



BEA WebLogic Portal®

Interaction Management Guide

Version 9.2
Revised: June 2007

Contents

1. Introduction

- Introducing Personalization 1-1
 - Using Interaction Management Tools. 1-2
 - Understanding the Features 1-3
- Interaction Management in the Portal Life Cycle 1-4
 - Architecture 1-5
 - Development. 1-6
 - Staging 1-7
 - Production. 1-7
- Getting Started 1-8

2. Planning an Interaction Strategy

- Choosing the Type of Interaction Management to Develop 2-1
 - Understanding Conditions 2-4
- Checklist for Planning Your User Interaction Strategy 2-5
- Checklist for Planning Your Campaign Strategy. 2-9
 - Understanding Campaigns 2-10
- Planning Your Behavior Tracking Strategy 2-11
 - Understanding When to Use a Predefined Event 2-11
 - Understanding When to Create a Custom Event 2-12
 - Understanding When to Create a Custom Event Listener 2-12
- Updating Interaction Management Features 2-13

Upgrading Interaction Features from Portal 8.1	2-13
--	------

Part I. Architecture

3. Setting up Content

Adding Content	3-1
Determining Content Priority	3-2

Part II. Development

4. Creating a Property Set

Setting up a Property Set	4-2
Creating a User Profile Property Set	4-3
Creating a User Segment Property Set	4-4
Creating a Session Property Set	4-8
Creating a Request Property Set	4-8
Creating a Community or Remote Portlet Property Set	4-9
Creating an Event Property Set	4-10
Creating a Catalog Property Set	4-10
Adding Properties or Conditions to a Property Set	4-11
Modifying Properties and Conditions	4-13
Editing Properties	4-14
Editing Property Values	4-14
Retrieving Properties from External Data Stores	4-14
Deleting a Property or a Property Set	4-15

5. Creating a User Segment

Creating a User Segment	5-1
Setting Dates and Times	5-4
Modifying a User Segment	5-4

6. Creating a Content Selector

Setting Up Content to Display	6-2
Displaying Content for Other MIME Types	6-2
Creating a Content Selector	6-4
Creating the Content Selector File	6-6
Using a JSP Tag to Display a Content Selector File	6-23
Using the <pz:div> Tag Instead of a Content Selector	6-27
Deleting a Content Selector Query	6-27
Deleting a Content Selector	6-28
Modifying a Content Selector	6-28

7. Creating a Placeholder

Selecting Content for a Placeholder	7-2
Displaying Additional MIME Types in a Placeholder	7-3
Adding Content to a Placeholder	7-5
Creating a Placeholder	7-5
Creating a Placeholder File	7-6
Building a Content Query	7-12
Determining Which Query and Content to Display	7-13
Adding a Placeholder to a JSP	7-14
Modifying a Placeholder	7-15
Using the <ad:adTarget> Tag Instead of a Placeholder	7-16

8. Building a Campaign

Performing the Prerequisite Tasks	8-3
Building a Campaign	8-3
Planning Your Campaign Logic	8-4
Creating a Campaign File	8-5

Adding a Scenario to a Campaign	8-11
Adding an Action to a Scenario	8-13
Adding an E-Mail Action to a Campaign.	8-16
Adding a Discount Action to a Campaign	8-17
Setting Up Automatic E-Mail Messages	8-18
Testing a Campaign	8-26
Triggering a Campaign	8-31
Troubleshooting Campaign Actions.	8-31
Turning Off a Campaign	8-32
Resetting a Campaign	8-32
Resetting a Campaign in the Development Environment	8-33
Resetting a Campaign in the Production Environment	8-34

9. Setting Up Events and Behavior Tracking

Choosing How to Handle Events.	9-2
Completing Your Behavior Tracking Strategy	9-7
Planning the Deployment of Custom Events, Listeners, and Property Sets.	9-7
Using Predefined Events	9-7
Using the SessionLoginEvent.	9-8
Using the SessionBeginEvent and SessionEndEvent.	9-8
Using the UserRegistrationEvent	9-9
Using the AddToCartEvent	9-9
Using the RemoveFromCartEvent	9-9
Using the PurchaseCartEvent.	9-10
Using the Rule Events	9-10
Using the DisplayCampaignEvent	9-10
Using the CampaignUserActivityEvent	9-11
Using the ClickCampaignEvent	9-12

Using the ClickProductEvent	9-12
Using the ClickContentEvent	9-12
Generating Events for Content Clicks	9-13
Using the ClickThroughEventFilter	9-13
Generating Content Events	9-14
Using the ContentConfigEvent	9-15
Using the ContentCreateEvent	9-15
Using the ContentDeleteEvent	9-15
Using the ContentUpdateEvent	9-16
Providing Event Attribute Values	9-16
Enabling Behavior Tracking	9-19
Configuring Behavior Tracking	9-20
Adjusting Behavior Tracking for Optimal Performance	9-22
Storing Behavior Tracking Data in Other Ways	9-23
Creating a Separate Database for Behavior Tracking Events	9-23
Creating Custom Events	9-23
Creating the Event Class	9-23
Creating an XML Schema for Behavior Tracking	9-36
Creating Custom Event Listeners	9-38
Dispatching Events	9-41
Using Events in Campaigns	9-43
Registering Events for Campaigns	9-45
Debugging the Event Service	9-45
Tracking Content Changes	9-46
Disabling Behavior Tracking	9-49
Unregistering the Behavior Tracking Listener	9-49
Removing an Individual Event	9-49

10. Creating Advanced Personalization with Rules

Using Rules in Portal Applications	10-1
Choosing Personalization Components	10-2
Understanding the Rules Service	10-4
Creating a Rule	10-7
Creating a Rule Set	10-8
Deploying a Rule Set	10-13
Adding Objects to Working Memory	10-14
Invoking the Rules Service to Evaluate Objects	10-16
Filtering the Results	10-22
Using the Results in Your Application	10-24
Rules Control Reference	10-25

Part III. Staging

11. Modifying Property Set Values

Editing a Property Value	11-2
Deleting a Property Value	11-3

12. Modifying a User Segment

Modifying a User Segment	12-5
Modifying a User Segment's Properties	12-6
Copying a User Segment	12-7
Removing a User Segment	12-7

13. Modifying a Content Selector

Modifying a Content Selector	13-1
Deleting a Content Selector and Query	13-3

14. Modifying a Placeholder

Changing Content for a Placeholder	14-2
Modifying a Placeholder	14-2
Deleting a Query or a Placeholder	14-3
Managing Placeholders for Optimal Performance	14-4

Part IV. Production

15. Managing a Campaign

Modifying a Campaign	15-1
Changing a Campaign's Description or Sponsor	15-2
Changing a Campaign Start or Stop Date	15-2
Activating and Deactivating a Campaign	15-3
Turning Off a Campaign	15-4
Resetting Campaign Settings	15-5
Duplicating a Campaign	15-5
Modifying a Rule	15-6
Modifying a Content Action	15-6
Modifying an E-Mail Action	15-8
Modifying a Discount Action	15-9
Previewing a Modified Campaign Action	15-9
Managing a Campaign for Optimal Performance	15-10

Introduction

This guide describes how to set up personalized content to enhance how users interact with your portal application.

Personalized content can include content or images targeted to specific users or audiences. For example, you can create dynamic images or links that are personalized for each user. You could dynamically guide users through a process (such as signing up for employee benefits or shopping online) that takes them to different places based on their personal preferences or characteristics.

You can even record the path users take through your portal to gauge the effectiveness of the portal, its design, or your process flows. This Behavior Tracking provides information that can validate your strategies or help you make improvements.

This chapter includes the following sections:

- [Introducing Personalization](#)
- [Interaction Management in the Portal Life Cycle](#)
- [Getting Started](#)

Introducing Personalization

This section contains the following topics:

- [Using Interaction Management Tools](#)
- [Understanding the Features](#)

Developers use Workshop for WebLogic to set up Personalization features, such as Campaigns, Content Selectors, Placeholders, and User Segments. Developers can also create rules for Personalization and events for Behavior Tracking. Portal administrators use the WebLogic Portal Administration Console to modify Campaigns, Content Selectors, Placeholders, and User Segments to fit the needs of the portal's audience.

Developing Interaction Management features often involves setting up related pieces. For example, if you want to target users with personalized content in a Campaign, you have to add content to BEA's Virtual Content Repository, create Placeholders that display the content, set up properties (such as User Profile or session properties) that are used to define the conditions under which users will be targeted with Campaign content, and finally, create the Campaign.

This chapter describes the tools you can use to create Interaction Management features and the logic that drives the tools. Each tool uses a rules engine to match users with appropriate content.

Using Interaction Management Tools

You can use the following tools to create and maintain Interaction Management features in the portal life cycle:

1. **BEA Workshop for WebLogic Platform** – Developers create the following items:
 - User Segments, Property Sets, Content Selectors, Placeholders, Campaigns and Behavior Tracking – Create these Personalization features and then use Java Server Page (JSP) tags or controls to enable the feature in a Page Flow or JSP.
 - JSP Tags – Use JSP tags to display personalized content to users. For example, Campaigns show web content using a JSP tag called a Placeholder: `<ph:placeholder name="myPlaceholder1"/>`. You can add JSP Placeholder tags (identified by the **name** attribute) anywhere in your portal's JSPs. For more information on Java classes, see the [Javadoc](#)
 - Java Controls – Use Java controls (predefined Java functionality) in your Page Flows and Web Services to display personalized content. For example, you can use the Rules Executor control to help determine a user's path through a Page Flow based on specific conditions, such as User Profile values or session properties. For more information on controls, see the [Controls Javadoc](#).
2. **Java API** – Developers can also utilize a full API to programmatically develop Interaction Management functionality.
3. **WebLogic Portal Administration Console** – Portal Administrators can modify values and some properties for User Segments, Property Sets, Content Selectors, Placeholders, and

Campaigns. They can also change the target audience that will see Personalization features, or modify Campaign dates. Administrators can use the Administration Console to test the new features you have developed and adjust the target audience. If you need to modify or fine tune any of them, you can use Workshop for WebLogic to return to the Development phase and make changes. You must redeploy your portal application to see the changes in the Staging environment.

Understanding the Features

Workshop for WebLogic provides the following features to help you deliver personalized content:

- **Property Sets** – A Property Set Editor lets developers define User Profile properties, request properties, session properties, and custom events to create conditions that uniquely identify users. For example, you can create a **NewHire** property to target new users with benefit enrollment information. An **Employee** property set could have an attribute (or property) called **NewHire**, as well as **HireDate**.
- **User Segments** – User Segments help you dynamically categorize users based on conditions or criteria that define the target visitor. For example, you can define conditions that dynamically identify gender, occupation, movie fans, or pet lovers.
- **Content Selectors** – A Content Selector targets users with personalized web content from BEA's Virtual Content Repository. For example, you can display a list of recommended movies to users identified as *movie fans*.
- **Campaigns** – Campaigns let you target specific users with a single piece of personalized content, automatically send them a predefined e-mail, or provide a discount in a commerce application. Campaigns run for a limited time and drive online behavior and Personalization to achieve a specific business goal.
- **Placeholders** – Campaigns use Placeholders to display personalized content on a portal page. A Placeholder is a predefined location in a JSP that displays a single piece of web content retrieved from the BEA Virtual Content Repository. A Placeholder uses queries to retrieve and display content, and can rotate the content to display something different on each browser refresh. Campaigns are targeted to Placeholders.
- **Events and Behavior Tracking** – Events let you create actions that happen in a user's interaction with your portal, or respond to actions that occur. Events, such as a user clicking a button or registering in your portal, can trigger a Campaign. You could generate an event when a user logs in or logs out. You can also use a Campaign to respond in real time to an event. For example, if a user clicks an image, an event is generated. You would

know which image is clicked and you can display other information in another portlet. For example, you clicked a camera image and camera accessories display in another portlet.

See [Chapter 2, “Planning an Interaction Strategy”](#) for more details and examples of each type of interaction.

The following terms are used in this guide:

- **Authentication** – Registers (verifies) the user and logs the user into the portal.
- **Authorization** – Determines what the user can access.

Interaction Management in the Portal Life Cycle

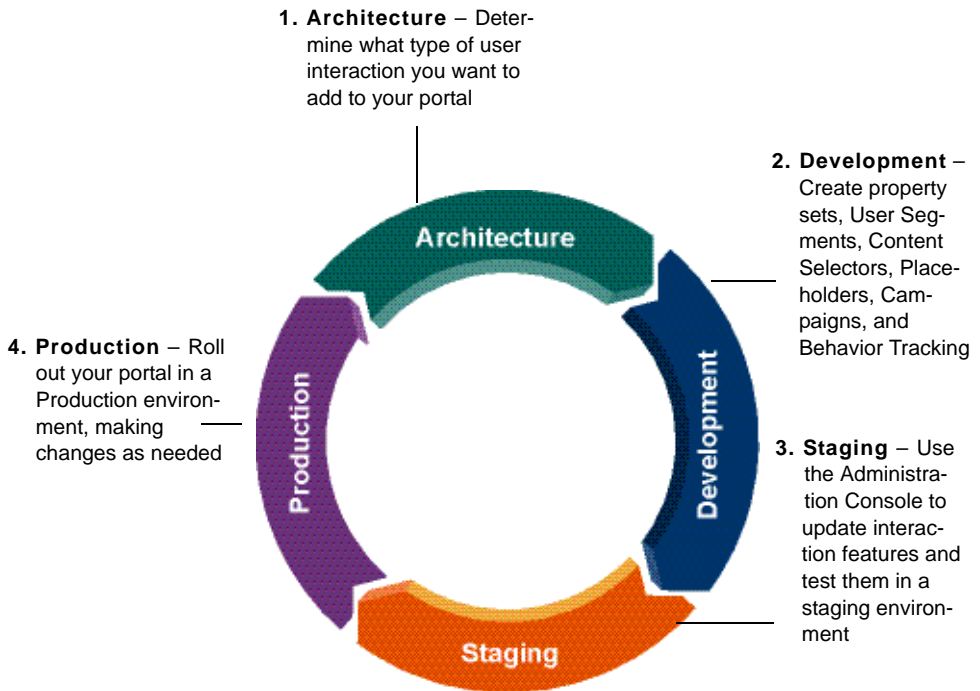
This section contains the following topics:

- [Architecture](#)
- [Development](#)
- [Staging](#)
- [Production](#)

The tasks in this guide are organized according to the portal life cycle. The portal life cycle contains four phases: Architecture, Development, Staging, and Production. Adding Personalization and user interaction to your portal is an important part of the portal life cycle. For more information about the portal life cycle, see the [WebLogic Portal Overview](#).

[Figure 1-1](#) shows which Interaction Management tasks occur in each phase.

Figure 1-1 Interaction Management Tasks in the Four Phases of the Portal Life Cycle



Architecture

In the [Architecture](#) phase, you plan the type of interaction that your portal users will experience. Architects decide who to target, what type of personalized content users will see, how often the content changes, and how to update the content. For more information about the portal life cycle, see the [WebLogic Portal Overview](#).

The following chapters provide guidance on Architecture tasks:

- [Chapter 2, “Planning an Interaction Strategy”](#) describes when to use different types of Personalization and explains the relationships between the Interaction Management features.
- [Chapter 3, “Setting up Content”](#) describes the properties you can add to content items in BEA’s Virtual Content Repository that support Interaction Management functionality.

Development

In the [Development](#) phase, developers use Workshop for WebLogic to create user property sets and properties, User Segments, Placeholders, Content Selectors, Campaigns, and Behavior Tracking to add Personalization without custom coding. Developers can also work directly with the Java API to add Personalization.

Personalization features allow you to target users with personalized web content, display a single piece of web content retrieved from the BEA Virtual Content Repository, automatically send a user a predefined e-mail, or provide a discount in a commerce application. Developers can also categorize users based on specific characteristics or criteria and then target those User Segments.

Tools: Workshop for WebLogic and the Java API.

The following chapters provide instructions on Development tasks:

- [Chapter 4, “Creating a Property Set”](#) provides instructions on how to set up conditions that drive Interaction Management and the choices you need to make.
- [Chapter 5, “Creating a User Segment”](#) describes how to dynamically group users based on conditions you define.
- [Chapter 6, “Creating a Content Selector”](#) describes Content Selectors and how to use them to display multiple personalized content items from the virtual content repository.
- [Chapter 7, “Creating a Placeholder”](#) describes Placeholders and how to use them to display single content items from the Virtual Content Repository. This chapter describes how to use Placeholders by themselves to display non-personalized content and how to use them with Campaigns to display personalized content.
- [Chapter 8, “Building a Campaign”](#) provides the setup steps and things to consider when building a Campaign.
- [Chapter 9, “Setting Up Events and Behavior Tracking”](#) describes the event framework and when to use predefined Behavior Tracking events. The chapter also discusses when and how to create regular events and custom events for Behavior Tracking, and how to use events in a Campaign.
- [Chapter 10, “Creating Advanced Personalization with Rules”](#) explains how to customize rules to create advanced Personalization. This type of Personalization can help you do things like control each user’s path through a Page Flow or use runtime information to determine conditional logic.

Staging

In the [Staging](#) phase, portal administrators use a browser to test the Content Selectors, Placeholders, Campaigns, and so on that developers created in the Development phase. If any of the functionality needs to change, you can use the Administration Console to make changes, or return to the Development phase and use Workshop for WebLogic and make changes. Developers can also utilize the Java API. Developers must redeploy the portal application to see the changes in the Staging environment. The Development phase and the Staging phase often occur simultaneously.

Tools: Administration Console.

The following chapters provide instructions on Staging tasks:

- [Chapter 11, “Modifying Property Set Values”](#) shows how to change the values in your User Profile property sets.
- [Chapter 12, “Modifying a User Segment”](#) describes how to change User Segment properties (conditions) to dynamically group users.
- [Chapter 13, “Modifying a Content Selector”](#) provides the steps to edit a Content Selector property, so that you can change the content that is displayed.
- [Chapter 14, “Modifying a Placeholder”](#) shows how to manage the content that populates Placeholders.
- [Chapter 15, “Managing a Campaign”](#) gives instructions on making changes to a Campaign, including the start or stop date and modifying the query user name.

Production

After developers test the portal application in the Staging phase, portal administrators use the [Production](#) phase to fine-tune the live production environment. For example, in the Production phase, administrators could use the Administration Console to modify Placeholders, Content Selectors, or Campaigns. They can change a Campaign’s effective dates, update web content, modify the discount offered in a catalog, or add a new User Segment to attract a different audience.

If you need to change any of these features, developers can use Workshop for WebLogic to return to the Development phase and make changes. Developers must redeploy the portal application to see the changes in the Staging and Production environments.

Tools: Administration Console.

See Part IV: [Production](#) for guidance on Production tasks.

Getting Started

If you are new to portal development, see the [WebLogic Portal Overview](#) for more information about the portal life cycle.

You can also consult the following information:

- [WebLogic Portal Overview](#)
- [Security Guide](#)
- [User Management Guide](#)

Planning an Interaction Strategy

This chapter describes when to use different types of Personalization and explains the relationships between the Interaction Management features.

This chapter includes the following sections:

- [Choosing the Type of Interaction Management to Develop](#)
- [Checklist for Planning Your User Interaction Strategy](#)
- [Checklist for Planning Your Campaign Strategy](#)
- [Planning Your Behavior Tracking Strategy](#)
- [Updating Interaction Management Features](#)
- [Upgrading Interaction Features from Portal 8.1](#)

Choosing the Type of Interaction Management to Develop

Use [Table 1](#) to determine which type of Interaction Management to develop.

This section contains the following topic:

- [Understanding Conditions](#)

Table 2-1 Types of Interaction Management

If you want to	...do this
<p>Display different graphics – each time an employee visits the Intranet portal, display a different picture from the company picnic.</p> <p>This action displays a binary property from a single content item from the virtual content repository that can change each time a user visits your portal or clicks Refresh.</p>	<ul style="list-style-type: none">• Create a generic rotation of content for all users by creating a Placeholder and adding a default query for the Placeholder that displays the range of content you want.• Create a targeted rotation of content for each user based on each user's characteristics by creating a Placeholder and a Campaign.• Set up the Campaign with content actions that put different types of content in the Placeholder for different types of users.• Define the necessary conditions and rules to use in the Campaign. <p>You can also use the <code><ad:adTarget></code> JSP tag as an alternative to a Placeholder to manually embed a content query in a JSP.</p> <p>See Chapter 7, “Creating a Placeholder” for more information.</p>
<p>Display a graphic specific to the user type – when a certain type of user visits the portal, display a graphic specific to the user type.</p> <p>For example, if a <i>manager</i> user views the portal, show the manager the <i>performance review reminder</i> graphic. If a <i>regular employee</i> user views the portal, show the employee the <i>benefits open enrollment</i> graphic.</p> <p>This action displays a binary property from a single content node from the Virtual Content Repository that shows the same content node for each type of user.</p>	<p>Target specific users differently by creating a Placeholder with a default query for all users or a Placeholder used by a Campaign to target specific users differently. Use one of the following ways to show the same content node without content rotation:</p> <ul style="list-style-type: none">• Set up your content with properties and values that can uniquely identify each piece of content.• Create highly focused content queries in the Placeholder or the Campaign to retrieve those single unique content items. <p>See Chapter 7, “Creating a Placeholder” and Chapter 8, “Building a Campaign” for more information.</p>

Table 2-1 Types of Interaction Management

If you want to	...do this
<p>Show each user a unique list of recommended books – the list is based on the user's characteristics.</p> <p>This action displays multiple content nodes and properties from the virtual content repository simultaneously.</p>	<p>Show multiple content nodes from the Virtual Content Repository simultaneously by creating Content Selectors and adding them to your JSPs.</p> <p>See Chapter 6, “Creating a Content Selector” for more information.</p>
<p>Show content that matches a user's characteristics – provide different sections of HTML content in a JSP but show users only the sections that match their characteristics.</p> <p>This action displays personalized content from an inline section of a JSP.</p>	<p>Display personalized inline JSP content by creating User Segments and using the <code><pz:div></code> JSP tag to wrap personalized content.</p> <p>You can also use the following JSP tags to display inline JSP content based on the device that is viewing the content (for example, a handheld device or a PC): <code><cscm:default></code>, <code><cscm:not-default></code>, <code><cscm:recognized></code>, <code><cscm:not-recognized></code>, <code><cscm:when></code>, and <code><cscm:when-not></code>. The <code>cscm</code> tags are Portal Multichannel JSP tags. They are contained in the <code>client_taglib.jar</code> file.</p> <p>See Chapter 6, “Creating a Content Selector” and the <i>Javadoc</i> for more information.</p>
<p>Send users automatic e-mails.</p>	<p>Create and store e-mail message files and create a Campaign that uses an e-mail action. See Chapter 8, “Building a Campaign.”</p>
<p>Give users automatic discounts.</p>	<p>Perform the following tasks:</p> <ul style="list-style-type: none"> • Add commerce services to your portal application. • Set up a shopping cart using the WebLogic Portal Commerce API. • Create a catalog in the virtual content repository. • Use the WebLogic Portal catalog classes in the Commerce API to retrieve catalog items from the virtual content repository and identify them with categories and SKU properties. • Create discounts and use the Commerce API to view the discounts in your shopping cart. You can also use the API to surface the discount's description next to the discount amount displayed in the shopping cart.

Table 2-1 Types of Interaction Management

If you want to	...do this
Use rules in Page Flows and Web Services to provide a personalized user experience	Use the Rules Executor control in your Page Flows and Web Services. See Chapter 10, "Creating Advanced Personalization with Rules" .

Understanding Conditions

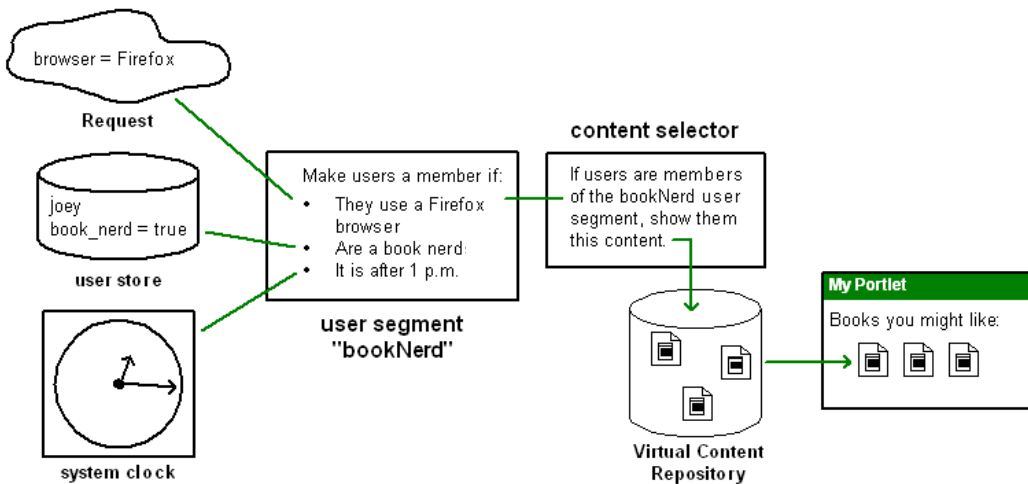
Interaction Management uses a variety of conditions that identify users and what they are doing.

When you build Interaction Management functionality, you use conditions to perform the following actions:

1. Specify the exact characteristics that identify the users you want to target
2. Define the actions that occur when users that match those conditions visit your portal

[Figure 2-1](#) illustrates how personalized content is dynamically displayed to users with a Content Selector. Conditions are captured, specific users are identified, and actions occur for those users.

Figure 2-1 Simple Example of Interaction Management Logic and Flow



A rules engine runs behind the scenes on a server in a portal domain, reads all available conditions in memory, evaluates those conditions against the rules you created, and executes the actions you defined if the conditions match your rules.

For example, the following list contains some of the conditions that a Campaign can use to determine which users to target with personalized content:

- Dynamically predefined groups of users (User Segments)
- Properties in a User’s Profile (such as personal preferences)
- Specific properties in the HTTP request or session
- An event that occurs (such as performing a click)
- Date and time factors
- Items or value of items in a shopping cart

Checklist for Planning Your User Interaction Strategy

Adding Personalization to your portal can involve setting up several related pieces. The checklist in [Table 2-2](#) includes items that you should consider when planning a personalized portal.

Table 2-2 User Interaction Strategy Checklist

Check box	Planning Item	Description
<input type="checkbox"/>	1. Create content	Determine the content you want to display, when it should display, and set specific properties on your content items. For example, you can use Workshop for WebLogic to add properties to your content that can make an image clickable, end a Campaign after a specific number of clicks, start a movie, provide a clickable URL, and so on. See Chapter 3, “Setting up Content” for instructions. You should also consult the <i>Content Management Guide</i> for information on creating and maintaining a virtual content repository.

Table 2-2 User Interaction Strategy Checklist (Continued)

Check box	Planning Item	Description
<input type="checkbox"/>	2. Set up Property Sets and properties	<p>Property sets use properties to create conditions that uniquely identify users. The properties you create in BEA Workshop for WebLogic Platform are used in the conditions you define for your Personalization logic. For example, you could create a NewHire property set to target these users with benefit enrollment information.</p> <p>BEA Workshop for WebLogic Platform provides editors to help you define the following properties to create conditions that identify users:</p> <ul style="list-style-type: none">• User Profile properties determine which user information to save. User Profile properties can also be used to define Visitor Entitlement and Delegated Administration roles.• Request properties capture and use specific HTTP request information to trigger Personalization.• Session properties capture and use specific HTTP session information to trigger Personalization.• Custom Events can trigger Personalization and Campaigns and track user behavior. <p>Based on the logic conditions, each user is dynamically served personalized, accurate web content, automatic e-mails, and discounts. See Chapter 4, “Creating a Property Set” for instructions.</p>
<input type="checkbox"/>	3. Set up users	<p>Access existing users in external databases or add new users to your portal. For instructions on setting up and managing users, see the User Management Guide.</p>

Table 2-2 User Interaction Strategy Checklist (Continued)

Check box	Planning Item	Description
<input type="checkbox"/>	4. Create User Segments	<p>You can create User Segments to dynamically categorize users based on conditions or criteria that define the target visitor. Those conditions can include characteristics, such as occupation, browser type, User Profile values, or other user properties.</p> <p>For example, you could classify all users who ordered more than five on-demand movies in the last 30 days. If visitors match the defined characteristics, they automatically become members of that User Segment and are shown specific web content with Content Selectors or are targeted with Campaign Actions.</p> <p>User Segments can be used over and over in Content Selectors, Placeholders, and Campaigns. See Chapter 5, “Creating a User Segment” for instructions.</p>
<input type="checkbox"/>	5. Create Content Selectors	<p>A Content Selector lets you target specific groups of people with content items from BEA’s Virtual Content Repository. For example, after you create a User Segment to trigger Personalization, you can create a Content Selector that defines the content that is shown to users in a specific User Segment. You could display a list of recommended movies to users identified as <i>movie fans</i>. For instructions, see Chapter 6, “Creating a Content Selector”.</p>
<input type="checkbox"/>	6. Create Placeholders	<p>A Placeholder displays a single personalized content item on a JSP. The content item is dynamically retrieved from BEA’s Virtual Content Repository.</p> <p>A Placeholder uses queries to retrieve and display one piece of content at a time. For example, if a user is identified as a bird lover, a Placeholder in a Campaign can display an image of a bird with a store discount. The image can change with a browser refresh to show other birds as well. You can also use Placeholders by themselves to display specific types of non-personalized content that is not provided by a Campaign. See Chapter 7, “Creating a Placeholder”.</p>

Table 2-2 User Interaction Strategy Checklist (Continued)

Check box	Planning Item	Description
<input type="checkbox"/>	7. Create Campaigns	<p>A Campaign lets you target specific users with a single piece of personalized content at a time, automatically send them a predefined e-mail, or provide a discount in a commerce application. Campaigns run for a limited time and drive online behavior and Personalization to achieve a specific business goal. Your Marketing Department generally drives the content of a Campaign. For instructions, see Chapter 8, “Building a Campaign”.</p>
<input type="checkbox"/>	8. Set up Behavior Tracking	<p>You can use events to trigger Campaigns, persist event data in the database, and other actions. Events are generated when users interact with a web interface, such as logging in, clicking or viewing a graphic, clicking a button, navigating to another page in a portal, and so on. These events that occur in a user’s path through your portal are logged to the database, so you can analyze the user behavior in your portal. For example, you could determine how many users have registered in a portal and then create a Campaign that automatically sends each user a welcome e-mail when the registration event occurs.</p> <p>You can also be notified of custom events at runtime and respond accordingly. You might decide to forward events to another system or make runtime decisions on the basis of those events.</p> <p>See Chapter 9, “Setting Up Events and Behavior Tracking” for more information.</p>

One of the most important benefits of using Interaction Management is that the logic is decoupled from your source code. The files you create (Campaigns, Placeholders, Content Selectors, and so on) contain the Personalization logic and content queries, and your code references those files. For example, Campaigns show web content using a JSP tag called a Placeholder.

The next step is to define your Campaign to use the existing Placeholders, each of which can display content unique to the Campaign and to the individual users. Campaigns can change and new ones can be added, but you never have to change your JSP code. The Placeholders you need in the JSPs stay the same.

Checklist for Planning Your Campaign Strategy

This section contains the following topic:

- [Understanding Campaigns](#)

Campaigns use pieces of Interaction Management to achieve a business goal. If you plan to use a Campaign in your portal, use the checklist in [Table 2-3](#).

Table 2-3 Campaign Strategy Checklist

Check box	Planning Item	Notes
<input type="checkbox"/>	1. Create a Portal application	See Creating a Portal Application and Portal Web Project in the Workshop for WebLogic help system.
<input type="checkbox"/>	2. Set up content	<p>When you show personalized content with a Campaign (using a content rule), the content is retrieved from BEA's Virtual Content Repository and displayed in a Placeholder. There are many properties you can add to your content that enable necessary and helpful features for Campaigns. For example, to increase the chances of a specific content item being shown in a Placeholder, create an adWeight property (as an Integer) for your content items. The greater the adWeight number you enter for a content item, the greater the chances that it will be displayed in a Placeholder if it is retrieved by a query.</p> <p>For more information on setting up content for use in Interaction Management, see Chapter 3, "Setting up Content".</p>
<input type="checkbox"/>	3. Decide if you will use Goal Setting	Goal Setting ends a Campaign based on the number of content items displayed or clicked. For more information on Goal Setting, see "Planning Your Campaign Logic" on page 8-4 and "Setting Goal Definitions" on page 8-7 .
<input type="checkbox"/>	4. Create Placeholders	<p>Campaigns use Placeholders to display personalized web content. If you display personalized content through Campaigns, create the Placeholders that will hold your Campaign queries and display the web content.</p> <p>For more information on Placeholders, see Chapter 7, "Creating a Placeholder".</p>

Table 2-3 Campaign Strategy Checklist (Continued)

Check box	Planning Item	Notes
<input type="checkbox"/>	5. Create User Segments	If you want to trigger a Campaign based on users who are grouped dynamically based on specific characteristics, create User Segments. For more information on User Segments, see Chapter 5, “Creating a User Segment” .
<input type="checkbox"/>	6. Create Property Sets	If you plan to trigger a Campaign based on properties from users, events, HTTP sessions, or HTTP requests, perform the following relevant procedures: <ul style="list-style-type: none"> • Create User Profile properties • Register Custom Events • Create Session Properties • Create Request Properties For more information on how these properties are used in Interaction Management, see “Setting up a Property Set” on page 4-2 .
<input type="checkbox"/>	7. Set up e-mail messages	You can send automatic e-mails in a Campaign. Follow the steps in “Setting Up Automatic E-Mail Messages” on page 8-18 .
<input type="checkbox"/>	8. Trigger the Campaign	Set up a Regular or Behavior Tracking event that start your Campaign. A commonly used event is <code>SessionLoginEvent</code> . For instructions, see “Triggering a Campaign” on page 8-31 . See “Using Events in Campaigns” on page 9-43 and “Registering Events for Campaigns” on page 9-45 for instructions.

Understanding Campaigns

The following examples show different ways to use Campaigns:

- A company provides open benefits enrollment for its employees, where employees can change their current benefits choices. In the internal Human Resources portal, the company creates a Campaign that runs from November 1 - 30. During that time the Campaign displays an Open Enrollment graphic in the portal header region and when employees

make changes to their benefits and click **Submit**, a confirmation e-mail is automatically sent.

- A large online retailer is running a holiday special for its external customers. The retailer creates a Campaign that provides a one-time discount of 30% off the cost of books when the total cost of books in any order is \$100 or more.
- A mobile devices ISP creates a Campaign that shows targeted add-on services that are specific to each type of mobile device when users click the **New Stuff!** link.

Note: Campaigns and Behavior Tracking are not currently supported for anonymous, non-trackable users. See the [User Management Guide](#).

Planning Your Behavior Tracking Strategy

WebLogic Portal's event framework provides many options for generating and handling events, to track the behavior of visitors to your portal. This section provides guidelines to help you determine the pieces of the event framework you want to use to implement the functionality you need.

This section contains the following topics:

- [Understanding When to Use a Predefined Event](#)
- [Understanding When to Create a Custom Event](#)
- [Understanding When to Create a Custom Event Listener](#)

Understanding When to Use a Predefined Event

WebLogic Portal provides many predefined Behavior Tracking events you can use in your applications, described in “[Using Predefined Events](#)” on page 9-7. Each event collects specific attributes and structures those attributes as XML, and the Behavior Tracking listener puts the XML in a buffer to insert into the BT_EVENT database table.

Most of the predefined events also have predefined event property sets in Workshop for WebLogic, stored in the portal application's `/data/src/events` directory. These property sets let you use events in your Campaign definitions to trigger Campaign actions when the events occur or when events have specific attribute values.

The following list explains when to use WebLogic Portal's predefined events:

- You want to store event data as XML in the BT_EVENT table

- The predefined events capture specified attributes
- The events capture the attributes you want, but you want to handle the events in a customized way by creating your own event listener
- You want to use the events in your Campaign

Understanding When to Create a Custom Event

If none of WebLogic Portal's predefined events capture the specific combinations of attributes you need, create a custom event. There are two types of custom events you can create: Behavior Tracking events and regular events.

See [Chapter 9, "Setting Up Events and Behavior Tracking"](#) for instructions on setting up events.

Planning Behavior Tracking Events

Create a custom Behavior Tracking event when none of Portal's predefined events captures the event attributes you want and you want to use Portal's Behavior Tracking framework to persist event data as XML in the BT_EVENT table. You can use these events in Campaigns and create a custom listener that performs special handling on the event, but unless you want to use the Behavior Tracking framework to store event data as XML, you do not need to create a custom Behavior Tracking event.

If you do not want to use the Behavior Tracking service, create a custom regular event.

Planning Regular Events

Create a custom regular event when none of WebLogic Portal's predefined events captures the event attributes you want and you do not want to use the Behavior Tracking service for persisting event data as XML in the BT_EVENT table.

The following list describes when to create a custom regular event:

- You want to capture a specific set of attributes with an event and use that event to trigger Campaigns
- You want to capture a specific set of attributes with an event and execute custom functionality when that event occurs (using a custom event listener)

Understanding When to Create a Custom Event Listener

WebLogic Portal provides two listeners: a Campaign listener and a Behavior Tracking listener.

The Campaign listener tells the Campaign service when an event has occurred (with the exception of the ignored events in the `wps.jar` file's `listener.properties` file). The Campaign service reads the current request and executes Campaign actions if the request data matches the conditions of any of your Campaigns. If your Campaign definitions include any event conditions, which you were able to supply with event property sets, the Campaign service evaluates those as well to determine if it must execute Campaign actions.

The Behavior Tracking listener listens for only the Behavior Tracking events that are registered with the Behavior Tracking service. When it receives an event it is interested in, it moves the XML document for that event into a buffer for later persistence into the `BT_EVENT` table at an interval you determine.

Create a custom event listener if you want to execute functionality not provided by the Campaign listener or the Behavior Tracking listener. For example, if you want to perform your own event data persistence, modify a User Profile, redirect the user to another part of a Page Flow, or provide any other type of real-time response to the event, create a custom event listener that provides the functionality you want.

See [Chapter 9, “Setting Up Events and Behavior Tracking”](#) for more information.

Updating Interaction Management Features

After you create Property Sets, User Segments, Content Selectors, Placeholders, and Campaigns in Workshop for WebLogic, you can modify the settings and queries for those components in the WebLogic Portal Administration Console. For instructions, see [Chapter 11, “Modifying Property Set Values”](#), [Chapter 12, “Modifying a User Segment”](#), [Chapter 13, “Modifying a Content Selector”](#), [Chapter 14, “Modifying a Placeholder”](#), and [Chapter 15, “Managing a Campaign”](#).

If you need to create new Interaction Management features or modify properties, use Workshop for WebLogic and then iteratively push your updates to the running server. For more information, see the [Production Operations Guide](#).

Upgrading Interaction Features from Portal 8.1

When you run the BEA WebLogic Upgrade Wizard, the wizard upgrades your WebLogic Portal 8.1 SP4, SP5, or SP6 interaction features, such as Content Selectors, Placeholders, Campaigns, and so on.

When you run the BEA WebLogic Upgrade Wizard and it detects your Portal 8.1 installation, you can select the **Upgrade RDBMSAuthenticator** option. Selecting this option replaces the existing authentication provider with the new *SQLAuthenticator* authentication provider and

upgrades all content, including personalization features. You can also choose to manually upgrade your personalization features from Portal 8.1 SP4, SP5, or SP6 to the Portal 9.2 RDBMS user store later. For step-by-step instructions on running the BEA WebLogic Upgrade Wizard, see the *Upgrade Guide*.

Part I Architecture

Part I includes the following chapters:

[Chapter 2, “Planning an Interaction Strategy”](#)

[Chapter 3, “Setting up Content”](#)

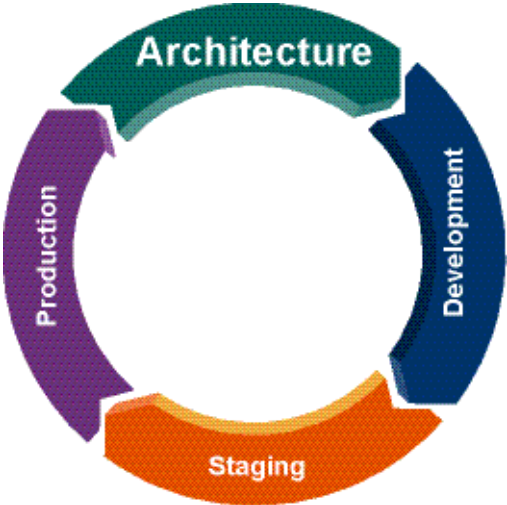
This section contains guidelines to help you plan the type of user interaction you will add to your portal. Developing a user interaction strategy can save you time during the other phases of the portal life cycle.

When you are planning how your users will interact with your portal, determine the following:

- What type of personalized content to display
- How to set up and store content
- If content will be controlled by the type of user
- How long to display each type of content
- How to cycle through content
- If you want to send automatic e-mails or give automatic discounts
- Where to store the user information you gather

Part I contains instructions for planning your strategy for user interaction, Campaigns, and Behavior Tracking in the Architecture phase. When you finish the Architecture phase, you can proceed to the [Development](#) phase, and then on to the other phases.

For a description of the Architecture phase of the portal life cycle, see the [WebLogic Portal Overview](#). The portal life cycle is shown in the following graphic:



Setting up Content

Targeting users with personalized content is an important part of Interaction Management. You can use Placeholders and Campaigns to control the type of content you display and how long the content appears. The first step is to add content types and content to your BEA Repository.

