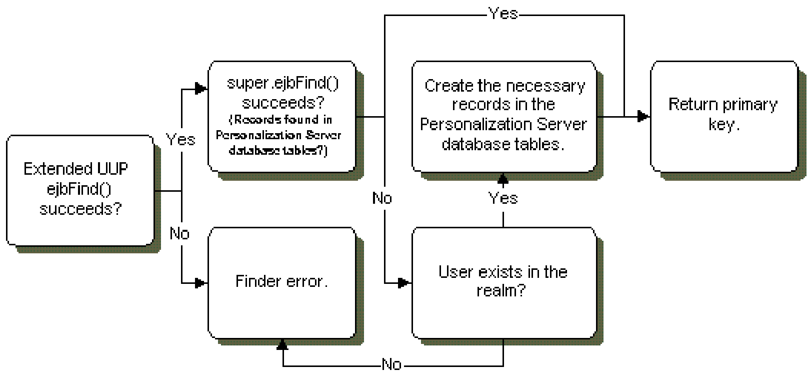
```
Object value     = myUser.getProperty(“my_property_set”,
                                     "my_integer_property",
                                      null,
                                      null);
                  Number tempNumber = (Number) value;
                   int    realValue  = tempNumber.intValue();
```

- `getProperty()` returns a java.lang.Double object if `setProperty()` is called with a java.lang.Float object. Code retrieving such a property should be written as follows:

```
Object value      = myUser.getProperty(“my_property_set”,
                                      “my_float_property”,
                                      null,
                                      null);
                 Number tempNumber = (Number) value;
                  float  realValue  = tempNumber.floatValue();
```

The `com.beasys.commerce.axiom.contact.User` object offers functionality for EJB find operations that makes integrating a UUP with the WebLogic Personalization Server easy. Once a UUP’s `ejbFind()` finds records in the existing data store, the call to `super.ejbFind()`—the User object `ejbFind()`—will create the necessary records for the UUP in the WebLogic Personalization Server tables if they do not yet exist and the following condition is met: If the User object `ejbFind()` fails, it checks the underlying security realm to see if the username corresponds to a valid user. If so, User's `ejbFind()` creates the necessary records, thereby eliminating finder errors and the need to spend time initially migrating user data into the WebLogic Personalization Server's User database tables (Figure 7-16).

Figure 7-16 Flow During an ejbFind() Operation


Extended UUP ejbFind() succeeds?
Yes
No
super.ejbFind() succeeds? (Records found in Personalization Server database tables?)
Finder error.
No
Yes
Create the necessary records in the Personalization Server database tables.
User exists in the realm?
Yes
No
Return primary key.

If your configuration is such that the realm cannot verify the existence of the user, but the user must be created, it is the responsibility of your EJB to create the superclass records if they are not found initially. The Unified User Example code demonstrates such a situation. Please refer to the `ejbFindByPrimaryKey()` method in the file `UnifiedUserBean.java`.

Six entries are required in the ejb-jar.xml file used when creating the unified user profile bean's descriptor. There entries are:

1. JNDIHomeName

   This environment entry is not to be confused with the actual JNDI lookup name of the extended EJB. Rather, it is used to relate profile entries for the UUP EJB with those of `com.beasys.commerce.axiom.contact.User.` The value must always be:

   `com.beasys.commerce.axiom.contact.User`

   Exact entry:

```
<env-entry>
  <env-entry-name>JNDIHomeName</env-entry-name>
  <env-entry-type>java.lang.String</env-entry-type>
  <env-entry-value>
  com.beasys.commerce.axiom.contact.User
  </env-entry-value>
</env-entry>
```

2. SchemaGroupName

   This environment entry is used to configure the EJB to pull property values from a particular classification of Property Sets. The value must always be:

   `USER`

   Exact entry:

```
<env-entry>
  <env-entry-name>SchemaGroupName</env-entry-name>
  <env-entry-type>java.lang.String</env-entry-type>
  <env-entry-value>USER</env-entry-value>
</env-entry>
```

3. SmartBMPClass

   This environment entry specifies which SmartBMP class to use when creating, refreshing, updating, and removing the EJB. If you have created a SmartBMP for your class which extends `com.beasys.commerce.axiom.contact.UserSmartBMP`, use the classname of your SmartBMP for this entry. If you do not use a particular SmartBMP with your class, use `com.beasys.commerce.axiom.contact.UserSmartBMP` as the value.

   Sample entry:

```
<env-entry>
  <env-entry-name>SmartBMPClass</env-entry-name>
  <env-entry-type>java.lang.String</env-entry-type>
  <env-entry-value>
    com.beasys.commerce.axiom.contact.UserSmartBMP
  </env-entry-value>
</env-entry>
```

4. EntityPropertyManagerHome

   This environment entry specifies which `EntityPropertyManager` bean to use when accessing user and group properties. If using the LDAP configuration (security realm is the LDAPRealm), the entry must be as follows.

   Exact Entry:

```
<env-entry>
  <env-entry-name>EntityPropertyManagerHome</env-entry-name>
  <env-entry-type>java.lang.String</env-entry-type>
  <env-entry-value>
com.beasys.commerce.foundation.property.EntityPropertyAggregator
  </env-entry-value>
</env-entry>
```

For any other configuration the `EntityPropertyManagerHome` entry should be specified as follows.

Exact Entry:

```
<env-entry>
  <env-entry-name>EntityPropertyManagerHome</env-entry-name>
  <env-entry-type>java.lang.String</env-entry-type>
  <env-entry-value>
    com.beasys.commerce.foundation.property.EntityPropertyManager
  </env-entry-value>
</env-entry>
```

The contents of the `ejb-jar.xml` file shipped with the Unified User Example are shown below. Note that this bean was not paired with its own SmartBMP implementation derived from UserSmartBMP.

5. PersistenceHelperPlugin

   This entry specifies which persistence helper class should be used by the BMP. If the standard UserSmartBMP is being used, the value should be “com.beasys.commerce.foundation.plugin.bmp.BMPPersistenceHelperPlugin”.

   Exact Entry:

```
<env-entry>
  <env-entry-name>PersistenceHelperPlugin</env-entry-name>
  <env-entry-type>java.lang.String</env-entry-type>
  <env-entry-value>
    com.beasys.commerce.foundation.plugin.bmp.BMPPersistenceHelperPlugin
  </env-entry-value>
</env-entry>
```

6. UnifiedProfileType

   This entry specifies the type of Unified Profile that this class belongs to. It is necessary to transparently create, edit, and delete UUP users through the admin tools. In the Unified User Example, the value is “Unified Profile Example”.

   Exact Entry:

```
<env-entry>
     <env-entry-name>UnifiedProfileType</env-entry-name>
     <env-entry-type>java.lang.String</env-entry-type>
     <env-entry-value>Unified Profile Example</env-entry-value>
</env-entry>

<ejb-jar>
    <enterprise-beans>
      <entity>
        <ejb-name>examples.usermgmt.UnifiedUser</ejb-name>
        <home>examples.usermgmt.UnifiedUserHome</home>
        <remote>examples.usermgmt.UnifiedUser</remote>
        <ejb-class>examples.usermgmt.UnifiedUserBean</ejb-class>
        <persistence-type>Bean</persistence-type>
        <prim-key-class>examples.usermgmt.UnifiedUserPk</prim-key-class>
        <reentrant>False</reentrant>
    <env-entry>
          <env-entry-name>JNDIHomeName</env-entry-name>
          <env-entry-type>java.lang.String</env-entry-type>

         <env-entry-value>com.beasys.commerce.axiom.contact.User</env-entry-value>
          </env-entry>
          <env-entry>
          <env-entry-name>SchemaGroupName</env-entry-name>
          <env-entry-type>java.lang.String</env-entry-type>
          <env-entry-value>USER</env-entry-value>
        </env-entry>
        <env-entry>
           <env-entry-name>SmartBMPClass</env-entry-name>
           <env-entry-type>java.lang.String</env-entry-type>

<env-entry-
value>com.beasys.commerce.axiom.contact.UserSmartBMP</env-entry-value>
        </env-entry>
        <env-entry>
        <env-entry-name>EntityPropertyManagerHome</env-entry-name>
          <env-entry-type>java.lang.String</env-entry-type>

<env-entry-
value>com.beasys.commerce.foundation.property.EntityPropertyAggregator</env-entry-
value>
        </env-entry>
        <env-entry>
           <env-entry-name>PersistenceHelperPlugin</env-entry-name>
           <env-entry-type>java.lang.String</env-entry-type>

<env-entry-
value>com.beasys.commerce.foundation.plugin.bmp.BMPPersistenceHelperPlugin</env-en
try-value>
        </env-entry>
        <env-entry>
           <env-entry-name>UnifiedProfileType</env-entry-name>
           <env-entry-type>java.lang.String</env-entry-type>
           <env-entry-value>Unified Profile Example</env-entry-value>
        </env-entry>

        <resource-ref>
           <res-ref-name>jdbc/commercePool</res-ref-name>
           <res-type>javax.sql.DataSource</res-type>
           <res-auth>Container</res-auth>
        </resource-ref>

          </entity>
    </enterprise-beans>
    <assembly-descriptor></assembly-descriptor>
  </ejb-jar>
```

Additionally, the following entry must be added to the `weblogic-ejb-jar.xml` file of the UUP so that it can access the commercePool database connection pool:

```
<weblogic-enterprise-bean>
  [...]

      <reference-descriptor>
         <resource-description>
            <res-ref-name>jdbc/commercePool</res-ref-name>
            <jndi-name>weblogic.jdbc.jts.commercePool</jndi-name>
         </resource-description>
      </reference-descriptor>
  [...]
</weblogic-enterprise-bean>
```

The last step in completing a custom UUP requires the UUP to be registered with the WebLogic Personalization Server through the User Management tools. In order to register the UUP, the User Management tools require the following:

Item	Description
Profile Type Name	Arbitrary name that is later used to refer to the profile type through the User Management system's `<um:getProfile>` JSP extension tag.
Profile Home Class	The home class of the new profile type.
Profile Remote Interface	The remote interface of the new profile type.
Profile Primary Key Class	The primary key class of the new profile type.
Profile JNDI Name	The JNDI lookup name of the new profile type.

By registering the UUP with the WebLogic Personalization Server, it becomes possible to ask for the new profile type with the `<um:getProfile>` JSP tag:

```
<um:getProfile profileType="UnifiedUserExample"
profileKey="<%=username%>"/>
```

It is then possible to use the `<um:getProperty>` and `<um:setProperty>` JSP tags with the UUP.

Using WebLogic Realms

A realm is a Java class that provides access to a store of Users, Groups, ACLs (access control lists), and related services. WebLogic Server uses a realm as a service, calling into the realm to retrieve Users, Groups, and ACLs as Java objects. WebLogic Server provides realms that access the WebLogic Server properties file, Windows NT, or UNIX networks, and LDAP servers for user, group, and ACL information. The WebLogic Personalization Server provides an additional RDBMSRealm which uses its own database tables containing user and group information as an out-of-the-box option. It is also possible to create your own realm if your situation requires accessing a datastore not supported by WebLogic Server.

The WebLogic Personalization Server must have access to a realm to retrieve information about users and groups, determine a group's members, and authenticate users. By depending on realms, the WebLogic Personalization Server can use existing stores of user and group information, allowing that information to remain in place. For instance, if you already have users and groups defined in an LDAP directory, they can be accessed by the WebLogic Personalization Server through the LDAPRealm without requiring any redundant data entry.

If you are using the WebLogic Personalization Server without an external data store of user and group information, then that information will be stored in the Personalization Server's database tables. In this case, the `com.beasys.commerce.axiom.contact.security.RDBMSRealm` must be used to access user and group information from the WebLogic Personalization Server tables. For this configuration to work, the appropriate realm properties for your database type must exist in the `commerce.properties` file.

Ensure Properties Are Set in the BEA WebLogic Personalization Server's commerce.properties File

If Using the WebLogic Oracle OCI Driver:

```
commerce.usermgmt.RDBMSRealm.driver=weblogic.jdbc.oci.Driver
commerce.usermgmt.RDBMSRealm.dbUrl=jdbc:weblogic:oracle
commerce.usermgmt.RDBMSRealm.dbServer=<machine name>
commerce.usermgmt.RDBMSRealm.dbUser=<database user>
commerce.usermgmt.RDBMSRealm.dbPassword=<database user's password>
```

If Using Cloudscape:

```
commerce.usermgmt.RDBMSRealm.driver=COM.cloudscape.core.
JDBCDriver
commerce.usermgmt.RDBMSRealm.dbUrl=jdbc:cloudscape:Commerce;\
                                   create=true;autocommit=false
commerce.usermgmt.RDBMSRealm.dbUser=none
commerce.usermgmt.RDBMSRealm.dbPassword=none
```

```
config/wlcsDomain/applications/wlcsApp/defaultWebApp/examples/uni
fieduserprofile/index.html
```

Verify That the Realm Is Active

To verify that WebLogic Server is configured to use this realm, follow these steps:

1. Open up the WebLogic Console in a browser.
2. Expand **wlcsDomain->Security->Realms**.
3. Verify that there is a realm (by default wlcsRealm) defined there. If not, create it.
4. Verify that the realm class name for that realm is `com.beasys.commerce.axiom.contact.security.RDBMSRealm`. If not, update it and click apply.
5. Expand **wlcsDomain->Security->Caching Realms**.
6. Verify that there is a caching realm (by default wlcsCachingRealm) defined there. If not, create it.
7. Verify that the basic realm for that caching realm is wlcsRealm. If not, update it and click apply.

8. Expand **wlcsDomain->Security**.
9. Verify that the Caching Realm specified is wlcsCachingRealm. If not, update it and click apply.
10. If any changes needed to be made in these steps, you must restart the server for them to take effect.

Implementing a New Custom Realm

It is important to note that if a realm other than the WebLogic Personalization Server's RDBMSRealm is being used, the administration tools for creating users and groups become inaccessible. This is because adding users and groups and administering credentials must be done through tools provided by the external datastore.

For use within the WebLogic Personalization Server, a realm must be a subclass of `weblogic.security.acl.AbstractListableRealm`. The WebLogic NTRealm, LDAPRealm, and UnixRealm are all subclasses of AbstractListableRealm.

Tools are provided that allow a properly-configured realm to be set up for use by the WebLogic Personalization Server. The realm configuration tools allow you to choose which groups from the realm you wish to use in the WebLogic Personalization Server, map group names that have changed in the realm to new group names, and clean up Personalization Server records that no longer correspond to valid realm users or groups.

Note: Changing the underlying realm can cause unpredictable behavior if the realm configuration tools are not immediately used to map and remove groups and clean up users as appropriate for the new realm.

In addition to user and group information, realms may also provide ACLs to determine an authenticated user's permissions within the system. An ACL guards an object or service in WebLogic Server. ACLs can guard servlets and JSP pages, JMS queues and topics, EJBs, JDBC connection pools, JNDI contexts, and ZAC packages. You can also create custom ACLs for use in your applications, and these ACLs will be supported by the WebLogic Personalization Server.

An ACL holds a list of AclEntries, each with a set of permissions for a user or group. A permission is an action that can be performed on the protected resource—for example, “execute,” “lookup,” “read,” or “write.” The exact permissions available depend on the type of resource the ACL protects. For example, a servlet requires “execute” permission, and a JMS queue requires “read” or “write” permission.

For more information on realms, including how to configure and administer realms, consult the WebLogic Server documentation for Using WebLogic Realms and ACLs. Also, for more information on implementing a custom realm, see the WebLogic Server documentation.

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`*`">`. For more information on these tags, see the topic “User Management JSP tags” in Chapter 12, “Personalization Server JSP Tag Library Reference.”

For an example, see
`%WL_COMMERCE_HOME%/server/public_html/anonymousprofile/index.html`

Privacy Statement

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.

User Manager

The UserManager Session EJB provides user management functionality in a WebLogic Personalization Server-specific context. Services provided by the UserManager include:

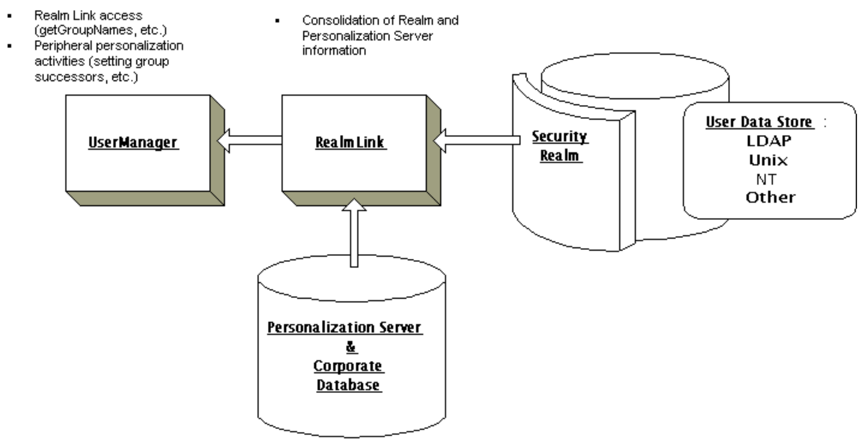
- Creating/removing users
- Creating/removing groups
- Adding users to groups/removing users from groups
- Adding groups to groups/removing groups from groups
- Retrieving usernames corresponding to a group
- Retrieving group names corresponding to a user
- Retrieving unique group and user IDs based on group/username
- Retrieving group/username based on unique ID
- Retrieving user/group objects based on name

For a complete list of UserManager services, please refer to the UserManager *Javadoc*.

Though it supplies the underlying functionality of the Group/User management JSP extension tags, the UserManager can be accessed directly. However, the UserManager is not intended for use outside the context of the WebLogic Personalization Server. To emphasize this point, the general relationship between the UserManager and the security realm support mechanism will be briefly explained, followed by a few examples.

Figure 7-17 shows the relationship between the UserManager, the RealmLink, and the security realm. The RealmLink is used to ensure that realm query results are consistent with WebLogic Personalization Server user and group data. The RealmLink is the only object aware of both the WebLogic Personalization Server data, and the Realm user and group data. An example of RealmLink activity is the query for group names associated with a particular user. Since the user manager administration tools allow for group registration with the WebLogic Personalization Server, the RealmLink will only return group names for a particular user that exist in both the security realm and in the WebLogic Personalization Server tables.

Figure 7-17 UserManager/RealmLink Cooperation


Realm Link access (getGroupNames, etc.)
Peripheral personalization activities (setting group successors, etc.)
Consolidation of Realm and Personalization Server information
UserManager
RealmLink
Security Realm
User Data Store :
LDAP
Unix
NT
Other
Personalization Server & Corporate Database

To ensure behavior consistent with WebLogic Personalization Server purposes, the UserManager employs two primary strategies:

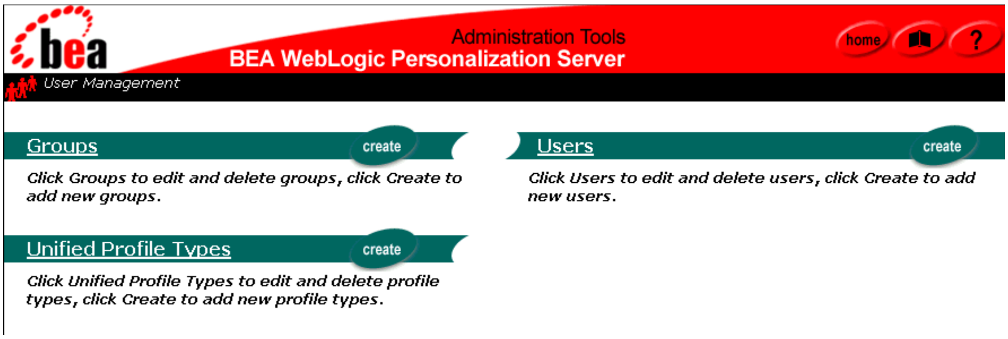
1. For certain operations, the UserManager qualifies the security realm being used before taking action. These operations can only be performed if the current security realm class is `com.beasys.commerce.axiom.contact.security.RDBMSRealm.` See UserManager EJB in the *Javadoc* for details.

   For example, the `createGroup()` method throws a `UserManagementException` if the out-of-the-box RDBMSRealm is not being used. The logic behind such an exception is that the UserManager is designed to work with the default Personalization database schema. If another realm is being used (for example, WebLogic LDAPRealm), it is assumed that the client has another means, besides the WebLogic Personalization Server Administration Tools, that should be used for adding and removing groups and users to/from the realm.

2. For all operations, the UserManager works in conjunction with the `com.beasys.commerce.axiom.contact.security.RealmLink` class to ensure results consistent with both security realm and WebLogic Personalization Server user and group data.

   For example, the `getGroupNamesForUser()` method returns only group names which exist in the current security realm and which are registered with the WebLogic Personalization Server via the Realm Configuration tools.

Using the User Management Tool



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 Unified Profile Types to edit and delete profile types, click Create to add new profile types.

The User Management Administration Tools allow you to create and associate users and groups or to link to and use existing directories of users. A user or group may then be personalized by overriding property values as defined in the Property Set Management tool. The Unified Profile Types tool allows you to configure access through User Management tag libraries to your existing application EJBs.

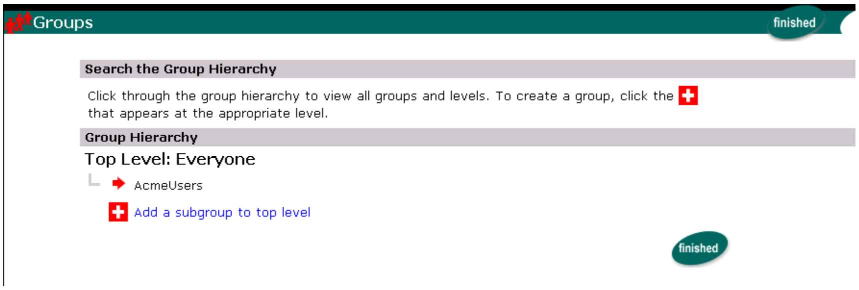
Note: If your system is configured for a third-party realm, the interface above would contain a Realm banner in addition to the ones presented and an LDAP banner if you are using the LDAPRealm. In addition, the Create buttons would not appear on the Users or Groups banners.

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. On the Administration Tools Home page, click the User Management icon. The User Management Home page appears.
2. On the User Management Home page, click Create in the Groups banner. The Create a New Group page appears.



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: Everyone

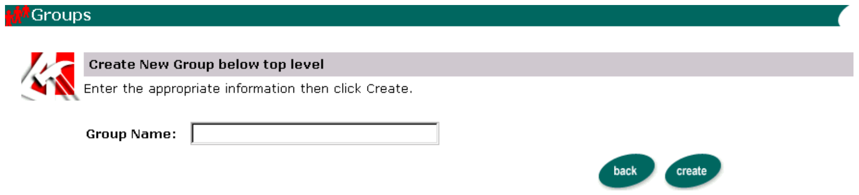
AcmeUsers

Add a subgroup to top level

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

4. Enter the name of the new group in the Group Name field.

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



Create New Group below top level

Enter the appropriate information then click Create.

Group Name:

6. Click Back to return to the Group Administration Tool or to enter another new group name (step 4).

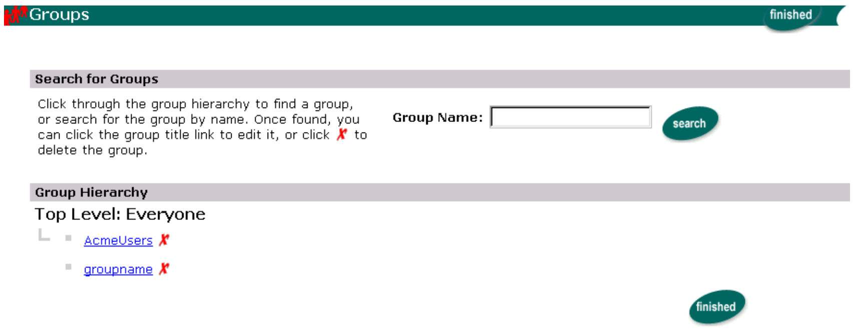
Note: The administration tools do not allow the creation of a user with username "system" or "guest" or a group called "everyone," as these are reserved WebLogic Server terms.

Deleting Groups

To delete groups:

1. On the Administration Tool Home page, click the User Management icon. The User Management Home page appears.

2. On the User Management Home page, click Groups in the Groups banner. The Search for Groups tool appears.



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: Everyone

- AcmeUsers ✗
- groupname ✗

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

   Note: The group name must be entered exactly.