This chapter includes the following sections:

- [Adding Content](#)
- [Determining Content Priority](#)

Adding Content

Use the *Content Management Guide* to learn how to add content to your BEA Repository. You can set up a hierarchy of folders and add content types (each of which contain properties). Content types determine the metadata you associate with a content file and how the content is retrieved in a search.

You can create your own custom content types, or use the following six content types that ship with WebLogic Portal:

- Ad Content Type
- Article Content Type
- Book Content Type
- Image Content Type

- Message Content Type

Plan your content types carefully. You can add a property definition to the content type after you have instances of the type. However, you cannot modify or delete existing property definitions of the type. Content type properties describe the content and help you manage that content. The more properties you associate with content items, the more granular your search results can be.

Adding specific properties to content items in your repository can make an image clickable, end a Campaign after a specific number of clicks, start a movie, provide a clickable URL, and so on. You can also perform repository management with content properties by viewing the date (a Content Type property) that the item was added to the repository.

Content type properties can be any of the following data types: Boolean, Long Integer, Number with a Decimal, String, Date/Time, Binary, Nested Content Type, or Link. See the [Content Management Guide](#) for more information.

Tip: You should set up your content type properties before you create Placeholders and Campaigns that access this content.

If you create Visitor Entitlements on a Content Management resource, these entitlements can prevent a portal visitor from seeing content they would normally see according to Personalization rules.

Determining Content Priority

A Placeholder—the JSP tag in a JSP that displays general content or personalized content for a Campaign—displays one piece of content at a time. When a content query in a Placeholder (a default Placeholder query or a query put in the Placeholder by a Campaign) returns multiple possible content items to a Placeholder, the Placeholder determines which content item to display. Use a content type property called **adWeight** to change the chance of displaying content in a Placeholder when the content items are retrieved with a query.

The **adWeight** property is an Integer property type. The higher the **adWeight** number you assign to a content item, the better the chance it will display in the Placeholder. See [Chapter 7, “Creating a Placeholder”](#) for instructions on using the **adWeight** property.

Part II Development

Part II includes the following chapters:

[Chapter 4, “Creating a Property Set”](#)

[Chapter 5, “Creating a User Segment”](#)

[Chapter 6, “Creating a Content Selector”](#)

[Chapter 7, “Creating a Placeholder”](#)

[Chapter 8, “Building a Campaign”](#)

[Chapter 9, “Setting Up Events and Behavior Tracking”](#)

[Chapter 10, “Creating Advanced Personalization with Rules”](#)

Developers use Workshop for WebLogic in the Development phase to create user property sets and properties, User Segments, Placeholders, Content Selectors, Campaigns, and Behavior Tracking. Portal administrators can use the WebLogic Portal Administration Console to update some of these features’ properties and values.

Developers can use these property sets, User Segments, Placeholders, and Campaigns that they create in the Development phase to personalize a portal by performing some of the following tasks:

- Target users with personalized web content (property sets and Content Selectors)
- Display a single piece of web content retrieved from the BEA Virtual Content Repository (Placeholders)
- Send a user an automatic predefined e-mail (Campaign)
- Provide a discount in a commerce application (Campaign)

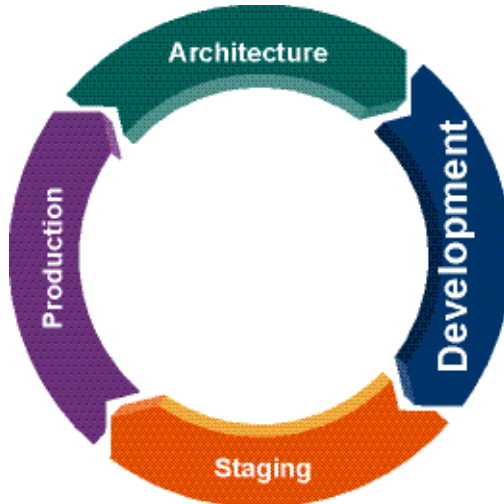
- Categorize users based on specific characteristics or criteria and then target those segments (User Segments)

The decisions you made during the [Architecture](#) phase shape what you do in the Development phase. For example, you plan Content Selectors in the Architecture phase and create them in the Development phase. A Content Selector is then tested in the Staging phase and fine-tuned in the Production phase. Each feature can progress through the portal life cycle.

When you finish the Development phase you can proceed to the [Staging](#) phase. Consider setting up a common development environment for the Development phase and the Staging phase. You might move iteratively between these two phases, developing and then testing what you created.

If you moved on to the [Architecture](#) phase and then go back to make changes that affect the Development phase, you must redeploy your portal application in order to view your changes. The BEA Propagation Utility performs the redeployment; see the [Production Operations Guide](#) for more information.

For a detailed description of the Development phase of the portal life cycle, see the [WebLogic Portal Overview](#). The portal life cycle is shown in the following graphic:



Creating a Property Set

Developing user interaction that uses Personalization and Campaigns can involve several steps. For example, if you want to target users with personalized content in a Campaign, you will add content to BEA's Virtual Content Repository, create Placeholders that display the content, set up properties (such as User Profile or Session properties) that are used to define the conditions under which users will be targeted with Campaign content, and then create the Campaign.

This chapter describes how to create these property sets that have conditions to identify users. The properties are used in the conditions you define for your Personalization logic. Each user is dynamically served personalized web content, automatic e-mails, or discounts based on the logic conditions.

Workshop for WebLogic provides editors to help you define the following properties and events to create conditions that identify and track users:

- **User Profile properties** – Determine which user information to save. User Profile properties can also be used to define Visitor Entitlement and Delegated Administration roles.
- **Request properties** – Capture and use specific HTTP request information to trigger Personalization. Request properties are associated with a request and are not persisted between requests. Request properties can also be used to define Visitor Entitlement and Delegated Administration roles.
- **Session properties** – Capture and use specific HTTP session information to trigger Personalization. Session properties are associated with a session and are not persisted between sessions. Session properties can also be used to define Visitor Entitlement and Delegated Administration roles.

- **Custom Events** – Trigger Personalization and Campaigns and track user behavior. You must register custom events so that your application recognizes them.

Developers use Workshop for WebLogic to create property sets and properties. Portal administrators can use WebLogic Portal Administration Console to update the values in the property set. See “[Checklist for Planning Your User Interaction Strategy](#)” on page 2-5 for information on designing a property set, and see [Chapter 11, “Modifying Property Set Values”](#) to learn how to update property set values.

This chapter includes the following sections:

- [Setting up a Property Set](#)
- [Adding Properties or Conditions to a Property Set](#)
- [Modifying Properties and Conditions](#)
- [Deleting a Property or a Property Set](#)

For information on setting up and managing users that will use Interaction Management features, see the [User Management Guide](#).

Setting up a Property Set

You can use Workshop for WebLogic to create a property set in for a User Profile, User Segment, HTTP session or request data, date and time condition, or an event.

For example, a User Profile consists of additional attributes you collect and store about a user. Each piece of metadata in a User Profile is called a property. User properties can range from statically-defined properties, such as a user’s phone number and e-mail address, to dynamically-created and persisted properties (web site tracking information for the user, for example).

You could create a property set called **human resources** that contains properties, such as `gender`, `hire date`, and `e-mail-address`. User Profile properties appear as input fields in the WebLogic Portal Administration Console when you edit a User Profile value. (You can also assign Group Profile property values to groups.) The properties you create are also used to define rules for Personalization, as well as Delegated Administration and Visitor Entitlement roles. Users and groups can have multiple profiles, if a profile equates to a property set. WebLogic Portal provides a default User Profile property set called `CustomerProperties.usr` that contains common properties.

There are specific properties you can set on content items to enhance Personalization in your applications. See [Chapter 6, “Creating a Content Selector”](#) for more information. For general information on Content Management, see the [Content Management Guide](#).

After you create the necessary Personalization properties and conditions and set up users and content, you can create Interaction Management functionality for your portal. For example, after you create a User Segment to trigger Personalization, you can create a Content Selector that defines the content that is shown to users in a specific User Segment.

The following section describes how to create a property set for a User Profile, User Segment, HTTP Session or Request, event, Community, or remote portlet. Property sets are application-scoped; any additional scoping or namespacing must be performed by the application.

Property Sets and other Interaction Management features use a variety of conditions that identify users and what they are doing. For more information on conditions, see [“Understanding Conditions” on page 2-4](#).

Note: The steps in this chapter refer to the `data\src` folder in the Package Explorer View. Your `data` and `src` directories might be named differently.

This section contains the following topics:

- [Creating a User Profile Property Set](#)
- [Creating a User Segment Property Set](#)
- [Creating a Session Property Set](#)
- [Creating a Request Property Set](#)
- [Creating a Community or Remote Portlet Property Set](#)
- [Creating an Event Property Set](#)
- [Creating a Catalog Property Set](#)

Creating a User Profile Property Set

Consult the [“Checklist for Planning Your User Interaction Strategy” on page 2-5](#) for more details on designing a property set.

Perform the following steps to create a property set for a User Profile:

Creating a Property Set

1. In the Portal Perspective in Workshop for WebLogic, right-click the **data\src\userprofiles** folder in the Package Explorer View and choose **New > Other**.

Tip: You can customize the menu so that **Property Sets** appears as a choice on the **New** menu. See the [Portal Development Guide](#) for instructions.

2. In the New Select a Wizard window, expand the **WebLogic Portal** folder and expand the **Property Sets** folder.
3. Select **User Property Set** and click **Next**.
4. In the New User Property Set window, enter a name for the User Profile property set in the **File name** field. Keep the `.usr` file extension.
5. Click **Finish**. The User Profile Editor appears.
6. Add properties to the property set by following the instructions in [“Adding Properties or Conditions to a Property Set”](#) on page 4-11.

Creating a User Segment Property Set

You can target visitors with web content, automatic e-mails, and discounts by defining and using groups called User Segments (similar to segments of a population). User Segments dynamically group users based on characteristics, such as group membership, browser type, profile values, and other user properties. If users match the characteristics, they automatically and dynamically become members of that User Segment. You can use User Segments in Content Selectors, Placeholders, and Campaigns.

Perform the following steps to create a User Segment:

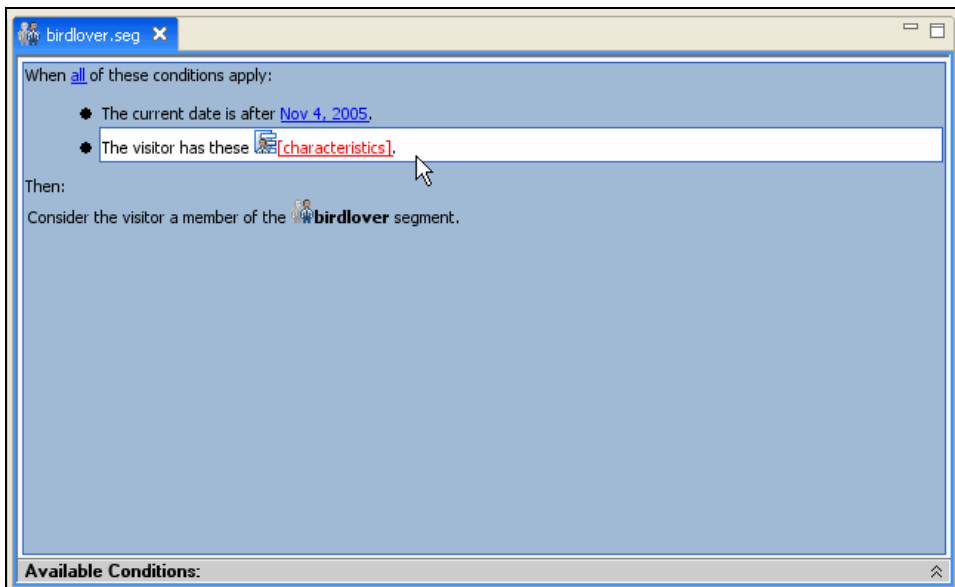
1. In the Portal Perspective in Workshop for WebLogic, right-click the **<data>\src** folder in the Package Explorer View and choose **New > Other**.

Tip: You can customize the menu so that **User Segment** appears as a choice on the **New** menu. See the [Portal Development Guide](#) for instructions.

2. In the New Select a Wizard window, expand the **WebLogic Portal** folder.
3. Select **User Segment** and click **Next**.

4. In the New User Segment window, enter a name for the User Segment in the **File name** field. Keep the `.seg` file extension.
5. Click **Finish**. The User Segment Editor appears.
6. In the **Palette** tab, drag the conditions you want to use into the User Segment Editor. See [Figure 4-1](#).

Figure 4-1 This User Segment is Called birdlover.seg



7. When you add a condition to the User Segment, click the condition link to set the conditions.

For example, if you drag the condition *The visitor has specific characteristics* into the User Segment Editor, click the link in the Editor to select the User Profile properties and their values that make a user a member of the User Segment.

Properties for the user, HTTP session, and HTTP request conditions are based on the User Profile, HTTP sessions, and HTTP request properties you create.

Choose conditions based on the descriptions in [Table 4-1](#).

Table 4-1 Conditions that Identify Users to Target and the Actions that Will Occur

Condition

Table 4-1 Conditions that Identify Users to Target and the Actions that Will Occur (Continued)

The visitor is a member of a predefined User Segment	If the visitor to your web site belongs to a predefined User Segment, execute the specified action. For example, if the visitor belongs to the Gold Customer User Segment , show the visitor a specific piece of web content (action).
The visitor has specific characteristics	If a visitor's characteristics are compared to values you specify and those comparisons evaluate as <code>true</code> , execute a specified action. For example, if the visitor's salary (characteristic) is greater than or equal to (comparison) \$50,000 (value), send the visitor an e-mail (action).
The visitor's HTTP session has specific properties	If the HTTP session's properties are compared to values you specify and those comparisons evaluate as <code>true</code> , execute a specified action. An HTTP session is information your organization might want to track to learn about a visitor's browsing session on the web site.
An HTTP request has specific properties	If the HTTP request's properties are compared to values you specify and those comparisons evaluate as <code>true</code> , execute a specified action.
The date is	If the current date is equal to the one you specify, execute a specified action. For example, if the date is equal to July 22, 2006, send users an e-mail about an upcoming sale (action). The current date refers to the date at the point that the condition is evaluated for a user visiting the web site. For more information, see “Setting Dates and Times” on page 4-7 .
It is after a given date	If the current date is after a date you specify, execute a specified action. For example, if the date is after December 18, 2005, offer users a discount (action). The current date refers to the date at the point that the condition is evaluated for a user visiting the web site. For more information, see “Setting Dates and Times” on page 4-7 .
It is after a given date and time	If the current date and time are after a date and time you specify, execute a specified action. For example, if the date and time are after July 22, 2006 at 3 p.m., send users an e-mail about an upcoming sale (action). The current date and time refer to the date and time at the point that the condition is evaluated for a user visiting the web site. For more information, see “Setting Dates and Times” on page 4-7 .

Table 4-1 Conditions that Identify Users to Target and the Actions that Will Occur (Continued)

It is between two times	If the current time falls within a range of times you specify, execute a specified action. For example, if the time is between 3 p.m. and 5 p.m., offer users a discount (action). The current time refers to the time at the point that the condition is evaluated for a given user visiting the web site. For more information, see “Setting Dates and Times” on page 4-7 .
It is between two dates	If the current date falls within a range of dates you specify, execute the specified action. For example, if the date is between December 18, 2005 and December 18, 2006, show users a sale ad (action). The current date refers to the date at the point that the condition is evaluated for a given user visiting the web site. For more information, see “Setting Dates and Times” on page 4-7 .
It is between two dates/times	If the current date and time fall within a range of dates and times you specify, execute the specified action. For example, if the date and time is between July 22, 2005 at 3 p.m. and July 22, 2006 at 3 p.m., show users a sale ad (action). The range of dates is inclusive. The current date and time refer to the date and time at the point that the condition is evaluated for a user visiting the web site. For more information, see “Setting Dates and Times” on page 4-7 .

8. Save the User Segment file by choosing **File > Save**.

Setting Dates and Times

When you set date and time conditions, the dates and times represent the time in your region. For example, if you are creating a Campaign action that will be triggered at 8 p.m., that means 8 p.m. in your region. For a time zone that is two hours behind you, the action will be triggered at 6 p.m. in that time zone.

This also affects dates you set. The date you set becomes effective at midnight in your time zone. In a time zone that is six hours ahead of yours, that date becomes effective for that time zone at 6 p.m. your time the day before.

Time changes also affect time-triggered actions. For example, you created a Campaign that begins October 1 at noon and ends October 31 at noon. If a change to standard time (one hour earlier) occurs on October 29, the Campaign will actually end on October 31 at 11 a.m. If you want the Campaign to end at noon on the new standard time, set the end time to 1 p.m.

Tip: Because of the different dates and times on which actions will be triggered around the country or world, it is important to tell users that dates and times are effective for your

time zone. This type of information allows users to calculate when in their time zone they can take advantage of your promotions.

Creating a Session Property Set

Session properties capture and use specific HTTP session information to trigger Personalization and Campaigns. Session properties are associated with a session and are not persisted between sessions. An example of a Session property set could be **PortalA**.

Perform the following steps to create a Session property set:

1. In the Portal Perspective in Workshop for WebLogic, right-click the `<data>\src\session` folder in the Package Explorer View and choose **New > Other**.

Tip: You can customize the menu so that **Property Sets** appears as a choice on the **New** menu. See the [Portal Development Guide](#) for instructions.

2. In the New Select a Wizard window, expand the **WebLogic Portal** folder and expand the **Property Sets** folder.
3. Select **Session Property Set** and click **Next**.
4. In the New Session Property Set window, enter a name for the Session property set in the **File name** field. Keep the `.ses` file extension.
5. Click **Finish**. The Session Editor appears.
6. Add properties to the property set by following the instructions in [“Adding Properties or Conditions to a Property Set”](#) on page 4-11.

Use code in a JSP or Java Page Flow to populate the session.

Creating a Request Property Set

Request properties capture and use specific HTTP request information to trigger Personalization. Request properties are associated with a request and are not persisted between requests.

Request properties can also store values that are populated programmatically. For example, the **DefaultRequestPropertySet.req** property set in the `<project>\data\src\request\` directory is included with every portal web project, and contains properties called **User-Agent** and **Client Classification**. These properties are populated automatically when a device accesses

a portal, providing a key component of the framework that delivers appropriate web content to mobile devices.

Perform the following steps to create a Request property set:

1. In the Portal Perspective in Workshop for WebLogic, right-click the `<data>\src\request` folder in the Package Explorer View and choose **New > Other**.

Tip: You can customize the menu so that **Property Sets** appears as a choice on the **New** menu. See the [Portal Development Guide](#) for instructions.

2. In the New Select a Wizard window, expand the **WebLogic Portal** folder and expand the **Property Sets** folder.
3. Select **Request Property Set** and click **Next**.
4. In the New Request Property Set window, enter a name for the Request property set in the **File name** field. Keep the `.req` file extension.
5. Click **Finish**. The Request Property Set Editor appears.
6. Add properties to the property set by following the instructions in “[Adding Properties or Conditions to a Property Set](#)” on page 4-11.

Creating a Community or Remote Portlet Property Set

You can create an application-defined property set to store profile data for entities that are not users or groups. These entities include Communities and remote portlets, or a custom entity created by an application programmer.

Perform the following steps to create a property set for an Application-defined property set:

1. In the Portal Perspective in Workshop for WebLogic, right-click the `data\src` folder in the Package Explorer View and choose **New > Other**.

Tip: You can customize the menu so that **Property Sets** appears as a choice on the **New** menu. See the [Portal Development Guide](#) for instructions.

2. In the New Select a Wizard window, expand the **WebLogic Portal** folder and expand the **Property Sets** folder.
3. Select the **Application-Defined Property Set** and click **Next**.

4. Enter a name for the Community or remote portlet property set in the **File name** field. Keep the `.propset` file extension.
5. Click **Finish**. The Application-Defined Property Set Editor appears.
6. Add properties to the property set by following the instructions in [“Adding Properties or Conditions to a Property Set”](#) on page 4-11.

Creating an Event Property Set

You must register custom events that you create so that your portal application recognizes the events. After you register your events, you can use them to trigger Personalization and Campaigns and track user behavior in your portals.

Events need event listeners to listen for them. Register the event listeners for your custom event in the `<PORTAL_APP>\META-INF\wps-config.xml` file.

Perform the following steps to create an Event property set:

1. In the Portal Perspective in Workshop for WebLogic, right-click the `<data>\src` folder in the Package Explorer View and choose **New > Other**.

Tip: You can customize the menu so that **Property Sets** appears as a choice on the **New** menu. See the [Portal Development Guide](#) for instructions.

2. In the New Select a Wizard window, expand the **WebLogic Portal** folder and expand the **Property Sets** folder.
3. Select **Event Property Set** and click **Next**.
4. In the New Event Property Set window, enter a name for the Event property set in the **File name** field. Keep the `.evt` file extension.
5. Click **Finish**. The Event Editor appears.
6. Add properties to the property set by following the instructions in [“Adding Properties or Conditions to a Property Set”](#) on page 4-11.

Creating a Catalog Property Set

The Catalog Structure Editor in Workshop for WebLogic lets you add properties to a catalog. After you add the properties, you must enable it in the WebLogic Portal Administration Console.

Catalog structure properties are name and value pairs that define the information you want to enter about items in your catalog, such as **SKU**, **Description**, and **Price**.

Before you create a Catalog property set, you should enable Catalog Management, which provides the Catalog tools that use the Catalog structure properties.

Perform the following steps to create a Catalog property set:

1. In the Portal Perspective in Workshop for WebLogic, right-click the `<data.\src\catalog` folder in the Package Explorer View and choose **New > Other**.

Tip: You can customize the menu so that **Property Sets** appears as a choice on the **New** menu. See the [Portal Development Guide](#) for instructions.

2. In the New Select a Wizard window, expand the **WebLogic Portal** folder and the **Property Sets** folder.
3. Select **Catalog Property Set** and click **Next**.
4. In the New Catalog Property Set window, enter a name for the property set in the **File name** field. Keep the `.clg` file extension.
5. Click **Finish**. The Catalog Structure Editor appears.
6. Add properties to the property set by following the instructions in [“Adding Properties or Conditions to a Property Set”](#) on page 4-11.

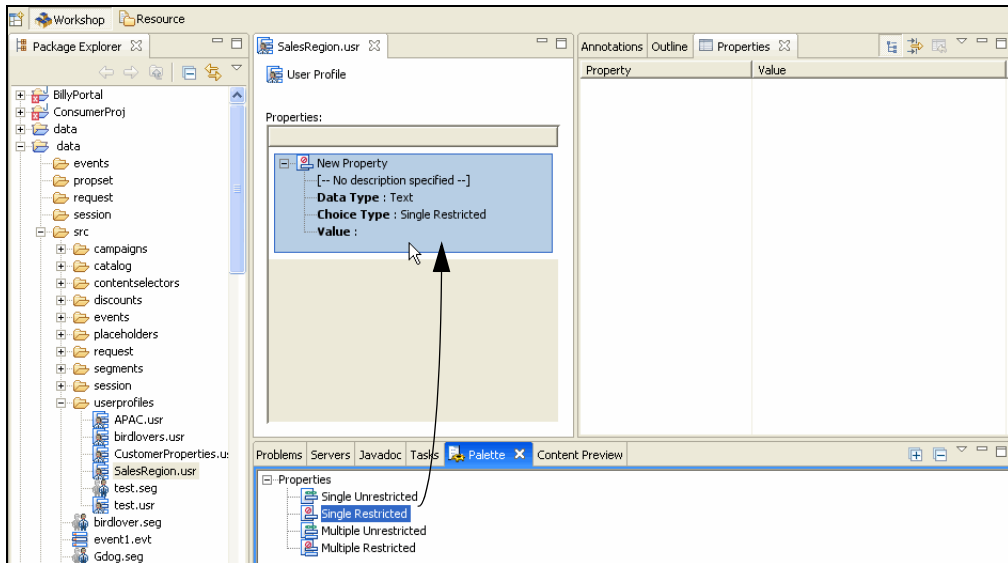
Adding Properties or Conditions to a Property Set

After you create a property set for a User Profile, HTTP Session or Request, Community, or WSRP, add the properties you want to it. The following steps assume you have created a property set according to the instructions in [“Setting up a Property Set”](#) on page 4-2.

Perform the following steps to add a property to a property set:

1. In Workshop for WebLogic, open the property set file you created in [“Setting up a Property Set”](#) on page 4-2.
2. In the **Palette** tab, drag one of property types into the Property Set Editor window, as shown in [Figure 4-1](#).

Figure 4-1 A Single Restricted Property Lets You Pick a Value From a List that You Define



The type defines the number of values that can be entered for the property. Following are descriptions of each type.

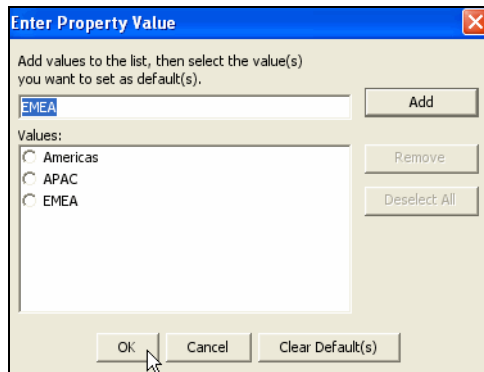
- **Single Unrestricted** – A single unrestricted property can have only one value, but you can enter any value. For example, **date of birth**.
- **Single Restricted** – A single restricted property can have only one value, and you are restricted to selecting that value from a predefined list. For example, a **Gender** property can be **Male** or **Female**.
- **Multiple Unrestricted** – A multiple unrestricted property can have multiple values, and you can enter any value. Generally, this property is used for a single category, such as **names of your children**. Other examples are a **favorite color** or **preferred browser**.
- **Multiple Restricted** – A multiple restricted property can have multiple values, and you are restricted to selecting the values from a predefined list. Other examples are **city** and **state**. You could also use a **check all that apply** type data.

3. Select the **Properties** tab and enter the following:

- Click the drop-down list and select the **Data Type** for the property value. For example, if you select **Boolean**, your property's **Values** can be only **true** or **false**. (Properties with a Boolean data type are automatically set to **Single Restricted**.)

- Enter a **Description** and a **Property Name**.
- In the **Selection Mode** and **Value Range** fields, you can change the property type. For example, you can change a property from **Single Unrestricted** to **Multiple Restricted**.
Note: Any change to the **Data Type**, **Selection Mode**, or **Value Range** fields removes anything previously entered in the **Values** field.
- Use the **Values** field to enter values for **Restricted** types or to set the default value for **Unrestricted** types. Click the ellipsis icon (...) to enter values. (In the Enter Property Value dialog box that appears for a **Restricted** type, enter a value and click **Add** after each entry. Click **OK** after you enter all values.) See [Figure 4-2](#).

Figure 4-2 This Property Has Three Possible Values



4. Save the properties by choosing **File > Save**.

WARNING: You can also use the Property control to create and manage properties. However, properties created with this control do not appear in the WebLogic Portal Administration Console. The properties must be modified and updated programmatically.

Modifying Properties and Conditions

You can edit properties and values in Workshop for WebLogic or you can edit just the property values in the WebLogic Portal Administration Console.

This section contains the following topics:

- [Editing Properties](#)

- [Editing Property Values](#)
- [Retrieving Properties from External Data Stores](#)

Editing Properties

Perform the following steps in Workshop for WebLogic to modify properties or conditions and their values for User Profiles, User Segments, HTTP session or request data, date and time conditions, or events:

1. In the Workshop Perspective in Workshop for WebLogic, double-click the property set file.
2. In the **Properties** tab, change the property or its default value.
3. Save your change by choosing **File > Save**.

You can also use the `<profile:setProperty>` JSP tag in your JSPs or the Property control in your Page Flows to modify existing property values for users.

Editing Property Values

Portal administrators can use the WebLogic Portal Administration Console to modify a property's value. For instructions, see [“Editing a Property Value” on page 11-2](#).

Retrieving Properties from External Data Stores

If you created a Unified User Profile (UUP) to access external user or group properties, you can use those properties to define rules for Personalization, Delegated Administration, or Visitor Entitlement.

After you create a UUP to access the properties stored in an external user store, such as an LDAP server, you can access those external properties only through WebLogic Portal's JSP tags, controls, or API. After you deploy a UUP, when the Administration Console focuses on the associated property set, it will call on the UUP to read values. A UUP must implement writable interfaces if you want to be able to write to the UUP properties.

If you want to use those external properties in defining rules for Personalization, Delegated Administration, or Visitor Entitlement, you must retrieve those properties in the WebLogic Portal Administration Console. Simply defining rules requires access only to the property set “schema”, which you created in [“Creating a User Profile Property Set” on page 4-3](#). When the rules are evaluated, the actual values are fetched. You do not have to use the Administration Console.

Note: The properties you retrieve from an external user store may be read-only, and their values cannot be updated in the WebLogic Portal Administration Console. To make those properties writable, your custom UUP must implement writable properties.

A password is not considered a property.

Perform the following steps to retrieve user or group properties from an external data store:

1. Create a UUP for the external data store. See the *User Management Guide* for instructions.
2. To create the User Profile property set for the external data store, locate the name of the property set to create. In your enterprise application root directory, inside the `p13n_app.jar` file, open the `/META-INF/ejb-jar.xml` file.
3. In the `<!-- User Profile Manager -->` section of the `ejb-jar.xml` file, locate the name entry for your external store, such as:

```
<env-entry-name>PropertyMapping/ldap</env-entry-name>
or
<env-entry-name>PropertyMapping/MyExternalStore</env-entry-name>
```

The name following `PropertyMapping/` is the name you must give to the new property set. For example, in the two examples shown, the property set is named `ldap.usr` or `MyExternalStore.usr`. Property set names are case sensitive. If you are using the LDAP UUP provided by WebLogic Portal, name the property set **ldap.usr**.

4. Add properties to the property set that exactly match the property names in the external store you want to surface.
5. Save the property set file.

Deleting a Property or a Property Set

You can delete individual properties from a property set, and you can delete an entire property set.

Perform the following steps to delete a property from a property set:

1. In the Portal Perspective in Workshop for WebLogic, double-click the property set file.
2. In the Editor window, select the property. For example, a User Segment could contain the condition *The visitor has specific characteristics*.
3. Right-click the property and choose **Delete**.

Perform the following steps to delete a property set:

1. In the Portal Perspective in Workshop for WebLogic, double-click the property set file.

Creating a Property Set

2. Right-click the property set and choose **Delete**.

You can also use the `<profile:removeProperty>` JSP tag in your JSPs or the Property control in your Page Flow to remove existing property values.

Creating a User Segment

You can use User Segments to dynamically group users based on conditions you define. Instead of creating groups of users, you can create groups of characteristics, such as gender, browser type, and date or time. If a user matches the characteristics, the user automatically and dynamically becomes a member of that User Segment. You can then target these groups with web content, automatic e-mails, and discounts based on the User Segment.

You can define User Segment conditions that identify gender, occupation, movie fans, or pet lovers. For example, you could classify all users who ordered more than five on-demand movies in the last 30 days. After you identify and group users into User Segments, you can target segments with personalized actions through Campaigns and Content Selectors.

Developers can use Workshop for WebLogic to create User Segments. Portal administrators can use the WebLogic Portal Administration Console to change the User Segment conditions to dynamically group users. Developer time is not required to update User Segments.

This chapter includes the following sections:

- [Creating a User Segment](#)
- [Modifying a User Segment](#)

Creating a User Segment

You can target visitors with web content, automatic e-mails, and discounts by defining and using groups called User Segments (as in segments of a population). Instead of being groups of hard-coded users, User Segments are groupings of characteristics, such as gender, the type of browser being used, and date or time information. If users match the characteristics, they are

automatically and dynamically members of that User Segment. When you use User Segments in Content Selectors and Campaigns, users that belong to those User Segments are targeted with the web content, e-mail, or discounts that you determine.

Note: The steps in this chapter refer to the `data\src` folder in the Package Explorer View. Your `data` and `src` directories might be named differently.

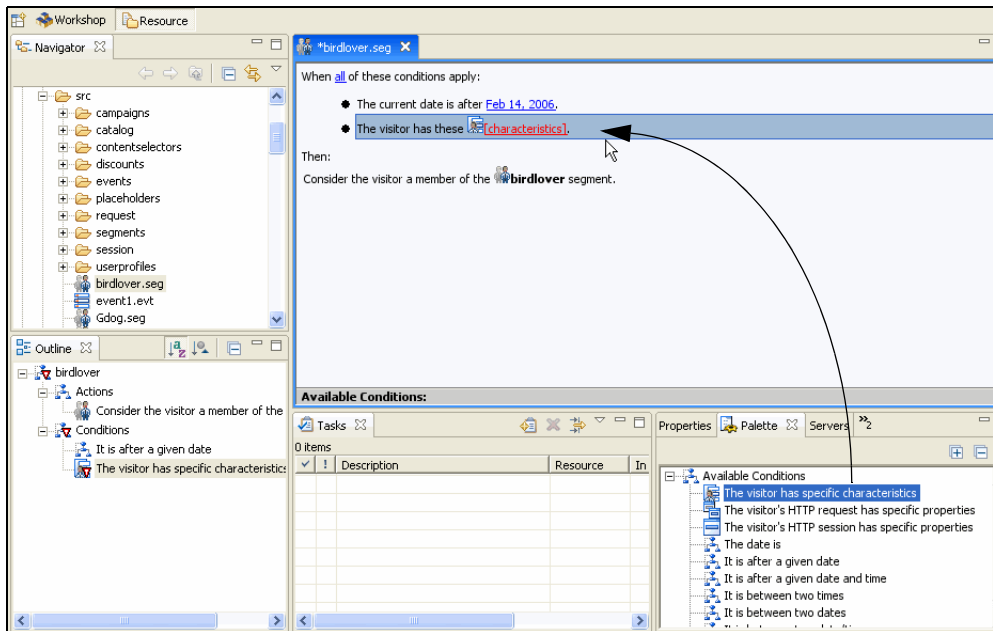
Perform the following steps to create a User Segment:

1. Start the WebLogic server by choosing **Run As > Run on Server** in Workshop for WebLogic. For instructions on configuring the WebLogic Server, see the [Portal Development Guide](#).
2. In the Portal Perspective, select the `<data>\src` folder in the Package Explorer View and choose **New > Other**.

Tip: You can customize the menu so that **User Segment** appears as a choice on the **New** menu. See the [Portal Development Guide](#) for instructions.

3. In the New Select a Wizard window, expand the **WebLogic Portal** folder.
4. Select **User Segment** and click **Next**.
5. In the New User Segment window, enter a name for the User Segment in the **File name** field. Keep the `.seg` file extension.
6. Click **Finish**. The User Segment Editor appears.
7. In the **Palette** tab, drag the conditions you want to use into the User Segment Editor. See [Figure 5-1](#).

Figure 5-1 Add a Condition to the User Segment



- For each condition you add to the User Segment, click the condition link to set the conditions.

For example, if you drag the condition *The visitor has specific characteristics* into the User Segment Editor, click the corresponding **characteristics** hyperlink in the User Segment Editor. The Visitor Characteristics window lets you select the User Profile properties and their values that will make a user a member of the User Segment. See [Table 4-1](#).

Properties for the user, HTTP session, and HTTP request conditions are based on the User Profile, HTTP session, and HTTP request properties you created. For more information on conditions, see [“Understanding Conditions” on page 2-4](#).

- Save the file by choosing **File > Save**.

Note: You can use User Segments in Content Selectors and Campaigns.

This section contains the following topic:

- [Setting Dates and Times](#)

Setting Dates and Times

When you set date and time conditions, the dates and times represent the time in your region. For example, if you are creating a Campaign action that will be triggered at 8 p.m., that means 8 p.m. in your region. For a time zone that is two hours behind you, the action will be triggered at 6 p.m. in that time zone.

The time zone also affects dates you set. The date you set becomes effective at midnight in your time zone. In a time zone that is six hours ahead of yours, that date becomes effective for that time zone at 6 p.m. your time the day before.

Time changes also affect time-triggered actions. For example, you created a Campaign that begins October 1 at noon and ends October 31 at noon. If a change to standard time (one hour earlier) occurs on October 29, the Campaign will actually end on October 31 at 11 a.m. So if you want the Campaign to end at noon on the new standard time, set the end time to 1 p.m.

Tip: Because of the different dates and times on which actions will be triggered around the country or world, it is important to tell users that dates and times are effective for your time zone. This type of information allows users to calculate when in their time zone they can take advantage of your promotions.

Modifying a User Segment

After you create a User Segment in Workshop for WebLogic, you can edit a User Segment using the following methods:

- **Workshop for WebLogic** – Developers can use Workshop for WebLogic to modify a User Segment. See [Chapter 12, “Modifying a User Segment”](#) for instructions.
- **BEA WebLogic Portal Administration Console** – Portal administrators can use the Administration Console to edit a User Segment’s value. See [Chapter 12, “Modifying a User Segment”](#) for instructions.

Creating a Content Selector

Content Selectors use rules to target specific groups of people with content items from BEA's Virtual Content Repository. Content Selectors return and display content. For example, if a user logs in and is identified in the User Profile as a **book fan**, a Content Selector can display a list of recommended books.

Developers can use Workshop for WebLogic to create and update Content Selectors and place them in a JSP. Portal administrators use the Administration Console to make changes to the Content Selectors that display content in your portal.

Users do not have to be authenticated (logged in) to be targeted by Content Selectors. For example, the presence of specific HTTP request or session properties, or specific date and time conditions can trigger content to be displayed. You can, for example, display targeted holiday content to users who visit your portal in the month of December and view your portal with a specific type of web browser.

However, authenticated users (as well as anonymous tracked users) have profile information associated with them that you can use to target them with personalized content. A **book fan** is an example of this type of User Profile data.

Note: The steps in this chapter refer to the `data\src` folder in the Package Explorer View. Your `data` and `src` directories might be named differently.

This chapter includes the following sections:

- [Setting Up Content to Display](#)
- [Creating a Content Selector](#)

- [Using the <pz:div> Tag Instead of a Content Selector](#)
- [Deleting a Content Selector Query](#)

Setting Up Content to Display

Content Selectors retrieve content properties, which are usually text or numeric values. For example, you want your Content Selectors to retrieve information about books. Book content can be assigned many properties, such as **title**, **author**, **publication date**, **ISBN**, and a **URL** to the publisher's web site. You may want your Content Selector to display a bulleted list of titles and link each title to the publisher's web site. This list and link is accomplished through using just the text values of the content properties.

However, your book content could also have a binary property that stores an image of the book's cover. The Content Selector can also display that binary property—the actual image—using the `ShowProperty` servlet, as described in [“Creating a Placeholder File” on page 7-6](#). The type of binary content that Content Selectors can display is dependent on the MIME types you have configured. By default, WebLogic Portal provides MIME support for displaying images and other types of files. See [“Displaying Additional MIME Types in a Placeholder” on page 7-3](#) for more information on other types.

This section includes the following topic:

- [Displaying Content for Other MIME Types](#)

Displaying Content for Other MIME Types

To display content for other MIME types, Content Selectors 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 `` tag that a browser requires for images. By default, **ad** Placeholders can generate the appropriate HTML only for the following MIME types:

- **Images** – A Placeholder generates an `` 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 content properties in the Virtual Content Repository.
- **Other types**– A Placeholder passes the text directly to the JSP.

WARNING: The `<EMBED>` tags do not always work correctly in all browsers. The behavior depends on the plug-ins you have configured.

If you are familiar with basic Java programming, you can write classes that enable Content Selectors to generate HTML for additional MIME types.

To support additional MIME types, you must complete the following tasks:

1. Create and compile a Java class to generate HTML.
2. Register the new class.

Creating and Compiling a Java Class to Generate HTML

To generate the HTML that the browser requires to display the MIME type, use Workshop for WebLogic to create a Java project in your application, and then create and compile a Java class in that Java project that implements the `com.bea.pl3n.ad.AdContentProvider` interface. For information on this interface, see the [Javadoc](#).

Registering the New Class

After you save the class in a directory that is in your `CLASSPATH`, you must notify WebLogic Portal Administration Console of its existence by performing the following steps:

1. Stop the server.
2. Create a backup copy of your application's `META-INF\wps-config.xml` file.
3. Open the `wps-config.xml` file in a text editor and find the `<AdService>` element.
4. Add the following code as a sub-element of the `<AdService>` element:

```
<AdContentProvider
Name="MIME-type"
Provider="YourClass.class"
Properties="optional-properties-for-your-class"
>
</AdContentProvider>
```

Provide the following values for the attributes of the `AdContentProvider` element:

- **Name** – The name of the MIME type that you want to support.
- **Provider** – The name of the compiled Java file. If you saved the file below a directory that your `CLASSPATH` environment variable names, you must include the file's path name, starting one directory level below the `CLASSPATH` directory.
- **Properties** – Any additional properties or parameters want to pass to your object. For example, if you added `<PORTAL_APP>/classes` to the system classpath, save your class to support AVI files as `<PORTAL_APP>/classes/myclasses/MimeAvi.class`.

The following code sample demonstrates AVI file support:

```
<AdContentProvider
Name="video/x-msvideo"
Provider="myclasses.MimeAvi"
Properties=" "
>
</AdContentProvider>
```

5. Save your modifications to the `wps-config.xml` file.
6. Restart the WebLogic Server.
7. Activate the content provider in the Administration Console by performing the following steps:
 - a. Start the WebLogic Portal Administration Console.
 - b. In the WebLogic Portal Administration Console, choose **Configuration Settings > Service Administration**.
 - c. In Resource Tree, select **Add/remove configurable item**.
 - d. In the list of configurable items that appears, select the check box next to the content provider you added.
 - e. Click **Update**.

Creating a Content Selector

Content Selectors are scoped to the enterprise application, so you can include a Content Selector in any JSP within the enterprise application.

A Content Selector consists of the following two parts:

1. **Content Selector file** – The Content Selector file that you create in Workshop for WebLogic contains the following parts:
 - a. Personalization rules – The conditions that define when the Content Selector runs its query and displays the content. The following conditions can be used to trigger a Content Selector:
 - The visitor is a member of a predefined User Segment
 - The visitor has specific characteristics (User Profile properties)
 - Specific HTTP session or request properties exist

- Date and time conditions
- b. A content query that retrieves a specific set of content properties.
2. **The <pz:contentSelector> JSP tag** – This companion tag performs the processing and is also created in Workshop for WebLogic. Content returned from Content Selectors is usually displayed in portlets. When a user views a portlet or a portal desktop that contains a Content Selector, the Content Selector's rules and logic look for a match of properties, such as User Profile information. If the properties match the Content Selector rules, the Content Selector runs a query and retrieves and displays all content matching the query.

Tip: During development, the rules files reload when they change (just like JSPs), so you can quickly develop with Content Selectors. However, when the server is in production mode, Content Selectors are loaded into the database (from the file-based definitions in the application) where they can be modified in the WebLogic Portal Administration Console without redeploying the application or restarting the server.

Use the following guidelines when you create Content Selectors.

- Create a Content Selector for each audience you want to target. If you have five audiences you want to target with content, create five Content Selectors, then add five <pz:contentSelector> tags to your JSP, each of which references its own Content Selector file.
- When a Content Selector is triggered and runs its query, the results (node objects) are returned to the <pz:contentSelector> JSP tag and stored in the tag's **id** attribute. At this point, you need other tags or code to process and display the content. The tags described in [Table 6-4](#) are useful tags for displaying retrieved content.
- To display binary content retrieved by Content Selectors, use the ShowProperty servlet. The following code example desensitizes the ShowProperty servlet:

```
<pz:contentSelector rule="classic" id="nodes" />
<utility:NotNull item="<%=nodes%>">
<ul>
<utility:forEachInArray array="<%=nodes%>" id="node"
type="com.bea.content.Node">
">
</utility:forEachInArray>
</ul>
</utility:NotNull>
```

The `` HTML tag's source path is constructed to use the path to the content item in the Virtual Content Repository. The path includes the `ShowProperty` servlet to render the binary file. In the `node.getPath()` method, `node` is a specific content item stored by the `<utility:forEachInArray>` tag's `id` attribute. If the `<utility:forEachInArray>` `id` attribute value was `foo`, the method would look similar to the following: `foo.getPath()`.

This section contains the following topics:

- [Creating the Content Selector File](#)
- [Using a JSP Tag to Display a Content Selector File](#)

Creating the Content Selector File

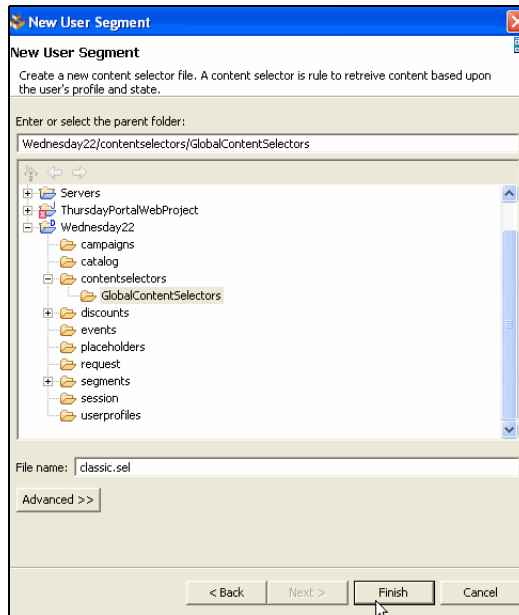
Perform the following steps to create a Content Selector file:

1. Start the WebLogic server by choosing **Run As > Run on Server** in Workshop for WebLogic. For instructions on configuring the WebLogic Server, see the [Portal Development Guide](#).
2. In the Portal Perspective, right-click the `data\src\contentselectors\GlobalContentSelectors` folder in the Package Explorer View and choose **New > Other**.

Tip: You can customize the menu so that **Content Selector** appears as a choice on the **New** menu. See the [Portal Development Guide](#) for instructions.

3. In the New Select a Wizard window, select **Content Selector** in the WebLogic Portal folder and click **Next**.
4. In the New Content Selector File window, enter a name for the Content Selector in the **File name** field. Keep the `.sel` file extension, as shown in [Figure 6-1](#).

Figure 6-1 Create a New Content Selector

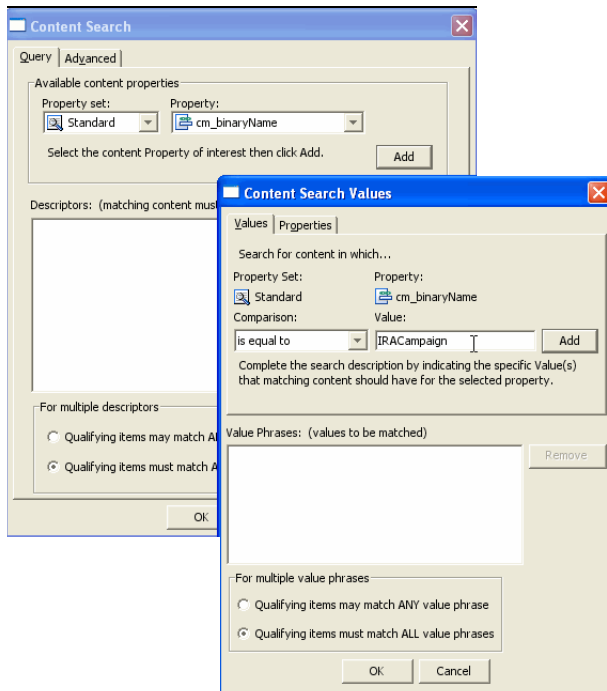


5. Click **Finish**. The Content Selector Editor appears.
6. Select the **Properties** tab to change the display **Name** for the Content Selector.
7. Select the **Palette** tab to see the Available Conditions under which the Content Selector will run. As you select conditions, corresponding links appear in the top of the Editor window.
8. In the Content Selector Editor window, click the corresponding links to create the conditions you selected, and enter the appropriate information.
9. In the Content Selector Editor window, click the query's **empty content search** link to define the query. This requires a connection to BEA's Virtual Content Repository, which is set up in the WebLogic Portal Administration Console.
10. You can define the query in advanced mode using WebLogic Portal's expression syntax (on the **Advanced** tab) or in graphical mode (on the **Query** tab):
 - **Advanced mode** – In the Content Search window, select the **Advanced** tab and build a query using the instructions in [“Building a Content Query with Expressions”](#) on page 6-10. The **Advanced** tab provides code coloring to highlight context errors in your queries. You can reuse existing queries by pasting them into the **Advanced** tab.

- **Graphical mode** – Use the following steps in the **Query** tab to build a content query by selecting content properties, comparators, and values to retrieve content items.
 - a. In the Content Search window, select the **Query** tab.
 - b. Select a property set and a property within the content type, and click **Add**.

Note: The properties you select are content properties (types) rather than property set properties such as user profile or session properties.
 - c. In the Content Search Values window that appears, use one of the following tabs:
 - **Values** tab – To define the query based on a comparison to a value you enter. For example, the query could be set to retrieve content with an **investorRiskLevel** property that is marked as **high**. You could also retrieve binary content with a name of **IRACampaign**. See [Figure 6-2](#) for an example and [Table 6-1](#) for more details.

Figure 6-2 A Content Query that Retrieves a Binary File Called IRACampaign



- **Properties** tab – To define the content query based on the property value that is dynamically fed in from another type of property, such as a user profile property. For

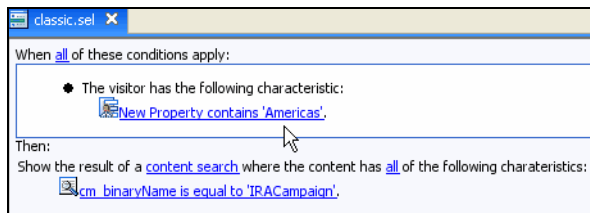
example, instead of creating a query based on static content properties, you can create a query that reads in the value of the current user's **investorRiskLevel** to populate the query. The query would be different for each user.

- d. Click **Add**. The query descriptor is added in the Content Search window.
 - e. You can add more value phrases to the query, then set the appropriate option in the **For multiple descriptors** area at the bottom of the window.
11. Click **OK** in the Content Search window to add the query to the search.
 12. You can preview the content that will be retrieved by the query by clicking the **Content Preview** tab. If you defined the query to use values from a User Profile property, the retrieved content will be different for each user, so you must enter the username of an existing user in the **Preview User** field to see which content will be retrieved for that user.
 13. Save the Content Selector file by choosing **File > Save**.
 14. To use the Content Selector, add the `<pz:contentSelector>` tag to the relevant JSPs. See [“Using a JSP Tag to Display a Content Selector File” on page 6-23](#) for more information on using the `<pz:contentSelector>` tag to display text content (including text binaries, such as HTML files) and non-text binaries (graphics). See the [Content Management Guide](#) to learn how to modify an existing Content Selector to display binary content.

Tip: Content Selectors can display binary content, such as images. See the [Content Management Guide](#) for instructions.

Figure 6-3 shows a Content Selector file called `classic.sel`. The condition that triggers the Content Selector is a user with a property set of **SalesRegion** with a value of **Americas**. The content that is retrieved is any content with a binary file name that contains `IRACampaign`.

Figure 6-3 Content Selector File in Workshop for WebLogic



Tip: When library services (for example, versioning and workflow status) are enabled for a BEA repository, system properties are always available (published) to queries unless the content item has a retired status. If you want to retrieve content based on its content type, you must use the **cm_objectClass** system property in your content query. If your queries use only system properties, the query retrieves all content items with matching system properties that are not retired.

Users do not have to be authenticated to be targeted by Content Selectors.

Building a Content Query with Expressions

You can use the WebLogic Portal Expression language in WebLogic Portal Administration Console to build content queries. You can build queries with the Advanced Query window from content-related JSP tags and when you are creating queries for Placeholders, Campaigns, and Content Selectors.

The query compares a content property to a value that you enter. A query contains the following three parts: <property><comparator><value>.

If the comparison is true, all matching content items are retrieved.

For example:

<pre>employee_type == 'manager'</pre>	<p>This query retrieves any content item from the BEA Virtual Content Repository with a property called employee_type that contains the exact (==) String manager.</p>
---------------------------------------	---

Queries are often made up of multiple clauses using **and** (&&) and **or** (| |) logic. For example:

<pre>(genre == 'rock' genre == 'alternative') && platinum_records > 2</pre>	<p>This query retrieves any content item with a genre property that has an exact value of rock or () alternative and (&&) with a platinum_records property value greater than 2. The parentheses separates one section of the query from the next, controlling the order of evaluation.</p>
<pre>(genre=='rock' genre=='country') && artist=='dead'</pre>	<p>This query retrieves any content item with a genre property that has an exact value of rock or country and an artist property value of dead.</p>

This section contains the following topics:

- [Using Rules to Build a Query](#)
- [Selecting Properties](#)
- [Using Comparators](#)
- [Supplying Values](#)

Using Rules to Build a Query

Use the following rules when you build a query:

- Queries are case sensitive.
- Queries can be simple, one-clause queries, or they can be more complex, multi-clause queries. In complex queries, parentheses control the order of evaluation for multiple query clauses, and clauses are evaluated using **and** (&&) and **or** (| |) logic.
- In a Placeholder, only one content item can be displayed at a time. If you are using a Placeholder to run queries from a Campaign (as well as its own default queries), you can run several queries, but only one query at a time. The query is determined by the Placeholder's weighting system. After the query runs, the query can return multiple content items, but the Placeholder displays only one of the retrieved content items.

Tip: With your development server running, create a test Placeholder or Content Selector in Workshop for WebLogic and construct queries for it using the **Advanced** tab in the Content Search window. The retrieved content is displayed in the **Content Preview** tab. If you utilize the `userProperty()` format to retrieve values for your query, you must enter a user name in the **Preview User** field of the **Content Preview** tab to retrieve the values for that user and display the retrieved content items.

Selecting Properties

The following two types of properties exist for content:

- User-defined content properties.
- Explicit (system) content properties. See [Table 6-1](#) for a list of explicit content properties.

You can use both properties in a query.

User-defined content properties are stored in sets called *types*, which are defined in the WebLogic Portal Administration Console. For example, you can create a type called **Book** and add properties, such as **title**, **author**, **pub_date**, and **isbn**.

Explicit content properties are automatically associated with all content items and are automatically assigned values for a content item when the content item is added to the content repository. These properties, which are also listed in the

`com.bea.content.expression.Search` class, include the items listed in [Table 6-1](#).

Table 6-1 Explicit Content Properties

Property	API	Description
cm_uid	<code>Node.id.uid</code>	The unique ID for a content item. You can view a content item's unique ID by selecting the content item in the Administration Console and viewing its description.
cm_createdDate	<code>Node.createDate</code>	The date on which a content item was created. You can view Creation Date information for content items by selecting their content folder in the Administration Console and selecting the Summary tab.
cm_createdBy	<code>Node.createdBy</code>	The user who created the content item. You can view Created By information for content items by selecting the content folder in the Administration Console and selecting the Summary tab.
cm_modifiedDate	<code>Node.modifiedDate</code>	The date the content item was last modified. You can view Modified Date information for content items by selecting their content folder in the Administration Console and selecting the Summary tab.
cm_nodeName	<code>Node.name</code>	The name of the content item, as shown in the Administration Console.

Table 6-1 Explicit Content Properties (Continued)

cm_path	<code>Node.path</code>	The Virtual Content Repository path to the content item. For example, <code>/BEARepository/juvenilebooks/TheCrazyAdventure</code> .
cm_isHierarchy	<code>Node.type == Node.HIERARCHY</code>	Identifies a content folder rather than a content item. Content folders can contain content items and child content folders. When using this property in queries, compare it using a Boolean value of <code>true</code> or <code>false</code> .
cm_isContent	<code>Node.type == Node.CONTENT</code>	Identifies a content item rather than a content folder. Content items contain properties from the type with which they are associated. When using this property in queries, compare it using a Boolean value of <code>true</code> or <code>false</code> .
cm_objectClass	<code>Node.objectClass.name</code>	The content type associated with a content item. You can view Type information for content items by selecting the Types tab in the Administration Console and selecting the Summary tab.
cm_objectClassInstance	The instance of an object class	Finds all instances of a given object class and all of its children.
cm_value	The value	A metadata search for any property with a given value. This property is similar to a wildcard search.
cm_contentType	<code>BinaryValue.contentType</code>	The MIME type for content item binary properties. For example, <code>image/jpeg</code> . You can view the MIME type of a content item by selecting it in the Administration Console and looking at the Primary Property Data Type field.

Table 6-1 Explicit Content Properties (Continued)

cm_binarySize	BinaryValue.size (For Node Properties)	<p>The size of the binary value of content items. You can view the size of a content item's binary value by selecting the content item in the Administration Console, selecting the Properties tab, and clicking Download File. Click Save and right-click the displayed binary file to view the file properties.</p> <p>When you use this property in a query, specify binary size in bytes.</p>
cm_binaryName	BinaryValue.name (For Node Properties)	<p>The file name of the binary value of a content item. You can view the name of a content item's binary value by selecting the content item in the Administration Console and viewing the Name value in the Summary tab.</p>

When you construct a query, you can specify property names by themselves without quotes. For example: **title** <comparator> <value>. However, if a property name contains spaces, double quotes, or dashes, you must enclose the property name within a `toProperty()` format. For example: **toProperty('Favorite Author')** <comparator> <value>.

The query looks for the property name in all content types and returns any content item with that property value (if the content item meets the conditions of the query). However, you can also isolate a property within a specific type. For example, if you have a **books** type and an **articles** type that both contain a **title** property, you can retrieve content items with a specific **title** from within only the **book** type using the **cm_objectClass** property.

For example: `title likeignorecase '*Adventure' && cm_objectClass = 'books'`

Using Comparators

Comparators provide the logic that compares a query's property to the value you enter. If a content item meets the conditions of the query, the content item is returned. [Table 6-2](#) contains a list of the comparators you can use in your queries.

Table 6-2 Available Comparators

Comparator	Description (Property Formats on Which the Comparator Can Act)
= or ==	Checks to see if a single-value property (including a single value containing a list) is exactly equal to the case-sensitive value you enter (Boolean, date and time, numeric, and string values). For example, <code>genre == 'fantasy'</code> retrieves any content item that has case-sensitive <code>fantasy</code> as the value for the genre property.
!=	Checks to see if a single-value property (including a single value containing a list) is not equal to the case-sensitive value you enter (Boolean, date and time, numeric, and string values). For example, <code>genre != 'mystery'</code> retrieves any content item that does not have case-sensitive <code>mystery</code> as the value for the genre property.
>	Checks to see if a single-value property is greater than the value you specify (date or time and numeric values). For example, <code>pub_date > toDate('MM-dd-yyyy', '01-01-2000')</code> retrieves any content item with a pub_date property set to a value later than January 1, 2006.
<	Checks to see if a single-value property is less than the value you specify (date or time and numeric values). For example, <code>books in series < 3</code> . Because the property name contains spaces, you must use <code>toProperty()</code> , as shown in <code>toProperty('books in series') < 3</code> to retrieve any content item with a value less than 3 for the books in series property.
>=	Checks to see if a single-value property is greater than or equal to the value you specify (date or time and numeric values). For example, <code>toProperty('books in series') >= 3</code> retrieves any content item with a value greater than or equal to 3 for the books in series property. The property value of the sample content item is 1.
<=	Checks to see if a single-value property is less than or equal to the value you specify (date or time and numeric values). For example, <code>toProperty('books in series') <= 3</code> retrieves any content item with a value less than or equal to 3 for the books in series property.

Table 6-2 Available Comparators (Continued)

like	<p>Checks to see if a single-value property is like the case-sensitive value you enter. You can use wildcard characters * (one or more characters) or ? (single character). For example, <code>author like 'P?nm*' </code> retrieves any content that contains case-sensitive Penman, Panmen, or any other variation with a different character between the P and the n and any characters after the m for the author property.</p> <p>Note: If you do not put the asterisk (*) at the end, the content item is not retrieved, because more text follows the name Penman in the property value.</p>
likeignorecase	<p>Checks to see if a single-value property is like the value you enter. You can use the wildcard characters * and ?. Character case is ignored. For example, <code>author likeignorecase 'pen*' </code> retrieves any content with an author value that begins with pen in any case combination, such as penman, Penman, Penfield, and so on.</p>
contains	<p>Checks to see if a multi-value property contains exactly the single value you specify (date or time, numeric, and string values). In some implementations, this comparator may also work against single-value properties. For example, <code>genre contains 'fantasy' </code> retrieves any content containing an genre property value of exactly <code>fantasy</code>.</p>
containsall	<p>Checks to see if a multi-value property contains all of the exact values you specify (date or time, numeric, and string values). For example, <code>genre containsall ('fantasy', 'children')</code> retrieves any content that contains genre property values of <code>fantasy</code> and <code>children</code>. In some implementations, this comparator can also work against single-value properties.</p>
containsany	<p>Checks to see if a multi-value property contains any of the exact values you specify (date or time, numeric, and string values). For example, <code>genre containsany ('fantasy', 'children', 'scifi')</code> retrieves any content that contains genre property values of <code>fantasy</code>, <code>children</code>, or <code>scifi</code>. In some implementations, this comparator can also work against single-value properties.</p>

Table 6-2 Available Comparators (Continued)

in	Checks to see if a single-value property contains any of the values you enter. If the value you enter is not a list of possible values (is a single value), in is the same as = or == (date or time, numeric, and string values). For example, <code>isbn in ('pending', 'not_available')</code> retrieves any content that contains the isbn property value of <code>pending</code> or <code>not_available</code> .
-----------	--

Supplying Values

Values represent the content you want the query to return. By supplying values to a query, you are telling the query which content to retrieve or ignore based on the values stored on the content items.

For example, if you have a content type called **book** with a property called **title**, all content items you associate with the **book** type have a **title** field. Each **title** value is unique, so in a query, you can enter the unique value you want, resulting in the query retrieving a specific content item (or ignoring the content item, depending on the comparator you use).

You can enter values in two ways:

1. **Hard code values in the queries** – Use hard-coded values so that you get predictability with your queries. Hard-coded values let you pinpoint the specific content you want to retrieve.
2. **Get values from User Profiles and other types of property sets (Session and Request)** – When you populate values from User Profile, session, or request property sets, the value in each specified User Profile, session, or request property is inserted programmatically into the query, letting you create personalized queries based on the current user's preferences or the current session or request.

For example, in a query where a content property is **author**, you can get the value of the user's **FavoriteAuthor** profile property to supply the value for the query, letting the query retrieve content associated with the user's favorite author.

Use the following guidelines to build values in queries:

- Enclose string literal values in single quotes. For example: `'pending'`.
- To supply a single or double quote, use a backslash for the quoting character (unicode characters are not supported). For example, if a book title is stored with double quotes, such as `"The Crazy Adventure"`, enter the value like this: `'\"The Crazy Adventure\"'`. If the title is stored with single quotes, enter the value like this: `title = \'The Crazy Adventure\''`.

- Unicode (such as `"\u6565"`), octal (such as `"\7"`, `"\65"`, `"\377"`), and standard Java escape sequences (such as `"\n"`, `"\r"`, `"\b"`) are allowed in the string literals.
- Boolean literals are either the `true` or `false` keyword (lowercase, without quotes).
- Number literals are Java form (scientific notation is supported).

Date or time literals are presented in `toDate('formatStr', 'dateStr')` format, where `formatStr` is the date and time format you want to use (such as `'MM/dd/YYYY'`) and `dateStr` is the actual date and time you enter (such as `'01/01/2006'`).

The `formatStr` must be a valid `java.text.SimpleDateFormat` string. If you omit the `formatStr`, the `toDate()` expects the date and time you enter to be in the following format: `'MM/dd/yyyy HH:mm:ss z'` (where `z` is the time zone, such as `MDT`). For example: `toDate('12/01/2004 06:00:00 MDT')`.

To specify date values only, enter the format you want for the `formatStr`. For example, for a date value only, specify `toDate('MM/dd/yyyy', '12/01/2004')`. You can also specify only the month and year, such as `'MM-yyyy'`.

Use the `now` keyword to specify the time at which the expression is being parsed at run time.

- To supply query values from user, request, or session properties, use the following format: `<type>Property(<propertyset>, <propertyname>)`, where the type is **user**, **request**, or **session**. For example: `userProperty('userpreferences', 'FavoriteAuthor')`.

Creating Complex Queries

You can combine multiple independent query clauses, tying them together with **and** (`&&`) and **or** (`||`) logic and controlling the order of evaluation with parentheses the way you would with algebraic expressions. This lets you create more complex queries. You can also include **not** logic (!) in complex queries by using the exclamation point in front of a parenthetical grouping.

See the [Content Management Guide](#) for more information about queries.

Using Sample Queries

Figure 6-4 shows the properties set on a book stored in BEA's Virtual Content Repository.

Figure 6-4 Properties on a Book

The screenshot shows a web interface for 'TheCrazyAdventure' with a 'Properties' tab selected. A red asterisk indicates a required property. The 'Primary Property' section contains a table with one row for a file named 'crazyadventure.jpg'. The 'Other Properties' section contains a table with the following data:

Property	Current Value(s)	Description	Edit	Clear Value(s)
author	Perman, Piper		<input type="checkbox"/>	<input type="checkbox"/>
title	The Crazy Adventure	Title of the book	<input type="checkbox"/>	<input type="checkbox"/>
isbn	pending	ISBN number of the book	<input type="checkbox"/>	<input type="checkbox"/>
keywords	warthogs, pixies, children	Keywords associated with book	<input type="checkbox"/>	<input type="checkbox"/>
pub_date	December 1, 2004 12:00:00 AM	Date the book was published	<input type="checkbox"/>	<input type="checkbox"/>
publisher	Self	Publisher of the book	<input type="checkbox"/>	<input type="checkbox"/>
books in series	1		<input type="checkbox"/>	<input type="checkbox"/>
genre	fantasy, children		<input type="checkbox"/>	<input type="checkbox"/>
will_visit_schools	true		<input type="checkbox"/>	<input type="checkbox"/>

The example queries in Table 6-3 use properties in the sample content item, and the description for each sample query tells you if the query will retrieve the content item.

Table 6-3 Example Queries (Continued)

Query	Will Retrieve Sample?	Description
<code>genre == 'fantasy'</code>	Yes	Retrieves any content item that has case-sensitive <i>fantasy</i> as the value for the genre property.

Creating a Content Selector

<code>will_visit_schools == true</code>	Yes	Retrieves any content item with the will_visit_schools property set to <code>true</code> . There are no single quotes around <code>true</code> , and <code>true</code> is lowercase.
<code>genre != 'mystery'</code>	Yes	Retrieves any content item that does not have case-sensitive <code>mystery</code> as the value for the genre property.
<code>pub_date > toDate('MM-dd-yyyy', '01-01-2005')</code>	Yes	Retrieves any content item with a pub_date property set to a value later than January 1, 2005.
<code>books in series < 3</code>	No	Because the property name contains spaces, you must use <code>toProperty()</code> , as shown in the next example.
<code>toProperty('books in series') < 3</code>	Yes	Retrieves any content item with a value less than 3 for the books in series property.
<code>toProperty('books in series') >= 3</code>	No	Retrieves any content item with value greater than or equal to 3 for the books in series property. The property value of the sample content item is 1.
<code>toProperty('books in series') <= 3</code>	Yes	Retrieves any content item with value less than or equal to 3 for the books in series property.
<code>author like 'P?nm*'</code>	Yes	Retrieves any content that contains case-sensitive Penman, Panmen, or any other variation with a different character between the <code>P</code> and the <code>n</code> and any characters after the <code>m</code> for the author property. Without an asterisk (*) at the end, the content item would <i>not</i> be retrieved, because more text follows the name Penman in the property value.

<code>author likeignorecase 'pen*'</code>	Yes	Retrieves any content with an author value that begins with pen in any case combination, such as penman, Penman, Penfield, and so on.
<code>genre contains 'fantasy'</code>	Yes	Retrieves any content containing an genre value of exactly fantasy.
<code>genre contains 'child*'</code>	No	The contains comparator does not allow wildcard characters.
<code>genre containsall ('fantasy', 'children')</code>	Yes	Retrieves any content that contains genre property values of fantasy and children.
<code>genre containsall ('fantasy', 'children', 'scifi')</code>	No	Retrieves any content that contains genre property values of fantasy, children, and scifi. The sample content item contains fantasy and children, but not scifi.
<code>genre containsany ('fantasy', 'children', 'scifi')</code>	Yes	Retrieves any content that contains genre property values of fantasy, children, or scifi.
<code>isbn in ('pending', 'not_available')</code>	Yes	Retrieves any content that contains the isbn property value of either pending or not_available.
Complex Queries		
<code>toProperty('books in series') >= 3 && pub_date > toDate('MM-yyyy', '1-2005')</code>	No	Retrieves books that are part of a trilogy that were published after January 2005.
<code>(genre contains 'children' keywords like '*children*') && will_visit_schools == true</code>	Yes	Retrieves books with a genre value set to children or has *children* in the keyword value; and whose author is available to visit schools.

Creating a Content Selector

```
((genre contains 'children' ||  
keywords like '*children*') &&  
will_visit_schools == true) &&  
isbn != 'pending'
```

No

Retrieves books with a **genre** value set to children or has **children** in the keyword value; and whose author is available to visit schools; and with an **isbn** value that does not equal `pending` (meaning the book is not yet published). The parenthetical nesting controls the order of evaluation.

```
(title likeignorecase  
'*adventure' || genre contains  
'fantasy') && (pub_date >=  
toDate('MM-yyyy', '01-2005')  
|| isbn == 'pending')
```

Yes

Retrieves books whose **title** contains **adventure** or whose **genre** contains *fantasy*; and whose **pub_date** is after January 2005 or whose **isbn** is still `pending` (not yet published).

```
(genre containsany  
userProperty('userpreferences'  
, 'BookGenre') && (keywords  
likeignorecase '*pixies')) ||  
author likeignorecase  
userProperty('userpreferences'  
, 'FavoriteAuthor')
```

Depends

Reads the User's Profile properties and uses specific property values to supply the values in the query. The query provides personalized content retrieval, because retrieved content is based on user preferences.

For example, if the current user has her **BookGenre** property set to `mystery` and her **FavoriteAuthor** set to `Penman, Piper`, this query will return the sample content. Even though the **BookGenre** value doesn't match in the first clause, the **FavoriteAuthor** in the `or (| |)` second clause does match.

If you are using the Property control or the `setProperty` JSP tag to set user property sets and properties programmatically (rather than creating property sets in Workshop for WebLogic), you can still use `userProperty()` in your queries.

Other Useful Queries (Not Related to the Sample Content)


```
language ==
userProperty( 'userpreferences'
, 'userPreferredLang' )
```

This approach lets you serve language-appropriate content to each user based on the user language preference (`userPreferredLang`) stored in a property set. You could also use `sessionProperty()` to get the language preference from the session; or you could use `requestProperty('DefaultRequestPropertySet', 'Locale')`, which returns the user's locale string, such as en-US.

```
((UserAge <= 35 && colors
contains 'red' ||
UserAge > 35 && !(colors
contains 'black')) &&
mimeType == 'text/html') &&
toProperty('Launch Date') <
now && !(expireDate >
toDate('MM-yyyy', '12-2004'))
```

This query uses **not** logic, as shown in the `!(colors contains 'black')` clause.

Using a JSP Tag to Display a Content Selector File

After you have created a Content Selector file (see [“Creating the Content Selector File” on page 6-6](#)), you must use the `<pz:contentSelector>` JSP tag to display the content. The following JSP tag uses the `classic` Content Selector shown in [Figure 6-3](#):

```
<pz:contentSelector rule="classic" id="nodes"/>
```

The JSP tag has two required attributes:

- The **rule** attribute – Contains the name of the Content Selector file (without the file extension). This attribute tells the JSP tag which personalization rules and query to use.
- The **id** attribute – When a Content Selector runs its query, the content is retrieved from the virtual content repository as an array of content properties. The **id** attribute, which is a String you enter when you set up the JSP tag, serves as a container that holds the array. At this point, you must use other JSP tags to handle the array and display the content. For details on what you can do with the array of properties to display the content, see [“Displaying Content for Other MIME Types” on page 6-2](#).

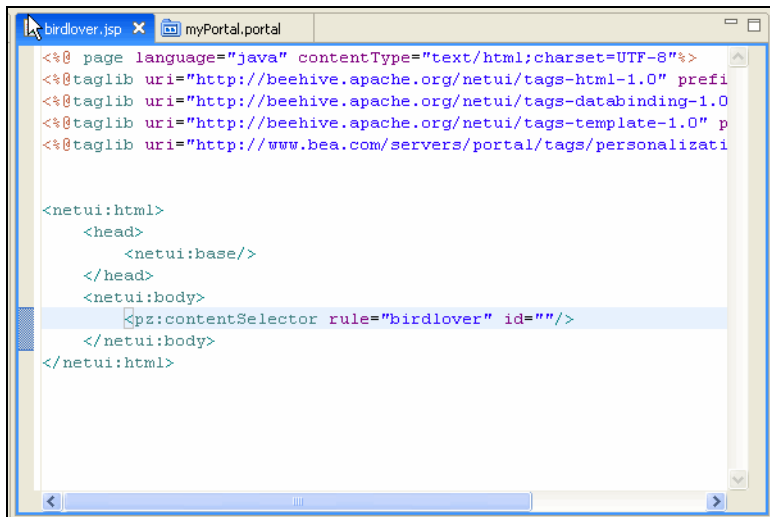
There are other attributes you can set on the `<pz:contentSelector>` tag. See the [Javadoc](#) for more information on the class.

Adding a Content Selector to a JSP

After you create a Content Selector file in Workshop for WebLogic, you can use any of the following three methods to add the Content Selector to a JSP in Workshop for WebLogic:

- Drag a Content Selector from the Workshop Perspective onto a page of an open portal file. See the instructions in [“Dragging a Content Selector to a Portal File”](#) on page 6-24.
- Open the JSP to which you want to add the Content Selector, and drag the Content Selector file from the Workshop Perspective into the JSP. The JSP tag is added automatically, the **rule** attribute is automatically populated with the name of the Content Selector, and the **id** attribute is included (without a value). The `include` statement for the tag library is automatically added. See [Figure 6-5](#).

Figure 6-5 Adding a Content Selector to a JSP in Workshop for WebLogic



- Drag the `<pz:contentSelector>` JSP tag from the JSP Design Palette window (in the Portal Personalization category) into an open JSP and populate the tag’s attributes manually. Get to the JSP Design Palette by choosing **Window > Show View > Other** and then select **Web** and **JSP Design Palette**. The `include` statement for the tag library is automatically added.

Dragging a Content Selector to a Portal File

Perform the following steps to drag a Content Selector to an open portal file:

1. In the Portal Perspective, locate a Content Selector file.
2. Drag a Content Selector from the Workshop Perspective onto a page of an open portal file. When you do this, three things occur:
 - a. The Portlet Wizard appears, letting you quickly create a portlet that will display the Placeholder.
 - b. The resulting portlet is automatically added to the portal page.
 - c. A JSP file is automatically created for the Content Selector.

The JSP file contains the Content Selector JSP tag with the **rule** and **id** attributes automatically populated, and the `include` statement for the tag library is automatically added. Other JSP and HTML tags are also added automatically, as shown in [Listing 6-1](#).

Listing 6-1 JSP File with Other JSP and HTML Tags

```
<pz:contentSelector rule="modern" id="nodes" />
<utility:NotNull item="<%=nodes%>">
<ul>
  <utility:forEachInArray array="<%=nodes%>" id="node"
    type="com.bea.content.Node">
    <li><cm:getProperty id="node" name="cm_nodeName"
      conversionType="html" /></li>
  </utility:forEachInArray>
</ul>
</utility:NotNull>
```

[Table 6-4](#) describes the JSP tags you can use with a Content Selector.

Table 6-4 JSP Tags for Content Selectors

JSP Tag	Description
<code><utility:NotNull></code>	Checks to see if the array actually contains content.
<code></code>	Provides a container for listing content items in a bulleted list.

Table 6-4 JSP Tags for Content Selectors

<code><utility:forEachInArray></code>	Iterates through the array and isolates each content item until all items in the array have been processed. Each item is given its own <code></code> bullet.
<code><cm:getProperty></code>	Takes each content item from <code><utility:forEachInArray></code> and designates which content property is going to be displayed.

Using More than One Content Selector

You can use multiple Content Selectors with conditional logic to get hierarchical personalized content, where you try to match the most specific to the least specific, or for mutually exclusive Content Selectors. [Listing 6-2](#) demonstrates how to use multiple Content Selectors for personalized content.

Listing 6-2 Using the `<pz:contentSelector>` JSP Tag for Multiple Content Selectors

```

<%@ taglib uri="content.tld" prefix="cm"%>
<%@ taglib uri="http://www.bea.com/servers/pl3n/tags/utility"
prefix="utility"%>
<%@ taglib uri="http://www.bea.com/servers/portal/tags/ad" prefix="ad"%>
<%@ taglib uri="http://www.bea.com/servers/portal/tags/personalization"
prefix="pz"%>
<pz:contentSelector rule="FemaleContent" id="nodes" sortBy="cm_nodeName
desc"/>
<% if (nodes == null || nodes.length <= 0) { %>
<pz:contentSelector rule="MaleContent" id="nodes" sortBy="cm_nodeName
desc"/>
<% }%>
<% if (nodes == null || nodes.length <= 0) { %>
    Sorry, you don't get a free lunch today.
<% }%>
    Found <%=nodes.length%> Node(s):
<dl>
<utility:forEachInArray array="<%=nodes%>" id="node"
type="com.bea.content.Node">

```

```
<dt><cm:getProperty id="node" name="title" conversionType="html" /></dt>
<dd><ad:render id="node" /></dd>
</utility:forEachInArray>
</dl>
```

Using the <pz:div> Tag Instead of a Content Selector

The <pz:div> JSP tag can provide in-line HTML Personalization. You can populate a JSP with sets of in-line content and wrap it with the tag. The tag uses a **rule** attribute that takes the name of an existing User Segment. Only members of that User Segment can see the in-line content. For example, you created a User Segment called `bookfanUserSegment.seg` in Workshop for WebLogic that makes anyone a member who has a *bookfan* User Profile property set value set of `true`. The following sample code illustrates this:

```
<pz:div rule="bookfanUserSegment">
  <p>Only users who are book nerds will see this text!</p>
</pz:div>
```

User Segment rules (conditions) are the same as those available to Content Selectors, so the <pz:div> tag provides a similar level of Personalization. The difference is that Content Selectors retrieve their content from the virtual content repository, while the <pz:div> tag encloses its content in-line in the JSP.

Tip: Content Selectors can display binary content, such as images. See the [Content Management Guide](#) for instructions.

Deleting a Content Selector Query

Deleting a Content Selector query removes the query from the Content Selector in Workshop for WebLogic and the Administration Console.

Note: The steps in this chapter refer to the <data>\src folder in the Package Explorer View. Your `data` and `src` directories might be named differently.

Developers perform the following steps to remove a query in a Content Selector:

1. In the Portal Perspective, open the Content Selector file in the `\data\src\contentselectors\GlobalContentSelectors` folder in the Package Explorer View.

2. Click the **query** link in the Content Selector Editor.
3. Select the query in the Content Search dialog.
4. Click **Remove**.
5. Click **OK**.

Deleting a Content Selector

Removing a Content Selector removes it from Workshop for WebLogic and from the Administration Console.

Perform the following steps to delete a Content Selector:

1. In the Portal Perspective, open the Content Selector file in the `\<data>\src\contentselectors\GlobalContentSelectors` folder in the Package Explorer View.
2. Right-click the Content Selector and choose **Delete**.
3. Click **Yes** to confirm the deletion.

Tip: You should also delete any `<pz:contentSelector>` tags in your JSPs that reference the Content Selector you deleted.

Modifying a Content Selector

You can use the following methods to edit the content that a Content Selector displays:

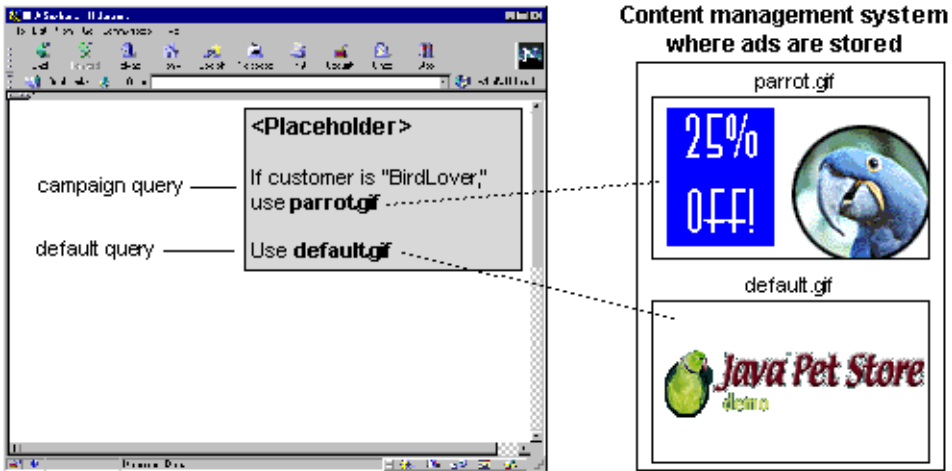
- **Workshop for WebLogic** – Developers can use Workshop for WebLogic to modify a Content Selector’s conditions and queries. For instructions, see [Chapter 13, “Modifying a Content Selector”](#).
- **Administration Console** – Portal administrators can use the Administration Console to edit a Content Selector’s properties or description. For instructions, see [Chapter 13, “Modifying a Content Selector”](#).

Creating a Placeholder

A Placeholder is a predefined location in a JSP that displays a single piece of web content at a time that is dynamically retrieved from the BEA Virtual Content Repository. If more than one content query is registered for a Placeholder, the Placeholder uses predefined queries and logic to determine which query to run and which content item to display. If a content query does not return data, the Placeholder runs another registered query, if there is one. Each query has a priority, or weight that determines the specific content to display.

For example, a Placeholder for a pet store could use a Campaign query to determine if the user is a bird lover and then display a special bird offer, as shown in [Figure 7-1](#).

Figure 7-1 Placeholders can Display Default Content and Campaign Content



A Placeholder is made up of two parts:

1. A Placeholder file you create in Workshop for WebLogic
2. A companion JSP tag that performs the processing

When you create a Placeholder, you select the content to display, choose a query for the Placeholder, and then create the Placeholder itself.

This chapter includes the following sections:

- [Selecting Content for a Placeholder](#)
- [Creating a Placeholder](#)
- [Modifying a Placeholder](#)
- [Using the <ad:adTarget> Tag Instead of a Placeholder](#)

Selecting Content for a Placeholder

Placeholders use a document's MIME-type attribute to generate the appropriate HTML tags that the browser requires. By default, Placeholders generate the appropriate HTML tags only for the following MIME types:

- **Images** – A Placeholder generates an `` 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.
- **Other types** – A Placeholder passes the text directly to the JSP.

This section contains the following topics:

- [Displaying Additional MIME Types in a Placeholder](#)
- [Adding Content to a Placeholder](#)

Displaying Additional MIME Types in a Placeholder

To display content, 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 `` tag that a browser requires for images. By default, ad Placeholders can generate the appropriate HTML only for the MIME types listed in the [Selecting Content for a Placeholder](#) section for images and other types of files.