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

3. Click the X to the right of the group name. A confirmation box appears.

4. Select OK. The group is deleted.

Adding Users to Groups

To add users to groups:

1. On the Administration Tools Home page, click the User Management icon. The User Management Home page appears.

2. On the User Management Home page, click Groups in the Groups banner. The Search for a Group page appears.

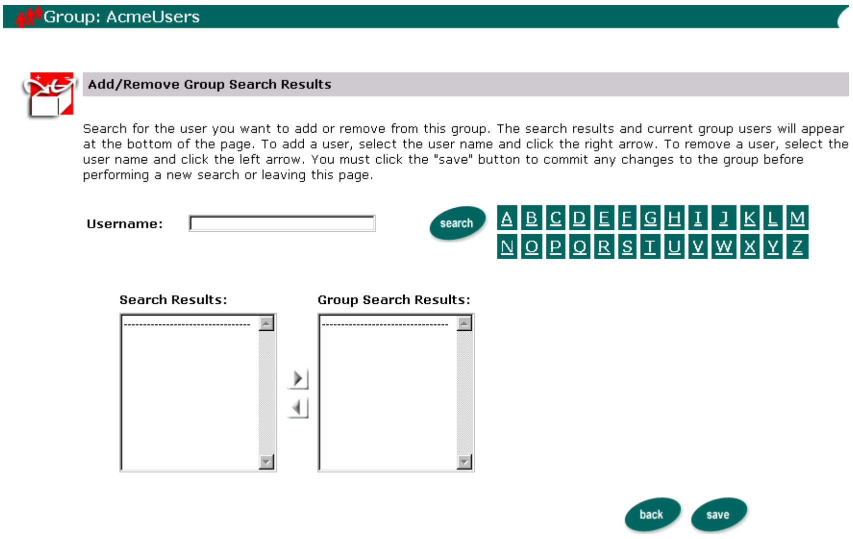
   To locate the appropriate group, do one of the following:

   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.

3. Select the group. The Group Properties view appears.

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



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.

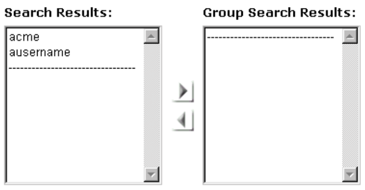
To locate a user, do one of the following:

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.

5. Select the username, or a group of names, from the Search Results field.



acme
ausername

6. Click the left-to-right directional arrow. The username(s) appears with the Group Users field.
7. Click Save.
8. Click Back to return to the Group Properties view.

Note: The search applies both list boxes.

Removing Users from Groups

To remove users from groups:

1. On the Administration Tools Home page, click the User Management icon. The User Management Home page appears.
2. On the User Management Home page, click Groups in the Groups banner. The Search for Groups tool appears.

   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.
3. Select the group. The Group Properties view appears.
4. Click the add/remove icon (+/-) at the bottom of the page. The Add/Remove Users tool appears.

   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.

5. Select the username, or a group of usernames, from the Group Users field.
6. Click the right-to-left directional arrow. The username(s) is removed from the Group Users field and appears in Search Results.
7. Click Save.
8. Click Back to return to the Group Properties view.

Editing Group Property Values

To edit group property values:

1. On the Administration Tools Home page, click the User Management icon. The User Management Home page appears.
2. On the User Management Home page, click Groups in the Groups banner. The Search for a Group page appears.

   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.

3. Select the group. The Group Properties view appears.

4. Select or search for a property set to view for this group. For specific instructions on property set management, see Chapter 6, "Creating and Managing Property Sets." The group's default property values appear if no other property set has been accessed during the tools session.
5. Click Search.
6. Click Edit on the appropriate Property bar. The associated Edit Property Values page appears.
7. Change the values on the Edit Property Values page.
8. Click Save.
9. Click Back to return to the Group Properties view.
10. Return to step 4 and edit other properties as necessary.

Notes: Non-default Property sets and properties not configured through the Property Set Management tools are not editable here.

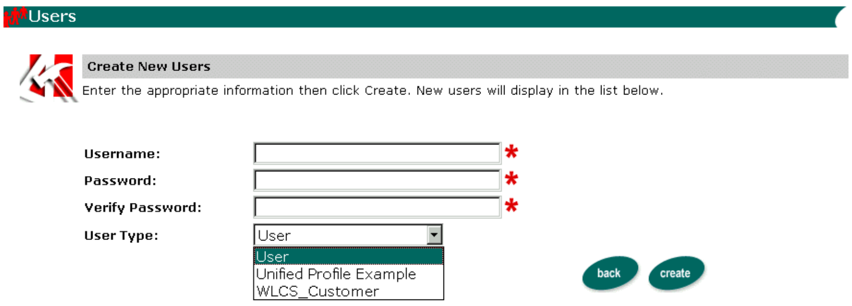
If you click the Reset button on the Property bar (instead of Edit as we did in step 6), the property is set to null for that user. This will have one of three results:

- First, if the property has a default value, the 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.

Creating User

To create users:

1. On the Administration Tools Home page, click the User Management icon. The User Management Home page appears.
2. On the User Management Home page, click Create in the Users banner. The Create New Users page appears.



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
Unified Profile Example
WLCS_Customer

3. Enter the username in the Username field.

   Note: Limit usernames to 25 characters.
4. Enter the password associated with the Username in the Password field.
5. Re-enter the password provided in step 4 in the Verify Password field.

   Note: Characters in password fields appear as asterisks.
6. From the User Type list, select a Unified Profile. The user will be an instance of this Unified Profile. This allows the system to access explicit properties in a Unified Profile type, and ensures proper data cleanup when the user is removed.
7. 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 (for example: User, user) user creation.

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

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:

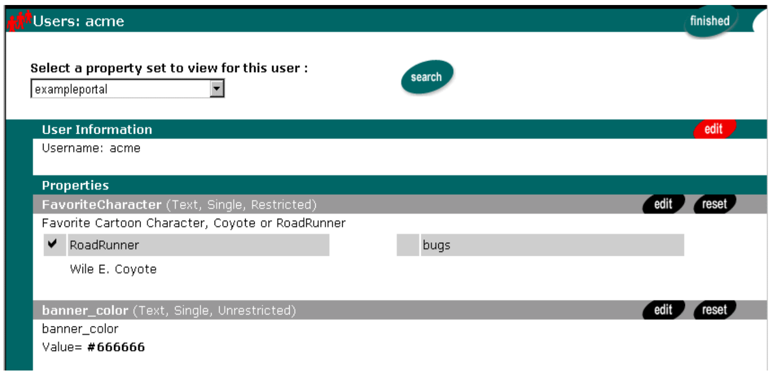
1. On the Administration Tools Home page, click the User Management icon. The User Management Home page appears.
2. On the User Management Home page, click Users in the Users banner. The Search for a User tool appears.

   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.
3. Select the user. The User Property view appears.
4. Select a property set to view for this user. For specific instructions on Property Set Management, see Chapter 6, "Creating and Managing Property Sets."
5. Click Search. The User Properties view appears.


Users: acme

Select a property set to view for this user : exampleportal

User Information

Username: acme

Properties

FavoriteCharacter (Text, Single, Restricted)

Favorite Cartoon Character, Coyote or RoadRunner

✓ RoadRunner | bugs

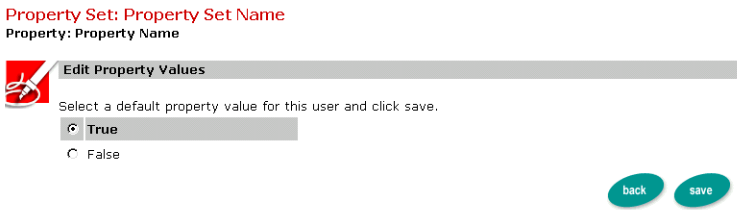
Wile E. Coyote

banner_color (Text, Single, Unrestricted)

banner_color

Value= #666666

6. Click Edit on the appropriate Property bar. The associated Edit Property Values page appears.


Property Set: Property Set Name

Property: Property Name

Edit Property Values

Select a default property value for this user and click save.

[✓ True] False

7. Change the user's values at the Edit Property Values page.
8. 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.
9. Click Back to return to the User Properties view.
10. 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:

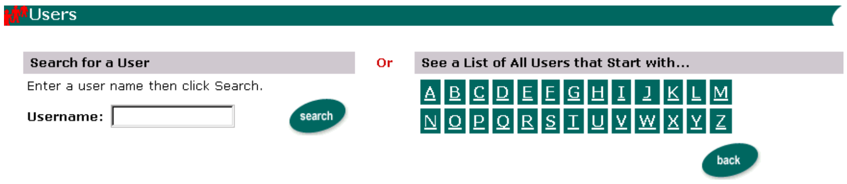
- First, if the property has a default value, the user will have that default value. Note that the default value is not copied into the user's settings. The user's value is just set to null so that the default value will be returned when getProperty() is called for that property. If the default value 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. On the Administration Tools Home page, click the User Management icon. The User Management Home page appears.
2. On the User Management Home page, click Users in the Users banner. The Search for a User tool appears.


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

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.

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

4. Click OK to confirm the deletion.

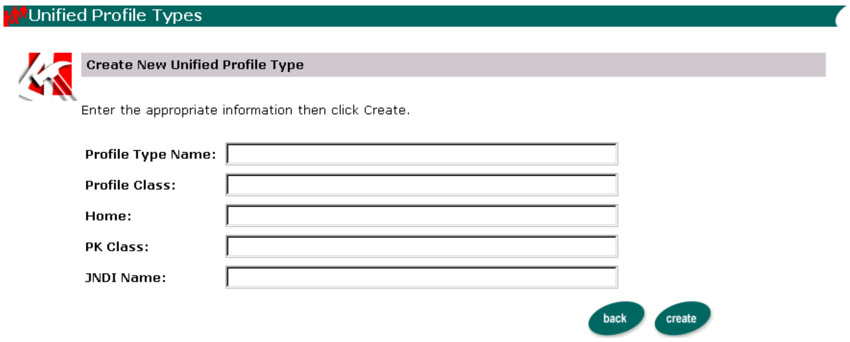
Note: When a use is deleted from the Delete Users screen, the corresponding User component and its properties will be deleted, but the username will continue to be returned from user searches.

Creating Unified Profile Types

To create unified profile types:

The Unified Profile Type tool facilitates the registration of profile types to be used as Unified User Profile (UUP) objects.

1. On the Administration Tools Home page, click the User Management icon. The User Management Home page appears.
2. On the User Management Home page, click Create in the Unified Profile Types banner. The Create New Unified Profile Type page appears.



Create New Unified Profile Type

Enter the appropriate information then click Create.

Profile Type Name:	
Profile Class:	
Home:	
PK Class:	
JNDI Name:	

The following table contains descriptions of the Create New Unified Profile Type fields:

Field	**Description**
Profile Type Name	This is an arbitrary name that is used to refer to the profile type through the User Management system's `<um:getProfile>` JSP extension tag.
Profile Remote Interface	The remote interface of the new profile type.
Home	The home class of the new profile type.
PK Class	The primary key class of the new profile type.
JNDI Name	The JNDI lookup name of the new profile type.

3. Enter the appropriate information in the fields provided.
4. Click Create and return to the Unified Profile Types list.
   Alternatively, click Back to return to the User Management Home page without saving your changes.

Editing Unified Profile Types

To edit unified profile types:

1. On the Administration Tools Home page, click the User Management icon. The User Management Home page appears.
2. On the User Management Home page, click the Unified Profile Types list. The Unified Profile Type page appears.
3. Click the appropriate link to edit a unified profile type. The Edit Unified Profile Type page appears.
4. Edit the appropriate field(s) of the unified profile type.
5. Click Save and return to the Unified Profile Types list or click Back to return to the User Management Home page without saving your changes.

Deleting Unified Profile Types

To delete unified profile types:

1. On the Administration Tools Home page, click the User Management icon. The User Management Home page appears.
2. On the User Management Home page, click Unified Profile Types. The Unified Profile Type page appears.
3. Click the X to right of the username to delete the user. A confirmation dialog box appears.
4. Click OK to confirm the deletion.

Using the LDAP Realm

The LDAP tools are accessible only if WebLogic's LDAPRealm is used.

Setting up LDAP in the WLS Administration Console

Before you begin, WebLogic Commerce Server must be properly configured to start up without throwing exceptions. You will start up WebLogic Commerce Server at the end of these instructions.

Note: These instructions are for a Netscape Directory Server.

Creating the LDAP Realm

1. Start the LDAP server.
2. Bring up the WebLogic Server Administration Console by launching a browser to the following URL: http://<server>:<port>\console
3. Click the Realms node of the tree in the left pane of the Administration Console.
4. Click the Create a new LDAPRealm link.

The General Tab

1. Enter a name for the LDAP realm you are creating (for example, wlcsLDAPRealm).
2. Click the Create button.

The LDAP Tab

1. Enter the URL with the listen port for your LDAP server (for example, ldap://mycomputer.beasys.com:389).
2. In the Principal field enter: uid=admin,ou=Administrators,ou=TopologyManagement,o=NetscapeRoot

3. In the Credential field enter the password for the user you want to connect as. For example, the LDAP administrator's password might be `admin`.
4. Leave the Enable SSL checkbox unchecked. (See note below.)
5. In the Auth Protocol drop-down list or menu, select: simple
6. Click the Apply button.

Note: When you see an LDAP property in the console that is not set in `config.xml`, enter a value *different* than the default and apply the change. Then enter the correct value and apply that change. You should then see the property set correctly within the `config.xml` file. For example, setting the attribute's default value will not put it into `config.xml` because it is a default value. Because WebLogic Commerce Server expects to see a default value in `config.xml`, it will throw an exception if it is not there. Set the attribute to a non-default value, then set it back to the default value, and you will see it appear in `config.xml`.

In step 4. above, check the box for Enable SSL, click the Apply button, then uncheck the box and click the Apply button again. This is to ensure that the default value of "false" appears in config.xml.

The Users Tab

1. In the User Authentication drop-down list, select: local
2. In the User Password Attribute field, enter: userpassword. (See note below.)
3. In the User DN field enter: o=beasys.com, ou=People
4. In the User Name Attribute field, enter: uid
5. Click the Apply button.

Note: When you see an LDAP property in the console that is not set in `config.xml`, enter a value *different* than the default and apply the change. Then enter the correct value and apply that change. You should then see the property set correctly within the `config.xml` file. For example, setting the attribute's default value will not put it into `config.xml` because it is a default value. Because WebLogic Commerce Server expects to see a default value in `config.xml`, it will throw an exception if it is not there. Set the attribute to a non-default value, then set it back to the default value, and you will see it appear in `config.xml`.

In the User Password Attribute field above, enter: x, click the Apply button, enter: userpassword, and click the Apply button again. This is to ensure that the default value of "userpassword" appears in config.xml.

The Groups Tab

1. In the Group DN field, enter: o=beasys.com, ou=Groups
2. In the Group Name Attribute field, enter: cn. (See note below.)
3. Ensure that the Group is Context checkbox is unchecked.
4. In the Group Username Attribute field, enter: uniquemember
5. Click the Apply button.

Note: When you see an LDAP property in the console that is not set in `config.xml`, enter a value *different* than the default and apply the change. Then enter the correct value and apply that change. You should then see the property set correctly within the `config.xml` file. For example, setting the `GroupNameAttribute="cn"` will not put it into `config.xml` because it is a default value. WebLogic Commerce Server expects to see a `GroupNameAttribute` in `config.xml` and will throw an exception if it is not there. Set the `GroupNameAttribute` to a non-default value, `"xx"`, and then set it to `"cn"`, and you will see it appear in `config.xml`.

In the Group Name Attribute field above, enter: x, click the Apply button, enter: cn, and click the Apply button again. This is to ensure that the default value of "cn" appears in config.xml.

Specifying/Creating the Caching Realm

1. Click the Caching Realms node of the tree in the left pane.

2. Most likely you will see a caching realm already created named `wlcsCachingRealm`. If this realm exists, click it. If no caching realm exists, skip to step 5.

3. In the Basic Realm list, select the name of the newly created LDAP realm (e.g., `wlcsLDAPRealm`).

4. Click the Apply button.

5. If a caching realm has not been created, click the Create a new Caching Realm link.

6. In the Name field, enter a name for the caching realm (for example, `wlcsCachingRealm`).

7. In the Basic Realm list, select the name of the newly created LDAP realm (for example, `wlcsLDAPRealm`)

8. Click the Create button.

Verifying the LDAP Properties in config.xml

1. Open the `$WL_COMMERCE_HOME/config/wlcsdomain/config.xml` file using an editor (not a browser).

2. Verify that the LDAP properties that were set using the WebLogic Server console are correctly set in config.xml.

   - You should see an LDAP realm element (`<LDAPRealm />`) with many attributes. Each attribute is a property that was set using the WLS console.

Note: If you see an LDAP property in the console that is not set in config.xml, enter a value *different* than the default and apply the change. Then enter the correct value and apply that change. You should then see the property set correctly within the config.xml file. For example, setting the attribute's default value will not put it into `config.xml` because it is a default value. Because WebLogic Commerce Server expects to see a default value in `config.xml`, it will throw an exception if it is not there. Set the attribute to a non-default value, then set it back to the default value, and you will see it appear in `config.xml`.

- You should see a caching realm element (`<CachingRealm />`) with three attributes. Verify that the `BasicRealm` attribute is set to your LDAP realm name.
- You should see a realm element (`<Realm />`) with three attributes. Verify that the `CachingRealm` attribute is set to the realm you specified in the Specifying/Creating the Caching Realm section of this document.

Note: Avoid editing the config.xml file manually. WebLogic Server prefers that config.xml is edited using the console. However, if you must make manual edits, WLS will accept them.

Example

```
<LDAPRealm AuthProtocol="simple" Credential="admin"
  GroupDN="o=beasys.com, ou=Groups" GroupIsContext="false"
  GroupNameAttribute="cn" GroupUsernameAttribute="uniquemember"
  LDAPURL="ldap://myLDAPserver:389" Name="wlcsLDAPRealm"
  Principal="uid=admin,ou=Administrators,
   ou=TopologyManagment,o=NetscapeRoot"
  SSLEnable="false" UserAuthentication="local"
  UserDN="o=beasys.com, ou=People" UserNameAttribute="uid"
  UserPasswordAttribute=”userpassword”/>
```

Startup WebLogic Commerce Server

1. WebLogic Commerce Server should now be set up to use the specified LDAP server. Start WebLogic Commerce Server.
2. If you are using the WebLogic Commerce Server Administation Tools, a login dialog box will display. Enter the username and password for an LDAP administrative user.

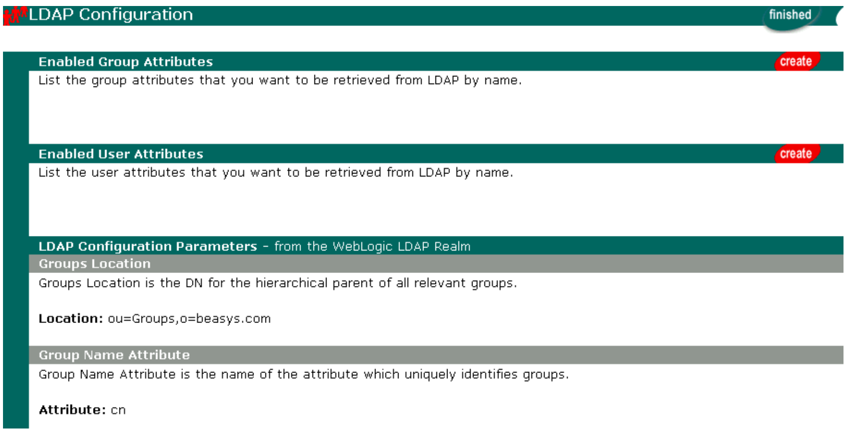
Registering User Attributes for Retrieval from LDAP

The LDAP Configuration screen is used to register user attribute names for run-time retrieval via the group profile.

Note: For the LDAP features to appear in the User Management tool, you must first install and configure the WebLogic LDAP security realm for your WebLogic Server, as described in the sections above.

Note: Your WebLogic Commerce Server Administration Tool is set up to allow access to the group called "admin." To access your WebLogic Commerce Server Administration Tool after you start your server with the alternate security realm, you will need to create a group called "admin" with an administrative user in it. By default, the exampleportal application is set to the AcmeUsers group profile when not authenticated. To use the exampleportal application with another security realm such as LDAP, you need to create a group called "AcmeUsers." Without this group, an exception is thrown to the console complaining about its absence. Another solution is to change the default group for exampleportal from AcmeUsers to another group being used within the security realm.

1. On the Administration Tools Home page, click the User Management icon. The User Management Home page appears.
2. On the User Management Home page, click LDAP in the LDAP banner. The LDAP Configuration view appears.



Enabled Group Attributes
List the group attributes that you want to be retrieved from LDAP by name.

Enabled User Attributes
List the user attributes that you want to be retrieved from LDAP by name.

LDAP Configuration Parameters - from the WebLogic LDAP Realm

Groups Location
Groups Location is the DN for the hierarchical parent of all relevant groups.

Location: ou=Groups,o=beasys.com

Group Name Attribute
Group Name Attribute is the name of the attribute which uniquely identifies groups.

Attribute: cn

3. Click Create on the Enabled User Attributes bar. The Add User Attribute page appears.
4. Enter a new attribute to retrieve from LDAP in the User Attribute Name field.
5. Click Save. Alternatively, click Back to return to LDAP Configuration view without saving your changes.
6. Repeat steps 4 and 5 as necessary.
7. When finished, click Back.

Registering LDAP Properties for Use With Rules

To use LDAP properties in rules, the rules need to know that the properties exist. For any properties that are registered for retrieval from LDAP, create a property set with the LDAP properties in it, and give each property the same name as the property that is registered in LDAP.

You cannot use the User Management Administration Tool or the WebLogic Personalization Server framework to modify properties that are stored in LDAP, but you can use the fact that other property sets are searched before LDAP if you want to override the LDAP value.

Unregistering User Attributes for Retrieval from LDAP

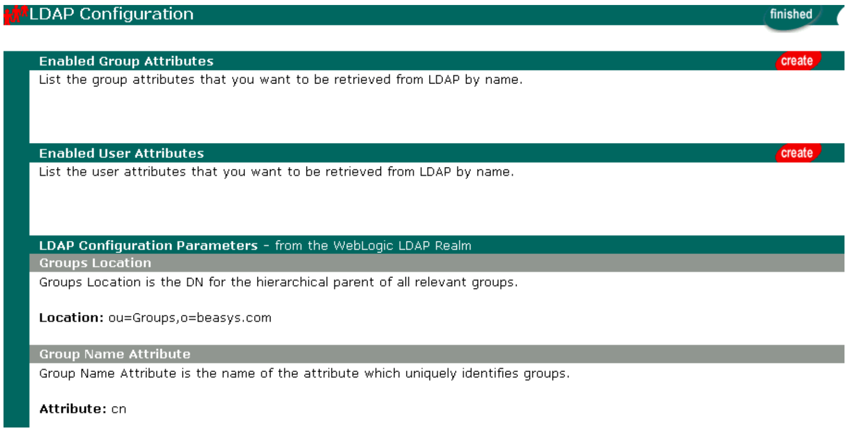
To unregister user attributes for retrieval from LDAP:

1. On the Administration Tools Home page, click the User Management icon. The User Management Home page appears.
2. On the User Management Home page, click LDAP in the LDAP banner. The LDAP Configuration view appears.
3. In the Enabled User Attributes list, click the X to the right of the attribute you want to delete. A confirmation dialog box appears.
4. Click OK to confirm the deletion.
5. Repeat steps 3 and 4 as necessary.
6. When finished, click Back.

Registering Group Attributes for Retrieval from LDAP

To register group attributes for retrieval from LDAP:

1. On the Administration Tools Home page, click the User Management icon. The User Management Home page appears.
2. On the User Management Home page, click LDAP in the LDAP banner. The LDAP Configuration view appears.



Enabled Group Attributes
List the group attributes that you want to be retrieved from LDAP by name.

Enabled User Attributes
List the user attributes that you want to be retrieved from LDAP by name.

LDAP Configuration Parameters - from the WebLogic LDAP Realm

Groups Location
Groups Location is the DN for the hierarchical parent of all relevant groups.

Location: ou=Groups,o=beasys.com

Group Name Attribute
Group Name Attribute is the name of the attribute which uniquely identifies groups.

Attribute: cn

3. Click Create on the Enabled Group Attributes bar. The Add Group Attribute tool appears.
4. Enter a new attribute in the Group Attribute Name field to retrieve from LDAP.
5. Click Save to add the attribute or click Back to return to LDAP Configuration view without saving your changes.
6. Repeat steps 4 and 5 as necessary.

Unregistering Group Attributes for Retrieval from LDAP

To unregister group attributes for retrieval from LDAP:

1. On the Administration Tools Home page, click the User Management icon. The User Management Home page appears.

2. On the User Management Home page, click LDAP in the LDAP banner. The LDAP Configuration view appears.
3. In the Enabled Group Attributes list, click the X to the right of the attribute you want to delete. A confirmation dialog box appears.
4. Click OK to confirm the deletion.
5. Repeat steps 3 and 4 as necessary.