If you are familiar with basic Java programming, you can write classes that enable Placeholders to generate HTML for additional MIME types. A video clip is an example of an additional MIME type.

Perform the following tasks to display additional MIME types in a Placeholder:

1. Create and Compile a Java Class
2. Add the new class to the CLASSPATH

Creating and Compiling a Java Class to Generate HTML

To generate the HTML that the browser requires to display the MIME type, use Workshop for WebLogic to create a Java Project in your application, and then create and compile a Java class in that Java Project that implements the `com.bea.p13n.ad.AdContentProvider` interface. For information on this interface, see the [Javadoc](#).

Adding the New Class to Your Classpath

After you save the class in a directory that is in your CLASSPATH, you must notify WebLogic Portal of its existence. You can do this manually or with the Administration Console. Choose [Method 1](#) or [Method 2](#) to add the class to your CLASSPATH.

Method 1

Perform the following steps to manually add the class to your CLASSPATH:

1. To add the class manually, stop the WebLogic Server.
2. Create a backup copy of your application's META-INF\wps-config.xml file.
3. Open the wps-config.xml file in a text editor and find the <AdService> element.
4. Add the following as a sub-element of <AdService>:

```
<AdContentProvider
Name="MIME-type"
Provider="YourClass.class"
Properties="optional-properties-for-your-class"
>
</AdContentProvider>
```

Provide the following values for the attributes of the AdContentProvider element:

- **Name** – The name of the MIME type that you want to support.
- **Provider** – The name of the compiled Java file. If you saved the file below a directory that your CLASSPATH environment variable names, you must include the file's path name, starting one directory level below the directory in CLASSPATH.
- **Properties** – Any additional properties or parameters want to pass to your object. For example, if you added <PORTAL_APP>/classes to the system CLASSPATH, save your class to support AVI files as <PORTAL_APP>/classes/myclasses/MimeAvi.class.

The following code sample shows sample attribute values:

```
<AdContentProvider
Name="video/x-msvideo"
Provider="myclasses.MimeAvi"
Properties=" "
>
</AdContentProvider>
```

5. Save your changes to the wps-config.xml file.
6. Restart the WebLogic Server.

You can also add the class to the CLASSPATH using the steps in [Method 2](#).

Method 2

Rather than manually adding the class to your CLASSPATH and activating the content provider, you can use the Administration Console to perform the following steps:

1. In Workshop for WebLogic, start the Administration Console by choosing **Run > Open Administration Console**.
2. In the Administration Console, choose **Configuration Settings > Service Administration**.
3. Select **Interaction Management** in the Resource Tree and click **Ad Service** in the **Browse** tab.
4. In the Ad Content Providers section, click **Add Ad Content Provider**.
5. Enter a **Description** and a **Name** for the new content provider. In the **Content Provider** field, specify the name of the class that generates the HTML elements that the browser requires to display an ad of this MIME type. If you write your own class, it must implement the `com.bea.commerce.platform.ad.adContentProvider` interface.
6. Click **Update**.
7. After you add the content provider, you can add additional properties.

Adding Content to a Placeholder

You can add specific properties to content items that support using content in Placeholders. See [Chapter 3, “Setting up Content”](#).

Creating a Placeholder

Creating a Placeholder involves two steps:

1. Create a Placeholder file in Workshop for WebLogic.
2. Use companion `<ph:placeholder>` JSP tags in any relevant JSPs. See the *Javadoc* for more information on the Java class.

Each `<ph:placeholder>` JSP tag must use its **name** attribute to reference a Placeholder you have created in Workshop for WebLogic.

This section contains the following topics:

- [Creating a Placeholder File](#)
- [Building a Content Query](#)
- [Determining Which Query and Content to Display](#)
- [Adding a Placeholder to a JSP](#)

Creating a Placeholder File

A Placeholder is a predefined location in a JSP that displays a single piece of web content retrieved from the BEA Virtual Content Repository. A Placeholder uses queries to retrieve and display content.

By default, Placeholders support the following MIME types: HTML, XML, plain text, and images. To display additional MIME types in Placeholders, see [Displaying Additional MIME Types in a Placeholder](#).

Content returned from a Placeholder is displayed in a portlet. When a user views a portlet or a portal desktop containing a Placeholder, the Placeholder's rules and back-end logic run a query, retrieve zero or more pieces of content matching the query, and display one of the returned items. If no content is retrieved, none is displayed.

The queries for a Placeholder originate from two different locations:

- **From the Placeholder itself** – When you create a Placeholder in the Placeholder Editor, you can also define default queries to run in the Placeholder. Default queries run for all anonymous and authenticated visitors.
- **From a Campaign** – When you create a Campaign in the Campaign Editor, one option is to display targeted personalized content in a Placeholder for a specific type of visitor.

Because a Placeholder can contain multiple queries, the Placeholder can display a different piece of content each time a user accesses the JSP containing the Placeholder. The following procedure includes a step that lets you set a query's priority to increase or decrease the chances that the query is run instead of other queries present in the Placeholder.

Placeholders are scoped to the enterprise application, so you can include Placeholders in any JSPs within the enterprise application.

Note: The steps in this chapter refer to the `data\src` folder in the Package Explorer View. Your `data` and `src` directories might be named differently.

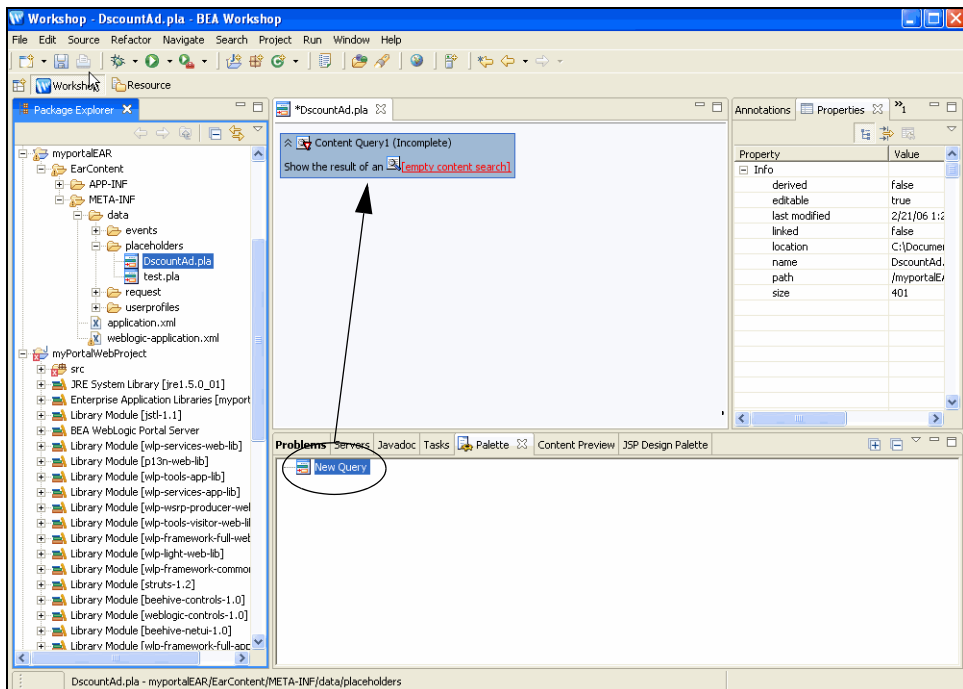
Perform the following steps to create a Placeholder:

1. Start the WebLogic Server in Workshop for WebLogic by choosing **Run As > Run on Server**. For instructions on configuring the WebLogic Server, see the [Portal Development Guide](#).
2. In the Portal Perspective, right-click the `data\src\placeholders` folder in the Package Explorer View and choose **New > Other**.

Tip: You can customize the menu so that **Content Placeholder** appears as a choice on the **New** menu. See the *Portal Development Guide* for instructions.

3. In the New Select a Wizard window, select **Content Placeholder** and click **Next**.
4. In the New Placeholder File window, enter a name for the Placeholder in the **File name** field. Keep the .pla file extension.
5. Click **Finish**. The Placeholder Editor appears.
6. In the Palette tab, drag the **New Query** into the Placeholder Editor to define a default query, as shown in [Figure 7-2](#).

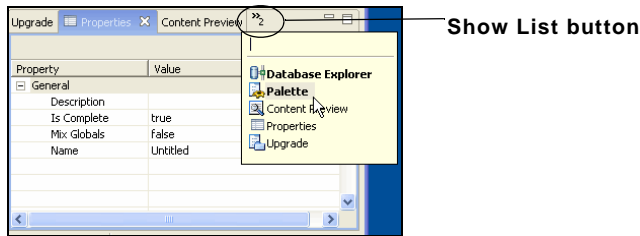
Figure 7-2 Define a Default Query by Dragging the New Query into the Placeholder Editor



Every query you add to the Placeholder in this way is a default query (non-Campaign). Campaigns can also put queries in this Placeholder, but Campaign queries are not defined in the Placeholders themselves. They are defined in the Campaigns.

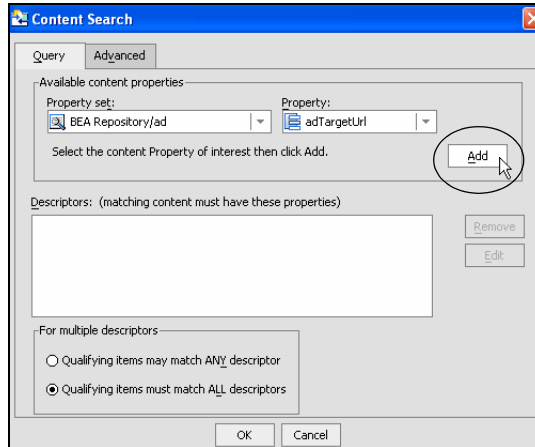
Note: If you do not see the **Palette** tab, click **Show List** to see other tabs that are not visible, as shown in [Figure 7-3](#).

Figure 7-3 Click Show List to View Hidden Tabs



7. In the Property Editor window, set values for the following:
 - **Name** – Enter a name for the query. The query name is used only for display purposes.
 - **Priority** – Select the query’s priority relative to other queries. A query with a higher priority is more likely to be run.
8. To define the query, click the **empty content search** link in the New Query item you added. You can define the query using WebLogic Portal's expression syntax (on the **Advanced** tab) or in graphical mode (on the **Query** tab).
 - **Advanced Mode** – In the Content Search window, click the **Advanced** tab and build a query using the instructions in “[Building a Content Query with Expressions](#)” on [page 6-10](#).
 - **Graphical Mode** – Begin with Step a and build a content query by selecting content properties, comparators, and values to retrieve content items.
 - a. In the Content Search window, select the **Query** tab.
 - b. Select a content type in the **Property set** field, select a content property in the **Property** field, and click **Add** as shown in [Figure 7-4](#). This example shows a content type of **ad** and a property of **adTargetUrl**. The **adTargetUrl** property makes an image clickable and provides a target for the clickthrough, expressed as a URL. See the [Content Management Guide](#) for more information.

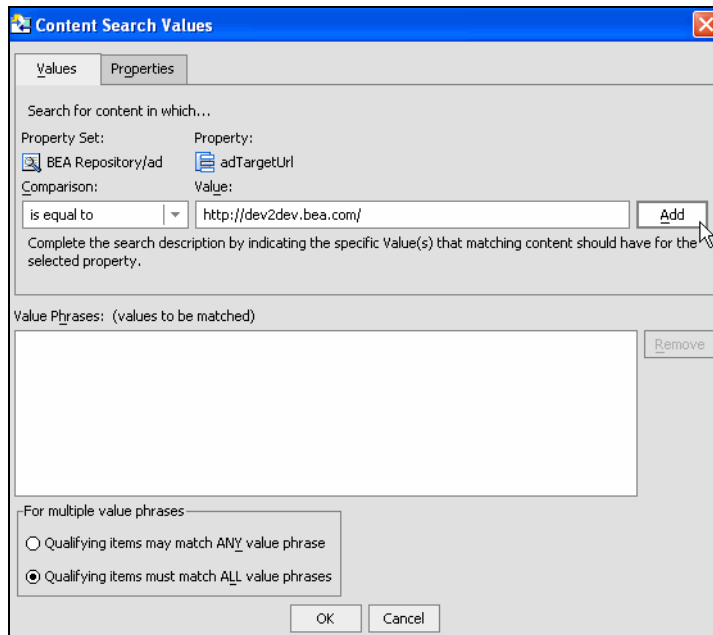
Figure 7-4 Choose a Content Type and a Content Property



Note: The properties you select are content properties (types) rather than property set properties, such as User Profile or session properties.

- c. In the Content Search Values window that appears, select one of the following tabs:
- **Values Tab** – Define the query based on a comparison to a value you enter. For example, the query could be set to retrieve content with an **investorRiskLevel** property that is marked as high. The example in [Figure 7-5](#) sets the query to retrieve an **adTargetUrl** of a BEA web site.
 - **Properties Tab** – To define the content query based on the property value that is dynamically obtained from another type of property, such as a User Profile property. For example, instead of creating a query based on static content properties, you can create a query that reads the value of the current user's **investorRiskLevel** to populate the query. The query would be different for each user.

Figure 7-5 Select an Item for the Comparison Field and Enter a Value



- d. Click **Add**. The query descriptor is added to the **Value Phrases** section.
 - e. You can add more value phrases to the query and then set the appropriate option in the **For multiple value phrases** section at the bottom of the window.
9. Click **OK** in the Content Search Values window.
 10. Create additional queries as needed in the Content Search window. You can add more value phrases to the query in the Content Search window, and then set the appropriate option in the **For multiple descriptors** section at the bottom of the window. Click **OK** when you are done.
 11. Click **OK** on the Content Search dialog.
 12. You can preview the content that will be retrieved by the query by selecting the **Content Preview** tab below the Editor. If you defined the query to use values from a User Profile property, the retrieved content will be different for each user, so you must enter the username of an existing user, such as *weblogic*, in the **Preview User** field to see the content that will be retrieved for that user.

The Content Preview window shows content that the query will retrieve. However, since a

Placeholder can show only one piece of content at a time, the single piece of content that is displayed varies depending on which query is run (determined by the priority you set for a query and the settings for Campaign queries) and the **adWeight** property setting on content.

You must add the **adWeight** property to a content type as a single-value Integer. Content with a higher **adWeight** number has a higher likelihood of being selected for display in a Placeholder.

13. Save the Placeholder file.

14. Add the Placeholder JSP tag called `<ph:placeholder>` to the relevant JSP. You must add the Placeholder filename to the Placeholder tag and run the JSP to see the results.

Note: An alternative to using Placeholders is using the `<ad:adTarget>` JSP tag to hard-code a content query within the tag. See [“Using the <ad:adTarget> Tag Instead of a Placeholder” on page 7-16](#) for more information.

Choosing the Type of Placeholder Query to Run

Placeholders can run two types of queries: default queries and Campaign queries.

Both types of queries have the same structure but originate from different places:

- **Default Query** – When you create a Placeholder file in Workshop for WebLogic, you can also add one or more queries to that Placeholder file that always remain with the Placeholder (unless you change or remove the queries in Workshop for WebLogic). These are called default queries. They return content from the virtual content repository regardless of who is viewing the portlet or JSP in your portal.

A Placeholder can run a default query every time a user loads a page that includes the Placeholder. For example, you define a header for a particular shell that contains a JSP file with a Placeholder. If that Placeholder contains a default query, a user is likely to see different content in the header each time the user visits or refreshes the desktop.

If a Campaign also targets the Placeholder, you can configure the Placeholder to show only content from that Campaign, or integrate Campaign content along with default query content.

Tip: You should create a default query for a Placeholder, because it ensures that a content item always appears in the Placeholder.

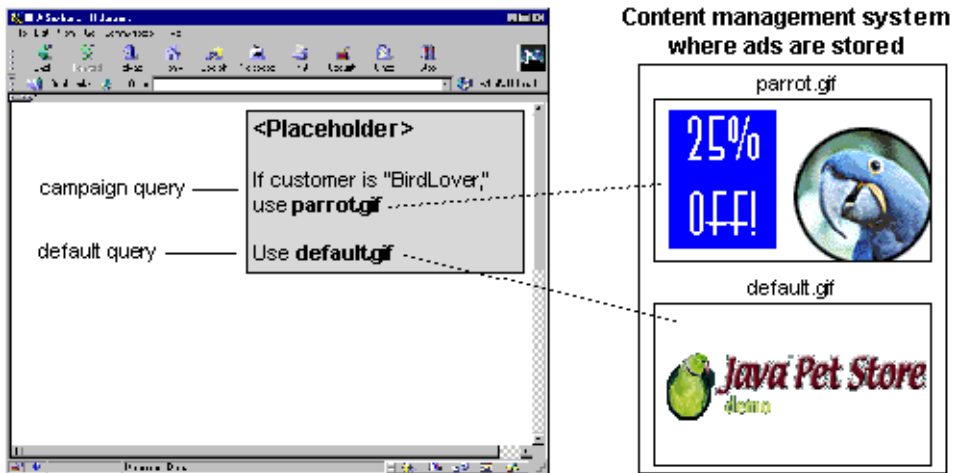
- **Campaign Queries** – When you create a Campaign in Workshop for WebLogic, one of the things you might want to do is trigger Campaign-specific content to appear in a specific

place on the desktop. Campaigns use Placeholders to display personalized content. When you define a content action in a Campaign, one of the steps is to select an existing Placeholder to run the Campaign query you define. Campaign content queries are dynamically placed by a Campaign when certain actions (events) occur. For example, a user who belongs to a certain User Segment logs into the desktop.

You can set default queries to not run when a query placed by a Campaign is present in the Placeholder.

Figure 7-6 shows the location on a page where the Placeholder displays an image, either from a default query or a Campaign query. The image is retrieved from BEA's Virtual Content Repository through a query. A Placeholder uses different factors to determine which query to run and then which retrieved content item to display.

Figure 7-6 Placeholders can Display Default Content or Campaign Content



Building a Content Query

You can use the WebLogic Portal Expression language in Workshop for WebLogic to build content queries. The advanced query-building window is available from content-related JSP tags and when building queries for Placeholders, Campaigns, and Content Selectors.

If more than one content query is registered for a Placeholder, for example, the Placeholder uses predefined queries and logic to determine which query to run and which content item to display. If a content query does not return data, the Placeholder runs another registered query, if there is one. Each query has a priority, or weight that determines the specific content to display.

A query contains three parts: <property> <comparator> <value>.

Using Expressions

The following two types of properties are available for content, and you can use both in queries:

- User-defined content properties
- Explicit (system) content properties

See [“Building a Content Query with Expressions” on page 6-10](#) for instructions and a list of explicit content properties.

Using Comparators

Comparators provide the logic that compares a query's property to the value you enter. If a content item meets the conditions of the query, the content item is returned. See [“Using Comparators” on page 6-14](#) for instructions.

Using Values

Values represent the content you want the query to return. By supplying values to a query, you're telling the query which content to retrieve (or ignore) based on the values stored on the content items. See [“Supplying Values” on page 6-17](#) for instructions.

Following Guidelines for Complex Queries

You can combine multiple independent query clauses, tying them together with **and** (&&) and **or** (| |) logic and controlling the order of evaluation with parentheses the way you would with algebraic expressions. This lets you create more complex queries. See the [Content Management Guide](#) for instructions and a list of example queries.

Determining Which Query and Content to Display

Placeholders perform a two-step process to choose which content to display. This section describes the following tasks:

1. How a Placeholder chooses which query to run
2. Once a query runs, how a Placeholder chooses which content item to display from the query

Choosing a Query to Run

When you add a default query to a Placeholder or create a Campaign query in Workshop for WebLogic, you can assign each query a priority: `highest`, `high`, `normal`, `low`, or `lowest`. The higher the priority, the higher the likelihood that the query will be run instead of other queries.

If a query does not find any documents, the Placeholder chooses another query and runs it.

You can also define default queries so that they do not run when Campaign queries are present. For instructions, see [Chapter 14, “Modifying a Placeholder”](#).

Choosing Which Content Item to Display

Depending on how broadly you define a query and on the number of documents in your Content Management system, a query could return multiple content items. By default, a Placeholder randomly selects a content item to display. To make this selection less of a random choice, you can add a property called **adWeight** to your content items, as described in [“Determining Content Priority” on page 3-2](#).

The higher the **adWeight** number you assign to a content item, the better the chance that the content item will be displayed in a Placeholder when the content item is retrieved by a query.

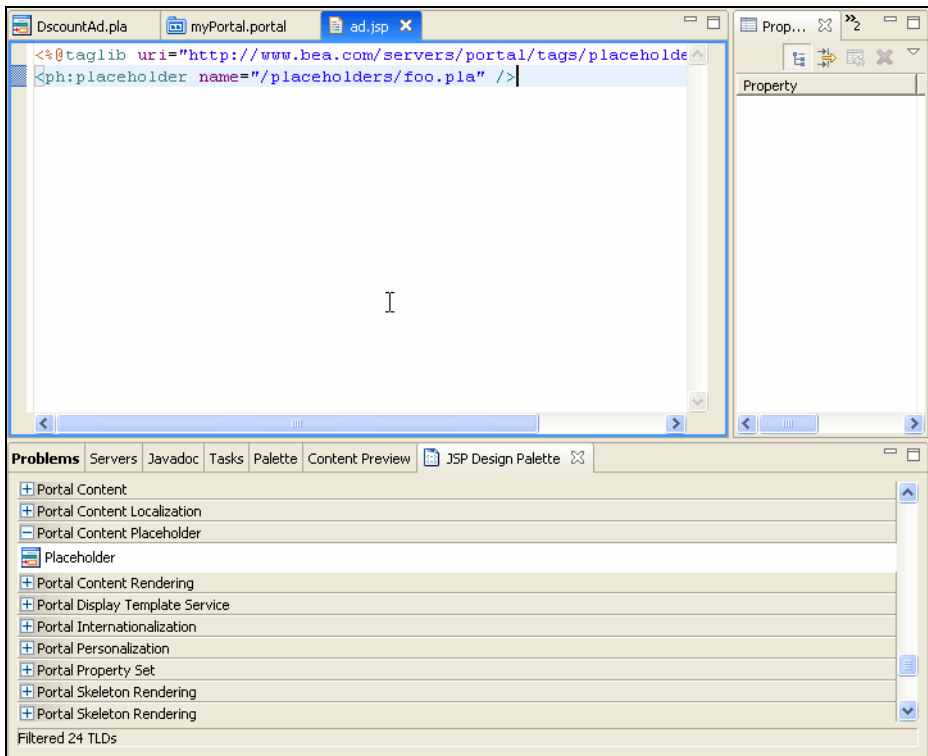
Adding a Placeholder to a JSP

After you create a Placeholder file in Workshop for WebLogic, you can use any of the following methods to add the Placeholder to a JSP in Workshop for WebLogic:

- Drag a Placeholder from the Workshop Perspective onto a page of an open portal file. When you do this, three things happen:
 - a. The Portlet Wizard appears, letting you quickly create a portlet that displays the Placeholder.
 - b. The resulting portlet is automatically added to the portal page.
 - c. A JSP file is automatically created for the Placeholder. The JSP file contains the Placeholder JSP tag with the **name** attribute automatically populated with the name of the Placeholder. The `include` statement for the tag library is automatically added.
- Open the JSP where you want to add the Placeholder, and drag the Placeholder file from the Workshop Perspective into the JSP in either Design View or Source View. The JSP tag is added automatically, the **name** attribute is automatically populated with the name of the Placeholder, and the tag library `include` statement is automatically added.

- Drag the `<ph:placeholder>` JSP tag from the JSP Design Palette (the Portal Content Placeholder category) into an open JSP and populate the tag's **name** attribute manually. The tag library `include` statement is added automatically. In [Figure 7-7](#), the Placeholder file is called `foo`, and the JSP tag references the Placeholder file.

Figure 7-7 The Two Parts of a Placeholder: a Placeholder File and a JSP Tag



Modifying a Placeholder

You can use either of the following two ways to modify a Placeholder:

- **Developers can use Workshop for WebLogic** – For instructions on modifying Placeholder properties or queries in Workshop for WebLogic, [Chapter 14, “Modifying a Placeholder”](#).

- **Portal administrators can use the BEA WebLogic Portal Administration Console** – For instructions on modifying Placeholder values in Administration Console, see [Chapter 14, “Modifying a Placeholder”](#).

See [“Managing Placeholders for Optimal Performance” on page 14-4](#) for information on changing Placeholders to improve performance.

Using the `<ad:adTarget>` Tag Instead of a Placeholder

You can also use the `<ad:adTarget>` JSP tag to display a content item on a JSP. This tag does not rely on a definition file like a Placeholder does. You simply add a query using the tag's **query** attribute. The query retrieves one or more content items (the same types of content items that Placeholders can display), and the tag chooses which content item to display in the same manner as the `<ph:placeholder>` tag. Campaigns do not put queries into an `<ad:adTarget>` tag, so the tag cannot display personalized content. You can also use `<pz:contentQuery>` and `<cm:search>` tags to execute runtime queries and return content that can be displayed.

Tip: You can get rotating banner-style content on your portal by picking one content item and cycle through the matching items on subsequent requests. You can use the `<ad:adTarget>` JSP tag, which uses the **AdConflictResolver** to pick which content to show. That will get each node's **adWeight** property (converted to a number) as the relative weight of each node and then use a random number to pick which content to use. The higher the weight, the more likely the content is to be displayed. If the item doesn't have an **adWeight** property, it assumes a value of 1.

For more information on the class for the `<ad:adTarget>` JSP tag, see the [Javadoc](#).

The following code sample shows how to set up rotating banner-style content:

```
<%@ taglib uri="http://www.bea.com/servers/portal/tags/ad" prefix="ad"%>
<ad:adTarget query=" color == userProperty('GeneralInfo', 'FavoriteColor')
"/>
```

Building a Campaign

WebLogic Portal's Campaigns help you accurately target visitors and trigger multiple actions—or even simultaneous actions—in a browsing session. Campaigns deliver the right content to the right user at the right time.

You can use Campaigns to personalize your portal in the following ways:

- **Display personalized web content** – When you use a Campaign to display personalized content, the content (for example, an image) is retrieved from BEA's Virtual Content Repository through a query and displayed in a Placeholder on a JSP. The JSP can exist in a portlet or in a desktop header region.
- **Send predefined e-mails automatically** – The Campaign service reads User Profile properties to obtain a user's e-mail address, and then sends a predefined e-mail to that user.
- **Offer personalized discounts in a commerce application** – If you built a commerce application with the necessary catalog and shopping cart functionality, you can provide a variety of discount types to specific users. Discounts enable you to put permanent or temporary price reductions on items, provide incentives to customers to buy additional products, buy products by a particular date, or use other criteria to define the discount. Discounts can run standalone or as the an action of a Campaign.

Marketing goals generally drive the content of a Campaign. For example, you might want people who log in with a certain browser type at lunch time to view content related to lunch specials. A bank might determine which portfolio to recommend, or a travel site would recommend a specific hotel chain.

Campaigns are flexible, because they let you create business logic without requiring code changes. For example, Campaigns show web content using a JSP tag called a Placeholder that is

similar to the following: `<ph:placeholder name="myPlaceholder1" />`. Add JSP Placeholder tags (uniquely identified by the **name** attribute) anywhere in your portal's JSPs. Then define your Campaigns to use the existing Placeholders, each of which can display content unique to the Campaign and to the individual users. You can change and add new Campaigns, but you never have to change your JSP code. The Placeholders you need stay the same.

In addition to providing flexibility and Personalization, Campaigns begin at a specific time and end when their purpose has been fulfilled (when specific goals are achieved or a time deadline is reached). Campaigns can even be set up to run only once for each visitor.

Structurally, a Campaign contains at least one scenario. Each scenario contains one or more actions that show personalized content, send an automatic e-mail, or provide a personalized discount. The advantage of scenarios as containers for actions is that you can use User Segments to determine which users are eligible to be targeted with the actions in a scenario (but you are not required to assign User Segments to scenarios). For example, you could create a Campaign with two scenarios: one that targets its actions only to males and the other that targets its actions to females. [Table 8-1](#) shows how the logic works in a scenario for females.

Table 8-1 A Campaign Scenario that Targets Females Who Recently Visited the Web Site

Are You a Member of the Female User Segment and Have Not Visited the Site in the Last 30 Days?

- | | |
|------------|--|
| Yes | Your User Profile indicates that you are female and have visited the site in the last 30 days. If this Campaign is triggered, you are targeted with any of the actions in this scenario that apply to you. |
| No | Your User Profile indicates that you are a male. If this Campaign is triggered, you are <i>not</i> targeted with any of the actions in this scenario. |

The chapter includes the following sections:

- [Performing the Prerequisite Tasks](#)
- [Building a Campaign](#)
- [Testing a Campaign](#)
- [Triggering a Campaign](#)
- [Turning Off a Campaign](#)
- [Resetting a Campaign](#)

Performing the Prerequisite Tasks

Complete the following tasks before you create a Campaign:

1. Read the [“Checklist for Planning Your Campaign Strategy”](#) on page 2-9.
2. If you decided to use Session or Request properties to trigger Campaign actions, verify that you performed the following tasks:
 - In the JSP containing the event to be fired, get the request attribute through a variable or set it directly in the JSP
 - In the JSP containing the event to be fired, get or set any event properties you want to use
 - If you want to use session properties to trigger Campaign actions, verify that the firing event is in the same session containing the session properties you want to use

For more information on Session or Request properties, see [“Creating a Session Property Set”](#) on page 4-8 or [“Creating a Request Property Set”](#) on page 4-8.

3. Determine if you plan to use goal setting to end your Campaign. Goal Setting can end a Campaign based on the number of content items displayed or clicked. See [“Setting Goal Definitions”](#) on page 8-7 and [“Adjusting Goal Definitions”](#) on page 8-9 for instructions.

Building a Campaign

Developers create Campaigns and administrators use those Campaigns as templates to modify Campaign characteristics and create new Campaigns with similar characteristics.

Building a Campaign requires that you plan your Campaign logic for scenarios and actions, create a Campaign file, add then add the scenarios and actions.

This section contains the following topics:

- [Planning Your Campaign Logic](#)
- [Creating a Campaign File](#)
- [Adding a Scenario to a Campaign](#)
- [Adding an Action to a Scenario](#)
- [Adding an E-Mail Action to a Campaign](#)
- [Adding a Discount Action to a Campaign](#)

- [Setting Up Automatic E-Mail Messages](#)

Planning Your Campaign Logic

A Campaign uses units called actions to perform specific Personalization tasks. Actions are triggered by specific conditions you set, and the actions are grouped into scenarios.

For example, a Campaign Action can be triggered by the following conditions: *When a user logs in between January 1 and January 31, and that user is a member of the non-manager User Segment, trigger the Campaign to do something.* The action could then do the following: *When the Campaign is triggered, send an automatic e-mail reminding the user to complete an annual performance review.*

A Campaign can contain multiple scenarios, each of which can contain multiple actions. An action is triggered when all of the following items are true:

- An event is fired and the Campaign service is listening for it. See [“Registering Events for Campaigns” on page 9-45](#) for more information.
- The conditions of the action are met, or a user belonging to a specific User Segment logs in, which triggers a scenario to fire all of its actions.
- The Campaign is set to active and has not expired (through a date deadline or if its goals were met).

[Table 8-2](#) shows Campaign Action rules that are created in Workshop for WebLogic.

Table 8-2 Campaign Action Rules

When all of the following conditions apply:	An HTTP request has the following properties: RequestPropertyOne is equal to <code>success</code>
Any of the following events has occurred:	SessionLoginEvent
Do the following:	Content Action, E-mail Action, or Discount Action

This rule is evaluated only if an event occurs for which the Campaign service is listening. (This event does not need to be used directly in the Campaign rule.) For example, if the Campaign service is configured to listen for the BEA-provided **UserRegistrationEvent** (which it is by default), then when a **UserRegistrationEvent** occurs, the event takes a snapshot of the Request object and the Campaign rules are evaluated.

The following list is the order in which the previous Campaign action rules are evaluated:

- Is there a request property called **RequestPropertyOne** with a value of `success`?
- Is this a **SessionLoginEvent**?
- Are all of these conditions `true`?

Because a **UserRegistrationEvent** woke up the Campaign service and took a snapshot of the request object, the Campaign Action is not triggered, because the rule requires that all of its conditions evaluate to `true`. The **SessionLoginEvent** rule is `false` (because it was the **UserRegistrationEvent** that woke up the Campaign service).

If the rule was defined differently so that any of the conditions evaluating to `true` would trigger the action (rather than all conditions), the Campaign action fires if the request property evaluates to `true`.

Creating a Campaign File

The steps in this chapter refer to the `data\src` folder in the Package Explorer View. Your `data` and `src` directories might be named differently.

Perform the following steps to create a Campaign file and set Campaign properties:

1. Start the WebLogic Server in Workshop for WebLogic by choosing **Run As > Run on Server**. For instructions on configuring the WebLogic Server, see the [Portal Development Guide](#).
2. In the Portal Perspective, right-click the `data\src\campaigns` folder in the Package Explorer View and choose **New > Other**.

Tip: You can customize the menu so that **Campaign** appears as a choice on the **New** menu. See the [Portal Development Guide](#) for instructions.

3. In the New Select a Wizard window, expand the WebLogic Portal folder, select **Campaign**, and click **Next**.
4. Enter a name for the Campaign in the **File name** field. Keep the `.cam` file extension.
5. Click **Finish**. Select the Campaign in the Campaign Editor when it appears.
6. Select the **Properties** tab and set the following properties for the **General** property. These properties help determine if the Campaign will run. Use the following description to set each property:

- **Active** – Set the value to `true` if you want the Campaign to be run. Set the value to `false` if you do not want the Campaign to run.
 - **Description** – Enter a detailed description of the Campaign. The description is appended to the text in the Description window.
 - **Is Complete** – Read-only value of `true` or `false`. If the conditions for the Campaign are all complete, the **Is Complete** property is set to `true`. If the field is `false`, you should check each action and scenario to determine which properties are missing.
 - **Name** – Read-only. The name of the Campaign (the Campaign filename).
 - **Sponsor** – Enter the name of the organization or person sponsoring the Campaign.
7. If you are displaying personalized content in a Campaign, expand the **Goals** property in the **Properties** tab and set the following properties:
- **Description** – Enter a description about the goals that will end a Campaign prior to its stop date. Click **OK** in the Property Text Editor dialog box when you are done.
 - **Goal Definitions** – End a Campaign prior to the Campaign stop date when specific images are viewed or clicked in a portal. Goal Setting can end a Campaign based on the number of content items displayed or clicked. See [Setting Goal Definitions](#) for instructions.
8. Expand the **Timing** property in the **Properties** tab and set the following properties:
- **Start Date** – Click the ellipsis icon [...] and set the month, day, and time (in your time zone) you want the Campaign to start. Click **OK**.
 - **Stop Date** – Click the ellipsis icon [...] and set the month, day, and time (in your time zone) you want the Campaign to end. Click **OK**.
- See [Figure 8-1](#).
9. Choose **File > Save** to save your work.

Figure 8-1 Sample Properties for a New Campaign

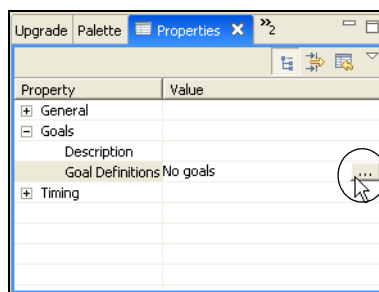
Property	Value
[-] General	
Active	true
Description	Frequent Eaters Campaign
Is Complete	true
Name	lunch
Sponsor	Moe's Bagels
[-] Goals	
Description	
Goal Definition	1 goal
[-] Timing	
Start Date	Wed Jan 25 08:51:44 MST 2...
Stop Date	Tue Feb 28 09:51:44 MST 2...

Setting Goal Definitions

Perform the following steps to define a Campaign goal:

1. In the Portal Perspective, select the Campaign you created in “Building a Campaign” on page 8-3.
2. In the **Properties** tab, expand the Goals property.
3. In the **Goal Definitions** field, click the ellipsis button [...], as shown in Figure 8-2.

Figure 8-2 Click the Ellipsis Button to Get to the Edit Campaign Goals Window

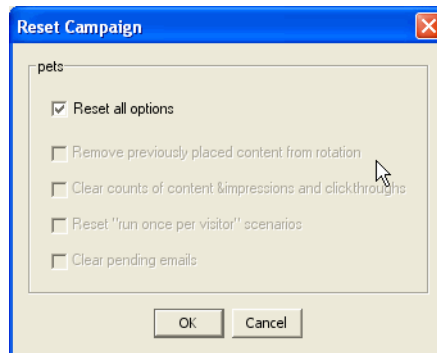


The Edit Campaign Goals window appears.

4. In the Edit Campaign Goals window, click **New**.
5. In the Campaign Goals section, enter a number in the **Count** field for the number of times the content item must be viewed or clicked to meet the goal.

6. Select an item from the drop-down list in the **Count Type** field that determines whether the content item must be viewed (*Impressions*) or clicked (*Click-throughs*).
Note: For a content item to use clickthrough functionality, it must have one of the following properties set: **adTargetUrl**, **adTargetContent**, or **adMapName**. The property value for any of these properties is a URL that, when clicked, takes the user to the location you want. In a portal, URLs are relative to the portal Web project root directory. See “[Creating URLs to Portal Resources](#)” on page 8-10 for more information.
7. Select an item from the drop-down list in the **Type** field. The type determines if the content impression or clickthroughs must be on content displayed by this Campaign (*From this Campaign only*) or on the content displayed by the Campaign or outside of the Campaign (*From anywhere*).
8. Select an item from the **Logic** field to determine if the count can be reached by adding the impressions or clickthroughs for all selected content items (*Summing the path counts*) or if it can be reached when any one content item in the list must reach the count number (*Against any one path*).
9. In the Goal Paths section, add content items to use in the goal. To populate the Goal Paths list, do one or both of the following:
 - In the **Add Path to Goal** field, enter the repository path to a content item in BEA's Virtual Content Repository, and click **Add**.
 - In the **Retrieve Query Paths** field, select a content action you have already defined in your Campaign. The content action has an associated query that will retrieve specific content items from the Virtual Content Repository. Click **Get** to retrieve a list of content items that the query will retrieve. Select the content items you want and click **Add**.
10. Create additional goals as required. Each goal you add in the Campaign Goals section has associated content items in the Goal Paths section. For example, you can add a goal that states, *If a piece of content is viewed (Impressions) three times (the Count), the goal is met.*
11. If you create more than one goal, select the appropriate **End the campaign** option below the Campaign Goals section.
12. Click **OK** in the Edit Campaign Goals window.
13. Save the Campaign by choosing **File > Save**.
14. After you save the Campaign, you are prompted to reset the Campaign by selecting options in the Reset Campaign dialog box and clicking **OK**, as shown in [Figure 8-3](#).

Figure 8-3 You Can Reset a Campaign by Selecting the Reset All Options Check Box



Adjusting Goal Definitions

The following examples goals show how you can modify Goal Setting and the consequences:

- Determine how frequently the Campaign service checks to see if goals have been met by performing the following steps:
 - a. In the Administration Console, choose **Configuration Settings > Service Administration**.
 - b. In the Resource Tree, expand the **Interaction Management** folder and then select **Campaign Service**.
 - c. In the **Configure** tab, click **Configuration Settings for Campaign Service**.
 - d. Set the **Goal Check Time** to the frequency you want. The default is 300000 milliseconds (five minutes). Less-frequent goal checks improve performance, but the Campaign service takes longer to determine if goals were met. For testing, set the value to 0 so that there are no delays in checking for goals.
 - e. Click **Update**.
- Determine how many impressions or clickthroughs occur before that number is written to the database. In the database, the Campaign service compares the current count to the impressions or clickthrough goal you set in the Campaign. Perform the following steps:
 - a. In the Administration Console, choose **Configuration Settings > Service Administration**.
 - b. In the Resource Tree, expand the **Interaction Management** folder and then select **Ad Service**.

- c. In the **Configure** tab, click **Configuration Settings for: Ad Service**.
- d. Set the **Display Flush Size** to the number you want. The default is 10. A larger flush size improves performance, but the Campaign service takes longer to determine if goals were met. For testing, set the flush size to a small number to ensure Campaigns end immediately after your goals are met.
- e. Click **Update**.

Creating URLs to Portal Resources

WebLogic Portal provides an extensible mechanism to create URLs to your portal resources in a portal web project that can transfer from domain to domain without breaking, especially when server names and port numbers change. This URL-creation mechanism also lets you switch between secure and non-secure URLs (HTTP and HTTPS).

Use the following two items to create portable URLs:

- The `<render:*Url>` JSP tags in the Portal Skeleton Rendering JSP tag library (see the JSP Design Palette)
- A portal web project's `WEB-INF/bee-hive-url-template-config.xml` file

The `bee-hive-url-template-config.xml` file contains multiple URL templates, each with a unique name. These template URLs contain variables such as `url:domain` and `url:port` that are read from the active server. The `<render:*Url>` JSP tags have a template attribute in which you can specify the name of a URL template in the `bee-hive-url-template-config.xml` file.

The following examples show how the JSP tags use the templates to create URLs:

- A sample URL template exists in the `bee-hive-url-template-config.xml` file:

```
<url-template name="secure-url">
  https://{url:domain}:{url:securePort}/{url:path}?{url:queryString}
</url-template>
```

- The `<render:resourceUrl>` JSP tag creates a URL using the template:

```
<% String reportpath = "reports/report1.html"; %>
  <a href="<render:resourceUrl template="secure-url"
    path="<%=reportpath%"/>">
View the Report
</a>
```

You can use any of the URL templates in the `bee-hive-url-template-config.xml` file provided by WebLogic Portal, and you can add as many templates as you want to the file.