Viewing LDAP Configuration Settings

To view LDAP configuration settings:

1. On the Administration Tools Home page, click the User Management icon. The User Management Home page appears.
2. On the User Management Home page, click LDAP in the LDAP banner. The LDAP Configuration view appears.
3. View the status of the parameters listed in the following table from the LDAP Configuration Parameters field

Parameter	**Description**
Groups Location	Distinguished name for the hierarchical parent of all relevant groups.
Group Name Attribute	The name of the attribute that uniquely identifies a group.
Group Username Attribute	The name of the attribute in group objects that has as its value the group members.
Users Location	Distinguished name for the hierarchical parent of all relevant users.
Username Attribute	The name of the attribute that uniquely identifies users in the system. Example: login name or unique ID.

Parameter	Description
LDAP System Principal	Distinguished name for a system level user. This user has read access to all information in the LDAP directory accessed by the application.
LDAP URL	The Universal Resource Locator (URL) of the LDAP directory server you are running.
SSL	Indicates whether communication from the WebLogic Personalization Server to the LDAP directory should be encrypted over SSL.

Note: The values above are "read only" and are specified when configuring the LDAPRealm.

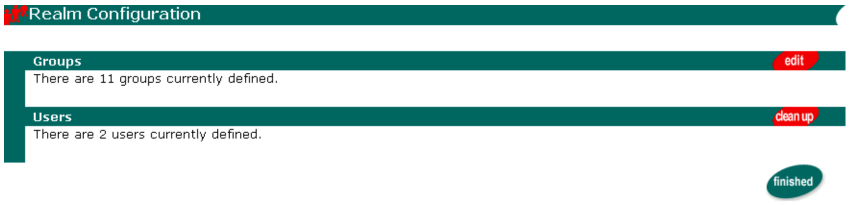
Using Other Realms

The remaining tools are accessible only if a realm other than WebLogic Personalization Server's RDBMSRealm is used.

Selecting Groups for Use in the WebLogic Personalization Server from the Realm

To select groups for use in the WebLogic Personalization Server from the realm:

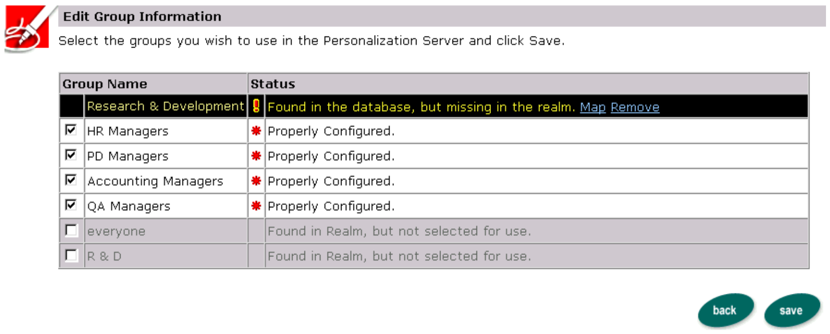
1. On the Administration Tools Home page, click the User Management icon. The User Management Home page appears.
2. On the User Management Home page, click Realm in the Realm banner. The Realm Configuration page appears.


Realm Configuration

Groups
There are 11 groups currently defined.

Users
There are 2 users currently defined.

3. Click Edit in the Groups bar. The Edit Group Information tool appears.


Edit Group Information

Select the groups you wish to use in the Personalization Server and click Save.

	Group Name		Status
	Research & Development	!	Found in the database, but missing in the realm. Map Remove
☑	HR Managers	✱	Properly Configured.
☑	PD Managers	✱	Properly Configured.
☑	Accounting Managers	✱	Properly Configured.
☑	QA Managers	✱	Properly Configured.
☐	everyone		Found in Realm, but not selected for use.
☐	R & D		Found in Realm, but not selected for use.

4. Select the group(s) you wish to use.
5. Click Save.

Mapping Realm Groups to the WebLogic Personalization Server

When a name changes in the realm, you must change it in the WebLogic Personalization Server too. Use this tool when a group name changes in the realm. Mapping works by changing the records in the WebLogic Personalization Server to reflect the new group name.

1. On the Administration Tools Home page, click the User Management icon. The User Management Home page appears.
2. On the User Management Home page, click Realm in the Realm banner. The Realm Configuration page appears.

3. Click Edit in the Groups bar. The Edit Group Information tool appears.

4. Click Map in the Status description of the corresponding group name. The Map Group tool appears.

Note: You are only given the option of mapping those groups that have been found in your database but are missing from the realm.

5. Select the appropriate group name from the Map To Group field.

6. Click Save. Alternatively, click Back to return to the Realm Configuration page without saving your changes.

Note: Group mapping works by simply changing the name of the group in the personalization tables to the group name in the realm. All property data is retained.

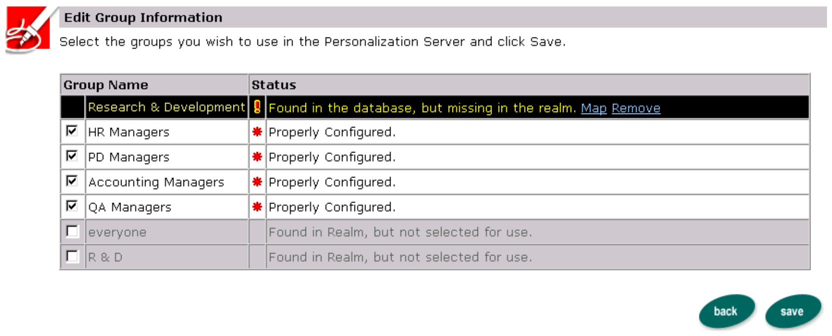
Deleting Groups from Your Database

To delete groups from your database:

1. On the Administration Tools Home page, click the User Management icon. The User Management Home page appears.

2. On the User Management Home page, click Realm in the Realm banner. The Realm Configuration page appears.

3. Click Edit in the Groups bar. The Edit Group Information tool appears.



Edit Group Information

Select the groups you wish to use in the Personalization Server and click Save.

	Group Name		Status
	Research & Development	!	Found in the database, but missing in the realm. Map Remove
☑	HR Managers	✱	Properly Configured.
☑	PD Managers	✱	Properly Configured.
☑	Accounting Managers	✱	Properly Configured.
☑	QA Managers	✱	Properly Configured.
☐	everyone		Found in Realm, but not selected for use.
☐	R & D		Found in Realm, but not selected for use.

4. Click Remove in the Status description of the corresponding group name.

   Note: You are only given the option of deleting those groups that are found in your database but are missing from the realm. A confirmation dialog box appears.

5. Click OK to confirm the deletion.

Deleting User Records That Do Not Exist in the Realm from the Personalization Database

To delete user records that do not exist in the realm from the Personalization database:

1. On the Administration Tool Home page, click the User Management icon. The User Management Home page appears.
2. On the User Management Home page, click Realm in the Realm banner. The LDAP Configuration view appears.
3. Click Edit in the Users bar. The Clean Up Users tools appears with a count of users found in the personalization database but not in the realm.
4. Click Clean Up if the usernames are no longer needed. All associated records are removed.

CHAPTER

8 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:

- Structuring a Query
- Using Comparison Operators to Construct Queries
- Using the BulkLoader to Load File-based Content
- Using Content Management JSP Tags

What Is the Content Manager?

The Content Manager run-time subsystem provides access to content 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 provideds 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 Querys 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 Document Content” on page 8-17 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.

The following table compares the methods of content retrieval that WebLogic Personalization Server provides.

Table 8-1 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. CBEs develop the infrastructure, then BAs 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.
Ad placeholders and `<ph:placeholder>` tags	■ Use a centrally maintained infrastructure for matching advertising documents with events, customer profiles, or customer segments. CBEs develop the infrastructure, then BAs 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 Manager for WebLogic). ■ 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.

Table 8-1 Methods for Retrieving and Displaying Documents (Continued)

Use This Method...	When You Want To...
`<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	■ 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. CBEs must modify each occurrence of this tag if you want to change the document that it retrieves.
`<cm:getProperty>` tag	■ Retrives 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	■ 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	■ 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. CBEs must modify each occurrence of this tag if you want to change the document that it retrieves.

Table 8-1 Methods for Retrieving and Displaying Documents (Continued)

Use This Method...	When You Want To...
`<cm:selectById>` tag	■ 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. CBEs must modify each occurrence of this tag if you want to change the document that it retrieves.

Constructing Queries Using Java

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

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

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

Differences Between Content Management and Document Management

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

`Documents` are a specialized type of `Content` that provide two methods for retrieval: a metadata-searching mechanism and retrieval of the pure bytes of the document's file. `Documents` should include additional explicit metadata properties related to the file and its versioning, including its size, name, path, author, and version. A document management system usually provides document-based services for documents that reside in the system's repository.

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

Using the Document Servlet

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

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

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

Example 1: Usage in a JSP

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

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

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

Example 2: Usage in a JSP

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

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

JSP Tags

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

- The `<cm:select>` tag uses only the search expression query syntax to select content.
- 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 12, "Personalization Server JSP Tag Library Reference," for more information on any of these tags.

Configuring the Content Manager

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

Once the deployment descriptor has been written, just build the EJBs as you normally would, then add the JAR file to your application through the WebLogic console.

Configuring the DocumentSchema EJB Deployment Descriptor

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

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

- *`SmartBMPUpdate`*: Set to `false`.
- *`UseDataSource`*: Controls whether `jdbc/docPool` (`true`) or `DocPoolURL` (`false`) is used to get connections. Defaults to `true`.
- *`DocPoolURL`*: Specifies the JDBC URL to the document JDBC connection to use (if `UseDataSource` is `false`). Should point to a connection pool.
  For example: `jdbc:weblogic:pool:docPool`.
- *`DocPoolDriver`*: Specifies the JDBC driver class to use to connect to the `DocPoolURL`. This is optional. If not specified, the EJB will try to determine the appropriate JDBC driver class from the DocPool URL.
- *`jdbc/docPool`*: A Data Source reference to the document JDBC connection Pool (see the topic “Setting Up Connection Pools” on page 8-13). This should correspond to the Data Source attached to the WebLogic connection pool that uses the document reference implementation JDBC driver.
- *`jdbc/commercePool`*: A DataSource reference to the `weblogic.jdbc.jts.commercePool`, which should be attached to the WebLogic connection pool `commercePool`.

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

Configuring the DocumentManager EJB Deployment Descriptor

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

- *`PropertyCase`*: This sets how the `DocumentImpl` modifies incoming property names. If this is *lower*, all property names are converted to lowercase. If this is *upper*, all property names are converted to uppercase. If this is anything else or not specified, property names are not modified. Use *lower* or *upper* if the `SmartBMP` class expects everything in a certain case. For the document reference implementation, do not specify the *`PropertyCase`*.
- *`jdbc/docPool`*: A Data Source reference to the document JDBC connection Pool (see the topic "Setting Up Connection Pools" on page 8-13). This should correspond to the Data Source attached to the WebLogic connection pool that uses the document reference implementation JDBC driver.
- *`ejb/ContentHome`*: EJB reference to the Document Home to which this should delegate for non-readOnly access.

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

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

Setting Up Connection Pools

For the document reference implementation, set up a specialized WebLogic connection pool and DataSource which will be used by the DocumentManager via the `jdbc/docPool` reference. (See the topic "Configuring the DocumentManager EJB Deployment Descriptor" on page 8-12.)

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

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

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

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

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

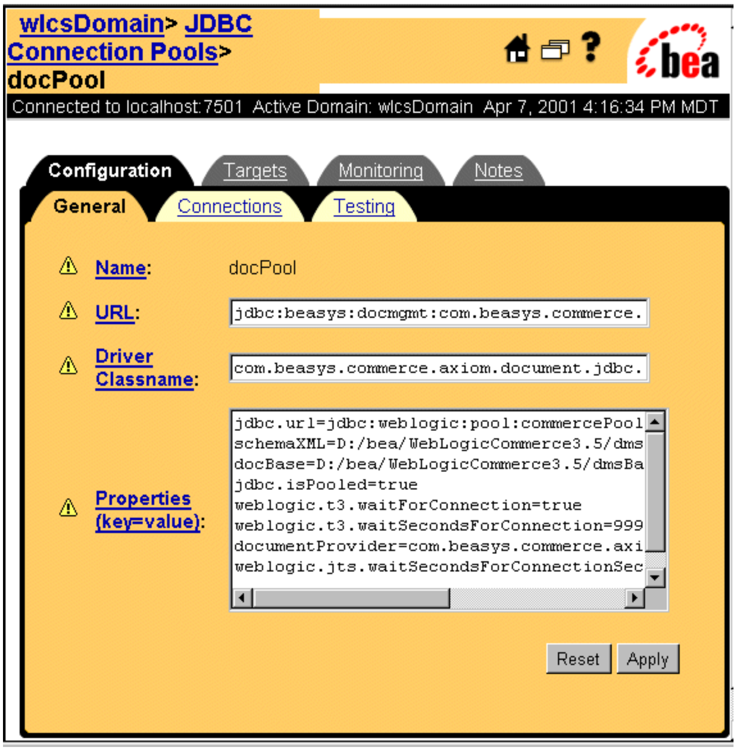
■ Set up a non-transactional DataSource pointing to the pool. The name of the DataSource should be the same as that configured with the DocumentManager and Schema.

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

Example Connection Pool Entry

Figure shows a sample configuration in the WebLogic Server Administration Console.

Figure 8-18 The docPool Screen in the WebLogic Server Console


wlcsDomain> JDBC Connection Pools> docPool
Connected to localhost:7501 Active Domain: wlcsDomain Apr 7, 2001 4:16:34 PM MDT
Configuration
Targets
Monitoring
Notes
General
Connections
Testing
Reset
Apply

Name: docPool

URL: `jdbc:beasys:docmgmt:com.beasys.commerce.`

Driver Classname: `com.beasys.commerce.axiom.document.jdbc.`

Properties (key=value):

```
jdbc.url=jdbc:weblogic:pool:commercePool
schemaXML=D:/bea/WebLogicCommerce3.5/dms
docBase=D:/bea/WebLogicCommerce3.5/dmsBa
jdbc.isPooled=true
weblogic.t3.waitForConnection=true
weblogic.t3.waitSecondsForConnection=999
documentProvider=com.beasys.commerce.axi
weblogic.jts.waitSecondsForConnectionSec
```

Configuring WebLogic Commerce Properties

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

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

`DEF_CONTENT_HOME`
: Specifies the default deployed ContentHome.

`DEF_CONTENT_MANAGER_HOME`
: Specifies the default deployed ContentManagerHome.

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

`DEF_DOCUMENT_HOME`
: Specifies the default deployed DocumentHome.

`DEF_DOCUMENT_MANAGER_HOME`
: Specifies the default deployed DocumentManagerHome.

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

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

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

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

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

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

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

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

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

Example:

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

Using the Show Document Servlet

To operate the Show Document servlet, it should be registered with WebLogic Server. The class name of the servlet is `com.beasys.commerce.content.ShowDocServlet`. To registerthe servlet with WebLogic, add the following XML to your Web application's `web.xml` file:

```
<servlet>
<servlet-name>ShowDocServlet</servlet-name>
<servlet-class>com.beasys.commerce.content.ShowDocServlet</servle
t-class>
</servlet>
<servlet-mapping>
<servlet-name>ShowDocServlet</servlet-name>
<url-pattern>/ShowDocServlet/*</url-pattern>
</servlet-mapping>
```

Reference the class in the URL as `/<webapp-name>/ShowDocServlet.`

To change the URL reference, change the `<url-pattern></url-pattern>` setting.

Querying Document Content

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

- JSP tags (see "Using Content Management JSP Tags" on page 8-28.)
- ContentHelper (see the *Javadoc* API documentation)

- ContentManager (see the *Javadoc* API documentation)
- ContentHome (see the *Javadoc* API documentation)

Structuring a Query

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

```
attribute_name comparison_operator literal_value
```

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

Several constraints apply to queries constructed using this syntax:

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

- To include a scope into the property name, use either `scope.propertyName` or the `toProperty` method with two arguments.
  - `toProperty ('myScope', 'myProperty')`

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

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

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

The following examples illustrate full expressions:

Example 1:

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

Example 2:

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

Using Comparison Operators to Construct Queries

To support advanced searching, the system allows construction of nested Boolean queries incorporating comparison operators. Table 8-2 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 1-11.)

Table 8-2 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`.

Table 8-2 Comparison Operators Available for Each Metadata Type (Continued)

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

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

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

Using the BulkLoader to Load File-based Content

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

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

Command-Line Usage

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

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

`-verbose`
: Emits verbose messages.

`+verbose`
: Runs quietly [default].

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

`+recurse`
: Does not recurse into directories.

`-delete`
: Removes document from database.

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

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

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

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

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

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

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

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

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

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

`-properties <name>`
: Specifies the location of the `loaddocs.properties` file which should contain the `connectionPool` definition.

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

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

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

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

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

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

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

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

`-mdext <ext>`
: Specifies the 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.

How the BulkLoader Finds Files

The following sequence describes how the BulkLoader locates files:

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

   Note: If the file or directory is not an absolute path, then it is assumed to be relative to the `docBase` specified by the `-d` option.
2. To determine if the BulkLoader should process a file or directory, it checks to see if the file is marked as a hidden file.

   Note: If it is a hidden file (or directory) and the `+hidden` option was not specified, then the file or directory is ignored.
3. If the file or directory does not exist or is not readable by the user executing the BulkLoader, a warning is displayed and the file or directory is ignored.
4. If the file or directory is a file, then it is loaded.

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

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

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

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

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

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

How the BulkLoader Finds Metadata Properties

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

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

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

Finally, the BulkLoader will pass the file to the `loadProperties` method of each registered LoaderFilter (the `-filter` option). The LoaderFilter may assign additional metadata to the file. When the BulkLoader starts up, it looks for a `com/beasys/commerce/axiom/document/loader/loader.properties` file in the classpath. From that, it looks for a `loader.defFilters` property. This is the colon-separated list of `LoaderFilter` class names the BulkLoader should always load. Unless that file is modified, the BulkLoader will load an `ImageLoaderFilter`, which will pull the width and height from `*.gif`, `*.jpg`, `*.png`, and `*.xbm` image files.

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

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

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

Cleaning Up the Database

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

Loading Internationalized Documents

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

For example, if the files under the Unicode files directory were saved in the Unicode encoding, you could do:
```
java com.beasys.commerce.axiom.document.loader.BulkLoader -verbose
-properties 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.beasys.commerce.axiom.document.loader.MetaParser` class. For example:
```
java com.beasys.commerce.axiom.document.loader.MetaParser
```

`unicode.htm unicode` would print out the `<meta>` tags found in the `unicode.htm` file, assumed to be Unicode encoded. Of course, any non-ASCII character probably will not print correctly to your console window, but you can tell what it thinks it found.

Generating Schema Files

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

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

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

Using Content Management JSP Tags

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

Content Cache

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

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

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

`cacheTimeout`

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

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

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

Example:

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

readOnly Content Tag

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

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

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

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

Object Interfaces

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

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

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

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

CHAPTER

9 Working with Ad Placeholders

An ad placeholder is one of several mechanisms that WebLogic Personalization Server 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 Campaign Manager for WebLogic, 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 Commerce Business Engineer (CBE) 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

To learn more about using a content management system with WebLogic Personalization Server, refer to Chapter 8, “Creating and Managing Content,” in this guide. For a comparison of content retrieval methods available with WebLogic Personalization Server, refer to “Methods for Retrieving and Displaying Documents” on page 8-4.

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, which are available only with Campaign Manager for WebLogic, 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 9-10 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 9-18 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 8, “Creating and Managing Content,” in this guide.

Table 9-1 describes the `adWeight` attribute, which you can associate with XHTML, image, and Shockwave documents.

Table 9-1 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 9-13 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 9-12 in this guide.

Table 9-2 describes attributes in addition to the `adWeight` attribute that you can associate with image files.

Table 9-2 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 9-2 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 9-3 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 9-3 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 9-3 Attributes for Shockwave Files (Continued)

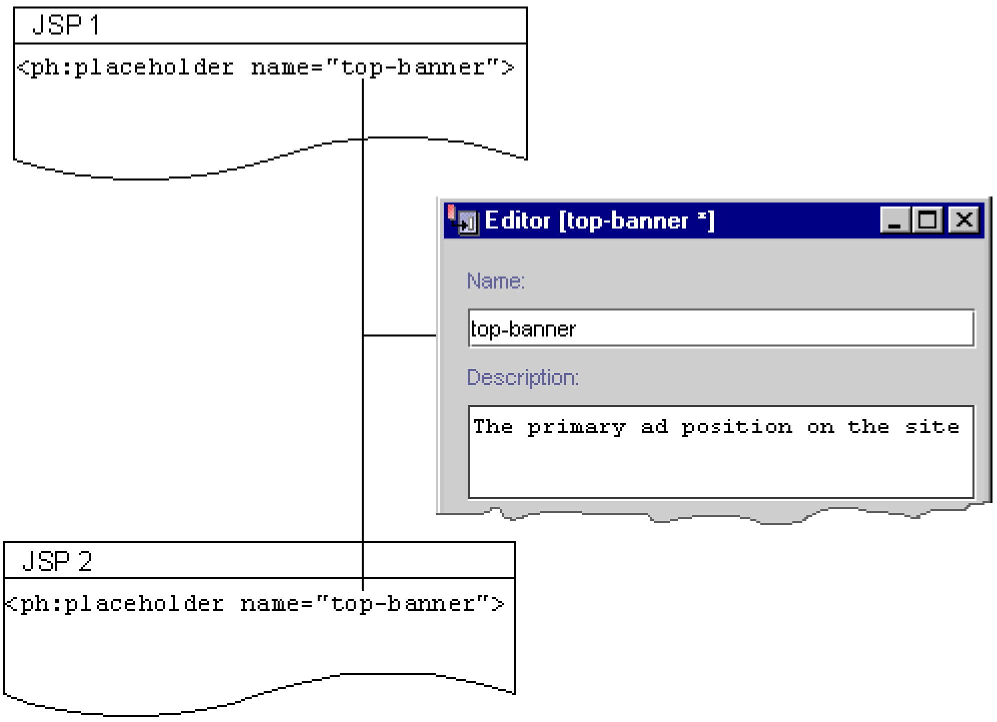
Attribute Name	Value Type	Description and Recommendations
swfSAlign	String	Determines the movie's alignment in relation to the browser window. Valid values are `l`, `t`, `r`, `b`, `tl`, `tr`, `bl`, `br`.
swfBase	String	Specifies the directory or URL used to resolve relative pathnames in the movie. Valid 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 9-19.)

For more information about placeholder tags, refer to <ph:placeholder> in Chapter 12, "Personalization Server JSP Tag Library Reference," in this guide.

Figure 9-19 Multiple Tags Using a Single Definition


JSP 1
<ph:placeholder name="top-banner">
Editor [top-banner *]
Name:
top-banner
Description:
The primary ad position on the site
JSP 2
<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 9-3.
- Use the document attributes that are described in “Ad Attributes in the Content Management System” on page 9-4.

- 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 9-13.

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 9-12.

For a more information about `<ad:adTarget>`, refer to Chapter 12, "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 if you use Campaign Manager for WebLogic, 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:

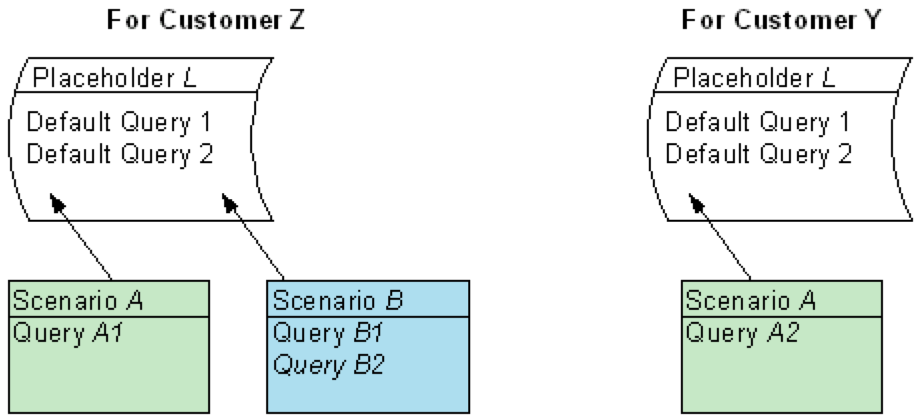
- How Ad Placeholders Contain Multiple Queries
- How the Ad Conflict Resolver Chooses a Query
- How an Ad Placeholder Chooses from Ad Query Results

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 9-9 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 9-20.)

Figure 9-20 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 use Campaign Manager for WebLogic and 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 Personalization Server 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 CBE 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" %>
   ```

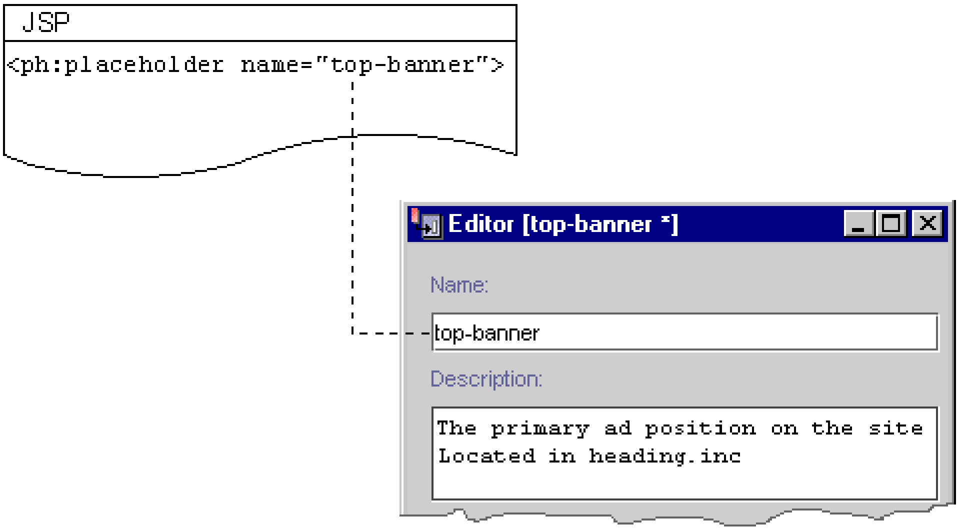
3. Find the location in which the Business Analyst wants to display the ad.