You can use any of the following variables when you build a URL template:

- **The {url:domain} variable** – Reads the name of the server from the current request.
- **The {url:port} variable** – Reads the listen port number of the server from the current request. See [Troubleshooting the URLs](#).
- **The {url:securePort} variable** – Reads the SSL port number of the server from the current request. See [Troubleshooting the URLs](#).
- **The {url:path} variable** – Reads the name of the web application. The URLs to all resources in a web application are relative to the web application directory.
- **The {url:queryString} variable** – Reads a `queryString` variable for the URL.

Troubleshooting the URLs

If you are using a proxy server or you are switching between non-secure and secure ports, you may find that URLs do not resolve if you use the `{url:port}` or `{url:securePort}` variables. The URLs do not resolve because the variables for those values are read from the request. For example, if a user in a non-secure port (port number 80) clicks a secure HTTPS link that was created with a URL template that uses the `{url:securePort}` variable, the port number of the request (80) is used for the `{url:securePort}` variable, which creates a secure request (HTTPS) on a non-secure port. The same result could occur if a user on a proxy server (port 80) clicks a link to a resource outside the proxy server (port 443).

In both cases, you should hard code port numbers in the URL templates to get URLs to resolve correctly.

The `beehive-url-template-config.xml` file automatically created in a portal web project also contains URL templates and variables for Web Service for Remote Portlets (WSRP). These templates must remain in the file if you plan to be a WSRP producer. See the [Federated Portals Guide](#) for more information.

Adding a Scenario to a Campaign

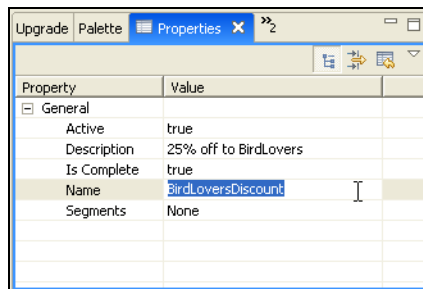
Campaigns contain actions that perform specific Personalization tasks and are triggered by specific conditions you set. Actions are grouped into scenarios. A Campaign can contain multiple scenarios; each scenario can contain multiple actions.

1. In the Portal Perspective in Workshop for WebLogic, select the Campaign you created in [“Building a Campaign” on page 8-3](#).

2. Drag the **New Scenario** item from the **Palette** tab into the Campaign Editor. You can use four scenario templates that contain predefined actions and conditions. If you drag one of these scenarios into the Campaign Designer, see the Description window for details on each.
3. In the **Properties** tab, expand the **General Property** item and set the following properties:
 - **Active** – Set the value to `true` if you want the Campaign to run. Set the value to `false` if you do not want the Campaign to run.
 - **Description** – Click the ellipsis icon [...] and enter a detailed description of the scenario. Your description is appended to the text in the Description window.
 - **Is Complete** – This field displays a read-only value of `true` or `false`. If the conditions are all complete (the conditions for the Campaign, each scenario, and each action), the **Is Complete** property is set to `true`. If the field is `false`, you should check each action and scenario to determine which properties are missing.
 - **Name** – Enter a name for the scenario.
 - **Segments** – If you want all actions in the scenario to run if the user is a member of one or more User Segments, click the ellipsis icon [...] and select the User Segments you want to use. For example, **BirdLovers**.

Figure 8-4 shows these three fields in a Campaign designed to offer a 25% discount to users who belong to the **BirdLovers** User Segment.

Figure 8-4 Campaign Property Editor



4. Add more scenarios as needed. For example, you could include an image of a parrot.
5. Save the Campaign by choosing **File > Save**.

Adding an Action to a Scenario

You can add a Content Action, an E-mail Action, or a Discount Action to a scenario, and you can add multiple actions in each scenario.

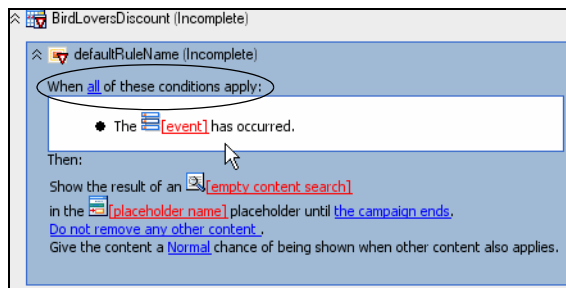
Adding a Content Action

Content actions retrieve Web content from a content repository and display the selected piece of content in a predefined Placeholder on a JSP.

Perform the following steps to add a Content Action to your Campaign:

1. In the Portal Perspective, select the Campaign you created in [“Building a Campaign” on page 8-3](#).
2. From the **Palette** tab, drag the **New Content Rule** icon (for example) onto the appropriate scenario icon in the Campaign Editor.
3. Select the **Properties** tab and enter a name for the action.
4. In the action, click the **all** link to toggle back and forth between **any** and **all** to determine which conditions will trigger the action. See [Figure 8-5](#).

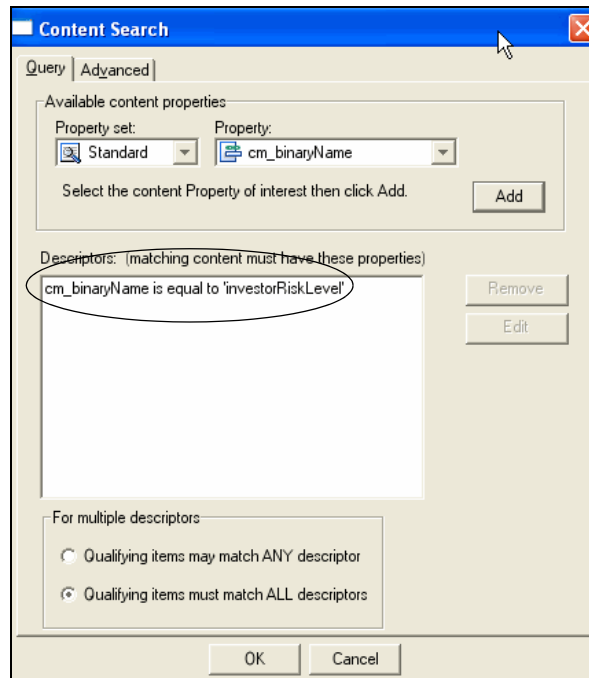
Figure 8-5 Click the All Link to Determine How What Will Cause the Action to Occur



5. To define the query, click the **empty content search** link. You can define the query in Advanced mode using WebLogic Portal's expression syntax (on the **Advanced** tab) or in Graphical mode (on the **Query** tab).
 - **Advanced mode** – In the Content Search window, select the **Advanced** tab and build a query using the instructions in [“Building a Content Query with Expressions” on page 6-10](#).
 - **Graphical mode** – Use the following steps to build a content query by selecting content properties, comparators, and values to retrieve content items.

- a. In the Content Search window, select the **Query** tab.
- b. Click the drop-down list to select a property set and then select a property and click **Add**. (The properties you select are content properties (types) rather than property set properties such as User Profile or session properties.)
- c. In the Content Search Values window that appears, select one of the following tabs:
 - **Values** – To define the query based on a comparison to a value you enter. For example, the query could be set to retrieve content with an **investorRiskLevel** property that is marked as `high`.
 - **Properties** – To define the content query based on the property value that is dynamically fed in from another type of property, such as a User Profile property. For example, instead of creating a query based on static content properties, you can create a query that reads in the value of the current user's **investorRiskLevel** to populate the query. The query would be different for each user.
- d. Click **Add**. The query descriptor is added in the Content Search window, as shown in [Figure 8-6](#).

Figure 8-6 Base the Content Query on a Comparison to a Value You Enter



- e. You can add more value phrases to the query, then set the appropriate option in the **For multiple descriptors** section.
6. Click **OK** in the Content Search window.
7. Click the **[placeholder name]** link in the action, and select the Placeholders that will display content when the scenario is triggered.
8. Click **Add** to move the Placeholders to the Selected Placeholders section, enter text to describe the Placeholder in the **Description** field, and click **OK**.
9. You can preview the content that will be retrieved by the query in the Content Preview window below the Editor. If you defined the query to use values from a User Profile property, the retrieved content will be different for each user, so you must enter the username of an existing user (such as *weblogic*) in the **Preview User** field to see which content will be retrieved for that user.

Note: The Content Preview window shows content that the query will retrieve. However, since a Placeholder can show only one piece of content at a time, the single piece of

content that is displayed varies depending on which query is run (determined by the priority you set for a query and the settings for Campaign queries) and the **adWeight** property setting on content.

You must add the **adWeight** property to a content type as a single-value Integer. Content with a higher **adWeight** number has a higher likelihood of being selected to display in a Placeholder.

10. If you want the content to stop being displayed prior to the end of the Campaign, click **the campaign ends** link to determine when the content will stop being displayed.
11. To increase the chances that content from the query will be displayed, click the **Do not remove any other content** link and determine the existing content that will be removed from the designated Placeholders when the action runs.
12. To set the priority that the query will be run compared to other queries that may exist in the Placeholders, click the **Normal** link and select a priority. Higher priorities give the query a greater chance of being run.
13. In the Available Conditions section, select the conditions you want to trigger the action. When you select a condition, a corresponding link appears in the action area. Click the link to define the condition.
14. Save the Campaign by choosing **File > Save**.

Adding an E-Mail Action to a Campaign

E-mail Actions send a predefined e-mail to a user in your Campaign. You might use an e-mail to alert users to specials that are customized to a specific User Segment.

Perform the following steps to add an E-Mail Action to your Campaign:

1. In the Portal Perspective in Workshop for WebLogic, select the Campaign you created in [“Building a Campaign” on page 8-3](#).
2. From the **Palette** tab, drag the **New E-mail Rule** icon onto the appropriate scenario icon in the Campaign Editor.
3. In the **Properties** tab, enter a name for the action.
4. In the action, click the **all** link to toggle back and forth between **any** and **all** to determine which conditions will trigger the action. The **any** choice means that only one of the conditions must be true for the action to occur.

5. Click the **[server url]** link to select the e-mail message to send. Choose **URL**, enter a **Subject**, and an optional default e-mail address. Click **Preview** if you want to view the e-mail now.
6. Click **OK**.
7. In the Available Conditions section in the **Palette** tab, select the conditions you want to trigger the action. When you select a condition, a corresponding link appears in the action area. Click the link to define the condition.
8. Save the Campaign file by choosing **File > Save**.

Note: If you are using emails in your Campaign, you can choose to send the emails in batch mode or real-time (batch mode is the default). In batch mode, when you run or test your Campaign, e-mails are not sent. See [“Setting Up Bulk E-Mail Messages” on page 8-20](#) to learn how to change the mailing behavior to real-time.

Adding a Discount Action to a Campaign

Discount actions give the user a discount on items, orders, or shipping. You can add a Discount Action to a Campaign.

Perform the following steps before you create a Discount Action in your Campaign:

1. Add commerce services to your portal application.
2. Set up a shopping cart using the WebLogic Portal Commerce API.
3. Create a catalog in the virtual content repository.
4. Use the WebLogic Portal catalog classes in the commerce API to surface catalog items from the Virtual Content Repository and identify them with **categories** and **SKU** numbers.
5. Create discounts and use the Commerce API to surface the discounts in your shopping cart. You can also use the API to surface the discount's description next to the discount amount displayed in the shopping cart.

Perform the following steps to add a Discount Action to your Campaign:

1. In the Portal Perspective of Workshop for WebLogic, select the Campaign you created in [“Building a Campaign” on page 8-3](#).
2. In the **Palette** tab, drag the **New Discount Rule** icon onto the appropriate scenario icon in the Campaign Editor.
3. In the Property Editor tab, enter a name for the action.

4. In the action, click the **all** link to toggle back and forth between **any** and **all** to determine which conditions will trigger the action.
5. Click the **[discount]** link and select the discounts you want to apply. You can choose to enter an optional **Discount Explanation** that can appear in the shopping cart next to the displayed discount.
6. Click **OK**.
7. In the Available Conditions section, select the conditions you want to trigger the action. When you select a condition, a corresponding link appears in the Action section. Click the link to define the condition.
8. Save the Campaign file by choosing **File > Save**.

Setting Up Automatic E-Mail Messages

Perform the following steps to send automatic e-mails as part of a Campaign:

1. Define an e-mail address property by create a property in a User Profile property set to store the user e-mail address. For example, the default **CustomerProperties.usr** property set contains an **Email** property that can contain a single, unrestricted string value for an e-mail address.
2. Set up the Campaign Service. You must tell the Campaign service where to get the e-mail address when sending automatic e-mails to users.
 - a. In the Administration Console, choose **Configuration Settings > Service Administration**.
 - b. In the Resource Tree, expand the **Interaction Management** folder and select **Campaign Service**.
 - c. Click **Configuration Settings for: Campaign Service**. [Figure 8-7](#) shows you where to enter the property set name and the name of the e-mail property you set up in the previous step.

Figure 8-7 You Can Set Up Where to Get E-Mail Addresses for Automatic E-Mails

Configuration Settings for: Campaign Service

Description: This service is used to check whether campaign goals have been met and to respond when they have been met.

Goal Check Time (milliseconds):* 300000

Base Directory for Email Browsing: campaigns/emails

⚠ Maximum URI Length: 254

Default From Email Address:* signupnow@bea.com

Email Address Property Name: Email

Property Set Name Containing Email Address Property: CustomerProperties

Email Opt In Property Name: Email_Opt_In

Property Set Name Containing Opt In Property: Demographics

[Update] [Cancel]

* Required information

⚠ Application redeployment required for service to use modified values

- **Goal Check Time** – The default is 300,000 milliseconds (five minutes). If you set the **Goal Check Time** to 0, there is no time delay in the amount of time the Campaign service checks to see if goals have been met. See [“Setting Goal Definitions”](#) on page 8-7 for more information.
- **Base Directory for Email Browsing** – The default directory for storing e-mail files is campaigns/emails. You must also change the <url-pattern> path in the web.xml file to secure the files in the new directory and redeploy the application after you make these changes. See [“Storing E-Mail Files in a Different Directory”](#) on page 8-25.
- **Maximum URI Length** – The maximum length of a deployable Campaign Uniform Resource Identifier (URI).
- **Default From Email Address** – The default address that receives any replies from email that the Campaign sends. In a standard mail header, this is the From address. Each Campaign scenario can specify its own From address that overrides this default property.
- **Email Address Property Name** – The name of the property that contains customer email addresses.

- **Property Set Name Containing Email Address Property** – The name of the property set that contains customer email properties.
 - **Email Opt In Property Name** – The name of the property that specifies whether customers want to receive Campaign-related email. You should define a User Profile property with the single, restricted values of `true` and `false`.
 - **Property Set Name Containing Opt In Property** – The name of the property set that contains the customer’s opt-in property. Emails will not be sent to users who have their property value set to `false`.
- d. Click **Update**.
3. Set SMTP for outgoing mail by configuring the Simple Mail Transfer Protocol (SMTP) host name for the Mail Service. Perform the following steps:
 - a. In the Resource Tree, select **Interaction Management** and then **Mail Service**.
 - b. Click **Configuration Settings for Mail Service** and use the **SMTP Host Name** field to enter the host name for your e-mail server's outgoing mail.
 - c. If you use a Sybase database, select the **Enable ORDER BY Workaround for Clobs** check box. This setting enables the Mail Service to work with Sybase since Sybase does not support using a TEXT data type in an ORDER BY clause.
 4. Create e-mail messages using a predefined e-mail message. E-mail messages can be in any of the following formats: TXT, HTML, JSP, or XML (with style sheets). Store the e-mails in the `<PortalWebProject>/campaigns/emails` directory. If you want to use a different directory for storing e-mail files, see [“Storing E-Mail Files in a Different Directory” on page 8-25](#). If you want to send bulk e-mails, see [“Setting Up Bulk E-Mail Messages” on page 8-20](#).
 5. Set e-mail security. Prevent unauthorized access to e-mail messages by following the steps in [“Setting Up E-Mail Security” on page 8-24](#).

Setting Up Bulk E-Mail Messages

You must use a command to periodically send the batch e-mails that the JSPs store in the WebLogic Portal data repository. You can also use the `cron` command or any other scheduler that your operating system supports to issue the `send-mail` command.

For Windows, the `send-mail` command is located in a `.bat` file wrapper script. For UNIX, the `send-mail` command is located in a `.sh` file. The following sections refer to the `.bat` file. UNIX users should substitute `.sh` for `.bat`.

The `send-mail` wrapper script specifies the name and listen port of the WebLogic Portal host that processes the `send-mail` request. By default, the wrapper script specifies `localhost:7501` for the hostname and listen port. However, `localhost:7501` is valid only when you run the script while logged in to a WebLogic Portal host in a single-node environment (and only if you did not modify the default listen port). If you use the `send-mail` script from any other configuration, you must modify the script.

Modifying the Send-Mail Script to Work from a Remote Host

Perform the following steps to run the `send-mail` script from a remote host (a computer that is not a WebLogic Portal host):

1. Open the following file in a text editor:

```
PORTAL_HOME\bin\mailmanager.bat (Windows)
or
PORTAL_HOME/bin/mailmanager.sh (UNIX)
```

2. In the `mailmanager` script, locate the `SET HOST=` line. Replace `localhost` with the name of a WebLogic Portal host.
3. If the host uses a listen port other than `7501`, replace `7501` in the `SET PORT=` line with the correct listen port.
4. Save the `mailmanager` script.

Modifying the Send-Mail Script to Work in a Clustered Environment

If you work in a clustered environment, you must modify the `send-mail` wrapper script to specify the name of a host in the cluster. The default `localhost` value is not valid for the Mail Service in a clustered environment.

Note: The following steps must be performed on each host that will run the script.

Perform the following steps on each host to use the `send-mail` script in a clustered environment:

1. Open the following file in a text editor:

```
PORTAL_HOME\bin\mailmanager.bat (Windows)
PORTAL_HOME/bin/mailmanager.sh (UNIX)
```

2. In the `mailmanager` script, replace `localhost` in the `SET HOST=` line with the name of a WebLogic Portal host. Because each host in a cluster can access the data repository that stores the e-mail messages, you can specify the name of any host in the cluster.

3. If the host uses a listen port other than 7501, replace 7501 in the `SET PORT=` line with the correct listen port.
4. Save the `mailmanager` script.

Using the Mailmanager Commands

The `mailmanager` command is a wrapper script that uses the `jav.com.bea.pl3n.mail.MailManager` class. The `mailmanager` commands help you send and manage bulk e-mails.

Use the following command syntax:

```
mailmanager.bat [ appName ] [ list | send | send-delete | delete ]  
batch-name ] (mailmanager.sh on UNIX)
```

If you specify only the `appName` arguments, the `mailmanager` command prints to standard output the names all e-mail batches in the application and the number of e-mails in each batch.

[Table 8-3](#) contains a list of the command arguments.

Table 8-3 Mailmanager Command Arguments

Command Argument	Description
<i>appName</i>	The name of the enterprise application that generated the e-mail batch.
<code>list</code>	Prints to standard output the names of all e-mail batches in the data repository and the number of e-mails in each batch.
<code>list batch-name</code>	Prints to standard output the subject and recipients of all e-mails in the batch that you specify.
<code>send batch-name</code>	Sends all e-mails in the batch that you specify.
<code>send-delete batch-name</code>	Sends all e-mails in the batch that you specify and then deletes the batch from the data repository.
<code>delete batch-name</code>	Deletes e-mails in the batch that you specify.
<i>batch-name</i>	The name of a batch that <code>mailmanager list</code> returns. This argument does not support wildcards.

Table 8-4 contains examples of mailmanager commands.

Table 8-4 Examples of Mailmanager Commands

Command Example	Description
<code>mailmanager.bat list</code>	Lists all available batches
<code>mailmanager.bat wlcsApp list /campaigns/campaign1.cam</code>	Lists the contents of a batch named <code>/campaigns/campaign1.cam</code> that the wlcsApp application generated
<code>mailmanager.bat wlcsApp send-delete /campaigns/campaign1.cam</code>	Sends the <code>campaign1.cam</code> batch and deletes it afterwards
<code>mailmanager.bat wlcsApp delete /campaigns/campaign1.cam</code>	Deletes the <code>campaign1.cam</code> batch

Sending Bulk E-Mail Messages

Perform the following steps to send bulk e-mail from a shell that is logged into a WebLogic Portal host:

1. Start the WebLogic Server by choosing **Run As > Run on Server**.
2. To determine the names and contents of the e-mail batches in the data repository, enter the following command:

```
mailmanager.bat appName list (Windows)
```

The `appName` is the name of the enterprise application that generated the e-mail batch. The command prints to standard output. You can use shell commands to direct the output to files.

3. To send a batch and remove it from the data repository, enter the following command:

```
mailmanager.bat appName send-delete batch-name
```

Note: If you are using e-mails in your Campaign, you can choose to send the e-mails in batch mode or real-time (batch mode is the default). In batch mode, when you run or test your Campaign, no e-mails will be sent. See [“Setting Up Bulk E-Mail Messages” on page 8-20](#) to learn how to send batch mode emails and how to change the mailing behavior to real-time.

Scheduling Bulk E-mail Delivery

You can use a scheduling utility to send the e-mail batches in the data repository. Because you must specify the name of a batch when you use the `mailmanager` command to send mail, you must schedule sending mail for each Campaign scenario separately. The name of a batch corresponds to the scenario's **containerId**. The **containerId** specifies the ID of the Campaign to which the scenario belongs.

For information in using a scheduling utility, refer to your operating system's documentation.

Deleting E-Mail Batches

You can delete e-mail batches as you send them (See [“Sending Bulk E-Mail Messages” on page 8-23](#)).

You can also perform the following steps to delete e-mail batches:

1. To determine the names and contents of the e-mail batches in the data repository, enter the following command:

```
mailmanager.bat appName list
```

The `appName` is the name of the enterprise application that generated the e-mail batch. The command prints to standard output. You can use shell commands to direct the output to files.

2. To delete a batch, enter the following command:

```
mailmanager.bat appName delete batch-name
```

Setting Up E-Mail Security

When a Campaign sends an automatic e-mail, it uses a predefined e-mail message stored on the file system within your portal web project. By default, WebLogic Portal prevents unauthorized access to those e-mail files when the files are stored in the `<PortalWebProject>/campaigns/emails` directory.

The following deployment descriptors secure your e-mail files:

- **The `<PortalApplication>/wps.jar/META-INF/weblogic-ejb-jar.xml` descriptor file** – The following line in this file provides the name of a user who is in the global *PortalSystemAdministrator* role:

```
<run-as-principal-name> portaladmin </run-as-principal-name>
```

Membership in the *global PortalSystemAdministrator* security role is defined in the WebLogic Portal Administration Console at the server level. In a portal domain created

with the Configuration Wizard, the *Administrators* and *PortalSystemAdministrators* groups that are provided by default are configured to be members of the global *PortalSystemAdministrator* role. Because the *portaladmin* user (also provided by default in a portal domain) is a member of the portal *Administrators* and *PortalSystemAdministrators* groups, *portaladmin* is a member of the global *PortalSystemAdministrator* role.

- **The <PortalWebProject>/WEB-INF/web.xml file** – The following line in this file secures the e-mail files in <PortalWebProject>/campaigns/emails, allowing only the Campaign service (through the *PortalSystemAdministrator* user defined in the previous section) to access and send the e-mails:

```
<url-pattern>/campaigns/emails/*</url-pattern>
```

Perform the following steps to use a different user for e-mail security:

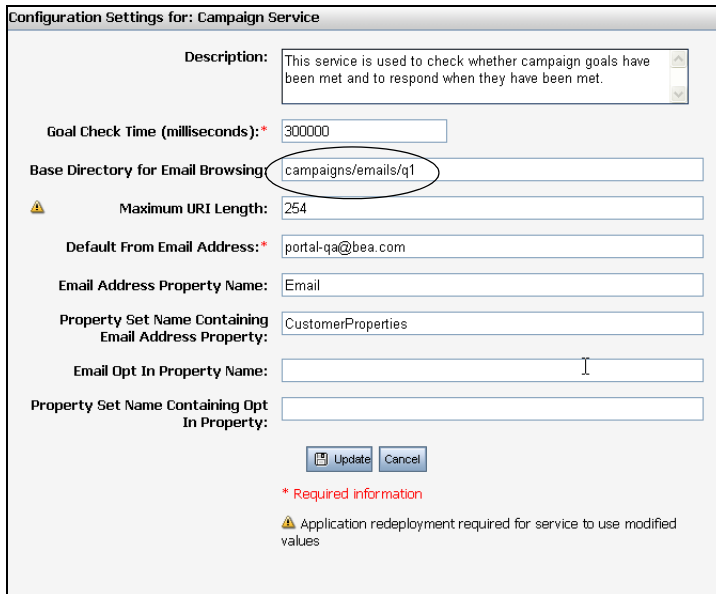
1. Back up the `wps.jar` file.
2. Un-jar the `wps.jar` file and change the name of the user in the `weblogic-ejb-jar.xml` file.
3. Verify that the user exists.
4. Verify that the user is a member of the global *PortalSystemAdministrator* security role; and
5. Re-jar and replace the old `wps.jar` and redeploy the application. If you enter the name of a user in `<run-as-principal-name>` that does not exist, or if you delete the *portaladmin* user without changing the `<run-as-principal-name>` entry, you will receive deployment errors on the `wps.jar` file.

Storing E-Mail Files in a Different Directory

Perform the following steps if you need to use a different directory to store e-mail files:

1. Change the `<url-pattern>` path in the `web.xml` file to secure the files in the new directory.
2. In the Administration Console, choose **Configuration Service > Service Administration**.
3. In the Resource Tree, expand the **Interaction Management** folder and select **Campaign Service**.
4. Click **Configuration Settings for Campaign Service** and change the directory in the **Base Directory for Email Browsing** field. For example, `campaigns/emails/q1`, as shown in [Figure 8-8](#).

Figure 8-8 Change the E-mail Directory



5. Redeploy the application or restart the server during development in Workshop for WebLogic).

Note: Using a wildcard character (*) in the URL pattern does not provide recursive directory protection. The wildcard protects only the files in the last directory listed. For example, if you want to store e-mail files in the /campaigns/emails/q1 directory, the url-pattern information in the /campaigns/emails/* directory does not protect the e-mail files in the /q1 directory. To protect those e-mail files, the url-pattern information must be in the /campaigns/emails/q1/* directory.

Testing a Campaign

Use the following guidelines to test Campaigns on your development server in your development environment:

1. Verify that the Campaign is complete. The entire Campaign, each scenario, and each action have specific conditions for being complete. When you select each, the **Is Complete** property in the Property Editor window displays a read-only value of true or false. If the **Is Complete** property is false for any part of a Campaign, select the property in the Property Editor window and read the Description to find out which properties are required.

2. Verify that the Campaign is active. With the Campaign selected (not a scenario or action), set the **Active** property in the Property Editor window to `true`.
3. If you see the text *Campaign is currently stopped* just below the Campaign Editor window, you must change the **Start Date** or **Stop Date** properties so that the current date falls between the two. When the current date is within the Campaign date range, the *Campaign is currently stopped* text disappears.
4. If your Campaign uses Goal Setting to end a Campaign based on content impressions or clickthroughs, perform the following steps to modify the settings:
 - a. In the Administration Console, choose **Configuration Settings > Service Administration**.
 - b. In the Resource Tree, expand the **Interaction Management** folder and select **Campaign Service**.
 - c. Click the **Configuration Settings for Campaign Service** link and set the **Goal Check Time** to 0. This creates no time delay in the amount of time the Campaign service checks to see if goals have been met. Click **Update**.
 - d. In the Resource Tree, select **Ad Service** and click the **Configuration Settings for: Ad Service** link
 - e. Set the **Display Flush Size** field to 1, as shown in [Figure 8-9](#).

Figure 8-9 Change the Display Flush Size to 1

The screenshot shows a dialog box titled "Configuration Settings for: Ad Service". It contains several configuration fields, each with a yellow warning icon to its left:

- Description:** This service is used to configure how ads are rendered and how events for ad display and ad click are handled.
- Display Flush Size:** 1
- Rendering Class:** com.bea.p13n.ad.AdContentProviderBase
- Event Tracker Class:** com.bea.campaign.AdTracking
- Ad Clickthru URI:** AdClickThru
- Show Doc URI:** ShowDoc

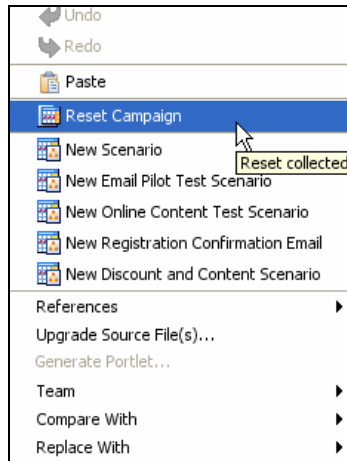
At the bottom of the dialog, there are two buttons: "Update" and "Cancel". Below the buttons, there is a red asterisk followed by the text "* Required information" and a yellow warning icon followed by the text "Application redeployment required for service to use modified values".

This setting writes each impression or clickthrough to the database each time it occurs and ends the Campaign on the exact number of impression or clickthrough counts you have established. For example, if you want to end a Campaign on five impressions, but your **Display Flush Size** was set to 10, you would need to see 10 impressions before the that number is written to the database. At that point, the Campaign service would detect that the five impressions had already been met, effectively ending the Campaign after 10 impressions rather than five.

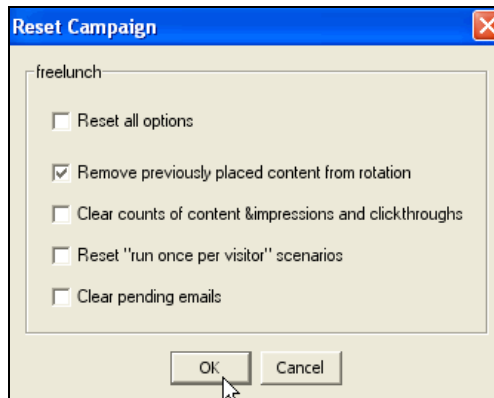
You must restart the server for this change to take effect.

Note: Do not deploy your application into a production environment with these settings. Performance will be adversely affected.

5. You can reset many aspects of Campaigns during testing, such as impression and clickthrough counts that can end a Campaign and scenarios that run only once for each user. With a Campaign file open in Workshop for WebLogic, choose **Portal > Reset Campaign** and reset any aspect of the Campaign.
6. Test **run once** Content Actions by performing the following steps to test Content Actions that you want to run only once per user:
 - a. With the Campaign open in Workshop for WebLogic, right-click in the Campaign Editor
 - b. Choose **Reset Campaign**.
 - c. In the Reset Campaign dialog box, select the **Reset all options** check box, and click **OK**.
 - d. View the portal and run the Content Action by starting an event for which the Campaign service is listening (such as logging in). Verify that the Campaign content is displayed.
 - e. Log out or click **Back** in the browser to return to where you can launch the event again.
 - f. Return to the Campaign in Workshop for WebLogic and select the Campaign.
 - g. Right-click the Campaign Editor space and choose **Reset Campaign**, as shown in [Figure 8-10](#).

Figure 8-10 Right-Click in the Campaign Editor to Reset the Campaign

- h. Select the **Remove previously placed content from rotation** check box and click **OK**. See [Figure 8-11](#).

Figure 8-11 Reset a Campaign

7. To troubleshoot Campaign content in Placeholders, you should understand how Placeholders handle default and Campaign queries. For example, default and Campaign queries have priorities that help determine which query runs. Also, you can set default queries so that they do not run when Campaign queries are present. For more information, see [Chapter 7, “Creating a Placeholder”](#).

8. Improve performance by disabling, enabling, and flushing content caches that are used for web content. With a Campaign open in Workshop for WebLogic, the **Edit > Portal Content Caches** menu provides those options:

- Flush Content Caches
- Disable Content Caches
- Enable Content Caches

For more information on managing your caches, use the **Run > Portal Cache Manager** menu and consult the *Portal Development Guide* for instructions.

The following caches are affected (you can view the caches in the Administration Console by choosing **Configuration Settings > Service Administration** and selecting **Personalization** and then **Cache Manager** in the Resource Tree).

- **The adBucketServiceCache** – Reserved for future use.
- **The searchCache** – Caches the results of content searches for the virtual content repository.
- **The documentMetadataCache** – Caches the results of document searches for the DocumentManager. This setting is not used by the content repositories.
- **The binaryCache.BEA Repository** – Caches binary property values for the BEA Repository.
- **The documentContentCache** – Caches the document bytes for the DocumentManager. This setting is not used by the content repositories.
- **The nodeCache.BEA Repository** – Caches content for the BEA Repository.
- **The documentIdCache** – Caches the results of document searches (ids only) for the DocumentManager. This setting is not used by the content repositories.
- **The adServiceCache** – Used by the **ad** service to cache the results of searches for content rendering.

Tip: For optimal performance, enable these caches in your production environment. See [“Setting Campaign Content Caches” on page 8-34](#) for instructions.

Triggering a Campaign

You must use a Regular or Behavior Tracking event to begin your campaign or trigger a campaign action based on events and their values. A commonly used event is `SessionLoginEvent`; see [“Using the SessionLoginEvent” on page 9-8](#) for instructions.

Campaign scenario rules are evaluated only when a single event occurs for which the Campaign service is listening.

Note: By default, the only events that cannot be used to trigger Campaigns are `DisplayContentEvent`, `DisplayProductEvent`, `BuyEvent`, `SessionBeginEvent`, and `SessionEndEvent`, as listed in the `<PortalApplication>/wps.jar/com/bea/campaign/internal/listeners.properties` file.)

If your Campaign conditions use request, session, or event properties, those properties are captured when a listened-for event is triggered. The event takes a snapshot of the current session properties, the single request property (contained in the session), and the event properties (contained in the request). The snapshot taken by the event is in the form of a request object, which the event passes to the Campaign service for evaluation. If the values in that snapshot evaluate to true against any Campaign action rules, those Campaign actions are triggered.

Tip: If you trigger a Campaign to test e-mails, the e-mails are not sent real-time if the `batch` flag default is still set to `true`. See [“Setting Up Bulk E-Mail Messages” on page 8-20](#) for instructions on changing the mailing behavior to real-time.

This section contains the following topic:

- [Troubleshooting Campaign Actions](#)

Troubleshooting Campaign Actions

When Campaign Actions are not triggered as expected using Session, Request, and Event properties, one of the following items might be the problem:

- The Campaign service was listening for events, but no event occurred
- The session, request, or event properties contained in the Campaign rule were not part of the request object snapshot taken when the event occurred
- In Campaign rules that are defined so that all conditions must apply for the Campaign action to be triggered, one or more of the conditions evaluated to `false`

Turning Off a Campaign

You can remove the `CampaignEventListener` in order to turn off all Campaigns so that they do not fire Campaign events.

Tip: The following steps in the Administration Console work for a portal that is deployed as an exploded EAR file. If your portal is a compressed EAR file, you will need to do these steps manually and then re-build and deploy the EAR file.

Perform the following steps to turn off Campaign events:

1. Start the Administration Console and choose **Configuration > Service Administration**.
2. In the Resource Tree, expand the **Personalization** folder and select **Event Service**.
3. In the **Browse** tab, select the `com.bea.campaign.internal.CampaignEventListener` check box and click **Delete**. Campaign events will not longer be fired, but if you set up other Behavior Tracking or other event listeners, those events will continue to fire.

Note: If you want to turn on the Campaign later, add the `CampaignEventListener` as a Synchronous Listener in the **Browse** tab.

Resetting a Campaign

You can reset different parts of your Campaigns. For example, you may want to do one or all of the following:

- Clear content from a Placeholder that had been previously put in the Placeholder. Doing this ensures that the users who were supposed to see personalized content only once see it only one time.
- Clear from the database the number of times an image has been viewed or clicked so that your Campaign does not reach its goals.
- Give users a second chance on **run once** Campaign Actions that they have already triggered.
- Clear any e-mail messages waiting to be sent.

You can reset Campaigns in the development environment (for testing) or in the production environment.

This section contains the following topics:

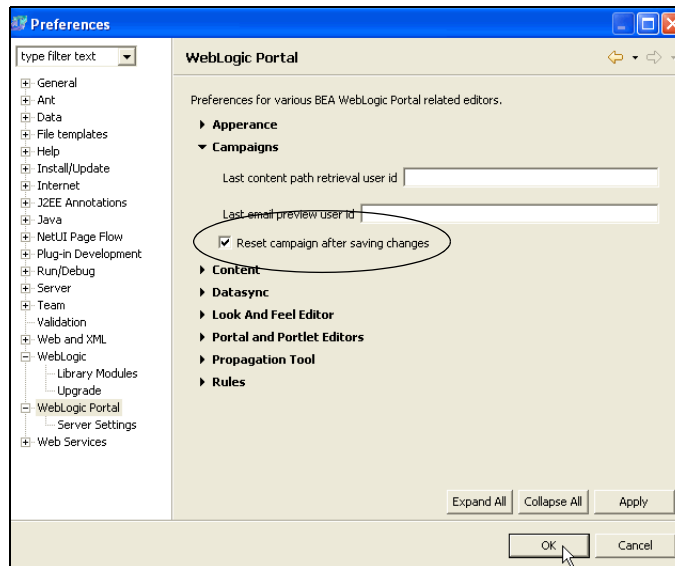
- [Resetting a Campaign in the Development Environment](#)
- [Resetting a Campaign in the Production Environment](#)

Resetting a Campaign in the Development Environment

Perform the following steps to automatically reset a Campaign in your development environment after you make changes to it:

1. Open a Campaign file in the Portal Perspective in Workshop for WebLogic.
2. Choose **Window > Preferences**.
3. In the Preferences window, select **WebLogic Portal**.
4. Click **Campaigns** and select the **Reset campaign after saving changes** check box, as shown in [Figure 8-12](#).

Figure 8-12 You Can Automatically Reset a Campaign After You Edit It



5. Click **OK**.

For more information on using this feature for testing, see [“Testing a Campaign”](#) on page 8-26.

Resetting a Campaign in the Production Environment

Perform the following steps to reset a Campaign in your production environment:

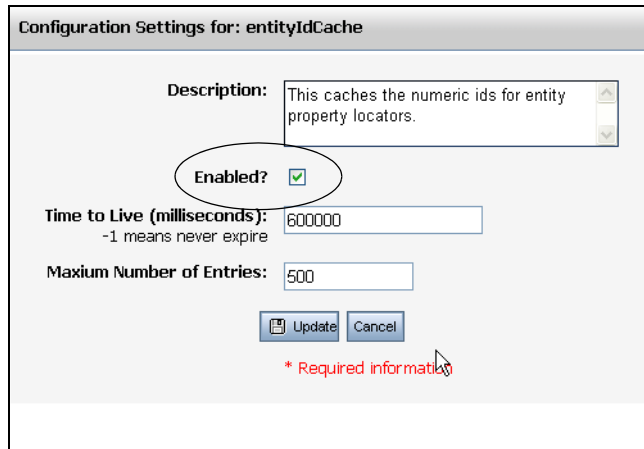
1. In Workshop for WebLogic, start the Administration Console by choosing **Run > Open Administration Console**.
2. In the Administration Console, select **Interaction > Campaigns**.
3. In the Resource Tree, select the Campaign that you want to reset.
4. In the **Campaign Details** tab, click **Reset Campaign**.
5. Click **OK**.

Setting Campaign Content Caches

For optimal performance, enable content caches in your production environment. From Workshop for WebLogic, you can disable, enable, and flush caches that are used for web content.

Perform the following steps to enable content caches:

1. In the Portal Perspective in Workshop for WebLogic, open a Campaign.
2. Choose **Edit > Portal Content Caches > Flush Content Caches** to clear each of the caches. Select **Edit > Portal Content Caches > Disable Content Caches** to stop caching for all caches. Select **Edit > Portal Content Caches > Enable Content Caches** to get the best performance for your Campaign. See [“Testing a Campaign” on page 8-26](#) for a list of content caches.
3. You can view the affected caches in the WebLogic Portal Administration Console by choosing **Configuration Settings > Service Administration**.
4. In the Resource Tree, expand the **Personalization** folder and expand the **Cache Manager** folder.
5. Click a specific cache name to view its settings. You can click **Flush** to clear this cache item.
6. Click **Configuration Settings for <cache name>** and select the **Enabled** or **Disabled** check box, as shown in [Figure 8-13](#).

Figure 8-13 You Can Enable or Disable a Cache Setting

Configuration Settings for: entityIdCache

Description: This caches the numeric ids for entity property locators.

Enabled?

Time to Live (milliseconds): 600000
-1 means never expire

Maxium Number of Entries: 500

Update Cancel

* Required information

7. Click **Update**.

Tip: You can adjust your Campaign and caches to run faster in a production environment.

See [“Managing a Campaign for Optimal Performance”](#) on page 15-10 for information on changing Campaigns to improve performance.

Building a Campaign

Setting Up Events and Behavior Tracking

An Event is generated when a user interacts with a web interface. Events can include logging in, clicking or viewing a graphic, clicking a button, navigating to another page in a portal, and so on.

WebLogic Portal provides an events framework that lets you leverage events in many ways: to trigger Campaigns, persist event data in the database, and provide other types of functionality when events occur.

Note: Interaction Management events are different than portlet events, which provide a framework for interportlet communication. See the [Portlet Development Guide](#).

The following examples show functionality you can provide with the event framework:

- **Capture the number of times users access a portal page.**
- **Determine how many users have registered in a portal.** You could also create a Campaign Action that automatically sends each user a welcome e-mail when the registration event occurs.
- **Identify which pieces of content are viewed or clicked.**
- **Determine which category of user logs in to your HR Intranet most often.** Categories of users could include managers and regular employees. You could also create a Campaign Action that displays a specific graphic when managers log in and displays another graphic when regular employees log in.

This chapter describes the components of the event framework, helps you plan an event strategy by explaining the purpose and use of each piece of the framework, describes WebLogic Portal's predefined events, and provides guidance and instructions on using events in your applications.

This chapter includes the following sections:

- [Choosing How to Handle Events](#)
- [Completing Your Behavior Tracking Strategy](#)
- [Using Predefined Events](#)
- [Generating Events for Content Clicks](#)
- [Generating Content Events](#)
- [Providing Event Attribute Values](#)
- [Enabling Behavior Tracking](#)
- [Configuring Behavior Tracking](#)
- [Creating Custom Events](#)
- [Creating Custom Event Listeners](#)
- [Dispatching Events](#)
- [Using Events in Campaigns](#)
- [Debugging the Event Service](#)
- [Tracking Content Changes](#)
- [Disabling Behavior Tracking](#)

Choosing How to Handle Events

Each Event is an instance of an Event object that is identified with a unique name, or type. Each Event type can get and set specific attributes, depending on its function. In each of the previous examples, the event must capture specific information. For example, to capture the number of times users access a portal page, a **ClickPage** event might get and set the **name** of the page that was clicked. To identify which pieces of content are viewed, a **DisplayContent** event might get and set the **ID** and **type** of each displayed content item.

After events set their attribute values, you can persist those values in any desired way. WebLogic Portal provides a default mechanism for persisting event attributes in a database as XML. When event data is stored in the database, you can mine that data to perform analytics, run reports, or even feed event data back into your applications. For example, you can create a portlet that runs

SQL queries against the database and returns the number of times each portal page was visited. You can also develop your own persistence functionality. For example, you can store event data in a file, or you can write the data to database tables without structuring the data in XML.

Sometimes, events do not require attributes or persistence. Their only purpose could be to trigger some other type of functionality. For example, if you want to determine how many times a download link is clicked regardless of who clicked it, a **ClickDownloadLink** event (and an accompanying event listener) can increment a database field value by 1.

You can also make Campaigns more powerful by using events in your Campaign definitions. For example, you can send a user a predefined e-mail automatically when the user generates the **UserRegistration** event by registering in a portal; or display a personalized piece of content when an event with specific attribute values is generated.

[Figure 9-1](#) shows the event framework, which gives you the flexibility to handle events in many ways. [Table 9-1](#) describes the pieces of the framework.

Figure 9-1 The Event Framework

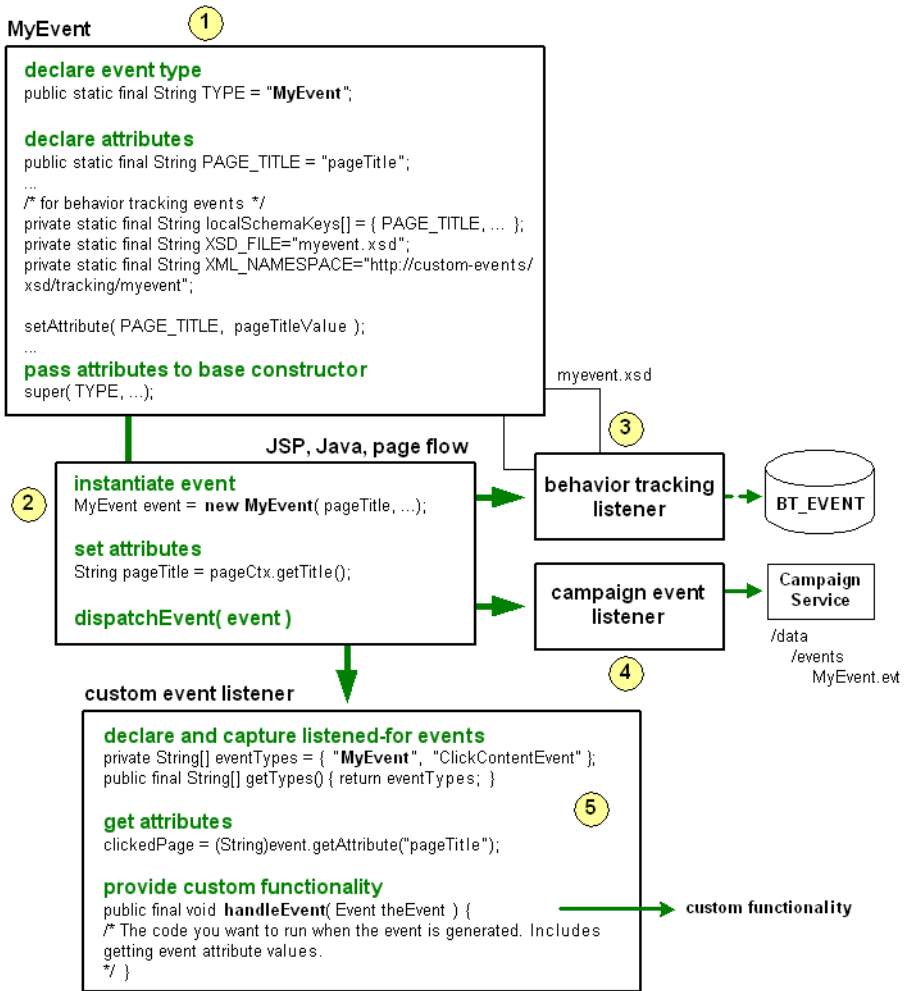


Table 9-1 The Event Framework

- 1 An event is an object that extends either the `Event` or `TrackingEvent` class. The event identifies itself to the Event service with a unique name, or type, declares the attributes it will use, and passes the event type and the attributes to the base class constructor.
 Events can contain whatever type of attributes you want to capture. For example, you can capture the name of a page or a portlet that is selected for viewing; you can capture the name of a JSP in a Page Flow to gauge which JSPs are being visited most often. You can trigger a Campaign when a specific JSP is viewed; you can capture information about content retrieved from the virtual content repository, or you can capture product information when a user adds an item to a shopping cart.
 Behavior Tracking events also declare the XML namespace and schema filename that the Event Service uses to store event attribute values in the database as XML. For each custom Behavior Tracking event you create, you should also create an XML schema.
- 2 Wherever you want to generate the event in your application (whether from a JSP, a Java class, or a Page Flow), create an instance of the event. In your code, set the attribute values the event needs, and pass them to the event as arguments in the order the event expects them. The argument order is defined in the event class. Tell the Event Service to dispatch the event. Dispatching an event tells all the interested event listeners that the event has occurred, causing them to perform their actions.
- 3 The Behavior Tracking listener listens for all events that extend the `TrackingEvent` class and are registered with the Behavior Tracking service.
 The Behavior Tracking listener's function is to move the XML document of event attributes, created by the event and the XML schema, to a buffer. The Behavior Tracking Service then moves the XML document to the `BT_EVENT` table in the database in an interval you determine.
 You can retrieve Behavior Tracking data from the database for reporting or analytical purposes, such as determining the amount of traffic a page or portlet receives.
 By default, the Behavior Tracking listener is not registered with the Event service. You must register the Behavior Tracking listener to enable Behavior Tracking, as described in [“Enabling Behavior Tracking” on page 9-19](#).

Table 9-1 The Event Framework

- 4 The Campaign event listener listens for and handles all events, except excluded events listed in the `wps.jar` file's `listeners.properties` file. When an event occurs, the Campaign event listener calls the Campaign service. The Campaign service takes a snapshot of the current HTTP request, and evaluates the data in the request against any Campaigns you have created to see if any Campaign actions need to be executed.

Campaigns are completely dependent on events. If no events occur, the Campaign service is never called, and no Campaign actions are executed.

In addition to the basic function of calling the Campaign service with an event, you can also use events within Campaign definitions by executing Campaign actions if a specific event occurs or if an event has specific properties. For example, you can define a Campaign in the following ways:

- If **MyEvent** occurs, show a specific piece of content.
- If **MyEvent** has a **published** property with a value greater than 2004, show a specific piece of content.

In order to use events and event properties in Campaign definitions, you must create an event property set for each event you want to use in Campaigns (stored in your application's `/data/src/events` directory in Workshop for WebLogic). An event property set contains the exact names of the attributes you are setting in your event. The Campaign Editor interface uses the event property set in drop-down fields that you use to create the Campaign definition.

For information on Campaigns, see [Chapter 8, "Building a Campaign"](#).

- 5 Create a custom event listener only if you want to perform custom functionality when an event occurs. A custom listener tells the Event service which events to monitor (which events trigger it to perform its custom functionality). For example, with a custom event listener, you can implement your own persistence mechanism to store event attributes, or you can respond to an event in real time by modifying a User Profile or displaying related products when a user clicks a product image.

The base class you implement, `EventListener`, provides two methods: `getTypes()`, which lets the listener advertise which event types it is interested in, and `handleEvent()`, which lets you perform your custom functionality.

A listener can listen for more than one event, whether the event is a custom event or any of WebLogic Portal's predefined events.

In performing custom event handling, you have access to the event properties with the event's `getAttribute()` method.

Completing Your Behavior Tracking Strategy

WebLogic Portal's event framework provides many options for generating and handling events, as described in the previous section. See the guidelines in [“Planning Your Behavior Tracking Strategy” on page 2-11](#) to determine the pieces of the event framework you need to implement.

This section contains the following topic:

- [Planning the Deployment of Custom Events, Listeners, and Property Sets](#)

Planning the Deployment of Custom Events, Listeners, and Property Sets

Creating custom events, listeners, and event property sets involves adding files to your application and updating your application `CLASSPATH`. If you are adding events and property sets to an application that is already deployed, these changes require application redeployment for the events and `CLASSPATH` updates, and running the BEA Propagation Utility to update the event properties in the database. For deployment instructions, see the [Production Operations Guide](#).

See Part IV [Production](#) for other deployment and production tasks.

Using Predefined Events

WebLogic Portal provides predefined Behavior Tracking events. The events capture different attributes and use the Behavior Tracking listener and the Behavior Tracking Service to persist the attributes as XML in the `BT_EVENT` table when they are generated, or dispatched. You must enable Behavior Tracking to persist the event attributes (as described in [“Enabling Behavior Tracking” on page 9-19](#)). You can also use these events to trigger Campaigns.

The following predefined events are provided for compatibility with legacy WebLogic Portal commerce applications: `AddToCartEvent`, `PurchaseCartEvent`, and `RemoveFromCartEvent`. If you want to dispatch these events in new commerce applications, you must create your own code and Content Management properties to set and get the event property values, and you must dispatch these events from your application code (for example, Page Flows and JSPs).

If you want to perform custom event handling when any of the predefined events is dispatched, create a custom event listener, as described in [“Creating Custom Event Listeners” on page 9-38](#).

This section contains the following topics:

- [Using the SessionLoginEvent](#)

- [Using the SessionBeginEvent and SessionEndEvent](#)
- [Using the UserRegistrationEvent](#)
- [Using the AddToCartEvent](#)
- [Using the RemoveFromCartEvent](#)
- [Using the PurchaseCartEvent](#)
- [Using the Rule Events](#)
- [Using the DisplayCampaignEvent](#)
- [Using the CampaignUserActivityEvent](#)
- [Using the ClickCampaignEvent](#)
- [Using the ClickProductEvent](#)
- [Using the ClickContentEvent](#)
- [Using the ClickThroughEventFilter](#)
- [Using the ContentConfigEvent](#)
- [Using the ContentCreateEvent](#)
- [Using the ContentDeleteEvent](#)
- [Using the ContentUpdateEvent](#)

Using the SessionLoginEvent

Use the `SessionLoginEvent` to dispatch an event when a user logs into a portal and is authenticated.

If Behavior Tracking is enabled, the event property values are written to the `BT_EVENT` table in the database when the event is generated and the event is registered with the Behavior Tracking Service as a persisted event, as shown in [Figure 9-3](#).

Using the SessionBeginEvent and SessionEndEvent

The `SessionBeginEvent` and the `SessionEndEvent` are generated automatically. A `SessionBeginEvent` is generated when a user accesses a web site running on WebLogic Portal.

A `SessionEndEvent` is generated when the session ends, such as when the user closes the browser or the session times out.

If Behavior Tracking is enabled, the event property values are written to the `BT_EVENT` table in the database when the events are generated and the event is registered with the Behavior Tracking service as a persisted event, as shown in [Figure 9-3](#).

The `SessionBeginEvent` and the `SessionEndEvent` do not have corresponding property sets in a portal application. By default, the Campaign listener does not listen for these events, so they cannot be used to trigger Campaigns. For more information on starting a Campaign, see [“Triggering a Campaign” on page 8-31](#).

Using the `UserRegistrationEvent`

Use the `UserRegistrationEvent` to dispatch an event when a user registers in a portal (when the user is added to the user store programmatically with a registration portlet, for example).

If Behavior Tracking is enabled, the event property values (in particular the user ID) are written to the `BT_EVENT` table in the database when the event is generated and the event is registered with the Behavior Tracking service as a persisted event, as shown in [Figure 9-3](#).

Using the `AddToCartEvent`

Use `AddToCartEvent` to dispatch an event when a user adds an item to a shopping cart. This event lets you capture information such as currency type, quantity of the item being added, unit list price, and SKU. These properties must be represented somehow in your shopping cart and content type to use this event.

If Behavior Tracking is enabled, the event property values are written to the `BT_EVENT` table in the database when the event is generated and the event is registered with the Behavior Tracking service as a persisted event, as shown in [Figure 9-3](#).

Using the `RemoveFromCartEvent`

The `RemoveFromCartEvent` generates an event when a user removes an item from a shopping cart. This event lets you capture information such as currency type, quantity of the item being added, unit list price, and SKU. These properties must be represented somehow in your shopping cart and content type to use this event.

If Behavior Tracking is enabled, the event property values are written to the `BT_EVENT` table in the database when the event is generated and the event is registered with the Behavior Tracking service as a persisted event, as shown in [Figure 9-3](#).

Using the PurchaseCartEvent

The `PurchaseCartEvent` dispatches an event when a user makes a purchase. This event lets you capture information such as currency type, order number, and total purchase price. These properties must be represented somehow in your shopping cart to use this event.

If Behavior Tracking is enabled, the event property values are written to the `BT_EVENT` table in the database when the event is generated and the event is registered with the Behavior Tracking Service as a persisted event, as shown in [Figure 9-3](#).

Using the Rule Events

WebLogic Portal provides a Rule Event control that lets you generate a Behavior Tracking event whenever you fire a rule in a page flow using the Rules Executor control. The Rule Event control gets all necessary properties, including the names of the rule set and the rule that was fired.

If Behavior Tracking is enabled, the event property values are written to the `BT_EVENT` table in the database when the event is generated and the event is registered with the Behavior Tracking service as a persisted event, as shown in [Figure 9-3](#).

The Rule Event does not have a corresponding property set in a portal application. Rules are often used instead of Campaigns, because they provide more flexibility and power; so creating a rule event property set to trigger a Campaign when a rule is fired is not a likely scenario. However, if you want to create a rule property set to trigger a Campaign when a rule is fired, create an event property set called **RuleEvent.evt** and add the following single, unrestricted string properties: **ruleset-name** and **rule-name**. For instructions on creating property sets, see [Chapter 4](#), “Creating a Property Set”.

For more information on using rules, see the [Rule Event Control](#) and [Chapter 10](#), “Creating Advanced Personalization with Rules”.

Using the DisplayCampaignEvent

If Behavior Tracking is enabled, a `DisplayCampaignEvent` is automatically generated when a Campaign places a content item in a Placeholder. The event property values are written to the `BT_EVENT` table in the database when the event is generated and the event is registered with the Behavior Tracking service as a persisted event, as shown in [Figure 9-3](#).

Using the Display Content Event Control

Using the Display Content Event control with a `<BehaviorTracking:displayContentEvent/>` JSP tag let you generate a Behavior Tracking event when you display a piece of content in a JSP.

See [Table 9-2](#) for details on how get the **document-id** and **document-type** properties.

If Behavior Tracking is enabled, the event property values are written to the `BT_EVENT` table in the database when the event is generated and the event is registered with the Behavior Tracking service as a persisted event, as shown in [Figure 9-3](#).

A Display Content Event does not have a corresponding property set in a portal application. By default, the Campaign service does not listen for these events, so they cannot be used to trigger Campaigns. For more information, see [“Triggering a Campaign” on page 8-31](#).

Using the Display Product Events JSP Tag

The `<productTracking:displayProductEvent>` JSP tag lets you generate a Behavior Tracking event when you display a product from your catalog.

See [Table 9-2](#) for details on how get the **application-name**, **category-id**, **document-id**, **document-type**, and **SKU** properties.

If Behavior Tracking is enabled, the event property values are written to the `BT_EVENT` table in the database when the event is generated and the event is registered with the Behavior Tracking service as a persisted event, as shown in [Figure 9-3](#).

A Display Product Event does not have a corresponding property set in a portal application. By default, the Campaign service does not listen for these events, so they cannot be used to trigger Campaigns. For more information, see [“Triggering a Campaign” on page 8-31](#).

Using the CampaignUserActivityEvent

The `CampaignUserActivityEvent` dispatches an event when a generic Campaign event occurs. This event creates a new `DisplayCampaignEvent` and associates users with specific Campaign and Scenario instances. These properties must be represented to use this event.

If Behavior Tracking is enabled, the event property values are written to the `BT_EVENT` table in the database when the event is generated and the event is registered with the Behavior Tracking Service as a persisted event, as shown in [Figure 9-3](#).

Using the ClickCampaignEvent

Using the `ClickCampaignEvent` with the `ClickThroughEventFilter` generates an event when a user clicks a content item displayed by a Campaign.

Perform the following steps to enable content clicking:

1. Include the appropriate entries in your portal web project's `web.xml` and `weblogic.xml` files, as described in [“Generating Events for Content Clicks” on page 9-13](#).
2. Configure your content items with specific properties, as described in [Chapter 3, “Setting up Content”](#).

If Behavior Tracking is enabled, the event property values are written to the `BT_EVENT` table in the database when the event is generated and the event is registered with the Behavior Tracking service as a persisted event, as shown in [Figure 9-3](#).

Using the ClickProductEvent

Use the `ClickProductEvent` with the `ClickThroughEventFilter` to generate an event when a user clicks a product content item. Product content items typically exist in a catalog or shopping cart. The `ClickProductEvent` lets you capture information such as product category and SKU. Both of those properties must be represented somehow in your content type to use this event.

To enable content clicking, include the appropriate entries in your portal web project's `web.xml` and `weblogic.xml` files, as described in [“Generating Events for Content Clicks” on page 9-13](#).

If Behavior Tracking is enabled, the event property values are written to the `BT_EVENT` table in the database when the event is generated and the event is registered with the Behavior Tracking service as a persisted event, as shown in [Figure 9-3](#).

Using the ClickContentEvent

Use the `ClickContentEvent` with the `ClickThroughEventFilter` to generate an event when a user clicks any content item that was retrieved from the virtual content repository, but not as the result of a Campaign.

To enable event generation on content clicking, include the appropriate entries in your portal web project's `web.xml` and `weblogic.xml` files, as described in [“Generating Events for Content Clicks” on page 9-13](#).

If Behavior Tracking is enabled, the event property values are written to the `BT_EVENT` table in the database when the event is generated and the event is registered with the Behavior Tracking service as a persisted event, as shown in [Figure 9-3](#).

Generating Events for Content Clicks

WebLogic Portal provides predefined events that can be generated when a user clicks a content item in a portal. In particular, the `<BehaviorTracking:clickContentEvent>` and the `<productTracking:clickProductEvent>` JSP tags enable content click events. The `ClickCampaignEvent` also generates content click events. To enable content to be clicked so that an event is dispatched to the Event service, use the `ClickThroughEventFilter`, add the `EventService` to the `web.xml` file and the `weblogic.xml` file, and enable Campaign clickthroughs.

This section contains the following topic:

- [Using the ClickThroughEventFilter](#)

Using the ClickThroughEventFilter

Use the `ClickThroughEventFilter` whenever you use `/ShowBinary` pattern in a URL. `ShowBinary` (which is mapped to the `ShowPropertyServlet`) displays binary web content, such as graphics. Use `/ShowBinary` in the content URL in your JSPs. After you map the `ClickThroughEventFilter` to the `/ShowBinary` URL pattern, use `/ShowBinary` in your JSP as part of the URL with a click event JSP tag. Then, when a user clicks the content, a click content event is generated by the `ClickThroughEventFilter`.

To enable this capability, add the following filter and filter mapping to your portal web project's `web.xml` file:

```
<filter>
  <filter-name>ClickThroughEventFilter</filter-name>
  <filter-class>
    com.bea.pl3n.tracking.clickthrough.ClickThroughEventFilter
  </filter-class>
</filter>

<filter-mapping>
  <filter-name>ClickThroughEventFilter</filter-name>
  <url-pattern>/ShowBinary/*</url-pattern>
</filter-mapping>
```

JSP Example

Assuming you have added the filter mapping to your `web.xml` file, the following JSP code displays a content item from the virtual content repository (that has already been retrieved from an iterator, not shown) and provides the mechanism for click content event generation:

```
<!-- The JSP tag gets the documentId of a content item, which provides
      the ClickThroughEventFilter with the parameters it needs to
      generate an event. The id attribute stores the data retrieved
      by the tag. This JSP tag alone does not generate the event. -->
<BehaviorTracking:clickContentEvent documentId="<%= node.getName() %>"
id="eventInfo" />

<!-- A URL variable uses /ShowBinary to provide the clickable link,
      which is mapped to the ClickThroughEventFilter. The eventInfo variable
      provides the ClickThroughEventFilter with the required event parameters
      when a user clicks the link. -->

<% String url = request.getContextPath() + "/ShowBinary"+node.getPath() +
      "?" + eventInfo;%>

<!-- Now if the user clicks the link, a ClickContentEvent is generated by
      the ClickThroughEventFilter. The ShowBinary servlet displays the
      content from the virtual content repository in its binary form
      (such as a graphic). -->

<a href="<%= url %>">" ></a>
```

Enabling Campaign Clickthroughs

To enable Campaign clickthroughs that trigger the predefined `ClickCampaign` Event, you must configure your content items with specific properties, as described in [Chapter 3, “Setting up Content”](#).

Generating Content Events

Other predefined events track changes made to the virtual content repository or to the repository configuration. The events capture different attributes and use the Behavior Tracking listener and the Behavior Tracking Service to persist the attributes as XML in the `BT_EVENT` table when they are generated, or dispatched. You must enable Behavior Tracking to persist the event attributes (as described in [“Enabling Behavior Tracking” on page 9-19](#)).

The following events track repository changes: `CampaignUserActivityEvent`, `ContentConfigEvent`, `ContentCreateEvent`, `ContentDeleteEvent`, and `ContentUpdateEvent`.

If you want to perform custom event handling when any of the predefined events is dispatched, create a custom event listener, as described in [“Creating Custom Event Listeners” on page 9-38](#).

Using the ContentConfigEvent

The `ContentConfigEvent` dispatches a new event when a user makes a configuration change to the virtual content repository. This event lets you capture information, such as the action that was performed on the repository. The properties must be represented to use this event.

If Behavior Tracking is enabled, the event property values are written to the `BT_EVENT` table in the database when the event is generated and the event is registered with the Behavior Tracking Service as a persisted event, as shown in [Figure 9-3](#).

Using the ContentCreateEvent

The `ContentCreateEvent` dispatches an event when a user adds content to the virtual content repository. This event lets you capture information, such as the content type, the path where the new content was created, the content’s status, and so on. The properties must be represented to use this event.

If Behavior Tracking is enabled, the event property values are written to the `BT_EVENT` table in the database when the event is generated and the event is registered with the Behavior Tracking Service as a persisted event, as shown in [Figure 9-3](#).

Using the ContentDeleteEvent

The `ContentDeleteEvent` dispatches an event when a user removes content from the virtual content repository. This event lets you capture information, such as the content type, the path where the content existed before it was removed, the content’s status, and so on. The properties must be represented to use this event.

If Behavior Tracking is enabled, the event property values are written to the `BT_EVENT` table in the database when the event is generated and the event is registered with the Behavior Tracking Service as a persisted event, as shown in [Figure 9-3](#).

Using the ContentUpdateEvent

The `ContentDeleteEvent` dispatches an event when a user changes content from the virtual content repository. This event lets you capture information, such as the content type, the path where the content existed before it was updated, the content's status, and so on. The properties must be represented to use this event.

If Behavior Tracking is enabled, the event property values are written to the `BT_EVENT` table in the database when the event is generated and the event is registered with the Behavior Tracking Service as a persisted event, as shown in [Figure 9-3](#).

Providing Event Attribute Values

WebLogic Portal's predefined events set many of their attribute values automatically. However, there are attributes that you must set manually in your code. [Table 9-2](#) describes the attributes needed by the predefined events and shows you the methods you can use to set the attributes in your code. You can also use these methods to set attributes in your custom events.

You can get some attributes from other generated events. For example, whenever a Campaign displays a piece of content, a `DisplayCampaignEvent` is generated. The `DisplayCampaignEvent` sets an attribute called **placeholder-id**. (It also sets other attributes.) If you want to set the **placeholder-id** for a custom event, you can get the attribute from the `DisplayCampaignEvent` using the `getAttribute()` method on that event. For example:

```
DisplayCampaignEvent.getAttribute( "aPlaceholderId" );
```

Table 9-2 Getting Attributes for Predefined Events

Event Attribute	How to Get the Attribute
application-name	The name of the enterprise application. All predefined events that use this attribute set it automatically. To set this attribute manually in a custom event, use the following method: <code>com.bea.p13n.events.Event.getApplication()</code> .
campaign-id	The unique ID of a Campaign. All predefined events that use this attribute set it automatically. To set this attribute manually in a custom event, use the following method: <code>com.bea.campaign.CampaignInfo.getUniqueId()</code> .

Table 9-2 Getting Attributes for Predefined Events (Continued)

Event Attribute	How to Get the Attribute
category-id	<p>The category that an item in the catalog belongs to. You must set this attribute manually. To get the category-id in the following way, your catalog must be built on the WebLogic Portal catalog API: <code>com.beasys.commerce.ebusiness.catalog.service.category.CategoryManager.getItemCategories()</code>. If your catalog is built in any other way, get the category-id attribute in the appropriate way.</p>
currency	<p>Gets the type of currency on an item. You must set this attribute manually. To get the currency, your commerce application must be built on the WebLogic Portal Commerce API: <code>com.beasys.commerce.axiom.units.Money.getCurrency()</code>. If your currency is set in any other way, get the currency in the appropriate way.</p>
document-id	<p>The unique virtual content repository ID of the retrieved content item. You could also get the unique document name. The Campaign events set this attribute automatically. You must set it manually for all other events. Content events that have corresponding JSP tags provide a tag attribute. After you have retrieved a content item from the virtual content repository with a Content Selector, a Campaign, a Placeholder, or by any other means, use one of the following to get the document-id:</p> <p><code>com.bea.content.Node.getId()</code> or <code>getName()</code> or Use the <code><cm:getProperty></code> JSP tag to get the cm_uid or cm_nodeName property.</p>
document-type	<p>Type is the name of the virtual content repository type, not the MIME type. The Campaign events set this attribute automatically. You must set it manually for all other events. Content events that have corresponding JSP tags provide a tag attribute. After you have retrieved a content item from the virtual content repository with a Content Selector, a Campaign, a Placeholder, or by any other means, use <code>com.bea.content.Node.getType()</code> to retrieve the content's type.</p>
order-id	<p>The unique ID of a customer's order. To get the order-id, your commerce application must use the WebLogic Portal order framework. Use the following method: <code>com.beasys.commerce.ebusiness.order.Order.getIdentifier()</code>. If your commerce application is built in any other way, get the order-id in the appropriate way.</p>
placeholder-id	<p>The unique ID of the content Placeholder displaying the content. The predefined events that use this attribute set it automatically.</p>

Table 9-2 Getting Attributes for Predefined Events (Continued)

Event Attribute	How to Get the Attribute
quantity	The number of a specific items in a shopping cart. To get the quantity , your commerce application must use the WebLogic Portal shopping cart framework. Use the following method: <code>com.beasys.commerce.ebusiness.shoppingcart.ShoppingCartLine.getQuantity()</code> . If your shopping cart is built in any other way, get the quantity in the appropriate way.
scenario-id	The unique ID of a Campaign scenario that contains the action that was executed. The predefined events that use this attribute set it automatically. To set it manually in a custom event, use the following method: <code>com.bea.campaign.action.Action.getScenarioId()</code> .
session-id	The unique ID of the current session. This is retrieved automatically by the predefined events, which extend the <code>TrackingEvent</code> class, which uses <code>javax.servlet.http.HttpSession.getId()</code> and assigns the return value to the session-id property.
sku	The sku number of the catalog item. To get the sku , you must use the commerce framework to set SKU numbers. Use the following method: <code>com.bea.commerce.ebusiness.tracking.tags.ProductEventTag.getSku()</code> . If your catalog is built in any other way, get the sku in the appropriate way.
total-price	The total price of an order in a shopping cart. To get total-price , your commerce application must use the WebLogic Portal shopping cart framework. Use the following method: <code>com.beasys.commerce.ebusiness.shoppingcart.ShoppingCart.getTotalPrice()</code> . If your shopping cart is built in any other way, get the total-price in the appropriate way.
unit-price unit-list-price	The unit price of an item in a shopping cart. To get unit-price or unit-list-price in the following way, your commerce application must use the WebLogic Portal shopping cart framework. Use the following method: <code>com.beasys.commerce.ebusiness.shoppingcart.ShoppingCartLine.getUnitPrice()</code> . If your shopping cart is built in any other way, get the unit-price or unit-list-price in the appropriate way.
user-id	The ID of the authenticated user. This is retrieved automatically by the predefined events, which extend the <code>TrackingEvent</code> class, and the return value is assigned to the user-id property. You can also use the following method to get the user-id from the request: <code>com.bea.pl3n.usermgmt.SessionHelper.getUserId(request)</code> .

Enabling Behavior Tracking

The default PointBase database in a WebLogic Portal domain (and the SQL scripts used to build a portal database for other database types) include Behavior Tracking tables that are ready to use for storing Behavior Tracking data. However, you must manually activate Behavior Tracking to use Behavior Tracking events.

Note: These steps assume you are working in a development or testing environment with an exploded application. If you try to enable Behavior Tracking for an application that is in an EAR file, the configuration is saved only in memory, and Behavior Tracking will not be enabled if you restart the server. This is because the configuration needs to be written to the `META-INF/wps-config.xml` file. That file is read-only in an EAR.

If you are using a database other than PointBase, see the [Database Administration Guide](#) for instructions on creating a separate database for Behavior Tracking events.

Perform the following steps to activate Behavior Tracking by registering the `BehaviorTrackingListener` class with the Event service:

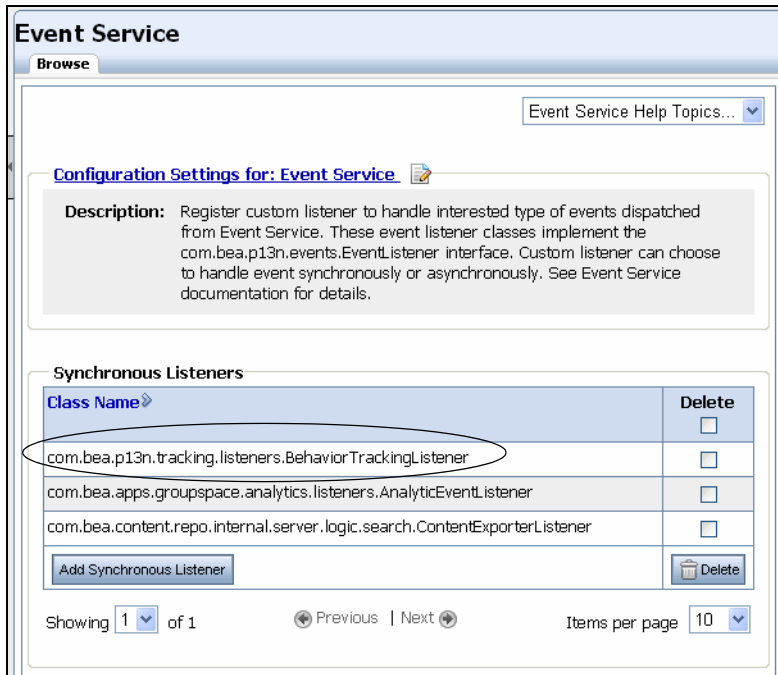
1. Start the Administration Console and log in as a system administrator.
2. Choose **Configuration Settings > Service Administration**.
3. In the Resource Tree, expand the **Personalization** folder and select **Event Service**.
4. In the Synchronous Listeners section, click **Add Synchronous Listener**.
5. Enter the following class in the **Class Name** field:

```
com.bea.p13n.tracking.listeners.BehaviorTrackingListener
```

Note: Synchronous listeners receive events immediately. Asynchronous listeners use a thread scheduler to receive events.

6. Click **Update**. The class appears in the Class Name list, and the `wps-config.xml` file is updated. Behavior Tracking is activated, as shown in [Figure 9-2](#). You do not need to restart the server or redeploy your application.

Figure 9-2 The New Class Appears in the Class Name Field



Configuring Behavior Tracking

By default, Behavior Tracking data is not written to the database immediately when a Behavior Tracking event occurs. The events are stored in a buffer. You can determine how often Behavior Tracking data is moved to the database from the buffer.

Perform the following steps to determine how often Behavior Tracking data is moved to the database:

1. Start the Administration Console and log in as a system administrator.
2. Choose **Configuration Settings > Service Administration**.
3. In the Resource Tree, expand the **Personalization** folder and select **Behavior Tracking Service**.
4. Click **Configuration Settings for: Behavior Tracking Service**. Figure 9-3 shows the Configuration Setting dialog box.

- Modify the settings, as described in [Table 9-3](#). Leave the default values for **Data Source JNDI Name** (the `p13n.trackingDataSource`) and **Custom Persistence Classname** (null). These fields provide the default behavior for moving event data from the buffer to the `BT_EVENT` table in the database. For alternative persistence, see [“Storing Behavior Tracking Data in Other Ways”](#) on page 9-23. For information on the **Persisted Event Types** field, see [“Creating a Behavior Tracking Event Class”](#) on page 9-29.

Figure 9-3 Configuring Behavior Tracking

Configuration Settings for: Behavior Tracking Service

Description: Configure Behavior-Tracking Service parameters, such as class and data source name for persisting behavior

Maximum Buffer Size: 100

Buffer Sweep Interval (seconds): 10

Buffer Sweep Maximum Time (seconds): 120

Data Source JNDI Name: p13n.trackingDataSource

Custom Persistence Classname:

Update Cancel

Application redeployment required for service to use modified values

- Click **Update**.
- Restart the server for your changes to take effect.

Use [Table 9-3](#) when you set your Behavior Tracking settings.

Table 9-3 Behavior Tracking Settings

Maximum Buffer Size	Determines the maximum number of events stored in the buffer before the event data is written to the database. The default value is 100 events. All events are stored in the buffer, but only the events listed in the Persisted Event Types field are written to the database. All others are flushed from the buffer.
Buffer Sweep Interval	Determines how often the events buffer is checked to determine whether the events in the buffer should be persisted to the database. Two conditions trigger events to be moved from the buffer to the database: 1) The maximum buffer size has been reached, or 2) The maximum time allowed in the buffer (Buffer Sweep Maximum) has been exceeded. The default value is 10 seconds.
Buffer Sweep Maximum Time	Sets the maximum time in seconds before the events in the buffer are written to the database (and the non-persisted event types are flushed from the buffer). The default value is 120 seconds.

This section contains the following topics:

- [Adjusting Behavior Tracking for Optimal Performance](#)
- [Storing Behavior Tracking Data in Other Ways](#)
- [Creating a Separate Database for Behavior Tracking Events](#)

Adjusting Behavior Tracking for Optimal Performance

In your development or testing environment, start with a set of baseline values for **Maximum Buffer Size**, **Buffer Sweep Interval**, and **Buffer Sweep Maximum**. Try different values while testing peak site usage with your web application until you find the ideal balance between the number of database operations and the amount of data being stored.

Tip: If you do not use Behavior Tracking, you should disable Event services. See [“Disabling Behavior Tracking”](#) on page 9-49.

Storing Behavior Tracking Data in Other Ways

Behavior Tracking event data, by default, is stored in the database in the `BT_EVENT` table. If you want to persist your event data in a different place or in a different way, such as to a different database table or to a file, create a custom event listener that provides the alternative persistence logic. For information on creating custom listeners, see [“Creating Custom Event Listeners” on page 9-38](#).

Creating a Separate Database for Behavior Tracking Events

If you are using Behavior Tracking, you can improve performance by storing Behavior Tracking data in a separate database. The *Database Administration Guide* contains instructions for creating a separate Behavior Tracking database for each type of database.

Creating Custom Events

If WebLogic Portal’s predefined events do not capture the specific combinations of attributes you need, you can create your own custom events. You can create two types of custom events: Behavior Tracking events and regular events.

For guidance on custom events and what type to create, see [“Understanding When to Create a Custom Event” on page 2-12](#) and [“Understanding When to Use a Predefined Event” on page 2-11](#). Creating a custom event involves creating the Event class and creating the XML Schema.

This section contains the following topics:

- [Creating the Event Class](#)
- [Creating an XML Schema for Behavior Tracking](#)

Creating the Event Class

WebLogic Portal provides the two base event objects that work with the Event Service: `Event` and `TrackingEvent`. These base classes provide the necessary methods required by the Event Service.

When you create an event class you extend one of the base classes, declare the event attributes you want, and pass the event data (such as the event type) to the base class constructor.

This section provides instructions on creating custom regular events and custom Behavior Tracking events.

Creating a Regular Event Class

Create a custom regular event when none of WebLogic Portal's predefined events capture the event attributes you want, and you do not want to use the Behavior Tracking service for persisting event data as XML in the `BT_EVENT` table. You can trigger Campaigns with custom regular events and perform your own event handling if you create a custom event listener.

The steps involve creating a utility project, which is generally used to develop general-purpose Java code that is not directly part of special entities, such as web services, controls, or EJBs.

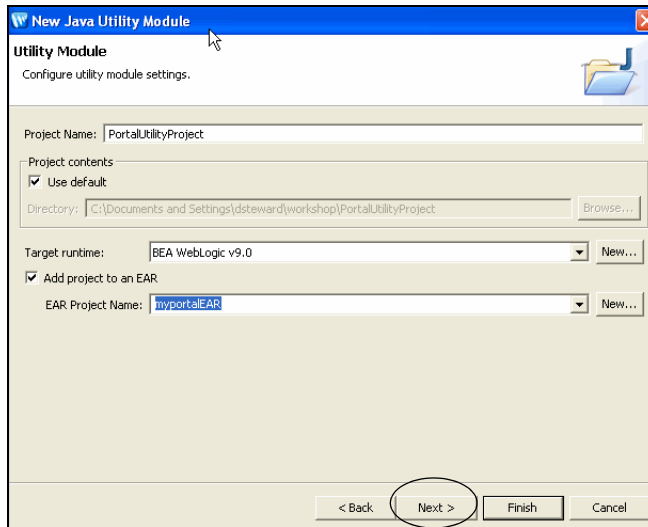
Note: You can also view the sample event class `ResourceDisplayedEvent.java` in the following file: http://edocs.bea.com/wlp/docs81/interm/src/sample_events.zip.

The steps in this chapter refer to the `\src` folder in the Package Explorer View. Your `src` directory might be named differently.

Perform the following steps to create a custom event class:

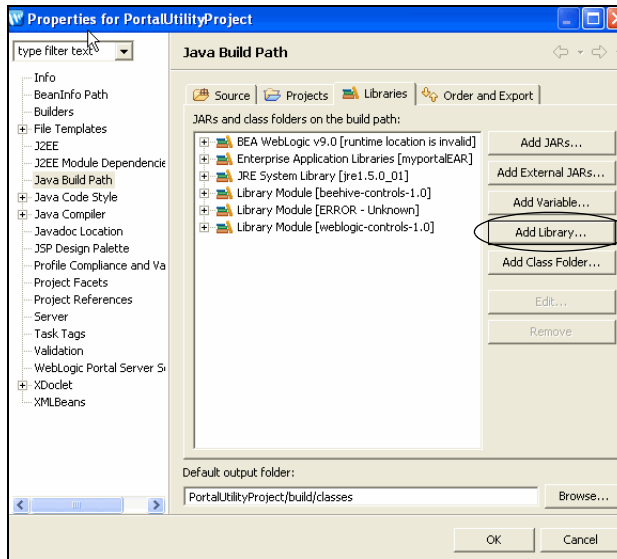
1. Create a utility project in Workshop for WebLogic by performing the following steps:
 - a. In the Portal Perspective, choose **File > New > Project**.
 - b. In the New Project - Select a Wizard window, expand the **J2EE** folder and select **Utility Project**. Click **Next**.
 - c. In the New Java Utility Module - Utility Module dialog, enter a name for the utility project and ensure that the **Use default** check box is selected. Select the **Add project to an EAR** check box and click **Next**, as shown in [Figure 9-4](#).