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 9-21) or where *`scriptlet`* returns the name of an existing placeholder.

Figure 9-21 Placeholder Names Must Match



```
<ph:placeholder name="top-banner">
```

top-banner

```
The primary ad position on the site
Located in heading.inc
```

Listing 9-1 shows an example from the heading include file of the e-commerce sample JSP templates (`$WL_COMMERCE_HOME\config\wlcsDomain\applications\wlcsApp\wlcs\commerce\includes\heading.inc`).

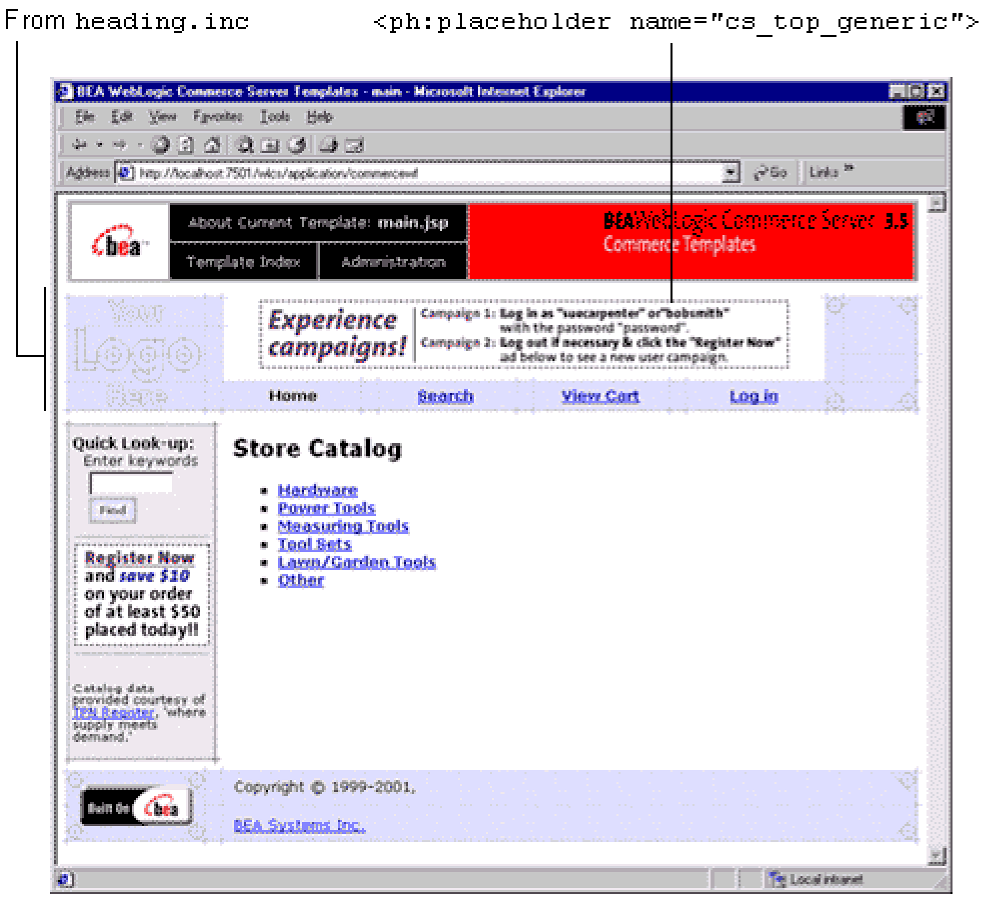
All JSP files in the e-commerce sample Web application include heading.inc to create consistency in the top banner. Instead of requiring that the banner on each page use the same placeholder, the placeholder in heading.inc uses a scriptlet to determine the value of the name attribute. A JSP can use the 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 9-1 Using a Scriptlet for the Placeholder Name

```
<%

   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">
  <img src="<%=WebflowJSPHelper.createGIFURL(request, response,
  "/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 9-22 illustrates how WebLogic Commerce Server renders the placeholder in the main.jsp file, which is the home page for the e-commerce JSP templates.

Figure 9-22 Placeholder in the E-Commerce JSP Templates


From heading.inc
<ph:placeholder name="cs_top_generic">
BEA WebLogic Commerce Server Templates - main - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Address http://localhost:7501/wlcs/application/commercewf
About Current Template: main.jsp
Template Index
Administration
BEA WebLogic Commerce Server 3.5 Commerce Templates
Your Logo Here
Experience campaigns!
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.
Home
Search
View Cart
Log in
Quick Look-up: Enter keywords
Find
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."
Store Catalog
Hardware
Power Tools
Measuring Tools
Tool Sets
Lawn/Garden Tools
Other
Built On bea
Copyright © 1999-2001,
BEA Systems, Inc.
Local intranet

For more information about the `<ph:placeholder>` tag, refer to Chapter 12, "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:

- Add the New Type to the Deployment Descriptor
- Create and Compile a Java Class to Generate HTML
- Register the New Class in weblogiccommerce.properties

Add the New Type to the Deployment Descriptor

Each Campaign Manager for WebLogic Web application must specify its deployment requirements in an XML file called a deployment descriptor. To add a new MIME type for ad placeholders, you must modify the deployment descriptor for your WebLogic Personalization Server Web application. You can use a text editor to modify the deployment descriptor.

If you use the example portal as a framework for developing your own Web application, then the deployment descriptor is located at the following pathname:

```
$WL_COMMERCE_HOME/config/wlcsDomain/applications/wlcsApp/exampleportal/WEB-INF/
web.xml
```

where `$WL_COMMERCE_HOME` is the location in which you installed Campaign Manager for WebLogic. Your Web application might be in another location. Contact your Campaign Manager for WebLogic administrator for information on which deployment descriptor to modify.

The deployment descriptor for your WebLogic Personalization Server Web application already contains a set of mappings for MIME type. Before you add a new type, review the existing mappings. Listing 9-2 illustrates a single MIME mapping from the example portal's deployment descriptor.

Listing 9-2 MIME Mapping in exampleportal/WEB-INF/web.xml

```
<mime-mapping>

    <extension>
      jpeg
    </extension>

    <mime-type>
      image/jpeg
    </mime-type>

</mime-mapping>
```

To add a new mapping, use the following syntax:

```
<mime-mapping>

    <extension>
      file-extension
    </extension>

    <mime-type>
      type/subtype
    </mime-type>

</mime-mapping>
```

where `file-extension` is the extension of the file type you want to map and `type/subtype` is a recognized MIME type and subtype.

Make sure that you provide end-tags for each of the XML elements.

When you save the modified deployment descriptor, you must restart the server to deploy the modifications. However, we recommend that you do not restart the server until you have registered the new Java class in `weblogiccommerce.properties` as described in “Register the New Class in weblogiccommerce.properties” on page 9-20.

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 `bea/commerce/platform/ad/AdContentProvider` interface. For information on the `bea/commerce/platform/ad/AdContentProvider` interface, refer to Campaign Manager for WebLogic *Javadoc*.

After you compile the class, you must save it in or below a directory that is specified in the system’s `CLASSPATH` environment variable. For example `$WL_COMMERCE_HOME/classes` is in the classpath. 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 in weblogiccommerce.properties

After you save the class in a directory that is in your classpath, you must notify Campaign Manager for WebLogic of its existence and purpose by adding a line to `weblogiccommerce.properties`. You can use a text editor to modify this file, which is located at the following pathname:

```
$WL_COMMERCE_HOME/weblogiccommerce.properties
```

where `$WL_COMMERCE_HOME` is the location in which you installed Campaign Manager for WebLogic.

To register your new class in the `weblogiccommerce.properties` file, find the section that Listing 9-3 illustrates. Then add a line that conforms to the following syntax:

```
adtargettag.rendering.mime-type.mime-extension=your-classname
```

Provide the following values for the variables in the previous syntax statement:

- `mime-type`. The name of the MIME type that you want to support.
- `mime-extension`. The filename extension that Campaign Manager for WebLogic uses to associate the file with the MIME type.
- `your-classname`. 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 the classpath.

For example, `$WL_COMMERCE_HOME/classes` is in the classpath. You saved your class to support AVI files as `$WL_COMMERCE_HOME/classes/myclasses/MimeAvi.class`
To register your classname, add the following line to `weblogiccommerce.properties`:

```
adtargettag.rendering.video.avi=myclasses.MimeAvi
```

Listing 9-3 Rendering Classes in weblogiccommerce.properties

```
###############################################

# AdTargetTag Properties

adtargettag.rendering=com.bea.commerce.platform.ad.AdClickThruSer
vlet
# This is the class that implements the AdEventTracker interface
# and is used to raise events
adtargettag.eventtracking=com.bea.commerce.campaign.AdTracking

# Additional classes to render content based upon mime type
# To use replace the "text.html" with the mime type, replacing any
#   '/' characters with '.'

# Place the name of the java class that handles the mime type after
the '='

#adtargettag.rendering.text.html=
```

How Placeholders Select and Display Ads

Placeholders use the following process to select and display ads in a given JSP (see Figure 9-23):

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 9-12 in this guide.
3. The Ad Service does the following:

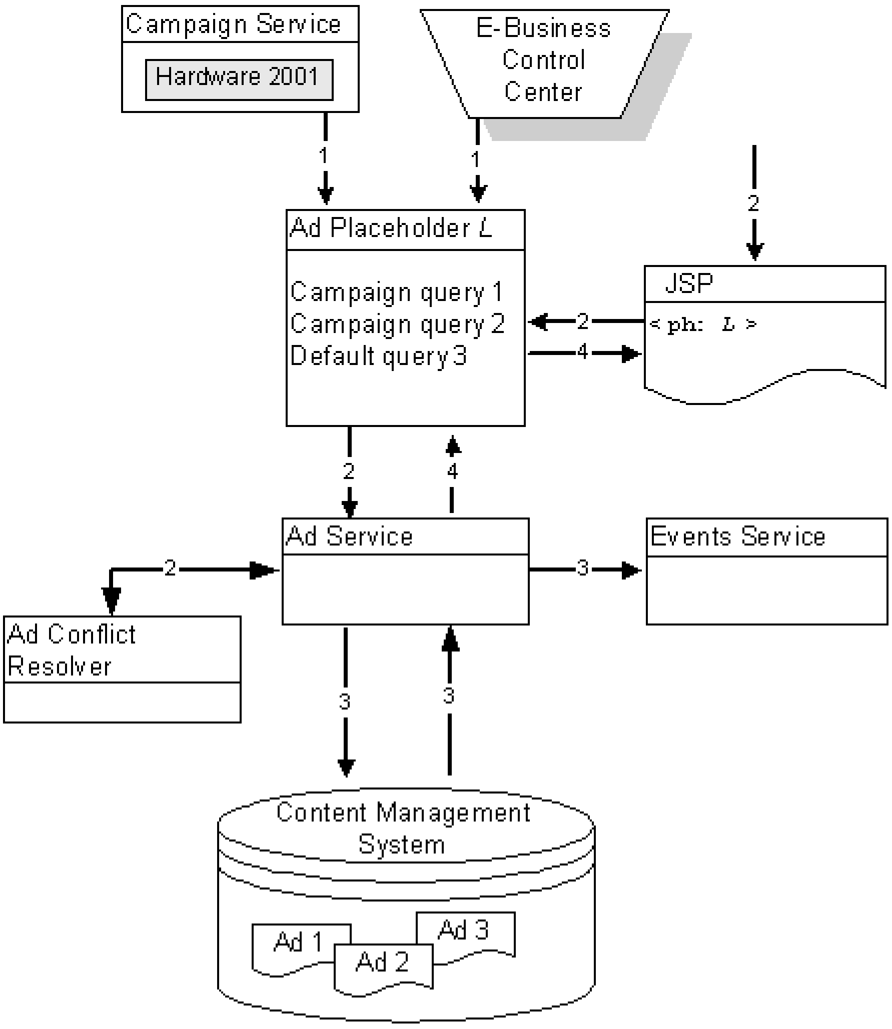
   a. It forwards the query to the content management system. If the query returns more than one ad, the ad placeholder uses the `adWeight` attribute of each ad to determine which one to retrieve.

   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 9-13 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 9-23 How Placeholders Display Ads



CHAPTER

10 Creating Localized Applications with the Internationalization Tags

This topic includes the following sections:

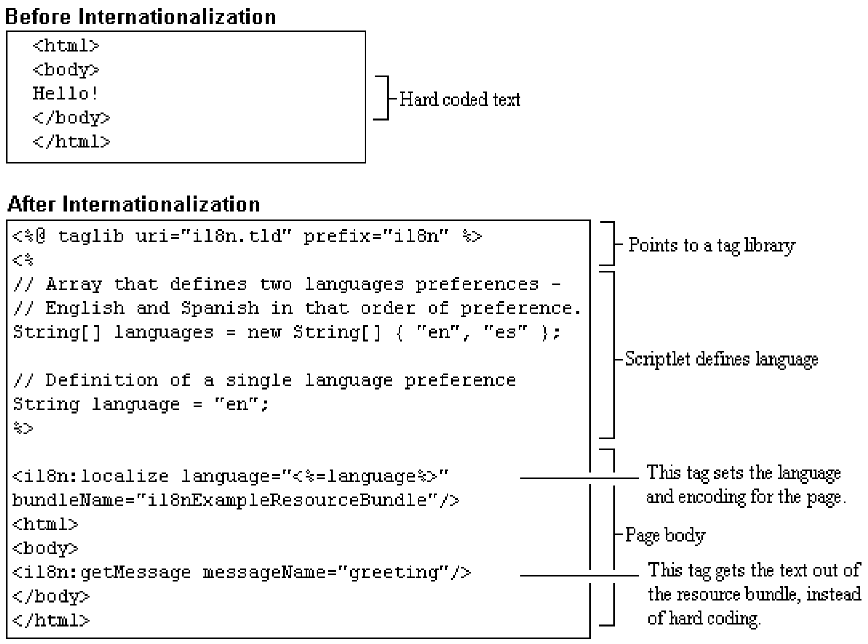
- What Is the I18N Framework?
- Localizing Your JSP
  - <i18n:getMessage>
  - <i18n:localize>
  - Character Encoding
  - Steps for Localizing Your Application
  - Code Examples
- Localizing the BEA WebLogic Personalization Server
  - Static Text
  - Constructed Messages
  - Resource Bundles Used in the WebLogic Personalization Server Tools

What Is the I18N Framework?

WebLogic Personalization Server provides a simple framework that allows access to localized text labels and messages. The internationalization (I18N) framework is accessible from JavaServer Pages (JSPs) through a small I18N tag library. An example is shown in Figure 10-24. The JSP extension tag library provides the following services:

1. Retrieves a static text label from a resource bundle (implemented as a properties file).
2. Retrieves a message from a resource bundle (implemented as a properties file).
3. Initializes a page context with a particular language, country, and variant for label and message retrieval throughout a page.
4. Properly sets the content type (text/html) and character encoding for a page.

Figure 10-24 An Example of Internationalization Code


Before Internationalization

```
<html>
<body>
Hello!
</body>
</html>
```

After Internationalization

```
<%@ taglib uri="i18n.tld" prefix="i18n" %>
<%
// Array that defines two languages preferences -
// English and Spanish in that order of preference.
String[] languages = new String[] { "en", "es" };

// Definition of a single language preference
String language = "en";
%>

<i18n:localize language="<%=language%>"
bundleName="i18nExampleResourceBundle"/>
<html>
<body>
<i18n:getMessage messageName="greeting"/>
</body>
</html>
```

Hard coded text
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.

Localizing Your JSP

The conventions used in the I18N tag library are based on the more general conventions used to internationalize Java applications. To understand the conceptual foundations for the `<i18n:getMessage>`tag, see the *Javadoc* for `java.text.MessageFormat` in the Sun Microsystem, Inc. *Java 2 SDK, Standard Edition* documentation. To better understand the ideas that served as the foundation for these tags, study the *Javadoc* for `java.util.ResourceBundle` and `java.util.Locale`.

The following tags are included in the I18N framework:

`<i18n:getMessage>`

`<i18n:localize>`

<i18n:getMessage>

This tag retrieves a localized label or message (based on the absence/presence of an `args` attribute). The tag optionally takes a bundle name, language, country, and variant to aid in locating the appropriate properties file for resource bundle loading.

This tag is used in the localization of JSP pages. All pages that have an internationalization requirement should use this tag.

For more information about the `<i18n:getMessage>` tag, see Chapter 12, “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 a larger page (for example, as portlets are included in portal pages), only the larger page can use the `<i18n:localize>` tag. This is because the `<i18n:localize>` tag sets the encoding for the page, and the encoding must be set in the parent (including) page before any bytes are written to the response’s output stream. Therefore, be careful that the encoding for the parent page is sufficient for all the content on that page as well as any included pages. The child (included) pages may continue to use the `<i18n:getMessage>` tag.

Note: Do not use the `<i18n:localize>` tag in conjunction with the `<%@ page contentType="<something>" >` page directive defined in the JSP specification. The directive is unnecessary if you are using this tag, and can result in inconsistent or wrong `contentType` declarations.

For more information about the `<i18n:localize>` tag, see Chapter 12, "Personalization Server JSP Tag Library Reference."

The JspMessageBundle

The `<i18n:getMessage>` tag uses the `com.beasys.commerce.i18n.jsp.JspMessageBundle` class. Unlike a ResourceBundle, the JspMessageBundle looks only for properties files (like the PropertyResourceBundle) within the ServletContext (on the doc path). This means that you can keep MessageBundle properties files relative to the associated JSP page, instead of having to have them on the `CLASSPATH`.

Another difference is that JspMessageBundles are specified using the `"/"` character instead of the `"."`. For instance, the path to a JspMessageBundle might look like this: `/jsp/ordersystem/placeOrder`.

If a bundle name is specified, then it can be specified *absolutely* or *relatively*. Absolute paths are treated as such if they begin with a `"/"`. Paths not beginning with `"/"` are searched for relative to the JSP page's location.

If no bundle name is specified, then bundle name defaults to the name of the JSP page. For instance, if you have a JSP page called placeOrder.jsp, then JspMessageBundle would look in the same directory for a placeOrder.properties file to serve as the JspMessageBundle for the placeOrder.jsp page.

When searching for a JspMessageBundle, both the doc root and repository directories are searched, in that order. Repository directories are directories specified during servlet registration and serve as a place to store common files such as images. If no message bundle can be found, a MissingResourceException occurs. For a more in-depth description of the repository directory convention, see "Repository" on page 5-8.

How the Localization Tag Works

The `<i18n:localize>` tag first examines all provided attributes and default attributes, and then performs the following three steps:

1. **Determines the base bundle name.**

   If a base bundle name is not provided, the bundle name defaults to the name of the JSP page with .properties appended.

   For example, if the name of the JSP page is placeOrder.jsp, then the default bundle name would be placeOrder.properties.

2. **Determines the language to use.**

   The tag will first look for resource bundles that correspond to the language parameter passed in to the tag.

   If no match between bundle and language is found, then the tag will try to find a match between resource bundles and languages defined in the request header.

   If a match can be made, the first language that matches is the language that is used.

   If no language is specified, the default is 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

Figure 10-1 shows how the `<i18n:localize>` tag maps languages to character encodings. These are the default settings.

You can override these defaults by providing any charset tag parameter you choose. For example, in the table below, the default charset for Japanese is Shift_JIS, but you could pass in x-sjis, EUC_JP, or iso-2022-jp instead. Or, as another example, to use Chinese Taiwan locale in place of Chinese, override GB2312 with Big5.

Table 10-1 Default Character Encodings

Language Code	Language Name	Character Encoding
ar	Arabic	ISO-8859-6
be	Byelorussian	ISO-8859-5
bg	Bulgarian	ISO-8859-5
ca	Catalan	ISO-8859-1
cs	Czech	ISO-8859-2
da	Danish	ISO-8859-1
de	German	ISO-8859-1

el	Greek	ISO-8859-7
en	English	ISO-8859-1
es	Spanish	ISO-8859-1
et	Estonian	ISO-8859-1
fi	Finnish	ISO-8859-1
fr	French	ISO-8859-1
hr	Croatian	ISO-8859-2
hu	Hungarian	ISO-8859-2
is	Icelandic	ISO-8859-1
it	Italian	ISO-8859-1
iw	Hebrew	ISO-8859-8
ja	Japanese	Shift_JIS
ko	Korean	EUC_KR
lt	Lithuanian	ISO-8859-2
lv	Latvian (Lettish)	ISO-8859-2
mk	Macedonian	ISO-8859-5
nl	Dutch	ISO-8859-1
no	Norweigan	ISO-8859-1
pl	Polish	ISO-8859-2
pt	Portuguese	ISO-8859-1
ro	Romanian	ISO-8859-2
ru	Russian	ISO-8859-5
sh	Serbo-Croatian	ISO-8859-5
sk	Slovak	ISO-8859-2

sl	Slovenian	ISO-8859-2
sq	Albanian	ISO-8859-2
sr	Serbian	ISO-8859-5
sv	Swedish	ISO-8859-1
th	Thai	TIS620
tr	Turkish	ISO-8859-9
uk	Ukrainian	ISO-8859-5
zh	Chinese	GB2312
other		UTF-8

Steps for Localizing Your Application

1. Familiarize yourself with the documentation for the Internationalization `<i18n:*>` tags in Chapter 12, "Personalization Server JSP Tag Library Reference.". For sample code, see Figure 10-24 "An Example of Internationalization Code" on page 10-2.

2. Include the `<i18n:localize>` tag in all pages with an internationalization requirement. The tag should be used as early in the page as possible—before anything is written to the output stream—so that the bytes are properly encoded.

   For example: `<%@ taglib uri="i18n.tld" prefix="i18n" %>`

   For example: `<i18n:localize language="<%=language%>"`

Note: When HTML pages are being included inside a larger page, only the larger page can use the `<i18n:localize>` tag.

3. Move all text that must be localized (including image URLs that must be localized) to property files that serve as resource bundles. Provide a resource bundle (property file) for each language you plan to support. One resource bundle per JSP page per language is the recommended approach.

Note: Changes to the property files will not be recognized until the server is restarted.

For example: Use `<i18n:getMessaage messageName="greeting"/>` instead of hardcoding "Welcome!"

4. Specify a directory path for the property files (resource bundles). The bundle location must be specified *relative* to the JSP location, or *absolutely*, under the document root.
5. Refer to all localized text in a JSP page by using the `<i18n:getMessage>` tag. Make sure the `<i18n:getMessage>` tag is referring to the correct resource bundle location (relative or absolute path).

   For example:
   If the JSP is in `public_html\mypage.jsp`, then the bundle location could be (absolute) `"/mypage/text_us.properties"` or
   (relative) `"text_us.properties"`.
6. Test the page for all languages that you support. Make sure that the localized text and images display correctly and that the page layout is correct.

Code Examples

The following examples show how to use the JSP internationalization framework with JavaScript and Java scriptlets.

Using the JSP Internationalization Framework with JavaScript

This example displays a JavaScript dialog with a localized message in it.

```
<%@ taglib uri="i18n.tld" prefix="i18n" %>
<%
String language="en";
%>
<i18n:localize language="<%=language%>"
bundleName="i18nJavaScriptExampleResourceBundle"/>

<script language="JavaScript">
function popDialog() {
alert("<i18n:getMessage messageName="greeting"/>")
}
</script>

<html>
<body>
<a href="javascript:popDialog();">Click here to see localized
text!</a>
</body>
</html>
```

Using the JSP Internationalization Framework with Java Scriptlets

This example gets a localized message, and uses that message in two Java scriptlets. One scriptlet prints to system out, the other inlines it into the page.

```
<%@ taglib uri="i18n.tld" prefix="i18n" %>
<%
String language="en";
%>
<i18n:localize language="<%=language%>"
bundleName="i18nJavaScriptExampleResourceBundle"/>

<html>
<body>
<i18n:getMessage messageName="greeting" id="theGreeting"/>
<p>
<%="Localized text for 'greeting': " + theGreeting%>
<p>
<%
System.out.println("Localized text for 'greeting': " +
theGreeting);
%>

</body>
</html>
```

Localizing the BEA WebLogic Personalization Server

Up to this point, this chapter has discussed localizing the application that you are building with the BEA WebLogic Personalization Server.

In developing your application, you may be required to localize some of the portal tools in the WebLogic Personalization Server. This section provides information for developers who need to localize the administration tools that are provided with this product, or who are deriving their application from examples that ship with the WebLogic Personalization Server.

The WebLogic Personalization Server Administration Tool is supported by JSP bean objects which employ Java internationalization conventions in the practice of presenting error and status messages. These beans use a BEA utility object called `com.beasys.commerce.i18n.MessageBundle` in conjunction with text-based properties files to produce two types of locale-specific display text. The two types of text are as follows:

- Static Text
- Constructed Messages

Static Text

WebLogic Personalization Server uses the following convention when naming static text entries in the properties files:

```
propertyName.txt=propertyValue
```

For example: `error.txt=Error Occurred`

A static text property is acquired from a loaded MessageBundle using the following method:

```
public String getString(String propertyName)
```

For example:
`System.out.printin(messageBundle.getString("error.txt"));`

For more information, see the *Javadoc* for the Portal API documentation.

Constructed Messages

The localized display text generated at run time often depends on one or more variables, and the order of these variables in a text segment is locale-specific. In this case, the WebLogic Personalization Server provides a means for constructing message segments for display.

WebLogic Personalization Server uses the following convention when naming message entries in properties files:

```
propertyName.msg=propertyValue
```

For example:

```
fieldRequired.msg={0} is a required field.
```

A constructed message is acquired from a loaded MessageBundle using the following method:

```
public String getMessage(Object[] args, String propertyName)
```

For example:

```
Object[] args = new Object[] {“ContentURL”};
```

```
System.out.println(messageBundle.getMessage(args,
"fieldRequired.msg"));
```

For more information, see the *Javadoc* for the Portal API documentation.

The MessageBundle's `getMessage()` method internally uses a `java.text.MessageFormat` object. To understand how the `getMessage()` method works, look at the *Javadoc* for `java.text.MessageFormat`.

Resource Bundles Used in the WebLogic Personalization Server Tools

Each properties file that supports a particular bean includes the bean name and a property extension. For example, the property file that supports the `com.beasys.portal.admin.jspbeans.PortalJspBean` bean resides in the `i18n` directory beneath `com/beasys/portal/admin/jspbeans`, and is called `PortalJspBean.properties`.

Localizing System Messages

You can localize the resource bundles that contain system messages related to the WebLogic Personalization Server Administration Tools and sample applications. Changes to the resource bundles will be recognized when the server is restarted.

Use the following properties files to localize system messages. These property files are found under `<WL_COMMERCE_HOME>/classes`:

```
com/beasys/commerce/axiom/util.i18n/JSPBeanBase.properties

com/beasys/commerce/user/jsp/beans/i18n/LDAPConfigBean.properties

com/beasys/commerce/user/jsp/beans/i18n/ProfileTypeBean.properties

com/beasys/commerce/user/jsp/beans/i18n/PropertyBean.properties

com/beasys/commerce/user/jsp/beans/i18n/PropertySetBean.properties

com/beasys/commerce/user/jsp/beans/i18n/RealmConfigBean.properties

com/beasys/commerce/user/jsp/beans/i18n/UserBean.properties

com/beasys/commerce/portal/admin/jspbeans/i18n/PortalJspBean.properties
```

```
com/beasys/commerce/portal/admin/jspbeans/i18n/PortletJspBean.properties

com/beasys/commerce/portal/admin/jspbeans/i18n/PortalPersonalization.properties

com/beasys/commerce/portal/admin/jspbeans/i18n/PortalRemoveJspBean.properties

com/beasys/commerce/portal.jspbeans/i18n/PortalAppearanceBean.properties

com/beasys/commerce/axiom.util/i18n/JspBeanBase.properties
```

CHAPTER

11 The WebLogic Personalization Server Database Schema

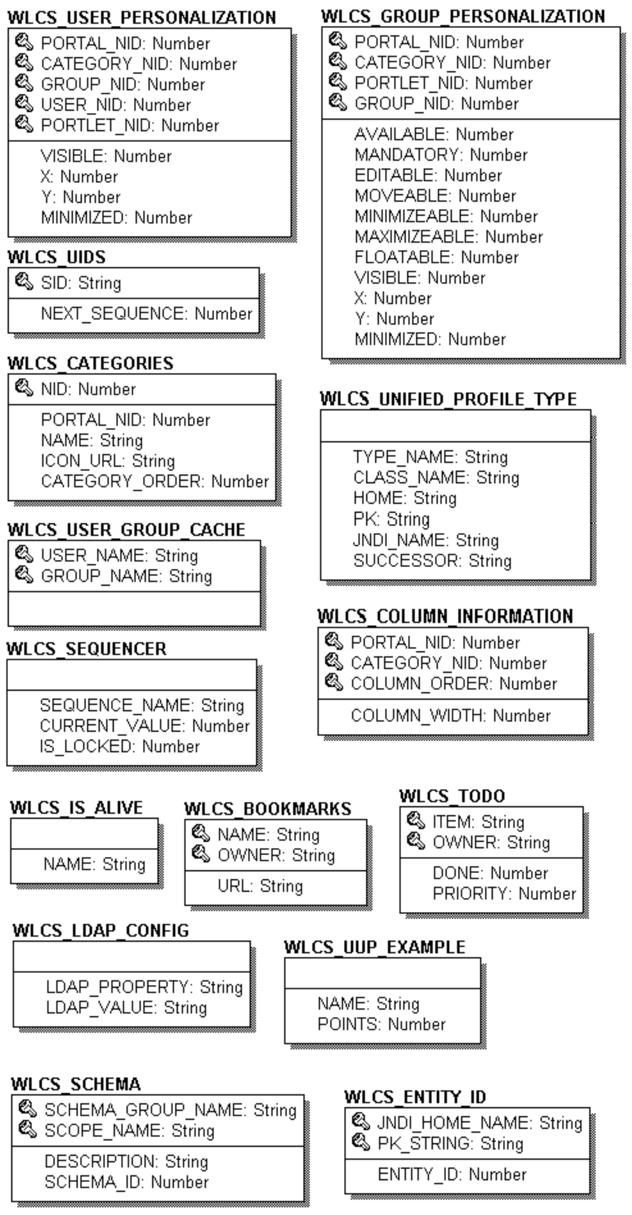
This chapter documents the database schema for the WebLogic Personalization Server. This topic includes the following sections:

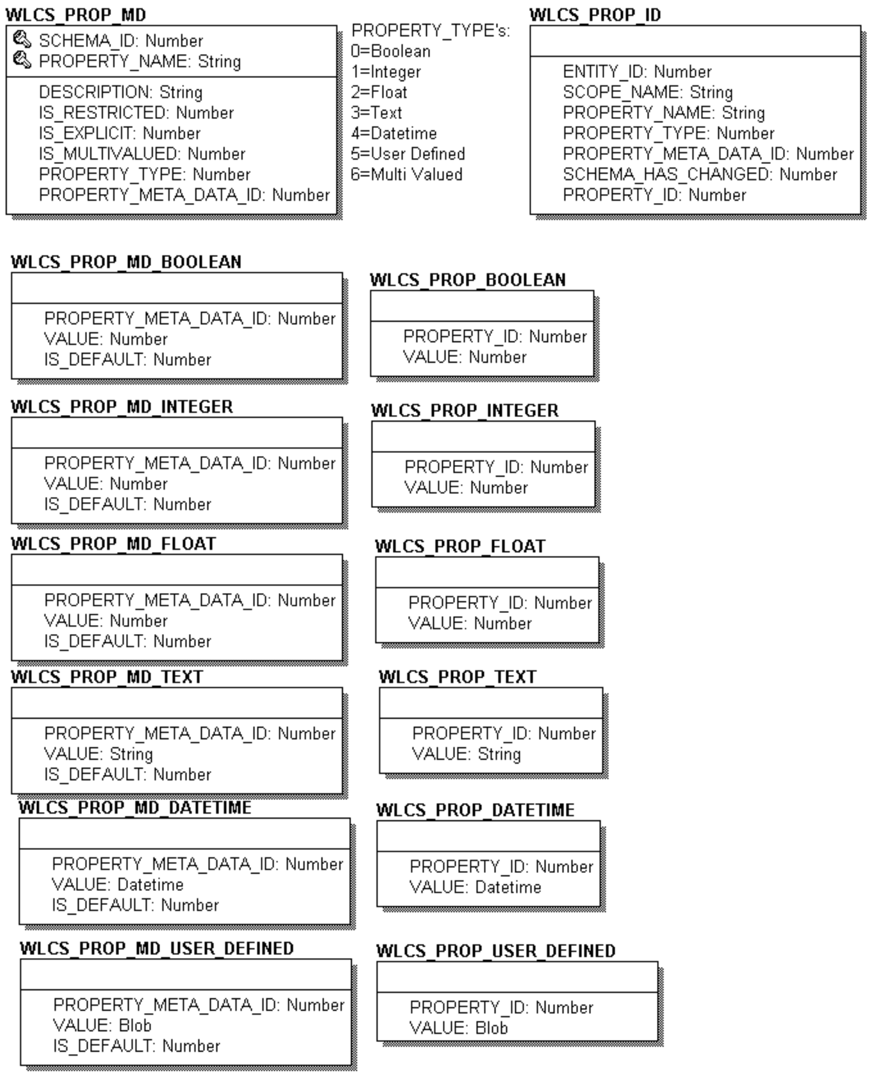
- The Entity-Relation Diagram
- List of Tables Comprising the WebLogic Personalization Server
- The Personalization Server Data Dictionary
- The SQL Scripts Used to Create the Database
- SQL Server

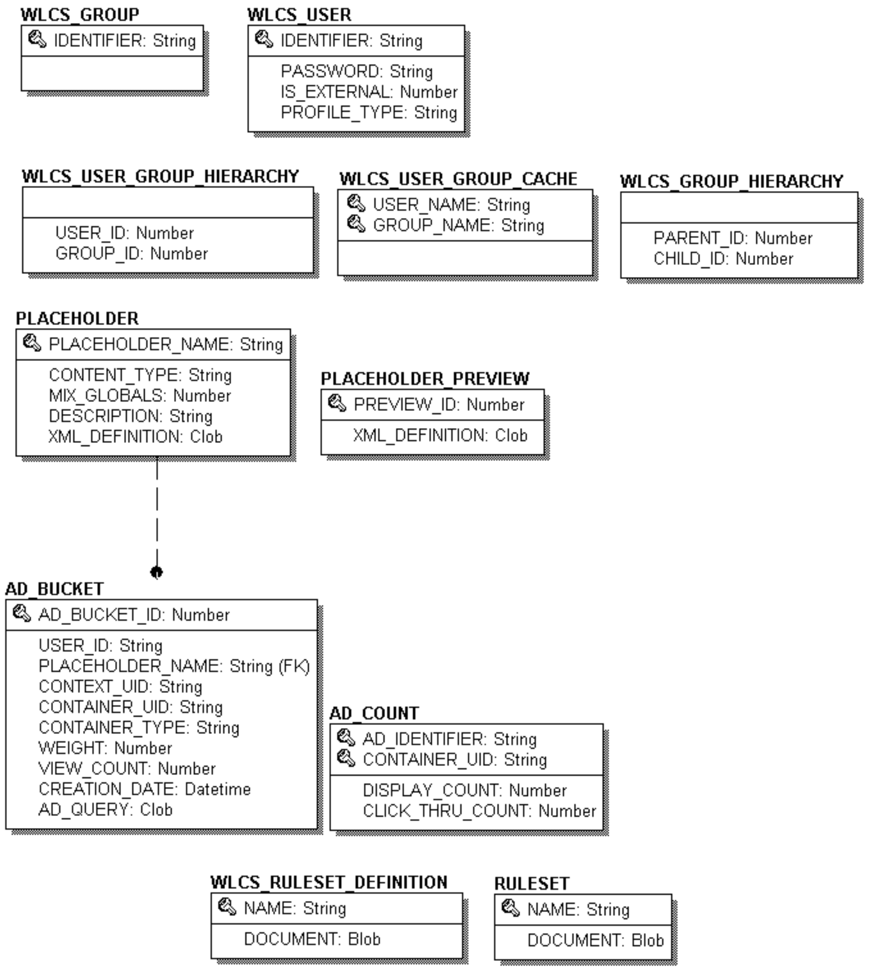
The Entity-Relation Diagram

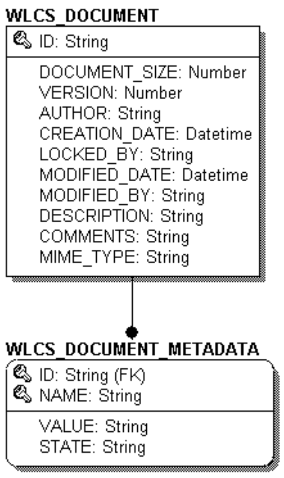
Figure 11-25 shows the logical Entity-Relation diagram for the WebLogic Personalization Server database. See the subsequent sections in this chapter for information about the data type syntax.

Figure 11-25 Entity-Relation Diagram for the WebLogic Personalization Server




WLCS_PROP_MD
SCHEMA_ID: Number
PROPERTY_NAME: String
DESCRIPTION: String
IS_RESTRICTED: Number
IS_EXPLICIT: Number
IS_MULTIVALUED: Number
PROPERTY_TYPE: Number
PROPERTY_META_DATA_ID: Number
PROPERTY_TYPE's:
0=Boolean
1=Integer
2=Float
3=Text
4=Datetime
5=User Defined
6=Multi Valued
WLCS_PROP_ID
ENTITY_ID: Number
SCOPE_NAME: String
PROPERTY_NAME: String
PROPERTY_TYPE: Number
PROPERTY_META_DATA_ID: Number
SCHEMA_HAS_CHANGED: Number
PROPERTY_ID: Number
WLCS_PROP_MD_BOOLEAN
PROPERTY_META_DATA_ID: Number
VALUE: Number
IS_DEFAULT: Number
WLCS_PROP_BOOLEAN
PROPERTY_ID: Number
VALUE: Number
WLCS_PROP_MD_INTEGER
PROPERTY_META_DATA_ID: Number
VALUE: Number
IS_DEFAULT: Number
WLCS_PROP_INTEGER
PROPERTY_ID: Number
VALUE: Number
WLCS_PROP_MD_FLOAT
PROPERTY_META_DATA_ID: Number
VALUE: Number
IS_DEFAULT: Number
WLCS_PROP_FLOAT
PROPERTY_ID: Number
VALUE: Number
WLCS_PROP_MD_TEXT
PROPERTY_META_DATA_ID: Number
VALUE: String
IS_DEFAULT: Number
WLCS_PROP_TEXT
PROPERTY_ID: Number
VALUE: String
WLCS_PROP_MD_DATETIME
PROPERTY_META_DATA_ID: Number
VALUE: Datetime
IS_DEFAULT: Number
WLCS_PROP_DATETIME
PROPERTY_ID: Number
VALUE: Datetime
WLCS_PROP_MD_USER_DEFINED
PROPERTY_META_DATA_ID: Number
VALUE: Blob
IS_DEFAULT: Number
WLCS_PROP_USER_DEFINED
PROPERTY_ID: Number
VALUE: Blob


WLCS_GROUP
IDENTIFIER: String
WLCS_USER
IDENTIFIER: String
PASSWORD: String
IS_EXTERNAL: Number
PROFILE_TYPE: String
WLCS_USER_GROUP_HIERARCHY
USER_ID: Number
GROUP_ID: Number
WLCS_USER_GROUP_CACHE
USER_NAME: String
GROUP_NAME: String
WLCS_GROUP_HIERARCHY
PARENT_ID: Number
CHILD_ID: Number
PLACEHOLDER
PLACEHOLDER_NAME: String
CONTENT_TYPE: String
MIX_GLOBALS: Number
DESCRIPTION: String
XML_DEFINITION: Clob
PLACEHOLDER_PREVIEW
PREVIEW_ID: Number
XML_DEFINITION: Clob
AD_BUCKET
AD_BUCKET_ID: Number
USER_ID: String
PLACEHOLDER_NAME: String (FK)
CONTEXT_UID: String
CONTAINER_UID: String
CONTAINER_TYPE: String
WEIGHT: Number
VIEW_COUNT: Number
CREATION_DATE: Datetime
AD_QUERY: Clob
AD_COUNT
AD_IDENTIFIER: String
CONTAINER_UID: String
DISPLAY_COUNT: Number
CLICK_THRU_COUNT: Number
WLCS_RULESET_DEFINITION
NAME: String
DOCUMENT: Blob
RULESET
NAME: String
DOCUMENT: Blob


WLCS_DOCUMENT
ID: String
DOCUMENT_SIZE: Number
VERSION: Number
AUTHOR: String
CREATION_DATE: Datetime
LOCKED_BY: String
MODIFIED_DATE: Datetime
MODIFIED_BY: String
DESCRIPTION: String
COMMENTS: String
MIME_TYPE: String
WLCS_DOCUMENT_METADATA
ID: String (FK)
NAME: String
VALUE: String
STATE: 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 Database Table
- The PLACEHOLDER _PREVIEW Database Table

Documentation Management tables
- The WLCS_COLUMN_INFORMATION Database Table
- The WLCS_DOCUMENT Database Table
- The WLCS_DOCUMENT_METADATA Database Table

Rule Editor tables
- The RULESET Database Table
- The WLCS_RULESET_DEFINITION Database Table

User Management tables
- The WLCS_GROUP Database Table
- The WLCS_GROUP_HIERARCHY Database Table
- The WLCS_GROUP_PERSONALIZATION Database Table
- The WLCS_UNIFIED_PROFILE_TYPE Database Table
- The WLCS_USER Database Table
- The WLCS_USER_GROUP_CACHE Database Table
- The WLCS_USER_GROUP_HIERARCHY Database Table
- The WLCS_USER_PERSONALIZATION Database Table
- The WLCS_UIDS Database Table

Common tables used by both WebLogic Personalization Server and WebLogic Commerce Server
- The WLCS_CATEGORIES Database Table
- The WLCS_SCHEMA Database Table
- The WLCS_ENTITY_ID Database Table
- The WLCS_BOOKMARKS Database Table

The WLCS_IS_ALIVE Database Table
The WLCS_LDAP_CONFIG Database Table
The WLCS_SEQUENCER Database Table
The WLCS_TODO Database Table
The WLCS_USER_PERSONALIZATION Database Table
The WLCS_UUP_EXAMPLE Database Table

The WLCS_PROP_MD Database Table
The WLCS_PROP_MD_BOOLEAN Database Table
The WLCS_PROP_MD_INTEGER Database Table
The WLCS_PROP_MD_FLOAT Database Table
The WLCS_PROP_MD_TEXT Database Table
The WLCS_PROP_MD_DATETIME Database Table
The WLCS_PROP_MD_USER_DEFINED Database Table

The WLCS_PROP_ID Database Table
The WLCS_PROP_BOOLEAN Database Table
The WLCS_PROP_INTEGER Database Table
The WLCS_PROP_FLOAT Database Table
The WLCS_PROP_TEXT Database Table
The WLCS_PROP_DATETIME Database Table
The WLCS_PROP_USER_DEFINED 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 11-1 describes the AD_BUCKET table. This table maintains content queries for ads.

The Primary Key is `AD_BUCKET_ID`.

Table 11-1 AD_BUCKET Table Metadata

Column Name	Data Type	Description and Recommendations
AD_BUCKET_ID	NUMBER(15)	PK—a unique, system-generated number used as the record identifier.
USER_ID	VARCHAR(50)	FK—foreign key to the `WLCS_USER.IDENTIFIER` column.
PLACEHOLDER_NAME	VARCHAR(50)	FK—foreign key to `PLACEHOLDER.PLACEHOLDER_NAME`.
CONTEXT_UID	VARCHAR(50)	The scenario unique identifier.
CONTAINER_UID	VARCHAR(50)	The campaign unique identifier.
CONTAINER_TYPE	VARCHAR(50)	Identifies the service associated with the `CONTAINER_UID`.
WEIGHT	NUMBER(15)	A weighting scheme used in prioritizing one placeholder over another.
VIEW_COUNT	NUMBER(15)	*Disabled. Reserved for future use.*
CREATION_DATE	DATE	The date and time this record was created.
AD_QUERY	CLOB	The actual content query.

The AD_COUNT Database Table

Table 11-2 describes the AD_COUNT table. This table tracks the number of times the ads are displayed and clicked through.

The Primary Keys are `AD_IDENTIFIER` and `CONTAINER_UID`.

Table 11-2 AD_COUNT Table Metadata

Column Name	Data Type	Description and Recommendations
`AD_IDENTIFIER`	`NUMBER(15)`	A unique, system-generated number used as a record identifier.
`CONTAINER_UID`	`VARCHAR(50)`	The campaign unique identifier.
`DISPLAY_COUNT`	`NUMBER(15)`	The number of times the ad has been displayed.
`CLICK_THROUGH_COUNT`	`NUMBER(15)`	The number of times the ad has been clicked on.

The PLACEHOLDER Database Table

Table 11-3 describes the PLACEHOLDER table. This table maps placeholder and content bucket services (e.g., ad bucket service).

The Primary Key is `PLACEHOLDER_NAME`.

Table 11-3 PLACEHOLDER Table Metadata

Column Name	Data Type	Description and Recommendations
`PLACEHOLDER_NAME`	`VARCHAR(50)`	PK—a textual name given to the placeholder to uniquely identify it from other placeholders.
`CONTENT_TYPE`	`VARCHAR(20)`	Identifies the type of service to work with (e.g., ad).
`MIX_GLOBALS`	`NUMBER(1)`	Determines whether or not this placeholder is to be used with a specific campaign or not. 0 = do not mix with other adshis placeholder is specific to certain campaign(s). 1 = mix with all ads.
`DESCRIPTION`	`VARCHAR(254)`	A description of the placeholder and its purpose.
`XML_DEFINITION`	`CLOB`	The content used to define the placeholder.

The PLACEHOLDER _PREVIEW Database Table

Table 11-4 describes the PLACEHOLDER_PREVIEW table. This table is used as a mechanism to hold the placeholder for previewing purposes only.

The Primary Key is `PPREVIEW_ID`.

Table 11-4 PLACEHOLDER_PREVIEW Table Metadata

Column Name	Data Type	Description and Recommendations
PREVIEW_ID	NUMBER(15)	PK—a unique, system-generated number used as the record identifier.
XML_DEFINITION	CLOB	The representation of the expression to be previewed.

The WLCS_BOOKMARKS Database Table

Table 11-5 describes the WLCS_BOOKMARKS table. This table is used by the Example portal and is not used except for demonstration purposes. It contains information used in the Bookmark portlet.

The Primary Key is `NAME` and `OWNER`.

Table 11-5 WLCS_BOOKMARKS Table Metadata

Column Name	Data Type	Description and Recommendations
NAME	VARCHAR(150)	The name of the bookmark.
OWNER	VARCHAR(150)	The owner of the bookmark.
URL	VARCHAR(50)	The URL of the bookmark.

The WLCS_CATEGORIES Database Table

Table 11-6 describes the WLCS_CATEGORIES table. This table is used to store category information for the portal portion of the WebLogic Personalization Server application.

Note: The `CATEGORY` feature has not been implemented at this time and, therefore, this table is not being used/populated.

The Primary Key is `NID`.

Table 11-6 WLCS_CATEGORIES

Column Name	Data Type	Description and Recommendations
`NID`	`NUMBER(15)`	Category identifier.
`PORTAL_NID`	`NUMBER(15)`	The Portal identifier. This column is a foreign key to the `NID` column of the `WLCS_PORTAL_DEFINITION` table.
`NAME`	`VARCHAR(100)`	The name for the category.
`ICON_URL`	`VARCHAR(100)`	The URL pointing to the icon associated with the category. This may be null.
`CATEGORY_ORDER`	`NUMBER(5)`	The sequence number identifying the order of display.

The WLCS_COLUMN_INFORMATION Database Table

Table 11-7 describes the WLCS_COLUMN_INFORMATION table. This table is used to store column definition information for each portal and category.

The Primary Key is comprised of `PORTAL_NID`, `CATEGORY_NID` and `COLUMN_ORDER`.

Table 11-7 WLCS_COLUMN_INFORMATION

Column Name	Data Type	Description and Recommendations
`PORTAL_NID`	`NUMBER(15)`	The Portal identifier. This column is a foreign key to the `NID` column of the `WLCS_PORTAL_DEFINITION` table.
`CATEGORY_NID`	`NUMBER(15)`	The Category identifier.
`COLUMN_ORDER`	`NUMBER(5)`	A sequence number identifying the display sequence for this column. Starting at the left-most part of the screen the `COLUMN_ORDER` would be 1.
`COLUMN_WIDTH`	`NUMBER(5)`	The value entered here is a percentage of the screen width. An example would be 30. This represents how wide this particular portal column is to be (30% of the screen).

The WLCS_DOCUMENT Database Table

Table 11-8 describes the WLCS_DOCUMENT table. This table is used to store information pertinent to each document used within the WebLogic Personalization Server.

The Primary Key is `ID`.

Table 11-8 WLCS_DOCUMENT Table Metadata

Column Name	Data Type	Description and Recommendations
`ID`	`VARCHAR(500)`	The identifier of the document. This specifies the relative path (case sensitive using forward slashes) to the actual file.
`DOCUMENT_SIZE`	`NUMBER(15)`	The size of the document in bytes.
`VERSION`	`NUMBER(15)`	The version of the document.
`AUTHOR`	`VARCHAR(50)`	The author’s name of this document.

Table 11-8 WLCS_DOCUMENT Table Metadata (Continued)

Column Name	Data Type	Description and Recommendations
`CREATION_DATE`	`DATE`	The date this document was created in the system.
`LOCKED_BY`	`VARCHAR(50)`	This column identifies who has this document locked for edits or updates.
`MODIFIED_DATE`	`DATE`	This tells you when this document record was last modified.
`MODIFIED_BY`	`VARCHAR(50)`	This column stores the name of the individual who last modified the document record.
`DESCRIPTION`	`VARCHAR(50)`	A description of the document.
`COMMENTS`	`VARCHAR(50)`	An area to store miscellaneous notes about the document.
`MIME_TYPE`	`VARCHAR(100)`	This column identifies which MIME type (or file type) is associated with this document. This is supposed to be MIME 1.0.

The WLCS_DOCUMENT_METADATA Database Table

Table 11-9 describes the WLCS_DOCUMENT_METADATA table. This table is used to store user-defined properties associated with each document.

The Primary Key is `ID` and `NAME`.

Table 11-9 WLCS_DOCUMENT_METADATA Table Metadata

Column Name	Data Type	Description and Recommendations
`ID`	`VARCHAR(500)`	The document identifier. This is a foreign key to the `ID` column of the `WLCS_DOCUMENT` table.
`NAME`	`VARCHAR(240)`	The metadata name.
`VALUE`	`VARCHAR(2000)`	The value to be associated with the metadata name (`NAME`).

Table 11-9 WLCS_DOCUMENT_METADATA Table Metadata (Continued)

Column Name	Data Type	Description and Recommendations
`STATE`	`VARCHAR(50)`	The current state of this metadata property. This is used by Interwoven and can be set to null.

The WLCS_ENTITY_ID Database Table

Table 11-10 describes the WLCS_ENTITY_ID table. Any ConfigurableEntity within the system will have an entry in this table.

The Primary Key is comprised of `JNDI_HOME_NAME` and `PK_STRING`.

Table 11-10 WLCS_ENTITY_ID Table Metadata

Column Name	Data Type	Description and Recommendations
`JNDI_HOME_NAME`	`VARCHAR(100)`	Defines what type of ConfigurableEntity this is.
`PK_STRING`	`VARCHAR(200)`	Unique identifier within the ConfigurableEntity.
`ENTITY_ID`	`NUMBER(15)`	A sequence-generated number providing a unique identifier used throughout the system (in the Property tables and so on).

The WLCS_GROUP Database Table

Table 11-11 describes the WLCS_GROUP table. This table is used to maintain each of the various Group identifiers.

The Primary Key is comprised of `IDENTIFIER`.

Table 11-11 WLCS_GROUP Table Metadata

Column Name	Data Type	Description and Recommendations
`IDENTIFIER`	`VARCHAR(50)`	The group name. This column is a foreign key to the `PK_STRING` column in the `WLCS_ENTITY_ID` table.

The WLCS_GROUP_HIERARCHY Database Table

Table 11-12 describes the WLCS_GROUP_HIERARCHY table. This table stores relationship information between groups.

The Primary Key is comprised of `PARENT_ID` and `CHILD_ID`.

Table 11-12 WLCS_GROUP_HIERARCHY Table Metadata

Column Name	Data Type	Description and Recommendations
`PARENT_ID`	`NUMBER(15)`	The parent group identifier. This column is a foreign key to the `ENTITY_ID` column in the `WLCS_ENTITY_ID` table.
`CHILD_ID`	`NUMBER(15)`	The child group identifier. This column is a foreign key to the `ENTITY_ID` column in the `WLCS_ENTITY_ID` table.

The WLCS_GROUP_PERSONALIZATION Database Table

Table 11-13 describes the WLCS_GROUP_PERSONALIZATION table. Portals can be associated to groups and this table helps establish those relationships and maintain specific information for the group.

The Primary Key is comprised of `PORTAL_NID`, `CATEGORY_NID`, `PORTLET_NID` and `GROUP_NID`.

Table 11-13 WLCS_GROUP_PERSONALIZATION

Column Name	Data Type	Description and Recommendations
`PORTAL_NID`	`NUMBER(15)`	The portal identifier. This column is a foreign key to the `NID` column of the `WLCS_PORTAL_DEFINITION` table.
`CATEGORY_NID`	`NUMBER(15)`	The category identifier. This column is a foreign key to the `NID` column of the `WLCS_CATEGORIES` table.
`PORTLET_NID`	`NUMBER(15)`	The portlet identifier. This column is a foreign key to the `NID` column of the `WLCS_PORTLET_DEFINITION` table.
`GROUP_NID`	`NUMBER(15)`	The group identifier. This column is a foreign key to the `ENTITY_ID` column of the `WLCS_ENTITY_ID` table.
`AVAILABLE`	`NUMBER(5)`	A switch to identify whether or not this portlet is available.
`MANDATORY`	`NUMBER(5)`	This flag, when set, overrides the `VISIBLE` flag and requires the portlet be displayed.
`EDITABLE`	`NUMBER(5)`	This flag determines whether a user is allowed to edit any content.
`MOVEABLE`	`NUMBER(5)`	This column is not being used.
`MINIMIZEABLE`	`NUMBER(5)`	This flag determines whether or not the user will be allowed to minimize the portlet.

Table 11-13 WLCS_GROUP_PERSONALIZATION (Continued)

Column Name	Data Type	Description and Recommendations
`MAXIMIZEABLE`	`NUMBER(5)`	This flag determines whether or not the user will be allowed to maximize the portlet.
`FLOATABLE`	`NUMBER(5)`	This flag determines whether the portlet can open up in its own browser window.
`VISIBLE`	`NUMBER(5)`	This flag determines whether or not the portlet is visible.
`X`	`NUMBER(5)`	The X coordinate determines the placement of the portlet on the screen. This is zero based and refers to the column placement (0=column 1, 1=column 2 and so on).
`Y`	`NUMBER(5)`	The Y coordinate determines placement of the portlet on the screen. Like the X coordinate, it is zero based. The Y coordinate refers to the row placement (0=row 1, 1=row 2 and so on).
`MINIMIZED`	`NUMBER(5)`	This flag determines whether or not the portlet should be displayed in a minimized format when initially displayed.

The WLCS_IS_ALIVE Database Table

Table 11-14 describes the WLCS_IS_ALIVE table. This table is used by the JDBC connection pools to insure the connection to the database is still alive.

Table 11-14 WLCS_IS_ALIVE Table Metadata

Column Name	Data Type	Description and Recommendations
`NAME`	`VARCHAR(100)`	Used by the JDBC connection pools to insure the connection to the database is still alive.

The WLCS_LDAP_CONFIG Database Table

Table 11-15 describes the WLCS_LDAP_CONFIG table. This table holds configuration information for LDAP functionality within the User Management module.

The Primary Key is `LDAP_PROPERTY`.

Table 11-15 WLCS_LDAP_CONFIG Table Metadata

Column Name	Data Type	Description and Recommendations
`LDAP_PROPERTY`	`VARCHAR(100)`	The property name.
`LDAP_VALUE`	`VARCHAR(254)`	The property value.

The WLCS_PROP_BOOLEAN Database Table

Table 11-16 describes the WLCS_PROP_BOOLEAN table. This table stores property values for boolean properties.

The Primary Key is `PROPERTY_ID`.

Table 11-16 WLCS_PROP_BOOLEAN Table Metadata

Column Name	Data Type	Description and Recommendations
`PROPERTY_ID`	`NUMBER(15)`	The identifier for each boolean property.
`VALUE`	`NUMBER(3)`	The value for each boolean property identifier.

The WLCS_PROP_DATETIME Database Table

Table 11-17 describes the WLCS_PROP_DATETIME table. This table stores property values for date and time properties.

The Primary Key is `PROPERTY_ID`.

Table 11-17 WLCS_PROP_DATETIME Table Metadata

Column Name	Data Type	Description and Recommendations
`PROPERTY_ID`	`NUMBER(15)`	The identifier for each date and time property.
`VALUE`	`DATE`	The value for each date and time property identifier.

The WLCS_PROP_FLOAT Database Table

Table 11-18 describes the LCS_PROP_FLOAT table. This table stores property values for float properties.

The Primary Key is `PROPERTY_ID`.

Table 11-18 WLCS_PROP_FLOAT Table Metadata

Column Name	Data Type	Description and Recommendations
`PROPERTY_ID`	`NUMBER(15)`	The identifier for each float property.
`VALUE`	`NUMBER`	The value associated with each float property identifier.

The WLCS_PROP_ID Database Table

Table 11-19 describes the WLCS_PROP_ID table. Any property assigned to a ConfigurableEntity has a unique PROPERTY_ID. This identifier and associated information is stored here.

The Primary Key is `ENTITY_ID`, `PROPERTY_NAME` and `SCOPE_NAME`.

Table 11-19 WLCS_PROP_ID Table Metadata

Column Name	Data Type	Description and Recommendations
`ENTITY_ID`	`NUMBER(15)`	A system generated value and foreign key to the `WLCS_ENTITY_ID` column.
`SCOPE_NAME`	`VARCHAR(100)`	This column may be null. If this property is defined in a property set, then the `SCOPE_NAME` will match the `SCHEMA_NAME` for that property set in the `WLCS_SCHEMA` table.
`PROPERTY_NAME`	`VARCHAR(100)`	The name of the property.
`PROPERTY_TYPE`	`NUMBER(3)`	This column identifies the type of property we are dealing with (for example, boolean, integer, float, text, and so on).
`PROPERTY_META_DATA_ID`	`NUMBER(15)`	The identifier for the Property metadata information. Again, we use the `PROPERTY_TYPE` column to identify which type of Property metadata we are looking at (for example, boolean, integer, and so on).
`SCHEMA_HAS_CHANGED`	`NUMBER(3)`	A flag informing to identify whether anything in the `WLCS_SCHEMA` or `WLCS_PROP_MD_xxx` tables has changed. If so, then certain cleanup activities must be performed prior to using this property next time.
`PROPERTY_ID`	`NUMBER(15)`	The property identifier is a unique system-generated number.

The WLCS_PROP_INTEGER Database Table

Table 11-20 describes the WLCS_PROP_INTEGER table. This table stores property values for integer properties.

The Primary Key is `PROPERTY_ID`.

Table 11-20 WLCS_PROP_INTEGER Table Metadata

Column Name	Data Type	Description and Recommendations
PROPERTY_ID	NUMBER(15)	The identifier of the integer property.
VALUE	NUMBER(20)	The value associated with the integer property.

The WLCS_PROP_MD Database Table

Table 11-21 describes the WLCS_PROP_MD table. This table stores information about defined properties in a property set.

The Primary Keys are `SCHEMA_ID` and `PROPERTY_NAME`.

Table 11-21 WLCS_PROP_MD Table Metadata

Column Name	Data Type	Description and Recommendations
SCHEMA_ID	NUMBER(15)	A foreign key to the `WLCS_SCHEMA` table.
PROPERTY_NAME	VARCHAR(100)	The name of a property.
DESCRIPTION	VARCHAR(254)	A description of the property.
IS_RESTRICTED	NUMBER(3)	If set `TRUE`, the value of the property is constrained to a set of values. 0 equates to `FALSE` and 1 equates to `TRUE`.
IS_EXPLICIT	NUMBER(3)	If set `TRUE`, the property value may be coming from an external source. 0 equates to `FALSE` and 1 equates to `TRUE`.

Table 11-21 WLCS_PROP_MD Table Metadata (Continued)

Column Name	Data Type	Description and Recommendations
`IS_MULTIVALUED`	`NUMBER(3)`	Some properties may have more than one value. 0 equates to `FALSE` and 1 equates to `TRUE`.
`PROPERTY_TYPE`	`NUMBER(3)`	Defines the property type (boolean, text and so on).
`PROPERTY_META_DATA_ID`	`NUMBER(15)`	The primary key is a unique, system-generated value.

The WLCS_PROP_MD_BOOLEAN Database Table

Table 11-22 describes the WLCS_PROP_MD_BOOLEAN table. This table stores property set definitions for the boolean property type.

The Primary Key is `PROPERTY_META_DATA_ID`.

Table 11-22 WLCS_PROP_MD_BOOLEAN Table Metadata

Column Name	Data Type	Description and Recommendations
`PROPERTY_META_DATA_ID`	`NUMBER(15)`	A unique identifier for this Property metadata and foreign key to the `WLCS_PROP_MD` table.
`VALUE`	`NUMBER(3)`	The value associated with the Property metadata.
`IS_DEFAULT`	`NUMBER(3)`	This flag tells us whether or not the `VALUE` column is the default value for this piece of Property metadata. 0 equates to `FALSE` and 1 equates to `TRUE`.

The WLCS_PROP_MD_DATETIME Database Table

Table 11-23 describes the WLCS_PROP_MD_DATETIME table. This table stores property set definitions for the date and time property type.

The Primary Key is `PROPERTY_META_DATA_ID`.

Table 11-23 WLCS_PROP_MD_DATETIME Table Metadata

Column Name	Data Type	Description and Recommendations
`PROPERTY_META_DATA_ID`	`NUMBER(20)`	A unique identifier for this Property metadata.
`VALUE`	`DATE`	The value associated with the Property metadata.
`IS_DEFAULT`	`NUMBER(3)`	This flag tells us whether or not the `VALUE` column is the default value for this piece of Property metadata. 0 equates to `FALSE` and 1 equates to `TRUE`.

The WLCS_PROP_MD_FLOAT Database Table

Table 11-24 describes the WLCS_PROP_MD_FLOAT table. This table stores property set definitions for the float property type.

The Primary Key is `PROPERTY_META_DATA_ID`.

Table 11-24 WLCS_PROP_MD_FLOAT Table Metadata

Column Name	Data Type	Description and Recommendations
`PROPERTY_META_DATA_ID`	`NUMBER(15)`	A unique identifier for this Property metadata.
`VALUE`	`NUMBER`	The value associated with the Property metadata.
`IS_DEFAULT`	`NUMBER(3)`	This flag tells us whether or not the `VALUE` column is the default value for this piece of Property metadata. 0 equates to `FALSE` and 1 equates to `TRUE`.

The WLCS_PROP_MD_INTEGER Database Table

Table 11-25 describes the WLCS_PROP_MD_INTEGER table. This table stores property set definitions for the Integer property type.

The Primary Key is `PROPERTY_META_DATA_ID`.

Table 11-25 WLCS_PROP_MD_INTEGER Table Metadata

Column Name	Data Type	Description and Recommendations
`PROPERTY_META_DATA_ID`	`NUMBER(15)`	A unique identifier for this Property metadata.
`VALUE`	`NUMBER(20)`	The value associated with the Property metadata.
`IS_DEFAULT`	`NUMBER(3)`	This flag tells us whether or not the `VALUE` column is the default value for this piece of Property metadata. 0 equates to `FALSE` and 1 equates to `TRUE`.

The WLCS_PROP_MD_TEXT Database Table

Table 11-26 describes the WLCS_PROP_MD_TEXT table. This table stores property set definitions for the text property type.

The Primary Key is `PROPERTY_META_DATA_ID`.

Table 11-26 WLCS_PROP_MD_TEXT Table Metadata

Column Name	Data Type	Description and Recommendations
`PROPERTY_META_DATA_ID`	`NUMBER(15)`	A unique identifier for this Property metadata.
`VALUE`	`VARCHAR(254)`	The value associated with the Property metadata.
`IS_DEFAULT`	`NUMBER(3)`	This flag tells us whether or not the `VALUE` column is the default value for this piece of Property metadata. 0 equates to `FALSE` and 1 equates to `TRUE`.

The WLCS_PROP_MD_USER_DEFINED Database Table

Table 11-27 describes the WLCS_PROP_MD_USER_DEFINED table. This table stores property set definitions for any user defined property type.

The Primary Key is `PROPERTY_META_DATA_ID`.

Table 11-27 WLCS_PROP_MD_USER_DEFINED Table Metadata

Column Name	Data Type	Description and Recommendations
`PROPERTY_META_DATA_ID`	`NUMBER(15)`	A unique identifier for this Property metadata.
`VALUE`	`BLOB`	The value associated with the Property metadata.
`IS_DEFAULT`	`NUMBER(3)`	This flag tells us whether or not the `VALUE` column is the default value for this piece of Property metadata. 0 equates to `FALSE` and 1 equates to `TRUE`.

The WLCS_PROP_TEXT Database Table

Table 11-28 describes the WLCS_PROP_TEXT table. This table stores property values for the text for the text property type.

The Primary Key is `PROPERTY_ID`.

Table 11-28 WLCS_PROP_TEXT Table Metadata

Column Name	Data Type	Description and Recommendations
`PROPERTY_ID`	`NUMBER(15)`	The identifier of the text property.
`VALUE`	`VARCHAR(254)`	The value associated with the text property.

The WLCS_PROP_USER_DEFINED Database Table

Table 11-29 describes the WLCS_PROP_USER_DEFINED table. This table stores property values for any user-defined property type.

The Primary Key is `PROPERTY_ID`.

Table 11-29 WLCS_PROP_USER_DEFINED Table Metadata

Column Name	Data Type	Description and Recommendations
`PROPERTY_ID`	`NUMBER(15)`	The identifier of the user-defined property.
`VALUE`	`BLOB`	The value associated with the user-defined property.

The RULESET Database Table

Table 11-30 describes the RULESET table. This table contains all of the rule sets.

The Primary Key is `NAME`.

Table 11-30 RULESET Table Metadata

Column Name	Data Type	Description and Recommendations
`NAME`	`VARCHAR(50)`	PK—the rule name. A unique name to differentiate it from other rules.
`DOCUMENT`	`CLOB`	The XML document containing the rule set definition.

The WLCS_RULESET_DEFINITION Database Table

Table 11-31 describes the WLCS_RULESET_DEFINITION table. This table contains all rule sets.

The Primary Key is NAME.

Table 11-31 WLCS_RULESET_DEFINITION Table Metadata

Column Name	Data Type	Description and Recommendations
NAME	VARCHAR(50)	The identifier, or name, of the rule set.
DOCUMENT	BLOB	The XML document containing the rule set definition.

The WLCS_SCHEMA Database Table

Table 11-32 describes the WLCS_SCHEMA table. This table stores property set definitions.

The Primary Keys are SCHEMA_GROUP_NAME and SCOPE_NAME.

Table 11-32 WLCS_SCHEMA Table Metadata

Column Name	Data Type	Description and Recommendations
SCHEMA_GROUP_NAME	VARCHAR(100)	The type of object this schema is used for.
SCOPE_NAME	VARCHAR(100)	The application name since it is defining names for the application.
DESCRIPTION	VARCHAR(254)	A description of the schema.
SCHEMA_ID	NUMBER(15)	A system-generated number used throughout the application.

The WLCS_SEQUENCER Database Table

Table 11-33 describes the WLCS_SEQUENCER table. The WLCS_SEQUENCER table is used to maintain all of the sequence identifiers (for example, property_meta_data_id_sequence, and so on) used in the application.

The Primary Key is `SEQUENCE_NAME`.

Table 11-33 WLCS_SEQUENCER Table Metadata

Column Name	Data Type	Description and Recommendations
`SEQUENCE_NAME`	`VARCHAR(50)`	A unique name used to identify the sequence.
`CURRENT_VALUE`	`NUMBER(15)`	The current value of the sequence.
`IS_LOCKED`	`NUMBER(1)`	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 WLCS_TODO Database Table

Table 11-34 describes the WLCS_TODO table. This table is used by the Example portal and is not used except for demonstration purposes. It contains information used in the To Do portlet.

The Primary Key is `ITEM` and `OWNER`.

Table 11-34 WLCS_TODO Table Metadata

Column Name	Data Type	Description and Recommendations
`ITEM`	`VARCHAR(50)`	The activity to be accomplished.
`OWNER`	`VARCHAR(150)`	The individual who owns, or is responsible for, this activity.
`DONE`	`NUMBER(5)`	The status identifying whether this item has been completed.

Table 11-34 WLCS_TODO Table Metadata (Continued)

Column Name	Data Type	Description and Recommendations
PRIORITY	NUMBER(5)	The priority of the activity.

The WLCS_UIDS Database Table

Table 11-35 describes the WLCS_UIDS table. This table stores sequence information in a generic database independent format.

The Primary Key is SID.

Table 11-35 WLCS_UIDS Table Metadata

Column Name	Data Type	Description and Recommendations
SID	VARCHAR(100)	The name of the sequence.
NEXT_SEQUENCE	NUMBER(15)	The next value available for use with the sequence.

The WLCS_UNIFIED_PROFILE_TYPE Database Table

Table 11-36 describes the WLCS_UNIFIED_PROFILE_TYPE table. This table allows registration of classes which extend the ProvidedUser class.

The Primary Key is TYPE_NAME.

Table 11-36 WLCS_UNIFIED_PROFILE_TYPE Table Metadata

Column Name	Data Type	Description and Recommendations
TYPE_NAME	VARCHAR(100)	Any unique name used for easy lookup.
CLASS_NAME	VARCHAR(100)	The name of the remote interface class.
HOME	VARCHAR(100)	The name of the home class.

Table 11-36 WLCS_UNIFIED_PROFILE_TYPE Table Metadata (Continued)

Column Name	Data Type	Description and Recommendations
PK	VARCHAR(100)	The name of the primary key class.
JNDI_NAME	VARCHAR(100)	The name to look up in the JNDI tree.
SUCCESSOR	VARCHAR(100)	This column allows you to define another class should the TYPE_NAME not exist. This column is a foreign key to TYPE_NAME of the WLCS_UNIFIED_PROFILE_TYPE table.

The WLCS_USER Database Table

Table 11-37 describes the WLCS_USER table. This table stores all user login/password combinations.

The Primary Key is IDENTIFIER.

Table 11-37 WLCS_USER Table Metadata

Column Name	Data Type	Description and Recommendations
IDENTIFIER	VARCHAR(50)	The user login. This column is a foreign key to the PK_STRING column of the WLCS_ENTITY_ID table.
PASSWORD	VARCHAR(50)	The encrypted password.
IS_EXTERNAL	NUMBER(3)	This flag determines whether a user came from an external realm as opposed to the internal database realm.
PROFILE_TYPE	VARCHAR(100)	A foreign key to the TYPE_NAME in the WLCS_UNIFIED_PROFILE_TYPE table.

The WLCS_USER_GROUP_CACHE Database Table

Table 11-38 describes the WLCS_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 11-38 WLCS_USER_GROUP_CACHE Table Metadata

Column Name	Data Type	Description and Recommendations
USER_NAME	VARCHAR(100)	FK—foreign key to WLCS_USER.IDENTIFIER.
GROUP_NAME	(VARCHAR(100)	FK—foreign key to WLCS_GROUP.IDENTIFIER.

The WLCS_USER_GROUP_HIERARCHY Database Table

Table 11-39 describes the WLCS_USER_GROUP_HIERARCHY table. This table allows you to store associated users and groups.

The Primary Key is comprised of `USER_ID` and `GROUP_ID`.

Table 11-39 WLCS_USER_GROUP_HIERARCHY Table Metadata

Column Name	Data Type	Description and Recommendations
USER_ID	NUMBER(15)	The `ENTITY_ID` of a user. This column is a foreign key to the `USER_ID` column of the `WLCS_ENTITY_ID` table.
GROUP_ID	NUMBER(15)	The `ENTITY_ID` of a group. This column is a foreign key to the `USER_ID` column of the `WLCS_ENTITY_ID` table.

The WLCS_USER_PERSONALIZATION Database Table

Table 11-40 describes the WLCS_USER_PERSONALIZATION table.This table contains personalized portal information for the user.

The Primary Key is comprised of PORTAL_NID, CATEGORY_NID, GROUP_NID, USER_NID and PORTLET_NID.

Table 11-40 WLCS_USER_PERSONALIZATION

Column Name	Data Type	Description and Recommendations
PORTAL_NID	NUMBER(15)	The portal identifier. This column is a foreign key to the NID column of the WLCS_PORTAL_DEFINITION table.
CATEGORY_NID	NUMBER(15)	The category identifier. This column is a foreign key to the NID column of the WLCS_CATEGORIES table.
GROUP_NID	NUMBER(15)	The group identifier. This column is a foreign key to the ENTITY_ID column of the WLCS_ENTITY_ID table.
USER_NID	NUMBER(15)	The user identifier. This column is a foreign key to the ENTITY_ID column of the WLCS_ENTITY_ID table.
PORTLET_NID	NUMBER(15)	The portlet identifier. This column is a foreign key to the NID column of the WLCS_PORTLET table.
VISIBLE	NUMBER(5)	This flag determines whether or not the portlet is visible. 0 equates to FALSE and 1 equates to TRUE.
X	NUMBER(5)	The X coordinate determines the placement of the portlet on the screen. This is zero based and refers to the column placement (0=column 1, 1=column 2 and so on).

Table 11-40 WLCS_USER_PERSONALIZATION (Continued)

Column Name	Data Type	Description and Recommendations
Y	NUMBER(5)	The Y coordinate determines placement of the portlet on the screen. Like the X coordinate, it is zero based. The Y coordinate refers to the row placement (0=row 1, 1=row 2 and so on).
MINIMIZED	NUMBER(5)	This flag determines whether or not the portlet should be displayed in a minimized format when displayed initially. 0 equates to FALSE and 1 equates to TRUE.

The WLCS_UUP_EXAMPLE Database Table

Table 11-41 describes the WLCS_UUP_EXAMPLE table. This is an example of how to use the Unified Profile Types.

The Primary Key is NAME.

Table 11-41 WLCS_UUP_EXAMPLE Table Metadata

Column Name	Data Type	Description and Recommendations
NAME	VARCHAR(100)	A username.
POINTS	NUMBER(15)	A point accumulator based on various actions taken by the user.

The SQL Scripts Used to Create the Database

The database schemas for the WebLogic Personalization Server, WebLogic Commerce Server and BEA's Campaign Manager for WebLogic are all created by executing the `create_all` script for the target database environment.

Cloudscape

For Cloudscape, execute one of the following:

- `WL_COMMERCE_HOME\db\cloudscape\3.5.1\create_all.bat` (Windows)
- `WL_COMMERCE_HOME/db/cloudscape/3.5.1/create_all.sh` (UNIX)

Script Name	Description
`create_all.bat`	The execution of this script will create the WLPS, WLCS and Campaign Manager database schema.
`create_all.sh`	The execution of this script will create the WLPS, WLCS and Campaign Manager database schema.
`create_campaign.sql`	Creates the Campaign Manager specific database objects (e.g., tables, indexes and constraints).
`create_common.sql`	Creates the database objects which are common to WLPS and WLCS.
`create_mail_ad.sql`	Creates all the database objects used by the mail messaging component.
`create_wlcs.sql`	Creates all the database objects for WLCS (including Catalog and Order Management).
`create_wlps.sql`	Creates all the database object for WLPS.
`drop_campaign.sql`	Drops all database objects associated with Campaign Manager.
`drop_common.sql`	Drops the database objects which are common between WLPS and WLCS.

Script Name	Description
`drop_mail_ad.sql`	Drops the database objects used by the mail messaging component.
`drop_wlcs.sql`	Drops the database objects associated with WLCS.
`drop_wlps.sql`	Drops the database objects associated with WLPS.
`insert_common.sql`	Inserts core data into the common tables between WLPS and WLCS.
`insert_wlcs.sql`	Inserts core data into some of the WLCS tables.
`insert_wlcs_sample_catalog.sql`	Inserts sample data into the product catalog.
`insert_wlcs_sample_customer.sql`	Inserts sample customer information into WLCS tables.
`insert_wlcs_sample_data.sql`	Inserts sample data into various WLCS tables.
`insert_wlps.sql`	Inserts core data into WLPS tables.
`insert_wlps_sample_data.sql`	Inserts sample data into various WLPS tables.

Oracle

For Oracle, from the command line, move to the following directory:

```
WL_COMMERCE_HOME/db/oracle/8.1.6
```

After logging into SQL*Plus, simply execute the `create_all.sql` script (e.g., `@create_all`).

Script Name	Description
`create_campaign.sql`	Creates the Campaign Manager specific database objects (e.g., tables, indexes and constraints).
`create_common.sql`	Creates the database objects which are common to WLPS and WLCS.

Script Name	Description
`create_mail_ad.sql`	Creates all the database objects used by the mail messaging component.
`create_wlcs.sql`	Creates all the database objects for WLCS (including Catalog and Order Management).
`create_wlps.sql`	Creates all the database object for WLPS.
`drop_campaign.sql`	Drops all database objects associated with Campaign Manager.
`drop_common.sql`	Drops the database objects which are common between WLPS and WLCS.
`drop_mail_ad.sql`	Drops the database objects used by the mail messaging component.
`drop_wlcs.sql`	Drops the database objects associated with WLCS.
`drop_wlps.sql`	Drops the database objects associated with WLPS.
`insert_common.sql`	Inserts core data into the common tables between WLPS and WLCS.
`insert_wlcs.sql`	Inserts core data into some of the WLCS tables.
`insert_wlcs_sample_catalog.sql`	Inserts sample data into the product catalog.
`insert_wlcs_sample_customer.sql`	Inserts sample customer information into WLCS tables.
`insert_wlcs_sample_data.sql`	Inserts sample data into various WLCS tables.
`insert_wlps.sql`	Inserts core data into WLPS tables.
`insert_wlps_sample_data.sql`	Inserts sample data into various WLPS tables.
`install_report.sql`	This script is used to summarize the database installation. Information such as the number of tables, indexes, etc., is displayed.
`statistics.sql`	This script is used in computing statistics on various database objects (e.g., tables and indexes) in an Oracle environment.

SQL Server

For SQL Server, you must first edit the `create_all.bat` file and properly identify the values for the variables used in identifying the target database environment (for example, `user_id`, `password` and `server`). Once the variables have been set properly, execute `create_all.bat` from the command line.

Script Name	Description
`create_all.bat`	The execution of this script will create the WLPS, WLCS and Campaign Manager database schema.
`create_campaign.sql`	Creates the Campaign Manager specific database objects (e.g., tables, indexes and constraints).
`create_common.sql`	Creates the database objects which are common to WLPS and WLCS.
`create_mail_ad.sql`	Creates all the database objects used by the mail messaging component.
`create_wlcs.sql`	Creates all the database objects for WLCS (including Catalog and Order Management).
`create_wlps.sql`	Creates all the database object for WLPS.
`drop_campaign.sql`	Drops all database objects associated with Campaign Manager.
`drop_common.sql`	Drops the database objects which are common between WLPS and WLCS.
`drop_mail_ad.sql`	Drops the database objects used by the mail messaging component.
`drop_wlcs.sql`	Drops the database objects associated with WLCS.
`drop_wlps.sql`	Drops the database objects associated with WLPS.
`insert_common.sql`	Inserts core data into the common tables between WLPS and WLCS.
`insert_wlcs.sql`	Inserts core data into some of the WLCS tables.
`insert_wlcs_sample_catalog.sql`	Inserts sample data into the product catalog.

Script Name	Description
`insert_wlcs_sample_customer.sql`	Inserts sample customer information into WLCS tables.
`insert_wlcs_sample_data.sql`	Inserts sample data into various WLCS tables.
`insert_wlps.sql`	Inserts core data into WLPS tables.
`insert_wlps_sample_data.sql`	Inserts sample data into various WLPS tables.
`readme.txt`	Documentation outlining the appropriate steps necessary for proper installation of the WLPS, WLCS and Campaign Manager database schema.

Defined Constraints

For some of the database tables described earlier in this chapter, the SQL files define constraints. Table 11-42 shows the table name and describes the constraint(s) defined for it.

Table 11-42 Constraints Defined on Campaign Manager Database Tables

Table Name	Constraints as Defined in create-catalog-oracle.sql
`PLACEHOLDER`	A check constraint (`CK_MIX_GLOBALS`) ensures the column `MIX_GLOBALS` is populated with either a 0 or 1.
`AD_BUCKET`	A referential integrity constraint (`FL_PLACEHOLDER_AD`) ensures that a `PLACEHOLDER` exists.

CHAPTER

12 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-commerce services.
The `esp:` prefix stands for e-commerce services portal.
The `pz:` prefix stands for personalization.

This topic includes the following sections:

<fm:removeCachedAttribute>
<fm:removeSessionAttribute>

- Internationalization
  <i18n:localize>
  <i18n:getMessage>
- Personalization Tags
  <pz:contentQuery>
  <pz:contentSelector>
  <pz:div>
- Placeholders
  <ph:placeholder>
- Property Sets
  <ps:getPropertyNames>
  <ps:getPropertySetNames>
- 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:changeGroupName>
  <um:createGroup>
  <um:createUser>
  <um:getChildGroupNames>
  <um:getChildGroups>
  <um:getGroupNamesForUser>
  <um:getParentGroupName>
  <um:getTopLevelGroups>
  <um:getUsernames>
  <um:getUsernamesForGroup>
  <um:removeGroup>
  <um:removeGroupFromGroup>
  <um:removeUser>
  <um:removeUserFromGroup>

- User Management: Security Tags
  <um:login>
  <um:logout>
  <um:setPassword>
- Utility Tags: Personalization Utilities
  <es:counter>
  <es:date>
  <es:forEachInArray>
  <es:isNull>
  <es:monitorSession>
  <es:notNull>
  <es:simpleReport>
  <es:simpleReport>
  <es:transposeArray>
  <es:uriContent>
- Utility Tags: WebLogic Utilities
  <wl:process>
  <wl:repeat>
  <wl:cache>

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 12-1) 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 12-1 <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 12-1 <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

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:get Property>` tag (Table 12-2) retrives the value of the specified content metadata property into a variable specified by `resultId`. It does not have a body. If `resultId` is not specified, the value will be inlined into the page, similar to the `<cm:printProperty>` tag. This tag operates on any ConfigurableEntity, not just the Content object. However, it does not support ConfigurableEntity successors.

Table 12-2 <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 12-2 <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 12-2 <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 12-3) 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 12-3 <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.beasys.commerce.axiom.document.Document to use. If this is specified and non-null, `id` will be ignored. Otherwise, `id` will be used.	R

Table 12-3 <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, the `<cm:printDoc>` will use the `FlowManagerHelper.getAppliactionFlowManager()` method to determine if the tag is operating under a FlowManager instance (a personalized application, a WebFlowed application, a portal).

Additionally, if `baseHref` is provided, then, after printing the document, the `<cm:printDoc>` tag will output a <BASE HREF> based upon the URL of the outermost page.

Example

To get a Document object from an `id` in the `request` attributes and inline the Document's text (which might contain relative links):

```
<%@ 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 12-4) 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 12-4 <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-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

Table 12-4 <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 12-76), zero can be used to iterate over the array of `Content` objects. This tag (Table 12-5) supports generic `Content` via a `ContentManager` interface.

Table 12-5 <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 12-5 <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
expr	No	Expression	The com.beasys.commerce.foundation.expression.Expression object to use to search for content. If this is null or not specified, then `query` must be specified. Otherwise, `query` is ignored.	R
useCache	No	String, Boolean	Determines whether Content is cached. This attribute can have one of two values: `False` (default value): ContentCache is not used. If `false` (not specified), the `cacheId`, `cacheScope` and `cacheTimeout` settings are ignored. `True`: ContentCache is used.	R
cacheId	No	String	The identifier name used to cache the Content. Internally, the cache is implemented as a Map; this will become the key. If not specified, the `id` attribute of the tag is used.	R

Table 12-5 <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
readOnly	No	String, Boolean	This attribute can have one of two values: If `true`, the ContentManager (specified via the `ContentHome` attribute) will try to return only lightweight (non-EJB) objects where possible. If `false` (not specified), the default value is used. Default= `ContentHelper.DEF_CONTENT_READ ONLY` (which is loaded from the `commerce.content.defaultReadOn ly` property in the weblogiccommerce.properties file).	R

Example

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

```
<%@ 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.beasys.commerce.axiom.content.Content">
       <li><cm:printProperty id="newsItem" name="Title"
       encode="html" />
   </es:forEachInArray>
</ul>
```

<cm:selectById>

The `<cm:selectById>` tag (Table 12-6) 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 12-6 <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

Table 12-6 <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 12-6 <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
readOnly	No	String, Boolean	This attribute can have one of two values: If `true`, the ContentManager (specified via the `ContentHome` attribute) will try to return only lightweight (non-EJB) objects where possible. If `false` (not specified), the default value is used. Default= `ContentHelper.DEF_CONTENT_READONLY` (which is loaded from the `commerce.content.defaultReadOnly` property in the weblogiccommerce.properties file).	R

Example

To fetch the `Document` (using ContentCaching) with an identifier of `1234` and inline its content:

```
<%@ 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" />
```

Flow Manager

Thr Flow Manager tags are used for accessing the session, session cache, or the global cache. For scalability reasons, it is best to limit what gets placed into the session. For large sessions, session replication across servers is very costly. This tag library will give the user the ability to write to data that can be scoped to the application or across applications.

<fm:getApplicationURI>

The `<fm:getApplicationURI>` tag (Table 12-7) gets the application from the URL: http://localhost:7001/portals/application/exampleportal.

When `includeContext="true"`, the tag returns `/context/path/pathinfo`, for example: `/portals/application/exampleportal`. This is required when a client browser needs to address the Web application context, for example, when using a form.

When `includeContext="false"`, the tag returns `/path/pathinfo`, for example `/application/exampleportal`. This is required when using Web applications and server side processing.

Table 12-7 <fm:getApplicationURI>

Tag Attribute	Required	Type	Description	R/C
id	Yes	String	The application as referenced by the Flow Manager. It can either get the value with the context or without. When used within a Web application, you must get the value without the context when using `<jsp:forward>`.	C
includeContext	No	boolean	Determines whether or not to include the servlet context with the application name. Defaults to `true`.	R

Example

```
<%@ taglib uri="fm.tld" prefix="fm" %>
<%@ taglib uri="weblogic.tld" prefix="wl" %>
.
.
.
<wl:process name="formSubmit">
       <fm:getApplicationURI id="uri" includeContext="false"/>
       <jsp:forward page="<%=uri%>"/>
</wl:process>
```

<fm:getCachedAttribute>

The `<fm:getCachedAttribute>` tag (Table 12-8) gets an attribute out of the session/global cache. This value can be scoped to the application or not.

Table 12-8 <fm:getCachedAttribute>

Tag Attribute	Required	Type	Description	R/C
id	Yes	Object	The variable to store the retrieved value.	C
name	Yes	String	The name of the name/value pair.	R
scoped	No	boolean	The name/value pair scoped to the application. Defaults to `true`.	R
global	No	boolean	The value scoped to the session or the global scope. Defaults to `false`.	R

Example

```
<%@ taglib uri="fm.tld" prefix="fm" %>
.
.
.
    <%Portal portal = null;%>
    <fm:getCachedAttribute id="tportal"
```

```
          name="<%=PortalTagConstants.CACHED_PORTAL%>"
global="true" />
   <es:isNull item="<%=tportal%>" >
       <esp:portalManager action="get" id="myPortal"
               portalName="<%=portalName%>"/>
       <%tportal=myPortal;%>
       <fm:setCachedAttribute
               name="<%=PortalTagConstants.CACHED_PORTAL%>"
value="<%=myPortal%>" global="true" />
   </es:isNull>
   <%portal=(Portal)tportal;%>
```

<fm:getSessionAttribute>

The `<fm:getSessionAttribute>` tag (Table 12-9) gets an attribute out of the HttpSession. The attribute may be scoped to the application (by default).

Table 12-9 <fm:getSessionAttribute>

Tag Attribute	Required	Type	Description	R/C
id	Yes	Object	The variable to store the retrieved value.	C
name	Yes	String	The name of the name/value pair.	R
scoped	No	boolean	The name/value pair scoped to the application. Defaults to `true`.	R

Example

```
<%@ taglib uri="fm.tld" prefix="fm" %>
.
.
.
<fm:getSessionAttribute id="username" name="portal.username"
     scoped="true" />
```

The name is: `<%=username%>`

<fm:removeCachedAttribute>

The `<fm:removeCachedAttribute>` tag (Table 12-10) removes an attribute from the session/global cache. This value can be scoped to the application or not.

Table 12-10 <fm:removeCachedAttribute>

Tag Attribute	Required	Type	Description	R/C
name	Yes	String	The name of the name/value pair.	R
scoped	No	boolean	The name/value pair scoped to the application. Defaults to `true`.	R
global	No	boolean	The value scoped to the session or the global scope. Defaults to `false`.	R

Example

```
<%@ taglib uri="fm.tld" prefix="fm" %>
.
.
.
<fm:removeCachedAttribute
     name="<%=PortalTagConstants.CACHED_PORTAL%>" global="true" />
```

<fm:removeSessionAttribute>

The `<fm:removeSessionAttribute>` tag (Table 12-11) removes an attribute from the HttpSession. The attribute may be scoped to the application (by default).

Table 12-11 <fm:removeSessionAttribute>

Tag Attribute	Required	Type	Description	R/C
name	Yes	String	The name of the name/value pair.	R

Table 12-11 <fm:removeSessionAttribute> (Continued)

Tag Attribute	Required	Type	Description	R/C
scoped	No	boolean	The name/value pair scoped to the application. Defaults to `true`.	R

Example

```
<%@ taglib uri="fm.tld" prefix="fm" %>
.
.
.
<fm:removeSessionAttribute name="portal.username" scoped="true" />
```

<fm:setCachedAttribute>

The `<fm:setCachedAttribute>` tag (Table 12-12) sets an attribute in the session/global cache. This value can be scoped to the application or not.

Table 12-12 <fm:setCachedAttribute>

Tag Attribute	Required	Type	Description	R/C
name	Yes	String	The name of the name/value pair.	R
scoped	No	boolean	The name/value pair scoped to the application. Defaults to `true`.	R
global	No	boolean	The value scoped to the session or the global scope. Defaults to `false`.	R
value	Yes	Object	The value to set.	R

Example

```
<%@ taglib uri="fm.tld" prefix="fm" %
.
.
.
  <%Portal portal = null;%>
  <fm:getCachedAttribute id="tportal"
         name="<%=PortalTagConstants.CACHED_PORTAL%>"
global="true" />
  <es:isNull item="<%=tportal%>" >
     <esp:portalManager action="get" id="myPortal"
         portalName="<%=portalName%>"/>
     <%tportal=myPortal;%>
     <fm:setCachedAttribute
         name="<%=PortalTagConstants.CACHED_PORTAL%>"
value="<%=myPortal%>" global="true" />
  </es:isNull>
  <%portal=(Portal)tportal;%>
```

<fm:setSessionAttribute>

The `<fm:setSessionAttribute>` tag (Table 12-13) sets an attribute in the HttpSession. The attribute may be scoped to the application (by default).

Table 12-13 <fm:setSessionAttribute>

Tag Attribute	Required	Type	Description	R/C
name	Yes	String	The name of the name/value pair.	R
scoped	No	boolean	The name/value pair scoped to the application. Defaults to `true`.	R
value	Yes	Object	The value to set.	R

Example

```
<%@ taglib uri="fm.tld" prefix="fm" %>
.
.
.
<% String val = “joe developer”; %>
<fm:setSessionAttribute name=“portal.username”
     value=“<%= val %>” scoped=“true” />
```

Internationalization

These tags are used in the localization of JSP pages that have an internationalization requirement.

Use the following code to import the utility tag library:
`<%@ taglib uri="i18n.tld" prefix="i18n" %>`

Note: In the following tables, 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 12-14) also specifies a character encoding and content type to be specified for a JSP page. Because of this, the tag should be used as early in the page as possible—before anything is written to the output stream—so that the bytes are properly encoded.

Note: When an HTML page is included in a larger page, only the larger page can use the `<i18n:localize>` tag. This is because the `<i18n:localize>` tag sets the encoding for the page, and the encoding must be set in the parent (including) page before any bytes are written to the response's output stream. The parent page must set an encoding that is sufficient for all the content on that page as well as any included pages.

Note: Do not use the `<i18n:localize>` tag in conjunction with the `<%@ page contentType="<something>" >` page directive defined in the JSP specification. The directive is unnecessary if you are using this tag, and can result in inconsistent or wrong `contentType` declarations.

Table 12-14 <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 12-14 <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 approprite for the chosen language is used.	R
contentType	No	String	The type of content contained in the page, defaults to "text/html".	R

Example

```
<%@ taglib uri="i18n.tld" prefix="i18n" %>
.
.
.
<%
// Array that defines two languages preferences - English and
// Spanish in that order of preference.
String[] languages = new String[] { "en", "es" };

// Definition of a single language preference
String language = "en";
%>

<i18n:localize language="<%=language%>"
bundleName="i18nExampleResourceBundle"/>
<html>
<body>
<i18n:getMessage messageName="greeting"/>
</body>
</html>
```

<i18n:getMessage>

This tag (Table 12-15) is used in conjunction with the `<i18:localize>` tag to retrieve localized static text or messages from a JspMessageBundle.

Table 12-15 <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
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

Table 12-15 <i18n:getMessage> (Continued)

Tag Attribute	Required	Type	Description	R/C
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>
```

Properies file

Here are the entries in the properties 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 retreive 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.

<pz:contentQuery>

The `<pz:contentQuery>` tag (Table 12-16) performs a content attribute search for content in a content manager. If the useCache attribute is set to `true`, the results of a content management query will be cached. The tag only has a begin tag and does not have a body or end tag. It returns an array of `Content` objects as determined by the Advisor.

Personalization content tags required for JSP developers to access the `Content` object returned might include:

An object array iterator tag. This tag provides a way to iterate over the `Content` objects in the array. Use the `<es:forEachInArray>` tag to iterate over an array of `Objects`. (See "<es:forEachInArray>" on page 12-76 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 12-6 for more information.)

Table 12-16 <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
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 12-37.	R
id	Yes	String	The array variable name that contains the content objects found. If it finds no objects, it returns an empty array (not null) of Content objects.	C

Table 12-16 <pz:contentQuery> (Continued)

Tag Attribute	Required	Type	Description	R/C
useCache	No	String, Boolean	Determines whether Content is cached. This attribute can have one of two values: `False` (default value): ContentCache is not used. If `false` (not specified), the `cacheId`, `cacheScope` and `cacheTimeout` settings are ignored. `True`: ContentCache is used.	R
cacheId	No	String	The identifier name used to cache the Content. Internally, the cache is implemented as a Map; this will become the key. If not specified, the `id` attribute of the tag is used.	R
cacheTimeout	No	String, long	The time, in milliseconds, for which the cached Content is valid. If more than this amount of time has passed since the Content was cached, the cached Content will be cleared, retrieved, and placed back into the cache. Use -1 for no-timeout (always use the cached Content). Default = -1.	R
cacheScope	No	String	Specifies the lifecycle scope of the content cache. Similar to `<jsp:useBean>`. Possible values: ■ `application` ■ `session` (the default) ■ `page` ■ `request`	R

Example

```
<%@ taglib uri="es.tld" prefix="es" %>
<%@ taglib uri="cm.tld" prefix="cm" %>
<%@ taglib uri="pz.tld" prefix="pz" %>
<%@ page input="com.beasys.commerce.content.ContentHelper" %>
.
```

```
.
.
<pz:contentQuery id="docs"
contentHome="<%=ContentHelper.DEF_DOCUMENT_MANAGER_HOME%>"
   query="author = 'Hemingway'" />

<ul>


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

<pz:contentSelector>

The `<pz:contentSelector>` tag (Table 12-17) allows arbitrary personalized content to be recommended based on a content selector rule.