Figure 9-4 Enter a Project Name



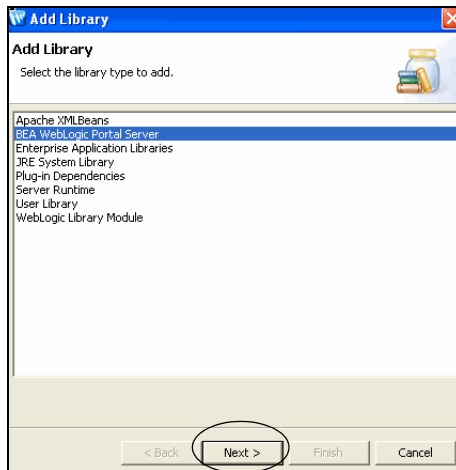
- d. In the New Java Utility Module - Select Project Facets dialog, select the facets that you want to enable and click **Finish**. Your new utility project is automatically associated with your EAR project.
2. To ensure that the project sees the p13n classes as the server will see them, add the WebLogic Portal Server and the p13n-app-lib library module CLASSPATH containers to the project. Perform the following steps:
 - a. In Package Explorer, right-click the portal utility project you created and choose **Properties**.
 - b. In the Properties dialog, select **Java Build Path** and select the **Libraries** tab.
 - c. Click **Add Library**, as shown in [Figure 9-5](#).

Figure 9-5 Add a Library



- d. In the Add Library dialog, select **BEA WebLogic Portal Server** as the library type and click **Next**, as shown in Figure 9-6.

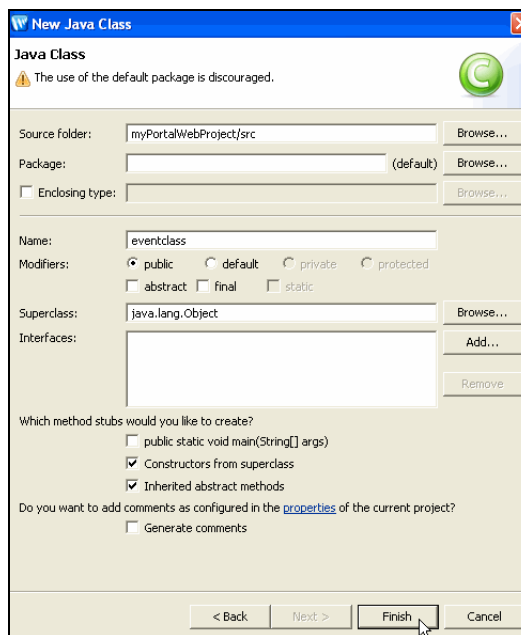
Figure 9-6 Select BEA WebLogic Portal Server



- e. In the Add Library - BEA WebLogic Portal Server dialog, configure the server CLASSPATH entries by selecting **All Configured entries** and clicking **Finish**.

- f. In the Properties dialog, click **Add Library** in the **Libraries** tab.
 - g. In the Add Library dialog, select **WebLogic J2EE Library** and click **Next**.
 - h. In the Add Library -WebLogic J2EE Library dialog, click **Browse** and select **p13n-app-lib** and click **OK**. The **Specification Version**, and **Implementation Version** fields are populated. Select the **Allow newer versions** check box and click **Finish**.
 - i. In the Properties dialog, click **OK**.
3. Make a new Java class by performing the following steps:
 - a. Select your web project in Package Explorer View and choose **File >New > Other**.
 - b. In the New - Select a Wizard dialog, expand the **Java** folder, select **Class**, and click **Next**.
 - c. In the New Java Class - Java Class dialog, enter a **Name** for the new class and for the **Superclass**. Select the **Constructors from superclass** check box and the **Inherited abstract methods** check box and click **Finish**. See [Figure 9-7](#).

Figure 9-7 Enter the Class Name and Superclass



The new class appears in the `\src` directory of your portal web project.

4. Perform the following steps to declare event attribute names and create the event object for the new Java class:

- a. Declare the event attribute names as keys that are passed back to the Event constructor. For example:

```
public static final String FOO_ATTRIBUTE = "fooAttribute";
public static final String SESSION_ID = "session-id";
public static final String USER_ID = "user-id";
```

- b. Create the event object. For example:

```
public MyEvent(
    String fooAttributeValue,
    String user_id,
    HttpServletRequest request,
    HttpSession session
```

Note: If you use events to trigger Campaigns, you must have a string called `user-id` that contains the User's Profile name. You must also have a **request** attribute of type `com.bea.p13n.http.Request`. The **request** attribute, however can be added at runtime with the following code:

```
event.setAttribute("request", new Request(request, true));
```

- c. Add a constructor, such as the one below, and pass the event type back to it:

```
super( TYPE );
```

- d. Declare the event attributes with the following code:

```
setAttribute( FOO_ATTRIBUTE, fooAttributeValue );
setAttribute( SESSION_ID, session_id );
if( user_id != null )
    setAttribute( USER_ID, user_id );
else
    setAttribute( USER_ID, "unknown" );
```

where `fooAttributeValue` is the variable that stores the value you retrieved in your code (not shown here).

5. The Java files in the `\src` folder will be compiled the normal way and deployed as part of the application. You can dispatch the event from a JSP, Java code, or a Page Flow, as described in [“Dispatching Events” on page 9-41](#). If you want to use the event in Campaign definitions, create an event property set for the event, as described in [“Registering Events for Campaigns” on page 9-45](#). If you want to perform custom functionality when the event is generated, create a custom event listener that listens for the event, as described in [“Creating Custom Event Listeners” on page 9-38](#).

Creating a Behavior Tracking Event Class

Create a custom Behavior Tracking event if none of WebLogic Portal's predefined events captures the event attributes you want, and you need to use WebLogic Portal's Behavior Tracking framework to persist event data as XML in the `BT_EVENT` table.

You can use these events in Campaigns and create a custom listener that performs special handling on the event, but unless you want to use the Behavior Tracking framework to store event data as XML, you do not need to create a custom Behavior Tracking event. If you do not want to use the Behavior Tracking service, create a custom regular event as described on “[Creating a Regular Event Class](#)” on page 9-24.

Your Behavior Tracking event works with its own XML schema to store the event data as XML in the `BT_EVENT` database table. Information about that schema must be included in your event class, as described in the following steps.

The steps involve creating a utility project, which is generally used to develop general-purpose Java code that is not directly part of special entities, such as web services, controls, or EJBs.

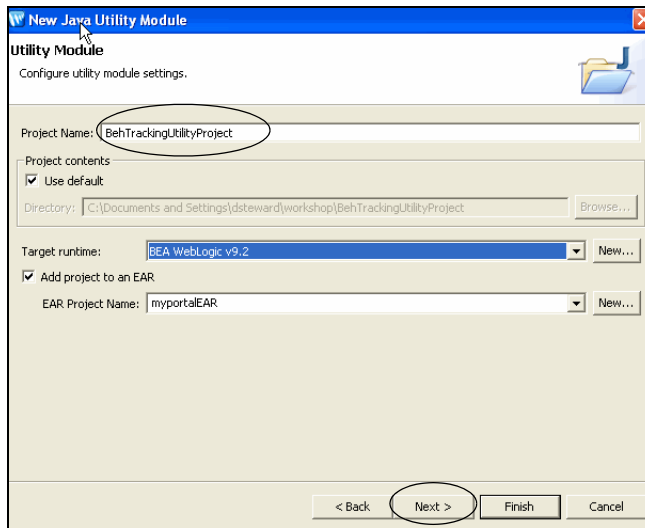
Note: You can also view the sample Event class `ResourceDisplayedEventBT.java` in the following file: http://edocs.bea.com/wlp/docs81/interm/src/sample_events.zip.

The steps in this chapter refer to the `src` folder in the Package Explorer View. Your `src` directory might be named differently.

Perform the following steps to create a Behavior Tracking event class:

1. Create a utility project in Workshop for WebLogic by performing the following steps:
 - a. In the Portal Perspective, choose **File > New > Project**.
 - b. In the New Project - Select a Wizard window, expand the **J2EE** folder and select **Utility Project**. Click **Next**.
 - c. In the New Java Utility Module - Utility Module dialog, enter a name for the utility project and ensure that the **Use default** check box is selected. Select the **Add project to an EAR** check box and click **Next**, as shown in [Figure 9-8](#).

Figure 9-8 Enter a Project Name

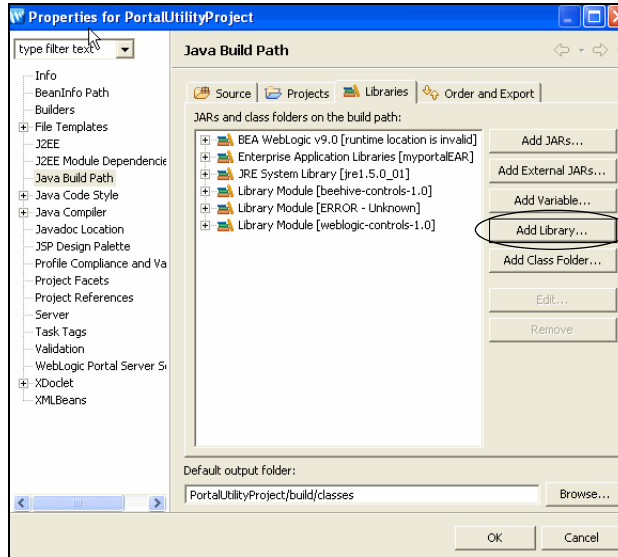


- d. In the New Java Utility Module - Select Project Facets dialog, select the facets that you want to enable and click **Finish**.

Your new utility project is automatically associated with your EAR project.

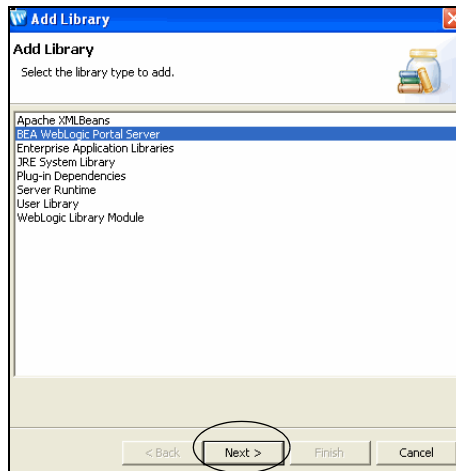
2. To ensure that the project sees the p13n classes as the server will see them, add the WebLogic Portal Server and the p13n-app-lib library module CLASSPATH containers to the project. Perform the following steps:
 - a. In Package Explorer, right-click the portal utility project you created and choose **Properties**.
 - b. In the Properties dialog, select **Java Build Path** and select the **Libraries** tab.
 - c. Click **Add Library**, as shown in [Figure 9-9](#).

Figure 9-9 Add a Library



- d. In the Add Library dialog, select **BEA WebLogic Portal Server** as the library type and click **Next**, as shown in Figure 9-10.

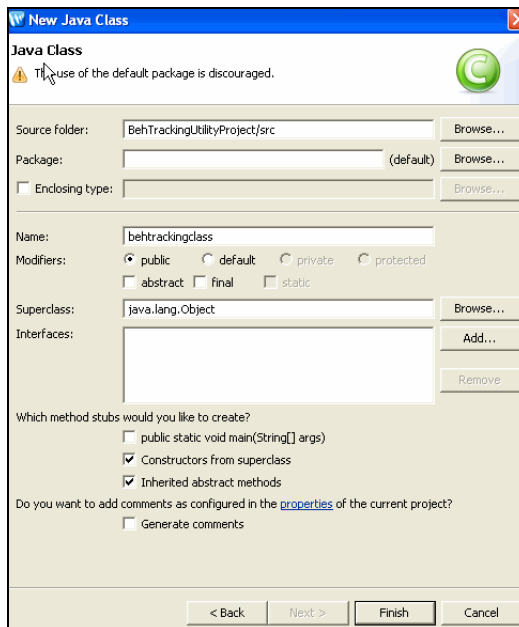
Figure 9-10 Select BEA WebLogic Portal Server



- e. In the Add Library - BEA WebLogic Portal Server dialog, configure the server CLASSPATH entries by selecting **All Configured entries** and clicking **Finish**.

- f. In the Properties dialog, click **Add Library** in the **Libraries** tab.
 - g. In the Add Library dialog, select **WebLogic J2EE Library** and click **Next**.
 - h. In the Add Library - WebLogic J2EE Library dialog, click **Browse** and select **p13n-app-lib** and click **OK**. The **Specification Version**, and **Implementation Version** fields are populated. Select the **Allow newer versions** check box and click **Finish**.
 - i. In the Properties dialog, click **OK**.
3. Make a new Java class by performing the following steps:
- a. Select your web project in Package Explorer View and choose **File >New > Other**.
 - b. In the New - Select a Wizard dialog, expand the **Java** folder, select **Class**, and click **Next**.
 - c. In the New Java Class - Java Class dialog, enter a **Name** for the new class and for the **Superclass**. Select the **Constructors from superclass** check box and the **Inherited abstract methods** check box and click **Finish**. See [Figure 9-11](#).

Figure 9-11 Enter the Class Name and Superclass



The new class appears in the `\src` directory of your portal web project.

4. Perform the following steps to declare event attribute names and create the event object for the new Java class:

- a. Declare the `XML_NAMESPACE` key. This is the namespace URL used by your Behavior Tracking event's XML schema to uniquely identify it. For example:

```
private static final String XML_NAMESPACE =
"http://www.yourdomain.com/myschemas/tracking/mytrackingschema";
```

- b. Declare the name of the XML schema file. For example:

```
private static final String XSD_FILE = "mytrackingschema.xsd";
```

- c. Declare the event attribute names as keys that are passed to the `TrackingEvent` constructor. For example:

```
public static final String SESSION_ID = "session-id";
public static final String USER_ID = "user-id";
public static final String PAGE_LABEL = "pageLabel";
```

- d. Declare the XML schema keys as an array. The schema keys are strings that are passed to the base `TrackingEvent` constructor. These keys are used to get the Behavior Tracking data that is put into the database. List the keys as an array of string objects.

```
private static final String localSchemaKeys[] =
{
    SESSION_ID, USER_ID, PAGE_LABEL
};
```

The `localSchemaKeys` order is important, because it corresponds to the order in which the XML schema needs the event properties for the XML output. An XML file will be invalid if elements are out of order.

The `SESSION_ID` and the `USER_ID` are data elements in the `localSchemaKeys` array that are useful in implementing a tracking event. The `SESSION_ID`, which must not be null, is the WebLogic Server session ID that is created for every session object. The `USER_ID` is the username of the user who triggered the event.

- e. Create the event object. For example:

```
public MyEvent(
    String fooAttributeValue,
    String user_id,
    HttpServletRequest request,
    HttpSession session
```

Note: If you use events to trigger Campaigns, you must have a string called `user-id` that contains the User's Profile name. You must also have a **request** attribute of type `com.bea.p13n.http.Request`. The **request** attribute, however can be

added at runtime with the following code:

```
event.setAttribute("request", new Request(request, true));
```

- f. Call the `TrackingEvent` constructor and pass the required arguments back to it in the required order. For example:

```
{
    super(
        TYPE,
        session,
        XML_NAMESPACE,
        XSD_FILE,
        localSchemaKeys,
        request );
}
```

- g. Declare the event attributes. For example:

```
setAttribute( PAGE_LABEL, pageLabelValue );
setAttribute( SESSION_ID, session_id );

if( user_id != null )
    setAttribute( USER_ID, user_id );
else
    setAttribute( USER_ID, "unknown" );
}
```

The `pageLabelValue` is the variable storing the value you retrieved in your code (not shown here).

5. Register the Behavior Tracking event with the Behavior Tracking service in the WebLogic Portal Administration Console. This event tells the Behavior Tracking listener to handle this type of event.
 - a. Start the Administration Console.
 - b. In the Administration Console, choose **Configuration Settings > Service Administration**.
 - a. In the Resource Tree, expand the Personalization folder and select **Behavior Tracking Service**.
 - b. In the **Configure** tab click **Configure Settings for: Behavior Tracking Service**.
 - c. Enter the name of your Behavior Tracking class in the **Custom Persistence Classname** field (such as `behtrackingclass`).
 - d. Click **Update**.

6. Create an XML schema that determines the structure of the XML document generated by the Behavior Tracking event. See [“Creating an XML Schema for Behavior Tracking” on page 9-36](#). If you want to use the event in Campaign definitions, create an event property set for the event, as described in [“Registering Events for Campaigns” on page 9-45](#). If you want to perform custom functionality when the event is generated, create a custom event listener that listens for the event, as described in [“Creating Custom Event Listeners” on page 9-38](#).
7. The Java files in the `\src` folder will be compiled the normal way and deployed as part of the application. You can dispatch the event from a JSP, Java code, or a Page Flow, as described in [“Dispatching Events” on page 9-41](#). If you want to use the Behavior Tracking event in Campaign definitions, create an event property set for the event, as described in [“Registering Events for Campaigns” on page 9-45](#). If you want to perform custom functionality when the event is generated, create a custom event listener that listens for the event, as described in [“Creating Custom Event Listeners” on page 9-38](#).

Creating an Event With a Scriptlet

You can create an event without writing an event class by using a scriptlet in a JSP. This technique is best suited for simple, Non-Behavior Tracking events that are used to trigger Campaigns. Using this technique for complex events clutters your JSP. You should use this technique in a JSP that has a form that can supply values to event properties.

Perform the following steps to create an event with a scriptlet:

1. In the Portal Perspective in Workshop for WebLogic, create an event property set. For instructions, see [“Registering Events for Campaigns” on page 9-45](#).
2. After you have created the event property set, open the JSP in which you want to create the event.
3. Drag the event property set file into the JSP where you want the event to occur (for example, after the **Submit** button on a form). The scriptlet is generated automatically.

For example, if you create an event property set called **MyEvent.evt** that contains a single, unrestricted attribute called **fooAttribute**, the following scriptlet is generated when you drag the property set file into a JSP:

```
<%
// Generate the Event object here.
// If you have a custom Event subclass for your event type,
// change this code to use it instead

com.bea.pl3n.events.Event event = new
com.bea.pl3n.events.Event("MyEvent");

// fooProperty should be a String
```

```
event.setAttribute("fooProperty", "");

// These attributes are standard to all Events.
event.setAttribute("request", new com.bea.p13n.http.Request(request,
true));

event.setAttribute("user-id",
com.bea.p13n.usermgmt.SessionHelper.getUserId(request));

// Dispatch the Event to the EventService.
com.bea.p13n.tracking.TrackingEventHelper.dispatchEvent(request,
event);

%>
```

The scriptlet automatically gets the **request** and the **user-id**, which are required for triggering Campaigns, and the code for dispatching the event. The dispatch code uses the Behavior Tracking API, but it also dispatches regular events.

You must supply the value for **fooProperty**, which could come from the value of a form field.

4. Save your work by choosing **File > Save**.

If you want to perform custom functionality when the event is generated, create a custom event listener that listens for the event, as described in [“Creating Custom Event Listeners” on page 9-38](#).

Creating an XML Schema for Behavior Tracking

Behavior Tracking events, by default, store their property values in the database as XML. For each type of Behavior Tracking event, the Event service uses a specific XML schema to create the XML. When you create a custom Behavior Tracking event, you must also create an XML schema for the Behavior Tracking service to use.

When creating an XML schema for a custom Behavior Tracking event, consider the following connection points between the schema and your event class:

- **Filename** – The value of the `XSD_FILE` key in your event class must match the name of the actual XSD file.
- **Namespace** – The value of the `XSD_NAMESPACE` key in your event class must match the `targetNamespace` attribute value in your XSD file.

- **Property Order** – The XSD file contains a list of event attributes you want to capture. The order in which these properties are listed in the XSD must match the order they are listed in your event class's `localSchemaKeys[]` array.

For example, if your event class contains this list of schema keys, your XSD file must list those properties in the same order:

```
private static final String localSchemaKeys[] =
{
SESSION_ID, USER_ID, PAGE_LABEL_KEY
};
```

You can view the sample XSD file, `ResourceDisplayedEventBT.xsd`, which is located in the following file: http://edocs.bea.com/wlp/docs81/interm/src/sample_events.zip.

When you use XSD, you need to change only the **targetNamespace** and **xmlns=** attribute values to your namespace, and add your custom event attributes, in order.

You can view the XSDs for WebLogic Portal's predefined events at the following location: `weblogic81\p13n\lib\p13n_app.jar`.

A user might not be associated with an event. In such a case, use the **minOccurs="0"** attribute for the **user-id** attribute in the XSD file. For example:

```
<xsd:element ref="user-id" minOccurs="0"/>
```

Packaging the Schema

After you create the schema, add it to your portal application's `p13n_app.jar` file using the following steps.

1. Back up your `p13n_app.jar` file by creating a copy of it and naming it `p13n_ejb.orig`, for example.
2. In your application directory, temporarily add the `<yourschema>.xsd` file to the `lib/schema/` directory.
3. Add the schema to the `p13n_app.jar` file. In a command window (that has the JAR utility in the environment), switch to the application directory and run the following command:

```
jar uvf p13n_app.jar lib\schema\<yourschema>.xsd
```

The schema is added to the JAR file.

4. Redeploy the `p13n_app.jar` file.

Creating Custom Event Listeners

An event listener serves one purpose: when an event occurs for which the listener is listening, the listener performs some type of programmatic functionality. WebLogic Portal provides the following two listeners that handle events in specific ways:

- **CampaignEventListener** – Listens for all events (except those it is told to ignore in the `listeners.properties` file in the `wps.jar` file) and calls the Campaign service to evaluate and trigger Campaign actions.
- **BehaviorTrackingListener** – Listens for all events registered with the Behavior Tracking service and puts data from Behavior Tracking events in a buffer, where it is later moved into the `BT_EVENT` database table. (You must manually register this listener to activate Behavior Tracking, as described in [“Enabling Behavior Tracking” on page 9-19.](#)) For example, you could create a custom event listener that listens for the `SessionLoginEvent`, `SessionBeginEvent`, and `SessionEndEvent`. You can add the **user-id** field of these events to a list to keep track of who has logged in.

If you create and register a custom Behavior Tracking event, that event is handled by the `BehaviorTrackingListener` and the `CampaignEventListener`. If you create a custom regular event, that event is handled by the `CampaignEventListener`.

However, there may be times when you want to provide more programmatic functionality when events occur, whether the events are custom events or WebLogic Portal’s predefined events. For example, you may want to persist event data to a file or another database table, show related products when a user clicks a product image, or modify a User’s Profile when the user submits a form. For these additional types of functionality, you must create custom event listeners.

WebLogic Portal provides a base event listener object called `EventListener`. This base class, which you must implement in your custom listener, provides two methods for listening for and responding to events:

- The `getTypes()` method – Tells the Event service which types of events that interest the listener.
- The `handleEvent()` method – Lets you insert the custom functionality you want to perform when the listener receives an event that interests it.

The steps to create a custom event listener involve creating a utility project, which is generally used to develop general-purpose Java code that is not directly part of special entities, such as web services, controls, or EJBs.

Note: You can also view the sample event listener `customEventListener.java` in the following file: http://edocs.bea.com/wlp/docs81/interm/src/sample_events.zip.

The steps in this chapter refer to the `\src` folder in the Package Explorer View. Your `src` directory might be named differently.

Perform the following steps to create a custom event listener for regular events or Behavior Tracking events:

1. Create a utility project in Workshop for WebLogic by performing the following steps:
 - a. In the Portal Perspective, choose **File > New > Project**.
 - b. In the New Project - Select a Wizard window, expand the **J2EE** folder and select **Utility Project**. Click **Next**.
 - c. In the New Java Utility Module - Utility Module dialog, enter a name for the utility project and ensure that the **Use default** check box is selected. Select the **Add project to an EAR** check box and click **Next**.
 - d. In the New Java Utility Module - Select Project Facets dialog, select the facets that you want to enable and click **Finish**.

Your new utility project is automatically associated with your EAR project.

2. To ensure that the project sees the `p13n` classes as the server will see them, add the WebLogic Portal Server and the `p13n-app-lib` library module CLASSPATH containers to the project. Perform the following steps:
 - a. In Package Explorer, right-click the portal utility project you created and choose **Properties**.
 - b. In the Properties dialog, select **Java Build Path** and select the **Libraries** tab.
 - c. Click **Add Library**.
 - d. In the Add Library dialog, select **BEA WebLogic Portal Server** as the library type and click **Next**.
 - e. In the Add Library - BEA WebLogic Portal Server dialog, configure the server CLASSPATH entries by selecting **All Configured entries** and clicking **Finish**.
 - f. In the Properties dialog, click **Add Library** in the **Libraries** tab.
 - g. In the Add Library dialog, select **WebLogic Library Module** and click **Next**.


```
    }
}
```

5. The Java files in the `\src` folder will be compiled the normal way and deployed as part of the application. You can dispatch the event listener from a JSP, Java code, or a Page Flow, as described in “[Dispatching Events](#)” on page 9-41. If you want to use the event listener in Campaign definitions, create an event property set for the event, as described in “[Registering Events for Campaigns](#)” on page 9-45.
6. Register the listener with the Event service by performing the following steps.
 - a. In the Administration Console, choose **Configuration Settings > Service Administration**.
 - b. In the Resource Tree, expand the **Personalization** folder and select **Event Service**.
 - c. In the **Browse** tab, click **Add Synchronous Listener** or **Add Asynchronous Listener**.
 - d. Enter the fully qualified class name. For example:


```
com.bea.pl3n.events.custom.listeners.MyEventListener
```

Note: Synchronous listeners receive events immediately. Asynchronous listeners use a thread scheduler to receive events.
 - e. Click **Update**. The listener is registered with the Event service. You do not need to restart the server.

Dispatching Events

With events and listeners in place, you can dispatch those events in your JSPs, Java code, and Page Flows. Dispatching an event means that the Event service sends an event object to any listeners interested in the event. Those listeners, in turn, handle the events in their own ways.

In the sample events provided in the

http://edocs.bea.com/wlp/docs81/interm/src/sample_events.zip, file a sample event called `ResourceDisplayedEventBT`, is dispatched from two portal framework skeleton JSP files: `book.jsp` and `page.jsp`. The `book.jsp` skeleton is responsible for rendering portal book and page navigation (such as tabs), and the `page.jsp` skeleton provides the area for portlets to be displayed.

The code shown in [Listing 9-1](#) is inserted in each of the `book.jsp` and `page.jsp` files. This example uses code from the `page.jsp` file. The code dispatches a `ResourceDisplayEventBT` event when portlets are viewed on a page.

Listing 9-1 Sample Code for Dispatching an Event from a JSP Page

```
<%@ page import="com.bea.pl3n.tracking.TrackingEventHelper,
    examples.events.ResourceDisplayedEventBT" %>
...
ResourceDisplayedEventBT rde;
...
ResourceDisplayedEvent rde = new ResourceDisplayedEvent(
    ppc.getLabel(),
    portletTitle,
    "portlet",
    sessionID,
    userId,
    "true", // portlet is being displayed
    request,
    session );
// New mechanism for dispatching an event in 9.2:
EventService es = TrackingEventHelper.getEventService();
TrackingEventHelper.dispatchEvent(es, rde);
...
```

The code performs the following actions:

- The file imports the custom event class and the `TrackingEventHelper`, which is used to dispatch the event.
- The event class is assigned to the **rde** variable.
- An instance of the event is created, and the attributes retrieved from the JSP are passed in as event arguments in the same order that the event expects them. It does not matter what names are used in the arguments as long as they supply the type of information the event needs.
- The event is dispatched to the Event Service with the `TrackingEventHelper.dispatchEvent (rde)` method.

The event is then sent to the listeners registered to receive it, and the listeners handle the event in their own ways. [Figure 9-1](#) illustrates the event life cycle, and in this example, the skeleton JSP is Item 2 in the diagram.

[“Creating an Event With a Scriptlet”](#) on page 9-35 also describes how to dispatch an event for which you have created no event class. Dispatching events when content is clicked requires special instructions, as described in [“Generating Events for Content Clicks”](#) on page 9-13. Some predefined events have their own dispatch methods. See [“Using Predefined Events”](#) on page 9-7.

Using Events in Campaigns

You can use events to activate the Campaign Service and to make your Campaigns more powerful by triggering Campaign actions based on events and their attribute values.

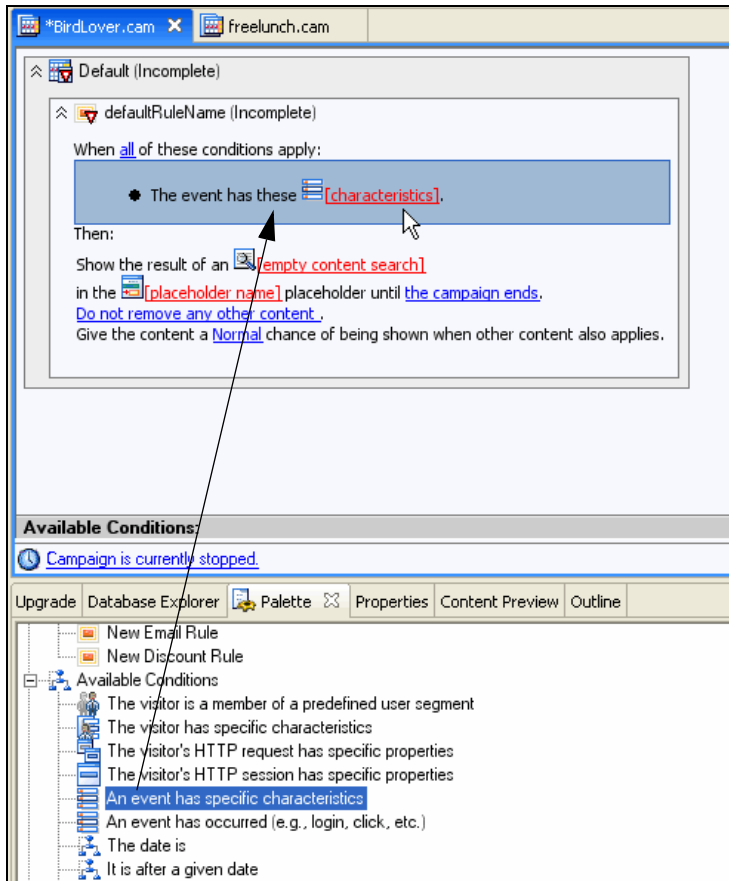
When you use an event in a Campaign, you do not have to explicitly tell the Campaign service about your events. The Campaign listener listens for all events that are not explicitly excluded in the `listeners.properties` file in the `wps.jar` file.

When an event occurs for which the Campaign listener is listening, the listener calls the Campaign service. The Campaign service takes a snapshot of the current request and evaluates the request data against all the Campaign rules you have defined to see if any actions need to be performed.

In addition, you can use events in Campaigns in another way: as part of a Campaign action. For example, you can define a Campaign action that displays personalized content only if the user clicks the Home page in a portal (triggered by some sort of click page event). To use events as part of a Campaign definition, you must create an event property set, as described in [“Registering Events for Campaigns”](#) on page 9-45.

An example of using an event in a Campaign definition is illustrated in [Figure 9-12](#), where a Campaign scenario is triggered if an event has specific property values (characteristics). When you add *An event has specific characteristics* to your Campaign scenario and click the **characteristics** link in the Campaign Editor, you can select the event properties and determine which property values will trigger the Campaign Action to occur. The event property set you created enabled the property selection.

Figure 9-12 Using an Event to Trigger a Campaign Scenario



To control your Campaign, especially when you want personalized content to display, create events that trigger Campaign actions at the key places in your application. For more information on controlling personalized content, see [“Managing Placeholders for Optimal Performance” on page 14-4](#). For instructions on creating a Campaign, see [“Building a Campaign” on page 8-1](#).

This section contains the following topic:

- [Registering Events for Campaigns](#)

Registering Events for Campaigns

If you want to use a custom event to trigger a Campaign, you must create an event property set. The properties you create for the event match the attribute names defined in your event class.

For example, the `sample ResourceDisplayedEvent` class uses the following properties: **resourceId**, **resourceLabel**, **resourceType**, **session-id**, **user-id**, and **resourceSelected**. In your event property set, you can define properties for any or all of those attributes, but the property names must exactly match the event attribute names.

For instructions on creating event property sets, see [“Creating Custom Events” on page 9-23](#). For instructions on registering event property sets, see [“Enabling Behavior Tracking” on page 9-19](#).

Changing Event Properties

If you create, modify, or delete event property sets after an application is deployed, you must update those property set definitions in the database using the BEA Propagation Utility. For more information, see the [Production Operations Guide](#).

Debugging the Event Service

You can debug the Event service and review the console output.

Perform the following steps to debug the event service:

1. To debug the Event service, create the following file:
`weblogic81\portal\debug.properties`.
2. Add the following to the file and modify the settings accordingly. These settings provide server console output for you to review:

```
usePackageNames: on
com.bea.p13n.cache: on
# Turns on debug for all classes under events
com.bea.p13n.events: on
# com.bea.p13n.events.internal.EventServiceBean: on
# Turns on debug for all classes under
# com.bea.p13n.tracking: on
com.bea.p13n.tracking.internal.persistence: on
# Selectively turn on classes
com.bea.p13n.mbeans.BehaviorTrackingListener: on
com.bea.p13n.tracking.listeners.BehaviorTrackingListener: on
com.bea.p13n.tracking.SessionEventListener: on
```

Tip: Events will fire for a content repository that was upgraded to 9.2 (unless you turned event tracking turned off at the repository level). Events can include repository configuration changes, as well as content additions, updates, and deletions to the repository. See the [Upgrade Guide](#) for more information on performing an upgrade.

Tracking Content Changes

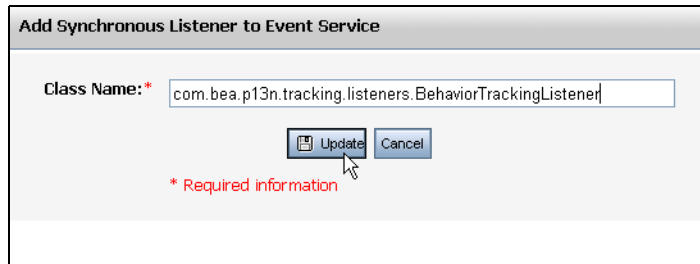
You can use content events to track content changes to your virtual content repository and modifications to the repository's configuration. Content changes include who added, updated, or deleted content or content properties and the date and time the change was made in the repository. You can also track who performed the changes (including the date and time the changes were made) to the repository's configuration, its content types, or content workflow. For more information on these events, see [“Generating Content Events” on page 9-14](#).

You could also configure a content event to watch for content changes and then perform an action. For example, when a user adds a resume document to the content repository, an e-mail is sent to the HR Director.

These content events are saved in the Behavior Tracking database for historical tracking. For more information on content management, see the [Content Management Guide](#).

Perform the following steps to track content changes:

1. Start the Administration Console.
2. Choose **Configuration Settings > Service Administration**.
3. In the Resource Tree, expand the Personalization folder and select **Event Service**.
4. Click **Add Synchronous Listener**.
5. Enter the listener name in the **Class Name** field. To capture content repository changes, enter `com.bea.p13n.tracking.listeners.BehaviorTrackingListener`. See [Figure 9-13](#).

Figure 9-13 Identify the Behavior Tracking Listener

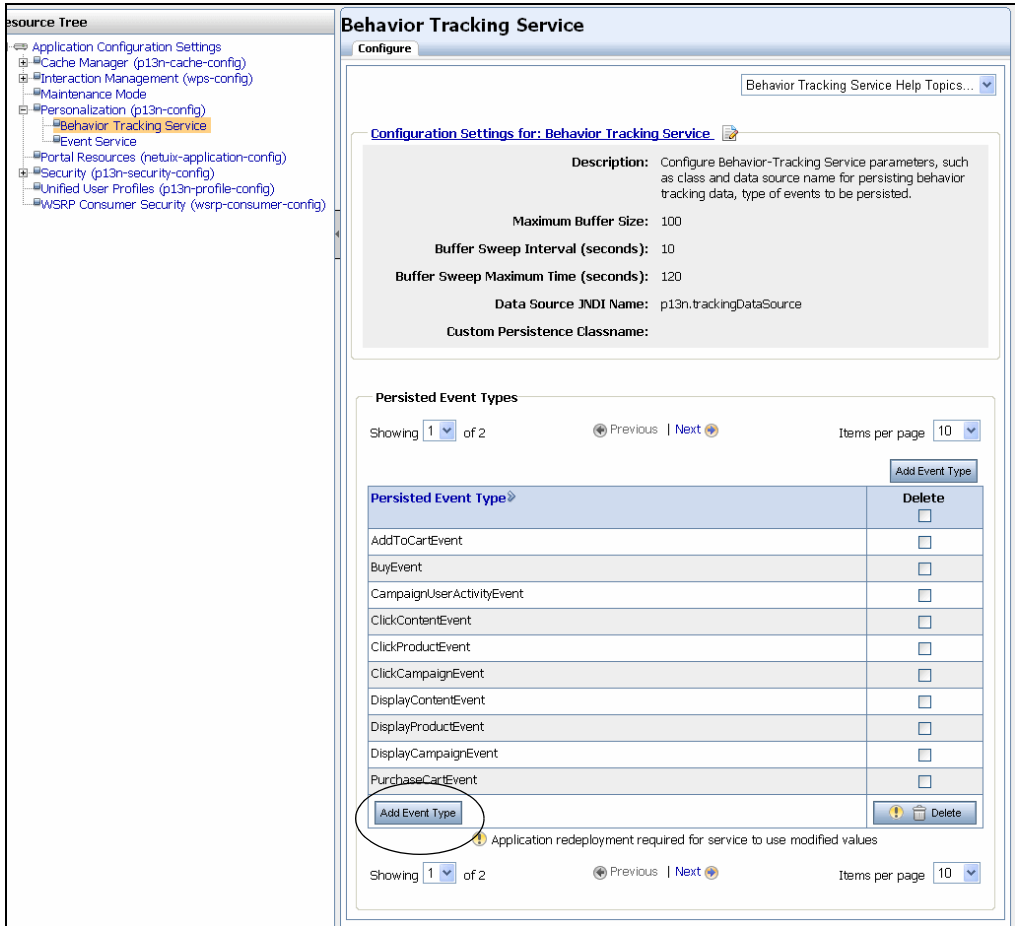
Add Synchronous Listener to Event Service

Class Name: *

* Required information

6. Click **Update**.
7. In the Resource Tree, select **Behavior Tracking Service**.
8. In the **Configure** tab, click **Add Event Type**, as shown in [Figure 9-14](#).

Figure 9-14 Click Add Event Type



9. In the **Persisted Event Type** field, enter the events for the content change you want to track. For example, you could add the `ContentCreateEvent` to determine new content that was added to the content repository, who added it, and when.

10. Click **Update**.

11. Refresh the display by clicking **Refresh Tree** in the Resource Tree.

Tip: To view the tracked changes to the content repository, you can create a log file of the repository content changes that are contained in the Behavior Tracking database tables, or you could enable listeners to provide specific logged output.

Disabling Behavior Tracking

You can disable the persistence of Behavior Tracking events by unregistering the Behavior Tracking listener or removing individual events.

This section contains the following topics:

- [Unregistering the Behavior Tracking Listener](#)
- [Removing an Individual Event](#)

Unregistering the Behavior Tracking Listener

Perform the following steps to unregister the Behavior Tracking listener:

1. Start the Administration Console.
2. In the Administration Console, choose **Configuration Settings > Service Administration**.
3. In the Resource Tree, expand the **Personalization** folder and select **Event Service**.
4. In the **Configure** tab, locate the Behavior Tracking listener and select the **Delete** check box next to it.
5. Click **Delete**.
6. Refresh the display by clicking **Refresh Tree** in the Resource Tree.

Note: Events could still be triggered to fire by the application, but they are not persisted to the database table.

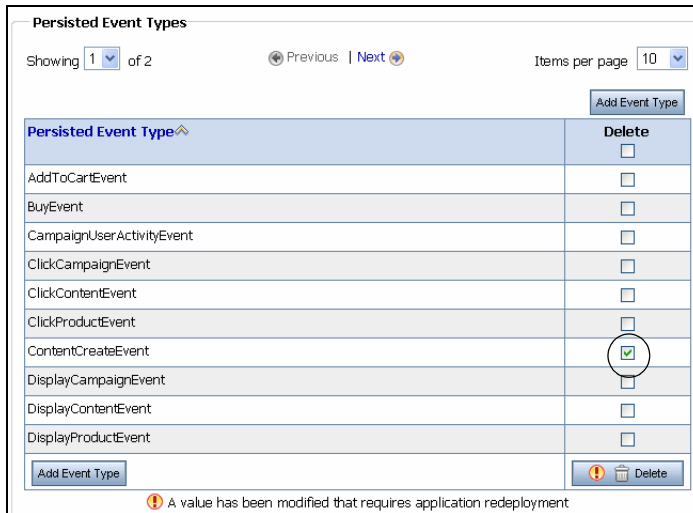
Removing an Individual Event

Perform the following steps to remove default events:

1. Start the Administration Console.
2. In the Administration Console, choose **Configuration Settings > Service Administration**.

3. In the Resource Tree, expand the **Personalization** folder and select **Behavior Tracking Service**.
4. In the **Configure** tab, select the **Delete** check box for each event in the Persisted Event Types section that you want to remove, as shown in [Figure 9-15](#).

Figure 9-15 Select the Delete Check Box



5. Click **Delete**.
6. You must redeploy your application for the changes to take effect.

Creating Advanced Personalization with Rules

Developing Personalization using User Segments, Campaigns, Placeholders, and Content Selectors can be done with JSP tags with very little Java coding. There might be times, however, when you want more flexibility in your Personalization. You can achieve this by creating and deploying a rule set, which uses the Rules Controls and the `RulesManager` EJB.

This chapter contains the following sections:

- [Using Rules in Portal Applications](#)
- [Creating a Rule](#)
- [Rules Control Reference](#)

The Rules service can help you create advanced Personalization features, which can help control each user's path through a Page Flow or using runtime information as dynamic input to conditional logic in your code. You must possess a working knowledge of XML and schemas (an advanced version of DTDs), as well as an intermediate understanding of Java development.