A content selector rule first determines whether a user fits the specified classification (for example, high income), and then selects content based on another qualifier (such as productType = sports cars.) It then evaluates a set of conditions that you define 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 segmentation. Because the Business Analysts are not exposed to the concept of rules, you will see content selector rules called simply “content selectors” and classifer 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 rule and 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 as determined by the Advisor.

Tags possibly required for JSP developers to access the `Content` objects returned might include:

- An object array iterator tag. This tag provides a way to iterate over the `Content` objects in the array. Use the `<es:forEachInArray>` tag to iterate over an array of `Objects`.
- Content access tags. Content tags access metadata attributes in the content and retrieve content and put it into the HTTP response output stream. (See the section "Content Management" on page 12-6 for more information.)

Table 12-17 <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
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 12-37.	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

Table 12-17 <pz:contentSelector> (Continued)

Tag Attribute	Req'd	Type	Description	R/C
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 it finds no objects, it returns an empty array (not null) of Content objects.	C
useCache	No	String, Boolean	Determines whether Content is cached. This attribute can have one of two values: `False` (default value): ContentCache is not used. If `false` (not specified), the cacheId, cacheScope and cacheTimeout settings are ignored. `True`: ContentCache is used.	R
cacheId	No	String	The identifier name used to cache the Content. Internally, the cache is implemented as a Map; this will become the key. If not specified, the `id` attribute of the tag is used.	R
cacheTimeout	No	String, long	The time, in milliseconds, for which the cached Content is valid. If more than this amount of time has passed since the Content was cached, the cached Content will be cleared, retrieved, and placed back into the cache. Use -1 for no-timeout (always use the cached Content). Default = -1.	R

Table 12-17 <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.beasys.commerce.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

```
<%@ taglib uri="es.tld" prefix="es" %>
<%@ taglib uri="cm.tld" prefix="cm" %>
<%@ taglib uri="pz.tld" prefix="pz" %>
<%@ page input="com.beasys.commerce.content.ContentHelper" %>
.
.
.
<pz:contentSelector rule="PremierCustomerSpotlight"
contentHome="<%=ContentHelper.DEF_DOCUMENT_MANAGER_HOME %>"
id="docs" />
<ul>
   <es:forEachInArray array="<%=docs%>" id="aDoc"
   type="com.beasys.commerce.axiom.content.Content">
      <li>The document title is: <cm:printproperty id="aDoc"
      name="Title" encode="html" />
   </es:forEachInArray>
</ul>
```

Note: The `sortBy` attribute, when used in conjunction with the `max` attribute, works differently for explicit (system-defined) and implicit (user-defined) attributes. If you sort on explicit attributes (`identifier, mimeType, size, version, author, creationDate, modifiedBy, modifiedDate, lockedBy, description,` or `comments`) the sort is done on the database; therefore if you combine `max="10"` and `sortBy`, the system will perform the sort and then get the first 10 items. If you sort on implicit attributes, the sort is done *after* the max have been selected.

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 12-18) allows a piece of content to be conditionally included as a result of a classifier rule being executed by a rules advislet. 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 12-18 <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.	C

Example

```
<%@ taglib uri="pz.tld" prefix="pz" %>
.
.
.
<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())
   {
   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 12-19) 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 “Displaying Ads” in *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 12-19 <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

Property Sets

The Property Set tags allow access to the list of available properties and property sets. Manipulation of property sets can be done either programatically or through the administration tools.

Use the following code to import the utility tag library:
`<%@ taglib uri="ps.tld" prefix="ps" %>`

Note: In the following tables, 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 12-20) is used to get a list of property names given a property set.

Table 12-20 <ps:getPropertyNames>

Tag Attribute	Required	Type	Description	R/C
propertySet	Yes	String	The name of the property set to add the new search.	R
schemaGroupName	Yes	String	Type of property set to search (as defined in com.beasys.commerce.foundation.property. SchemaManagerConstants).	R
id	Yes	String	The `id` of the variable to hold the list of property names, as a String array.	C

Table 12-20 <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* `schemaGroupName` *could not be found*: `PropertySetTagConstants.INVALID_PROPERTY_SET`	C

Example

```
<%@ taglib uri="ps.tld" prefix="ps" %>
.
.
.
<ps:getPropertyNames propertySet="<%myPropertySet%>"
     schemaGroupName="<%SchemaManagerConstants.USER_TYPE%>"
     id="propertyNames" result="myResult"/>
```

<ps:getPropertySetNames>

The `<ps:getPropertySetNames>` tag (Table 12-21) is used to get a list of property sets given a property set type.

Table 12-21 <ps:getPropertySetNames>

Tag Attribute	Required	Type	Description	R/C
schemaGroupName	Yes	String	The type of the property set to search (as defined in com.beasys.commerce.foundation.property. SchemaManagerConstants).	R
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* `schemaGroupName` *could not be found*: `PropertySetTagConstants.INVALID_PROPERTY_SET`	C

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.beasys.commerce.user.jsp.tags.UserManagerTagConstants"
%>
```

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 12-22) retrieves the profile corresponding to the provided profile key and profile type. The tag has no enclosed body. The retrieved profile can be treated simply as a `com.beasys.commerce.foundation.ConfigurableEntity`, or can be cast to the particular implementation of `ConfigurableEntity` that it is. Along with the profile key and profile, an explicit successor key and successor type can be specified, as specified by the `profileType` attribute. This successor will then be used, along with the retrieved profile, in subsequent invocations of the `<um:getProperty>` tag to ensure property inheritance from the successor. If no successor is retrieved, standard `ConfigurableEntity` successor search patterns will apply to retrieved properties.

Table 12-22 <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

Table 12-22 <um:getProfile> (Continued)

Tag Attribute	Required	Type	Description	R/C
successorType	No	String	The profile successor type to be retrieved. If specified, this profile type *must* correspond to a profile type registered via the Unified Profile Type tool in the User Management suite of administration tools, and its bean must conform to the rules of Unified User Profile creation. By default, the tag retrieves a profile of type `com.beasys.commerce.axiom.contact.Group`, unless otherwise specified. Example: "AcmeGroup"	C
scope	No	String	The HTTP scope of the retreived profile. Pass `"request"` or `"session"` as the values. Defaults to `session`.	C
groupOnly	No	String	Specifies to retrieve a `com.beasys.commerce.axiom.contact.Group`, rather than `com.beasys.commerce.axiom.contact.User`, for the default profile type. No successor will be retrieved when this value is `true`. Defaults to `false`.	C
profileId	No	String	A variable name from which the retrieved profile is available for the duration of the JSP's page scope.	C
profileType	No	String	Allows theJSP developer to specify what type of User profile object to return. If the given profileKey refers to a baseUser object, this attribute should be left blank. Otherwise, if it returns to an extended User object defined by a Unified Profile Type, the name of the Unified Profile Type should be supplied in this field.	C

Table 12-22 <um:getProfile> (Continued)

Tag Attribute	Required	Type	Description	R/C
successorId	No	String	A variable name from which the retrieved successor is available for the duration of the JSP's page scope.	C
result	No	String	A variable name from which the result of the operation is available. Possible values: *Success:* UserManagerTagConstants.GET_PROFILE_OK *Error encountered:* UserManagerTagConstants.GET_PROFILE_FAILED UserManagerTagConstants.NO_SUCH_PROFILE UserManagerTagConstants.NO_SUCH_SUCCESSOR	C

Example 1

This example shows a profile of type `AcmeUser` being retrieved with no successor specified, and an explicitly-supplied `session` scope.

```
<%@ taglib uri="um.tld" prefix="um" %>
.
.
.
<um:getProfile profileKey="bob" profileType="AcmeUser"
profileId="myProfile" scope="session"/>
```

Example 2

This example shows a default profile type (`com.beasys.commerce.axiom.contact.User`) being retrieved with a default successor type (`com.beasys.commerce.axiom.contact.Group`), and an explicitly-supplied `request` scope.

```
<%@ taglib uri="um.tld" prefix="um" %>
.
.
.
<um:getProfile profileKey="bob" successorKey="engineering"
scope="request"/>
```

Example 3

This example shows a profile type of `AcmeUser` being retrieved with a successor type of `AcmeGroup`, and an implicitly-supplied `session` scope.

```
<%@ taglib uri="um.tld" prefix="um" %>
.
.
.
<um:getProfile profileKey="bob" profileType="AcmeUser"
    successorKey="engineering" successorType="AcmeGroup"
    profileId="myProfile"/>
```

<um:getProperty>

The `<um:getProperty>` tag (Table 12-23) 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 7, “Creating and Managing Users,” in this guide.

Table 12-23 <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: "Demo Portal" **Note:** If no property set is provided, the property is retrieved from the profile's default (unscoped) properties.	R
propertyName	Yes	String	The name of the property to be retrieved. Example: "background_color"	R
id	No	String	If the `id` attribute is supplied, the value of the retrieved property will be available in the variable name to which `id` is assigned. Otherwise, the value of the property is inlined.	C

Example 1

```
<%@ taglib uri="um.tld" prefix="um" %>
.
.
.
<um:getProperty id="myTitlebarBGColor" propertySet="exampleportal"
propertyName="titlebar_bgcolor"/>
My titlebar bg color is <%=myTitlebarBGColor%>.
```

Example 2

```
My titlebar bg color is <um:getProperty propertySet="exampleportal"
propertyName="titlebar_bgcolor"/>.
```

<um:getPropertyAsString>

The `<um:getPropertyAsString>` tag (Table 12-24) 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 12-24 <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: "Demo Portal" **Note:** If no property set is provided, the property is retrieved from the profile's default (unscoped) properties.	R
propertyName	Yes	String	The name of the property to be retrieved. Example: "background_color"	R
id	No	String	If the `id` attribute is supplied, the value of the retrieved property will be available in the variable name to which `id` is assigned. Otherwise, the value of the property is inlined.	C

Example

```
<%@ taglib uri="um.tld" prefix="um" %>
.
.
.
<um:getPropertyAsString id=”myFaveColors”
propertySet=”exampleportal” propertyName=”fave_colors”/>
My favorite colors are <%=myFaveColors%>.
```

<um:removeProperty>

The `<um:removeProperty>` tag (Table 12-25) 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 12-25 <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: "Demo Portal" **Note:** The property is removed from the profile's default (unscoped) properties if no property set is provided.	R
propertyName	Yes	String	The name of the property to be removed. Example: "background_color"	R

Example

```
<%@ taglib uri="um.tld" prefix="um" %>
.
.
.
<um:removeProperty propertySet="<%=thePropertySet%>"
propertyName="<%=thePropertyName%>"/>
```

<um:setProperty>

The `<um:setProperty>` tag (Table 12-26) updates a property value for either the session's current profile, or for the Anonymous User Profile. This tag has no enclosed body.

Table 12-26 <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: "Demo Portal" **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: "background_color"	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:* UserManagerTagConstants.SET_PROPERTY_OK *Error encountered:* UserManagerTagConstants.SET_PROPERTY_FAILED	C

Example

```
<%@ taglib uri="um.tld" prefix="um" %>
.
.
.
<% String myName = request.getParameter("name"); %>
<um:setProperty propertySet="exampleportal" propertyName="name"
value="<%=myName%>"/>
```

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.beasys.commerce.user.jsp.tags.UserManagerTagConstants"
%>
```

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 12-27) 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 class `com.beasys.commerce.axiom.contact.security.RDBMSRealm` is defined as the active security realm. This can be verified through the WebLogic Server Administration Console.

Table 12-27 <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*: UserManagerTagConstants.ADD_GROUP_OK *Error encountered*: UserManagerTagConstants.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 12-28) 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 `customRealm element in config.xml` is `com.beasys.commerce.axiom.contact.security.RDBMSRealm`.

Table 12-28 <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:* UserManagerTagConstants.ADD_USER_OK *Error encountered:* UserManagerTagConstants.ADD_USER_FAILED	C

Example

```
<%@ taglib uri="um.tld" prefix="um" %>
.
.
.
<um:addUserToGroup userName="<%=userName%>"
groupName="<%=groupName%>" result="result"/>
```

<um:changeGroupName>

The `<um:changeGroupName>` tag (Table 12-29) changes the name of the group corresponding to the specified `oldGroupName` to the specified `newGroupName`. This tag has no enclosed body.

Note: This tag should only be invoked when the `customRealm element in config.xml` is `com.beasys.commerce.axiom.contact.security.RDBMSRealm`.

Table 12-29 <um:changeGroupName>

Tag Attribute	Required	Type	Description	R/C
oldGroupName	Yes	String	The old group name. Example: “<%=oldGroupName%>”	R
newGroupName	Yes	String	The new group name. Example: “<%=newGroupName%>”	R
result	Yes	String	The name of an Integer variable to which the result of the change group name operation is assigned. Possible values: *Success:* UserManagerTagConstants.GROUP_CHANGE_OK *Error encountered:* UserManagerTagConstants.GROUP_CHANGE_FAILED	C

Example

```
<%@ taglib uri="um.tld" prefix="um" %>
.
.
.
<um:changeGroupname oldGroupName=“<%=oldGroupName%>”
newGroupName=”<%=changeGroupName%>” result=”result”/>
```

<um:createGroup>

The `<um:createGroup>` tag (Table 12-30) creates a new `com.beasys.commerce.axiom.contact.Group` object. This tag has no enclosed body.

Note: This tag should only be invoked when the class `com.beasys.commerce.axiom.contact.security.RDBMSRealm` is defined as the active security realm. This can be verified through the WebLogic Administration Console.

Table 12-30 <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:* UserManagerTagConstants.CREATE_GROUP_OK *Error encountered:* UserManagerTagConstants.CREATE_GROUP_FAILED *A group with the specified group name already exists*: UserManagerTagConstants.GROUP_EXISTS	C

Example

```
<%@ taglib uri="um.tld" prefix="um" %>
.
.
.
<um:creategroup groupName=”<%=groupName%>” result=”result”/>
```

<um:createUser>

The `<um:createUser>` tag (Table 12-31) creates a new `com.beasys.commerce.axiom.contact.User` object. This tag has no enclosed body. Although classified as a Group-User management tag, this tag can be used in conjunction with run-time activities, in that it will persist any properties associated with a current Anonymous User Profile if specified.

Note: This tag should only be invoked when the class `com.beasys.commerce.axiom.contact.security.RDBMSRealm` is defined as the active security realm. This can be verified through the WebLogic Administration Console.

Table 12-31 <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

Table 12-31 <um:createUser> (Continued)

Tag Attribute	Required	Type	Description	R/C
saveAnonymous	No	String	Whether to persist current anonymous user profile attributes in the newly-created user's profile. Defaults to `false`. Example: "`false`"	R
id	No	String	A variable name to which the created User object is available for the duration of the page's scope.	C
result	Yes	String	The name of an Integer variable to which the result of the create user operation is assigned. Possible values: *Success:* UserManagerTagConstants.CREATE_USER_OK *Error encountered*: UserManagerTagConstants.CREATE_USER_FAILED *A user with the specified username already exists*: UserManagerTagConstants.USER_EXISTS	C

Example

```
<%@ taglib uri="um.tld" prefix="um" %>
.
.
.
<um:createUser userName="<%=username%>" password="<%=password"%>
result="result"/>
```

<um:getChildGroupNames>

The `<um:getChildGroupNames>` tag (Table 12-32) returns the names of any groups that are children of the given group.

Table 12-32 <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 identfier which will be assigned the String array of child group names.	C

<um:getChildGroups>

The `<um:getChildGroups>` tag (Table 12-33) retrieves an array of `com.beasys.commerce.axiom.contact.Group` objects that are children of the Group corresponding to the provided `groupName`. The information is taken from the personalization database tables, and reflects the group hierarchy information as set up from the Group Administration and Realm Configuration Administration Tools. This tag has no enclosed body.

Table 12-33 <um:getChildGroups>

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

Example

```
<%@ taglib uri="um.tld" prefix="um" %>
.
.
.
<um:getchildgroups groupName=”<%=groupName%>” id=”childGroups”/>
```

<um:getGroupNamesForUser>

The `<um:getGroupNamesForUser>` tag (Table 12-34) 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 12-34 <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 12-35) retrieves the name of the parent of the `com.beasys.commerce.axiom.contact.Group` object associated with the provided `groupName`. The information is taken from the personalization database tables, and reflects the group hierarchy information as set up from the Group Administration and Realm Configuration Administration Tools. This tag has no enclosed body.

Table 12-35 <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 12-36) retrieves an array of com.beasys.commerce.axiom.contact.Group objects, each of which has no parent group. The information is taken from the personalization database tables, and reflects the group hierarchy information as set up from the Group Administration and Realm Configuration Administration Tools. This tag has no enclosed body.

Table 12-36 <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 12-38) retrieves a String array that contains the usernames matching the provided search expression. The search expression supports only the asterisk (*) wildcard character, and is case insensitive. As many asterisks as desired may be used in the search expression. This tag has no enclosed body.

Note: This tag should only be invoked when the class com.beasys.commerce.axiom.contact.security.RDBMSRealm is defined as the active security realm. This can be verified through the WebLogic Administration Console.

Table 12-37 <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:* UserManagerTagConstants.CREATE_GROUP_OK *Error encountered:* UserManagerTagConstants.CREATE_GROUP_FAILED *A group with the specified group name already exists*: UserManagerTagConstants.GROUP_EXISTS	C

Table 12-38 <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 12-38 <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:* UserManagerTagConstants.USER_SEARC H_OK *General error*: UserManagerTagConstants.USER_SEARC H_FAILED	C

Note: The `USER_SEARCH_FAILED` value is returned only when a general error occurs while searching for the user, such as a database connection failure. If no user matches the search criteria, the result will not be equal to `UserManagerTagConstants.USER_SEARCH_FAILED`, but the length returned by the array in `id` will be zero.

Example

```
<%@ 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 12-39) retrieves a `String` array that contains the usernames matching the provided search expression and correspond to members of the provided group. The search expression supports only the asterisk (*) wildcard character, and is case insensitive. As many asterisks as desired may be used in the search expression. This tag has no enclosed body.

Note: This tag should only be invoked when the class `com.beasys.commerce.axiom.contact.security.RDBMSRealm` is defined as the active security realm. This can be verified through the WLS administration console.

Table 12-39 <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 12-39 <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:* UserManagerTagConstants.USER_SEARCH_OK *General error*: UserManagerTagConstants.USER_SEARCH_FAILED	C

Note: The `USER_SEARCH_FAILED` value is returned only when a general error occurs while searching for the user, such as a database connection failure. If no user matches the search criteria, the result will not be equal to `UserManagerTagConstants.USER_SEARCH_FAILED`, but the length returned by the array in `id` will be zero.

Example

```
<%@ 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 12-40) removes the `com.beasys.commerce.axiom.contact.Group` object corresponding to the provided `groupName`. This tag has no enclosed body.

Note: This tag should only be invoked when the class `com.beasys.commerce.axiom.contact.security.RDBMSRealm` is defined as the active security realm. This can be verified through the WebLogic Server Administration Console.

Table 12-40 <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:* UserManagerTagConstants.REMOVE_GROUP_OK *Error encountered:* UserManagerTagConstants.REMOVE_GROUP_FAILED	C

Example

```
<%@ taglib uri="um.tld" prefix="um" %>
.
.
.
<um:removeGroup groupName=”<%=groupName%>” result=”result”/>
```

<um:removeGroupFromGroup>

The `<um:removeGroupFromGroup>` tag (Table 12-41) removes a child group from a parent group.

Table 12-41 <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:* UserManagerTagConstants.REMOVE_GROUP_OK *Failure:* UserManagerTagConstants.REMOVE_GROUP_FAILED	C

<um:removeUser>

The `<um:removeUser>` tag (Table 12-42) removes the `com.beasys.commerce.axiom.contact.User` object corresponding to the provided `username`. It can remove any type of extended user that has its profileType set in the database. This tag has no enclosed body.

Note: This tag should only be invoked when the class `com.beasys.commerce.axiom.contact.security.RDBMSRealm` is defined as the active security realm. This can be verified through the WebLogic Server Administration Console.

Table 12-42 <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:* UserManagerTagConstants.REMOVE_USER_OK *Error encountered*: UserManagerTagConstants.REMOVE_USER_FAILED	C

Example

```
<%@ taglib uri="um.tld" prefix="um" %>
.
.
.
<um:removeUser userName=”<%=username%>” result=”result”/>
```

<um:removeUserFromGroup>

The `<um:removeUserFromGroup>` tag (Table 12-43) removes a user from a group.

Note: This tag should only be invoked when the class `com.beasys.commerce.axiom.contact.security.RDBMSRealm` is defined as the active security realm. This can be verified through the WebLogic Server Administration Console.

Table 12-43 <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:* UserManagerTagConstants.REMOVE_US ER_OK *Failure:* UserManagerTagConstants.REMOVE_US ER_FAILED	C

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.beasys.commerce.user.jsp.tags.UserManagerTagConstants"
%>
```

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 12-44) 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 12-44 <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:* UserManagerTagConstants.LOGIN_OK *General error when performing authentication*: UserManagerTagConstants.LOGIN_ERRO R *Authentication failed because of invalid username/password combination*: UserManagerTagConstants.LOGIN_FAILE D	C

<um:logout>

The `<um:logout>` tag (Table 12-45) ends the current user's WebLogic Server session. This is independent of the FlowManager's user session tracking, and should be used in combination with the `<um:login>` tag.

Table 12-45 <um:logout>

Tag Attribute	Required	Type	Description	R/C
No attributes				

<um:setPassword>

The `<um:setPassword>` tag (Table 12-46) updates the password for the user corresponding to the provided username.

Note: This tag should only be invoked when the class `com.beasys.commerce.axiom.contact.security.RDBMSRealm` is defined as the active security realm. This can be verified through the WebLogic Server Administration Console.

Table 12-46 <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

Table 12-46 <um:setPassword>

Tag Attribute	Required	Type	Description	R/C
result	Yes	String	The name of an Integer variable to which the result of the set password operation is assigned. Possible values: *Success*: UserManagerTagConstants.SET_PASSWORD_OK *Failure*: UserManagerTagConstants.SET_PASSWORD_FAILED	C

Personalization Utilities

The `<es:jsptaglib>` tag contains generic tags you can use to create JSP pages.

Use the following code to import the utility tag library:
`<%@ taglib uri="es.tld" prefix="es" %>`

Note: In the following tables, 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:counter>

The `<es:counter>` tag (Table 12-47) is used to create a `for` loop.

Table 12-47 <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 12-48) is used to get a date- and time-formatted String based on the user's time zone preference.

Table 12-48 <es:date>

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

Example

```
<%@ taglib uri="es.tld" prefix="es" %>
.
.
.
<es:date formatStr="MMMM dd yyyy" timeZoneId="MST" />
```

<es:forEachInArray>

The `<es:forEachInArray>` tag (Table 12-49) is used to iterate over an array.

Table 12-49 <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 12-50) 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 12-50 <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:monitorSession>

The `<es:monitorSession>` tag (Table 12-51) can be added to the beginning of any JSP page to disallow access to the page if the session is not valid or if the user is not logged in.

Table 12-51 <es:monitorSession>

Tag Attribute	Required	Type	Description	R/C
goToPage	No	String	The error page that you want displayed if the page is not accessible. The default value is `portalerror.jsp`.	R
loginRequired	No	String	Indicates whether the user is required to be logged in to access the JSP page including the tag. The default value is `false`.	R

Example

```
<%@ taglib uri="es.tld" prefix="es" %>
.
.
.
<es:monitorSession loginRequired="true" />
```

<es:notNull>

The `<es:notNull>` tag (Table 12-52) 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 12-52 <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:simpleReport>

The `<es:simpleReport>` tag (Table 12-53) is used to create two-dimensional array out of a simple query.

Table 12-53 <es:simpleReport>

Tag Attribute	Required	Type	Description	R/C
id	Yes	String	The variable that holds the resultant two-dimensional array converted from the java.sql.ResultSet specified by the resultSet tag attribute.	R
resultSet	Yes	java.sql.ResultSet	The result set that holds the java.sql.ResultSet.	R

Example

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

<es:transposeArray>

The `<es:transposeArray>` tag (Table 12-54) is used to transpose a standard [row][column] array to a [column][row] array.

Table 12-54 <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 12-55) is used to pull content from a URL. It is best used for grabbing text-heavy pages.

Table 12-55 <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 the *Javadoc* for further descriptions of the `wl` tags.

<wl:process>

The `<wl:process>` tag (Table 12-56) 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 12-56 <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:

	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 12-57) is used to iterate over a variety of Java objects, as specified in the set attribute.

Table 12-57 <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

<wl:cache>

The `<wl:cache>` tag specifies that its contents do not necessarily need to be updated every time it is displayed.

Table 12-58 <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
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

Index