Using Rules in Portal Applications

WebLogic Portal provides a set of tools to personalize the user experience in your portal applications. You have control over the content each user sees, the automatic e-mail messages each receives, and, in a commerce application, the type of discounts each user gets. To achieve these Personalization results, you create User Segments, Content Selectors, and Campaigns in Workshop for WebLogic. Developing Personalization with these tools involves very little Java coding because you can use JSP tags. After this type of Personalization is developed, portal

administrators can use the WebLogic Portal Administration Console to modify the behavior of the Personalization with no coding at all.

There may be times, however, when you want even more power and flexibility in the Personalization you develop. For example, you may want to use Personalization to control each user's path through a Page Flow or use run-time information as dynamic input to conditional logic in your code.

You can access the Rules Service by using the following two types of components:

- **The Rules Controls** – The two rules controls (the Rules Executor Control and the Rules Manager Control) are used in Page Flows or Web services, and provide a convenient way to add rules functionality to your application without writing code. For example, using a drag-and-drop interface, you can add a Rules Control to a Page Flow, select the methods in the control you want to use, and configure the control in the Workshop for WebLogic Property Editor. For more information, see the [Controls Javadoc](#).
- **The RulesManager EJB** – The Rules Controls, which delegate calls to the underlying `RulesManager` EJB, are the preferred way to interact with the Rules Service. However, if you want to use the Rules Service somewhere besides a Page Flow or Web service, you can use the `RulesManager` EJB directly in your code to access the Rules Service.

This overview section includes the following topics:

- [Choosing Personalization Components](#)
- [Understanding the Rules Service](#)

Choosing Personalization Components

[Table 10-4](#) describes the personalization tools provided by WebLogic Portal. User Segments, Campaigns, Content Selectors, and Personalization JSP tags are described only to highlight the increased programmatic power you have by directly accessing the Rules Service.

The Input Objects and Action columns in [Table 10-4](#) show the flexibility and power you have with the rules controls and the `RulesManager` EJB.

Table 10-4 WebLogic Portal Personalization Components

Component	Description	Input Objects	Action (if the input objects match the rules criteria)
User Segments	<p>Dynamically assign users to a grouping, or segment, when the users meet specific conditions.</p> <p>Segment rules are created in Workshop for WebLogic with the User Segment Editor. You can modify rules in the WebLogic Portal Administration Console.</p>	Segment rules can be defined with User Profile properties, HTTP session or request properties, and date or time values and ranges.	One action: If all conditions evaluate to true, the user is considered a member of the segment. Segments can be used in Campaigns, Content Selectors, and in the <code><pz:div></code> JSP tag.
Campaigns	<p>Trigger personalized actions to occur for users who meet specific conditions or perform specific actions.</p> <p>Campaign rules are created in Workshop for WebLogic with the Campaign Editor. Rules are modifiable in the WebLogic Portal Administration Console.</p>	Campaign rules can be defined with User Segments, User Profile properties, HTTP session or request properties, event characteristics, date or time values and ranges, shopping cart or catalog conditions, and random sampling.	Up to three types of actions: Show a single personalized content item, automatically send a predefined e-mail, provide a discount.
Content Selectors	<p>Show specific content items to users who meet specific conditions.</p> <p>Content Selector rules are created in Workshop for WebLogic with the Content Selector Editor. You can modify rules in the Administration Console.</p>	Content Selector rules can be defined with User Segments, User Profile properties, HTTP session or request properties, and date or time values and ranges.	One action: Show one or more personalized content items.
Personalization (Interaction Management) JSP tags	Some of these tags are used to render the results of segment, Campaign, and Content Selector rules.	Displayed content is based on User Segment, Campaign, or Content Selector rules.	One action: Show personalized content items.

Table 10-4 WebLogic Portal Personalization Components (continued)

Component	Description	Input Objects	Action (if the input objects match the rules criteria)
Rules Controls	<p>The Rules Executor control lets you evaluate any input objects (such as a user’s profile properties) against a predefined set of rules (rule set). If the rules evaluate to “true” based on the input objects, any predefined action(s) can be triggered (such as assigning the user to a certain classification). The Rules Manager control lets you look up information about rule sets. The rules controls serve as an interface to the RulesManager EJB.</p> <p>You add and configure rules controls in your Page Flows or Web services with a graphical user interface in Workshop for WebLogic. You create rules manually in XML.</p>	<p>Unlimited types of input objects: You can use the rules you create in XML to evaluate any object put into working memory (See “Invoking the Rules Service to Evaluate Objects” on page 10-16).</p>	<p>Unlimited types of actions: Can filter objects in working memory (see “Filtering the Results” on page 10-22) and perform any action defined in the rule set XML.</p>
RulesManager EJB	<p>Provides the same capabilities as the rules controls in areas of your application other than Page Flows and Web services. Using the RulesManager EJB to access the Rules Service involves Java coding.</p>		

Understanding the Rules Service

The Rules Service reads objects you have put into working memory and evaluating those objects against a set of rules you have predefined in an XML file. (Working memory is the place where objects are temporarily stored as the Rules Service is processing the rules.) If the objects in memory match the conditions defined in the rule set (which can be made up of multiple rules), the corresponding rule set actions are triggered. For example, if you put a user’s credit score into

working memory (from the User Profile, from the return of a Web service calculation, or any other way), a rule in the rule set can be defined in XML to perform the following action: *If the user has a credit score equal to or greater than 10, classify that user as a 'gold customer'.*

You can use the results of this rule processing in your applications any way you choose. For example, if you are developing a Page Flow, you can send a “gold customer” to the `gold.jsp` and send all other customers to another JSP.

- **Rules Controls** – WebLogic Portal provides two rules controls that you can use to invoke the Rules Service from a Page Flow or Web service. The Rules Executor control lets you evaluate objects in working memory against a rule or set of rules, filter the results, and perform actions if the rules evaluate to `true`. The Rules Manager control provides methods for getting rule set information.

Note: The Rules Manager control is most useful as a rules development debugging tool.

- **RulesManager EJB** – The `RulesManager` EJB is the interface into the Rules Service. The rules controls delegate calls to the `RulesManager` EJB. Use the `RulesManager` EJB if you want to use the Rules Service in code outside of a Page Flow or Web service.

Using the Rules Service

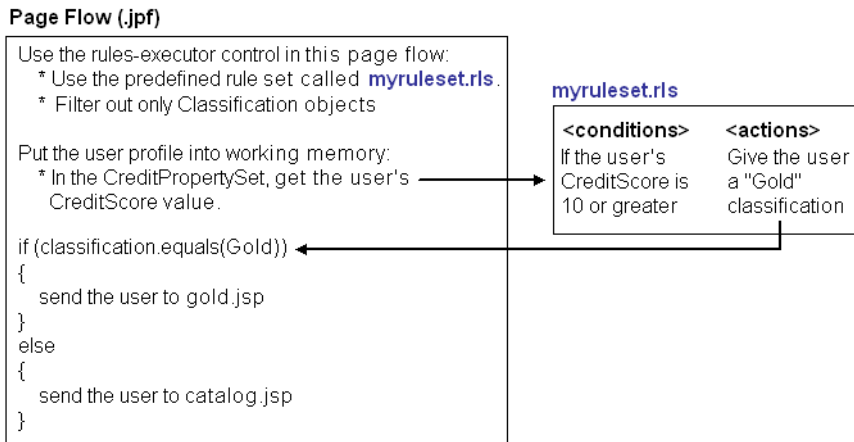
The Rules Service is based on the Rete algorithm, which is optimized for forward chaining reasoning. In the rule evaluation process outlined in the following steps, the Rules Executor control is used as an example:

1. The Portal Rules Service is initialized, creating its working memory.
2. The Rules Executor control will identify which rule set to use, which rules to evaluate (the default is `all`), and optionally, whether to filter the results. These are all parameters that can be configured on the control.
3. The developer creates and adds objects to working memory. Example objects could include the User's Profile, the Request, and so on. These parameters are passed in as an argument to the rule control's `evaluate*()` method.
4. The Rules Service is invoked by the Rules Executor control and uses the following algorithm:
 - a. **Match** – Evaluates the left hand side (LHS) of the rules to determine which are satisfied given the current contents of working memory.
 - b. **Conflict resolution** – Selects one rule with a satisfied LHS. If no rules satisfied the LHS, the interpreter is stopped.

- c. **Act** – Performs the actions in the right hand side (RHS) of the selected rule.
 - d. **Repeat the process** – Go to [step a](#).
5. The Rules Service fires repeatedly, executing rules according to the state of the input objects and rule conditions. Only one rule can be fired at a time. As conditions are met and rules are fired, more objects may be added to working memory for evaluation.
 6. After the Rules Service has reached a state where no more rules will fire, it stops. In addition to the original input objects, new objects created as a result of the rule evaluation may also be in working memory.
 7. Because the input objects are part of the results, you may choose to filter the results based on a class. For example, you can specify that only results of Java class `com.bea.p13n.usermgmt.profile.ProfileWrapper` are returned.
 8. The objects are returned to the caller, who then decides what to do with the returned data. For example, the user may be directed to a new page, or the User's Profile might have its properties updated.

[Figure 10-16](#) provides a basic illustration of the rule evaluation process with the Rules Executor control used in a Page Flow. The returned results from the Rules Service process are used to determine the user's path through the Page Flow. In the figure, natural language is used instead of code for illustration purposes. (To see the actual parameterization and invocation of the control in a Page Flow, see [“Using the Control to Determine the User's Path in the Page Flow” on page 10-19.](#))

Figure 10-16 Using Rules to Control a Page Flow



Understanding the Advantages of Using the Rules Service

The Rules Service is more dynamic than a simple conditional in your code. After a Web application or some other application component has been hard-coded with condition statements (if/then), there is no way to change that without recompiling the code and re-deploying the application. In comparison, rules can be changed and loaded as the Portal server is running. This means the administrator may get the business logic from domain experts, formulate a rule to reflect that logic, and load the rule into the application without ever having to stop the server.

Creating a Rule

This section shows you how to develop personalization using the rules controls and RulesManager EJB. The following steps are involved and are described in this section:

- [Creating a Rule Set](#)
- [Deploying a Rule Set](#)
- [Adding Objects to Working Memory](#)
- [Invoking the Rules Service to Evaluate Objects](#)
- [Filtering the Results](#)

- [Using the Results in Your Application](#)

Creating a Rule Set

Rule sets are sets of instructions written in XML that the Rules Service uses to evaluate objects in working memory. A rule set says, in essence, “if something in working memory meets these conditions, then do this.”

WebLogic Portal does not have a rules editor, so you must create rule sets manually. Rule sets must conform to particular schema. (The rule set schemas are located in the `p13n_app.jar` file in the `<WL_Home>\common\p13n\lib` directory.) The language of the rules is actually a usage of the WebLogic Portal expressions package, extended to meet additional requirements for the Rules Service. See the `com.bea.p13n.expression.operator.*` packages in WebLogic Portal JavaDoc for descriptions of the expressions you can use.

Rule set XML files, which must end in `.rls`, all contain the following required elements:

- **The `<rule-set>`** – Includes all references to schema used in the rule set.
- **The `<rule>`** – Contains the definition of the rule, which consists of at least one condition and at least action. A rule set can have more than one `<rule>`.
- **The `<conditions>` and `<actions>`** – Each `<rule>` contains its own if/then clauses that consist of at least one `<condition>` (if) and one or more `<action>` (then). The Rules Service evaluates the objects in working memory against the conditions. If an object meets a condition, the related actions are executed.

[Listing 10-2](#) contains a simple rule set example that says, *If the string ‘Make an Integer 10’ is in working memory, add an Integer object ‘10’ to working memory.*

Listing 10-2 Rule Set that Adds an Object to Working Memory

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- edited with XMLSPY v5 rel. 4 U (http://www.xmlspy.com) by Your Name
(Your Company) -->
<rule-set xmlns="http://www.bea.com/servers/p13n/xsd/rules/core/2.1.1"
xmlns:exp="http://www.bea.com/servers/p13n/xsd/expression/expressions/2.1.1"
xmlns:literal="http://www.bea.com/servers/p13n/xsd/expression/literal/1.0.1"
xmlns:string="http://www.bea.com/servers/p13n/xsd/expression/string/1.0.1"
```

```

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.bea.com/servers/pl3n/xsd/rules/core/2.1.1
rules-core-2_1_1.xsd" is-complete="true">
  <rule is-complete="true">
    <name>Add Integer</name>
    <description>Test Rule</description>
    <conditions>
      <exp:equal-to>
        <exp:variable>
          <exp:type-alias>java.lang.String</exp:type-alias>
        </exp:variable>
        <literal:string>Make an Integer 10</literal:string>
      </exp:equal-to>
    </conditions>
    <actions>
      <add-object>
        <exp:type-alias>java.lang.Integer</exp:type-alias>
        <exp:arguments>
          <literal:string>10</literal:string>
        </exp:arguments>
      </add-object>
    </actions>
  </rule>
</rule-set>

```

Unless you prefer to read schemas and manually construct valid XML according to the schema's rules, use an XML editor such as XMLSpy (which can be installed from the WebLogic Platform product CD). An XML editor reads a schema, as well as all the schemas the schema imports, and shows you elements and attributes that can be added to an XML document at any location, showing you available elements and attributes and helping you create a valid XML rule set.

Tip: The easiest way to create a rule set is to start with an existing one and modify it. Several example rule sets exist on the dev2dev web site at <http://dev2dev.bea.com/products/wlportal81/articles/portalruleservice.jsp>.

Before you create the rule set in XML, write the rule in natural language to understand all its pieces (conditions and actions) and types of data. What is being put into working memory? What conditions do you want the objects to meet, and what should happen (actions) when objects in working memory meet the conditions?

Use the following guidelines to create a rule set (or modify an existing rule set) with an XML editor:

1. Extract the following schemas from the `p13n_app.jar` file in the `<WL_Home>\common\p13n\lib` directory into the same directory where you will create your rule set:

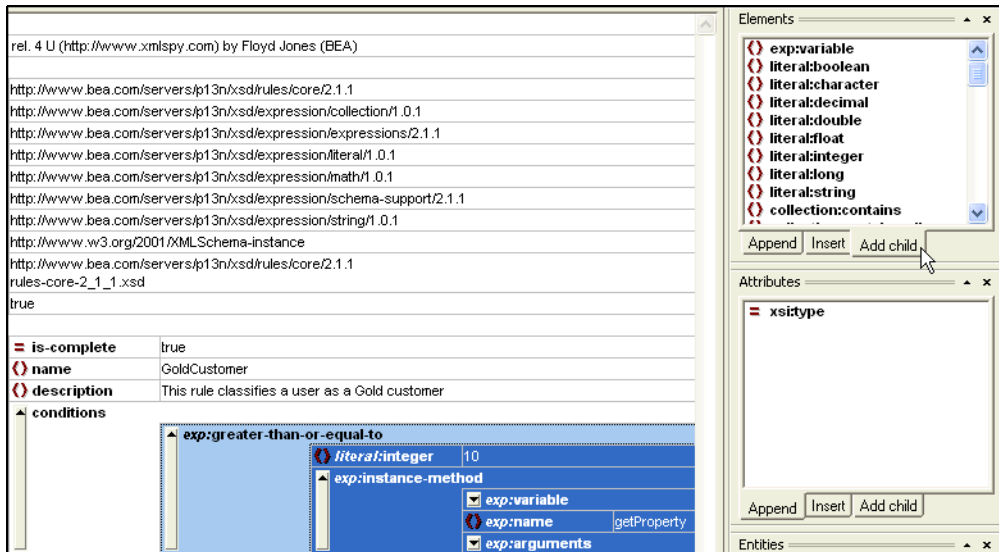
```
lib/schema/expression*.xsd  
lib/schema/rules*.xsd
```

2. Start a new document (or open an existing document) in your XML editor and associate the document with the `rules-core-2_1_1.xsd` schema. This schema also includes imports of other schemas, especially expression schemas, that are helpful in building rule sets.

In a new XML document, your XML editor should automatically insert the XML header, automatically import any schemas listed for import, and insert required base elements, such as `<rule-set>`, `<rule>`, `<name>`, `<conditions>`, and `<actions>`.

3. Select the `<conditions>` and `<rules>` elements and begin building (or modifying) the conditions and rules. [Figure 10-17](#) shows a rule set being built in XMLSpy. With the `<exp:greater-than-or-equal-to>` element selected, the Elements section shows which elements can be added as children. The Attributes section shows the attributes that can be set on the element.

Figure 10-17 Building a Rule with an XML Editor



- After you finish building the rule set, use the XML editor's features to check them for being well-formed. Then validate the rule set against the schema. (In XMLSpy, press **F7** and **F8** to perform these steps.) See [“Working with Invalid Rule Sets”](#) on page 10-13 for instructions on fixing invalid rule sets.
- Save the rule set. (XMLSpy prompts you if you are trying to save an invalid rule set, but you can still save it.)
- Copy the rule set to your portal application's META-INF/data directory, or to one of its subdirectories. For example, create a META-INF/data/rulesets directory and save the rule set there.

Using a Method in a Rule

Table 10-5 shows how an XML rule set uses a method to retrieve a User Profile property value so it can be evaluated by the Rules Service.

Table 10-5 How an XML Rule Uses a Method

<p>Condition</p>
<pre><exp:greater-than-or-equal-to> <literal:integer>10</literal:integer> <exp:instance-method> <exp:variable> <exp:type-alias>User</exp:type-alias> </exp:variable> <exp:name>getProperty</exp:name> <exp:arguments> <literal:string>CreditPropertySet</literal:string> <literal:string>CreditScore</literal:string> </exp:arguments> </exp:instance-method> </exp:greater-than-or-equal-to></pre>
<p>Method to which the condition maps</p>
<p>The ProfileWrapper takes an Object of type User:</p> <pre>ProfileWrapper pw = SessionHelper.getProfile(request); Object value = pw.getProperty("CreditPropertySet", "CreditScore");</pre>

Use the following information when working on the mapping between the XML and the method:

- The <exp:type-alias> identifies the type of object the method will work on. For a list of object type mappings defined in the parser-mapping-type.properties file in p13n_app.jar, see [“Using Type Mappings” on page 10-15](#).
- The <exp:instance-method> indicates a method, and the <exp:name> provides the name of the method.

Note: To invoke methods from a rule, the appropriate classes must be imported in the calling code.
- The <exp:argument> includes two <literal:string> elements that provide the String arguments to the method.

- The `<literal:integer>` identifies a value that the Rules Service uses. The evaluation of the object in working memory (in this case the User Profile `CreditScore` value) determines if the rule's action is fired.
- The result is that the rule's condition (in XML) retrieves the value of the user's `CreditScore`. If the value is greater than or equal to 10, in this example, the associated actions are fired.

Working with Invalid Rule Sets

If a rule set does not validate, you can see the invalid area. Perform the following steps to fix the invalid rule set:

- Verify that you imported all schemas referenced in the `.rls` file. Those schemas are listed at the top of the `*.rls` file. The schemas must be in the same directory as the `.rls` file.
- Ensure that the XML sections defined in your `.rls` file are valid according to the schema. You can open the schema in XMLSpy to check your `.rls` against the schema definitions. In XMLSpy, you can view the schema in Design view for a graphical representation of the schema. You can view the `.rls` in Text view and Enhanced Grid view.

Note: There is no guarantee that a rule set validated in an XML editor will be validated in the Rules Service.

Deploying a Rule Set

This section explains how to deploy a rule set in development (Workshop for WebLogic) and in Staging or Production environments.

This section contains the following topics:

- [Deploying a Rule Set in Workshop for WebLogic](#)
- [Deploying a Rule Set in a Staging or Production Environment](#)

Deploying a Rule Set in Workshop for WebLogic

After you create a rule set and store it in your application's `META-INF/data` directory (or in a subdirectory you create, such as the `META-INF/data/rulesets` directory), the rule set is automatically deployed if the server is running. If the server is not running, the rule set is automatically deployed at server startup. (The `META-INF/data` directory also contains the User Segments, Content Selectors, Campaigns, and other application metadata you have created.) Rule sets must be deployed to the `data` directory, and rule set file names must have an `.rls` extension to be used by the Rules Service.

When you modify a rule set in the `data` directory, the rule set is automatically refreshed on the running server.

Deploying a Rule Set in a Staging or Production Environment

Perform the following steps to add, modify, or remove a rule set that exists in a deployed application:

1. Modify the rule set in the Development environment and replace it in the deployed application. If the application is in a compressed EAR file, you must recreate the EAR file to include the updated rule set and then replace the EAR file on the server. When you replace the EAR file on the server, you do not need to redeploy the application.
2. Update the rule set with the BEA Propagation Utility. See the *Production Operations Guide* for instructions.

Adding Objects to Working Memory

Rule sets must have objects in working memory to evaluate. For example, a rule set might contain a rule that has the following condition: “If the user’s credit score is greater than 10.” This implies there is either the credit score input, or there is a way to get at the credit score. We could add a credit score to working memory in one of two ways:

- Adding the credit score to memory from an integer
- Adding the credit score to memory from a User Profile

Adding a Credit Score to Working Memory from an Integer

You could provide a credit score value in your code in the following ways:

- Directly – For example, `Integer value = new Integer(10);`
- Indirectly – For example, through a form input value:

```
Object [] inputObjects = { value }; (This is a required argument to the evaluate*() methods.)
```

You could then create a rule condition that evaluates the `value Integer`.

Adding a Credit Score to Working Memory from a User Profile

Retrieving a credit score from a User Profile is more flexible and dynamic. First, you would use code to retrieve the User Profile and put it into working memory:

```
ProfileWrapper pw = SessionHelper.getProfile(request);

Object [] inputObjects = { pw }; (This is a required argument to the
evaluate*() methods.)
```

In your rule set, you would then create a condition that uses a method (`getProperty`) that retrieves a specific property (`CreditScore`) from a specific property set (`CreditPropertySet`). See the code example in [Table 10-5](#). The condition in the code example checks to see if the retrieved `CreditScore` is greater than or equal to the `<literal:integer>` value of 10.

Note: The `User` type is actually an alias for an object of class `ProfileWrapper`. This mapping of `User` to `ProfileWrapper`, along with the mappings of other well-known types, are defined in the `parser-mapping-type.properties` file in the `p13n_app.jar` file, shown in [“Using Type Mappings” on page 10-15](#).

Using Type Mappings

The following object type mappings are from the `parser-mapping-type.properties` file in the `p13n_app.jar` file:

Using Mappings for `<type-alias>` Tags

[Listing 10-3](#) shows mappings for `<type-alias>` tags.

Listing 10-3 Mappings for `<type-alias>` Tags

```
User=com.bea.p13n.usermgmt.profile.ProfileWrapper
Classifier=com.bea.p13n.user.Classification
Capability=com.bea.p13n.entitlements.common.Capability
Role=com.bea.p13n.entitlements.common.Role
Context=com.bea.p13n.rules.internal.engine.Context
Email=com.bea.campaign.rules.MailActionDef
Placeholders=com.bea.campaign.rules.AddAdToPlaceholderActionDef
Discount=com.bea.commerce.ebusiness.campaign.AddUserDiscountActionDef
EndScenario=com.bea.campaign.rules.EndScenarioActionDef
CatalogQuery=com.beasys.commerce.ebusiness.catalog.rules.CatalogQueryWrapper
ContentQueryAdvice=com.bea.p13n.content.advislets.ContentQueryAdvice
ShoppingCartFacade=com.beasys.commerce.ebusiness.shoppingcart.ShoppingCartRulesFacade
```

The code examples in [Listing 10-2](#) and [Table 10-4](#) show the `<type-alias>` element with a User type.

Mappings for `<variable>` Tags

[Listing 10-4](#) shows mappings for `<variable>` tags.

Listing 10-4 Mappings for `<variable>` Tags

```
user=com.bea.p13n.usermgmt.profile.ProfileWrapper
request=com.bea.p13n.http.Request
session=com.bea.p13n.http.Session
event=com.bea.p13n.events.Event
randomNumber=java.lang.Number
classification=com.bea.p13n.user.Classification
date=com.bea.p13n.xml.schema.Date
time=com.bea.p13n.xml.schema.Time
timeInstant=com.bea.p13n.xml.schema.TimeInstant
role=com.bea.p13n.entitlements.common.Role
resource=java.lang.String
shoppingCart=com.beasys.commerce.ebusiness.shoppingcart.ShoppingCart
```

Invoking the Rules Service to Evaluate Objects

After you have created a rule set and you have objects in working memory, you can invoke the Rules Service to evaluate the objects in working memory with the rules you created. This section provides an example to show you how to invoke the Rules Service with the Rules Executor control in a Page Flow.

Using an Existing Rule Set

The example used throughout the [Invoking the Rules Service to Evaluate Objects](#) section assumes that a rule set already exists in the `/data/rulesets` directory that classifies users as “GoldCardMembers” or “SilverCardMembers” by reading a User’s Profile (similar to the

example used in “[Adding a Credit Score to Working Memory from a User Profile](#)” on [page 10-14](#)). The Page Flow example in this section shows the User Profile being added to working memory. Since the User’s Profile is needed in this example, you should assume the Profile Control was added to an existing Page Flow to enable the getting and setting of User Profile properties.

This sample also uses the User Login Control for authentication so that the Page Flow knows which User Profile to retrieve.

For instructions on creating a Page Flow and adding a Portal Control to the Page Flow, see the [Javadoc](#).

Inserting the Control in the Page Flow

When you insert a control in a Page Flow (a `.jpf` file in Workshop for WebLogic), all the Actions that are part of that control are available to use.

The Rules Executor Control contains two actions:

- **The evaluateRule Action** – Lets you evaluate the objects in working memory against a single rule in a rule set.
- **The evaluateRuleSet Action** – Lets you evaluate the objects in working memory against all rules in a rule set.

When you are looking at a Page Flow in Action View, you can select a control you have inserted (by selecting the control’s border) and set properties on that control. The Property Editor is a convenient way to send arguments to the `RulesManager` EJB (which interfaces directly with the Rules Service) without writing Java code.

For example, [Table 10-6](#) shows how the Rules Executor Control properties shown in [Figure 10-16](#) map to method and constructor arguments in the `RulesManager EJB`.

Table 10-6 How Control Properties Map to Method and Constructor Arguments

Rules Executor Control properties	RulesManager EJB methods (see com.bea.p13n.rules.manager)
<code>rulesetUri</code> <code>ruleName</code>	<code>evaluateRule(String ruleSetUri, String ruleName, Object[] inputObjects)</code>
<code>filterResults</code> <code>filterClassName</code>	<code>evaluateRule(String ruleSetUri, String ruleName, Object[] inputObjects, ObjectFilter filter)</code>
<code>filterClassNames</code>	<code>evaluateRuleSet(String ruleSetUri, Object[] inputObjects)</code>
	<code>evaluateRuleSet(String ruleSetUri, Object[] inputObjects, ObjectFilter filter)</code>
<code>filterRuleName</code>	Filter constructors (see com.bea.p13n.rules.manager.RuleResultClassFilter) <code>RuleResultClassFilter(String ruleName, Class targetClass)</code> <code>RuleResultClassFilter(String ruleName, Class[] targetClassArray)</code>

The `RulesManager EJB` has the same Actions contained in the Rules Executor Control: `evaluateRule()` and `evaluateRuleSet()`. The difference is that the Rules Executor Control Actions take only one argument—for example, `evaluateRuleSet(Object[] inputObjects)`—and provide the rule and filter arguments through the control properties.

If you set the `filterResults` property to `true` on the Rules Executor Control, the EJB method with the `filter` argument is used and the filtering properties you enter are automatically sent to that argument.

The `filterClassName` and `filterClassNames` properties are different options for populating the `filter` argument (with one or more types of filters). Set either `filterClassName` or `filterClassNames` on the control, but do not set both.

Use the `filterRuleName` property to filter on the results of a specific rule in a rule set that has fired. If you use this property, the `RuleResultClassFilter` constructor is called. Notice that the constructor is overloaded to use either a single class filter (that you entered in the `filterClassName`) property or multiple class filters (that you entered in the `filterClassNames`) property. The result of using the `filterRuleName` property is that you not only filter the results of a specific rule that has fired, you can also filter on specific data types.

Following are more detailed definitions of the control properties:

- **The `rulesetUri` (required) URI of the rule set to use** – This URI is relative to the application's `META-INF/data` directory. For example, if you created a rule set called `myruleset.rls` and stored it in a `data/rulesets` directory, the URI would be `/rulesets/myruleset.rls`. Use the Rules Manger Control to list rule sets and rules.
- **The `ruleName` (optional) Name of the rule to use** –The rule must be contained in the rule set specified in the `rulesetUri` property. If not specified, all the rules in the rule set are evaluated. The default is `null`.
- **The `filterResults` (optional)** – This property determines whether to filter the results after the rules have been evaluated. If the value is `false`, all objects remaining in working memory are returned. The default is `false`. For information on filtering, see [Filtering the Results](#).
- **The `filterClassName` (optional)** – Enter the class name (for example, `java.lang.String`) of the type of results to return. If this is left empty, results are not filtered. Specify this or the `filterClassNames`, but not both.
- **The `filterClassNames` (optional)** – Enter a comma-separated list of class names of the types of results to return (for example, `java.lang.String, java.lang.Integer`). If this is left empty, results are not filtered. Specify this or the `filterClassName`, but not both.
- **The `filterRuleName` (optional)** – Filter the results of a specific rule that was fired. If this is left empty, results from all rules will be returned. Otherwise, results from only this rule are returned. This filter is applied with the Class filters, if those are specified.

Understanding the Benefits of Using the Control

Understanding how properties map to methods and constructors can help you understand the benefits of using the Rules Executor control. Filling in property values provides the following benefits:

- You do not have to write Java code
- If you are filtering the results, the filter is constructed automatically for you

For more information on the Rules Executor Control, see the [Controls Javadoc](#).

Using the Control to Determine the User's Path in the Page Flow

After you add the Rules Executor Control to the Page Flow, set the properties on the control, and select one of the control's **execute*** actions to use, you should verify that all other prerequisite

details are in place (see [Using an Existing Rule Set](#)). Then you can add code to the Page Flow that sends users to a different page depending on the classification they receive from the rule evaluation process.

The sample Page Flow code in [Listing 10-5](#) shows how a user is directed to a particular page based on the results from the Rules Service. A similar sample is included in *The Portal Rules Service* on dev2dev at

<http://dev2dev.bea.com/products/wlportal81/articles/portalaruleservice.jsp> (in the `examples/pageFlows/classifyAndFlow` subdirectory).

Listing 10-5 Sample Code to Direct a User to a Page, Based on the Results of the Rules Service

```
public class Controller extends PageFlowController
{
    /**
     * @common:control
     */
    private com.bea.p13n.controls.login.UserLoginControl
userLoginControl;
    /**
     * @common:control
     */
    private com.bea.p13n.controls.profile.ProfileControl
myProfileControl;
    // The Rules Executor control is added. Properties are configured
    // in the Property Editor.
    // This is all done in the Page Flow's Action View.
    /**
     * @common:control
     * @jc:rules-executor
    filterClassName="com.bea.p13n.user.Classification"
    filterResults="true" rulesetUri="/rulesets/myruleset.rls"
    */
    private com.bea.p13n.controls.rules.RulesExecutorControl
myRulesExecutorControl;
    /**
     * @jpf:action
     * @jpf:forward name="default" path="default.jsp"
     * @jpf:forward name="goldCard" path="goldCard.jsp"
     * @jpf:forward name="silverCard" path="silverCard.jsp"
     * @jpf:catch
    type="com.bea.p13n.controls.exceptions.P13nControlException"
    path="error.jsp"
     * @jpf:forward name="error" path="error.jsp"
     */
}
```



```

        protected Forward evaluateRuleSetAction(EvaluateRuleSetActionForm
form)
        throws P13nControlException
        {
// Start with an empty list into which we add objects to populate
// the working memory of the Rules Service
            List wmObjects = new ArrayList();
            ProfileWrapper pw =
myProfileControl.getProfileFromRequest(this.getRequest());
            if ( pw == null)
            {
                throw new P13nControlException("Undable to retrieve profile
from
                request. " + "Make sure PortalServletFilter is configured
in web.xml for an anonymous user, " + "or that a user
                has logged in.");
            }
// This one will be the condition that fires the rule
            Integer value = new Integer(6);
            myProfileControl.setProperty(pw, "FooPropertySet", "CreditScore",
value);

            wmObjects.add(pw);
// Evaulate all rules in the rule set. Parameters have been declared on the
// control in the Page Flow Property Editor (in Action View).
            Iterator iter =
myRulesExecutorControl.evaluateRuleSet(wmObjects.toArray());
            List results = new ArrayList();
// Let's say we're looking for GoldCardMembers
            Classification goldCardMembers = new
Classification("GoldCardMembers");
            Classification silverCardMembers = new
Classification("SilverCardMembers");
// And we'll direct them to a certain page depending
// on how the rule evaluates
            Classification classification = (Classification)iter.next();
// Now you would do something with that,
// like show them a different page
            if (classification.equals(goldCardMembers))
            {
// Direct them to high-price stuff
                return new Forward("goldCard");
            }
        }

```

```
        else if (classification.equals(silverCardMembers))
        {
// Direct them to lower-price stuff
            return new Forward("silverCard");
        }
// Otherwise, it defaults. Something went wrong.
// Check the rule conditions or turn off filtering on the control
// to see what's in working memory
        }
    }
    return new Forward("default");
}
```

If you want to use only a specific type of object in working memory after the Rules Service has stopped, you can filter the objects in working memory. Filtering is set using Java types. On the Rules Executor Control, you can set the filter type in the Property Editor.

Get more information on the following subjects:

- **Filter Type** – See [Filtering the Results](#)
- **The Rules Executor Control** – See the *Controls Javadoc*.
- **The RulesManager EJB** – See the *Javadoc*.
- **Rules controls** – See the *Portal Rules Service* on dev2dev at <http://dev2dev.bea.com/products/wlportal81/articles/portalsruleservice.jsp>

Filtering the Results

When you execute the Rules Service to evaluate objects in working memory, as described in “[Invoking the Rules Service to Evaluate Objects](#)” on page 10-16, you can filter the objects in working memory when the Rules Service has stopped running to return only the objects of a specific type.

Objects exist in working memory as a result of one of the following actions:

- The caller puts them there (in the `inputObjects` array)
- A new object is instantiated in one of the rules’ actions

When the Rules Service has stopped, several objects might remain in working memory, including those the user initially added. For example, your rule may instantiate a new **Classification** object into working memory if the rule evaluates to `true`. Another example is that a rule action might have updated the User's Profile, so you need to retrieve the profile from working memory.

When the Rules Service's API executes a rule, it returns an Iterator over the entire contents of working memory *unless you filter the results*. If you are looking only for **Classification** objects, then you can specify a filter that returns only **Classification** objects. You can design this filter based on a single class name, multiple class names, or a given rule, as described in [“Inserting the Control in the Page Flow” on page 10-17](#).

When you implement the Rules Executor Control, the control takes care of constructing the filter automatically. You need to specify whether to filter and the filter class names as control properties. The filter is applied for you automatically by the control, if you specify this.

Filtering with the RulesManager EJB

Filtering with the `RulesManager` EJB is a more cumbersome than filtering with the Rules Service, because you must design the filter yourself with the `RulesManager` EJB. [Listing 10-6](#) shows how to design a filter.

Listing 10-6 Design a Filter with the RulesManager EJB

```
String filterRuleName = null;

Class filterClass = com.bea.p13n.user.Classification.class;

ObjectFilter filter = new RuleResultClassFilter(filterRuleName,
filterClass);

Class [] filterClasses = { java.lang.String.class,
com.bea.p13n.usermgmt.profile.ProfileWrapper.class};

ObjectFilter filter = new RuleResultClassFilter(filterRuleName,
filterClasses);
```

The filter can then be used as part of the `RulesManager` EJB, as shown in the following example:

```
public Iterator evaluateRule(String ruleSetUri, String ruleName, Object[]
inputObjects, ObjectFilter filter)
```

If you filtered the results, the Iterator should only contain results of the class types you specified. The code sample in [Listing 10-7](#) shows a Classification object of **SilverCardMembers**.

Listing 10-7 Sample Code that Retrieves Silver Card Members

```
while (iter.hasNext())
{
Classification c = (Classification)iter.next();
    if (c.equals(silverCardMembers))
    {
        // do something
    }
}
```

Using the Results in Your Application

The Rules Service makes decisions for you at run-time. The rules framework is more flexible than hard-coding logic (if/then) into your components, because you can modify rules without modifying your code.

Following are some examples of using rules and rule results:

- If the time is between 8-5, direct users to pages that relate to brokerage services. If the time is outside the range, direct users to pages related to investment research.
- If the user lives in Boulder and is female, show her an advertisement for the Boulder Rock Club.
- If the user's credit score is > 10, the User Profile (sets a property) to classify the user as a Gold Member.
- If the date is between December 1 and December 31, send the user to the New Year's promotional JSP.

For additional code examples of the Rules Controls and `RulesManager EJB`, see *The Portal Rules Service* on dev2dev at

<http://dev2dev.bea.com/products/wlportal81/articles/portalruleservice.jsp>.

Rules Control Reference

You can use the Rules Control elements to provide Personalization in your portal application.

[Table 10-7](#) lists the control names and all possible values you can use to create rules.

Table 10-7 Rules Control Elements for the Rules Engine

Rules Control Name	Description	All Possible Values
literal	Used to specify a particular, unchanging value	boolean, character, decimal, double, float, integer, long, string
multi-and	Evaluates to true when all of the supplied expressions evaluate to true.	
multi-or	Evaluates to true when any of the supplied expressions evaluate to true.	
not	Negates the logical value of one expression.	
not-equal-to	Evaluates to true when both expressions are not equivalent.	
or	Evaluates to true when either expression evaluates to true.	
method	An abstract, complex type.	static-method, instance-method, new-instance, contains, contains-all, abs, acos, add, asin, atan, atan2, ceil, cos, divide, exp. floor, ieee-remainder, log, maximum, minimum, multiply, pow, rint, round, sin, sqrt, subtract, tan, to-degrees, to-radians, char-at, compare-to-ignore-case, concat, ends-with, equals-ignore-case, length, like, replace, starts-with, substring, to-lower-case, to-upper-case, trim
static-method	Invokes the named method on an object in working memory of the type specified by the type-alias, passing any provided arguments to the method.	type-alias, name, arguments

Table 10-7 Rules Control Elements for the Rules Engine

Rules Control Name	Description	All Possible Values
instance-method	Invokes the named method on an object in working memory of the type specified by the variable type-alias, passing any provided arguments to the method.	variable, name, arguments
new-instance	Instantiates an object of the type specified in the type-alias. Arguments are supplied as expressions, and the type of each argument is specified in the type-alias list in the arguments- signature.	type-alias, arguments-signature, arguments
contains	Returns a boolean indicating if the first Collection object contains the second object.	
contains-all	Returns a boolean indicating if the first Collection object contains the entire second Collection object.	
abs	Absolute value operator, accepting a number and returning a number.	
acos	Arc cosine operator, accepting a number and returning a number.	
add	Addition operator, accepting two numbers and returning their sum.	
asin	Arc sine operator, accepting a number and returning a number.	

Table 10-7 Rules Control Elements for the Rules Engine

Rules Control Name	Description	All Possible Values
atan	Arc tangent operator, accepting a number and returning a number.	
atan2	Cartesian to polar coordinates operator, accepting two numbers and returning a number.	
ceil	Ceiling operator, accepting a number and returning a number.	
cos	Cosine operator, accepting a number and returning a number.	
divide	Division operator, accepting two numbers and returning a number.	
exp	Exponential operator, accepting a number and returning a number.	
floor	Floor operator, accepting a number and returning a number.	
ieee-remainder	IEEE 754 remainder operator, accepting two numbers and returning a number.	
log	Natural logarithm operator, accepting a number and returning a number.	
maximum	Maximum operator, accepting two numbers and returning a number.	

Table 10-7 Rules Control Elements for the Rules Engine

Rules Control Name	Description	All Possible Values
minimum	Minimum operator, accepting two numbers and returning a number.	
multiply	Multiplication operator, accepting two numbers and returning a number.	
pow	Power of operator, accepting two numbers and returning a number.	
random-number	Random number operator, optionally accepts a lower and upper bound and returns a random number.	
rint	Round to next integer operator, accepting a number and returning a number.	
round	Round operator, accepting a number and returning a number.	
sin	Sine operator, accepting a number and returning a number.	
sqrt	Square root operator, accepting a number and returning a number.	
subtract	Subtraction operator, accepting two numbers and returning a number.	
tan	Arc tangent operator, accepting a number and returning a number.	

Table 10-7 Rules Control Elements for the Rules Engine

Rules Control Name	Description	All Possible Values
to-degrees	Radians to degrees operator, accepting a number and returning a number.	
to-radians	Degrees to radians operator, accepting a number and returning a number.	
char-at	Returns the character at the given position within the string, accepting a string and a number then returning a character.	
compare-to- ignore-case	String comparison operator that ignores case, accepting two strings and returning an integer.	
concat	String concatenation operator, accepting two strings and returning a string.	
ends-with	Returns a boolean indicating if the first string ends with the second string.	
equals-ignore -case	Returns a boolean indicating if the first string is the same as the second string, ignoring case.	
length	Accepts a string and returns an integer representing the number of characters in the string.	
like	Returns a boolean indicating if the second string is contained in the first string, ignoring case.	

Table 10-7 Rules Control Elements for the Rules Engine

Rules Control Name	Description	All Possible Values
replace	Accepts a string and two characters, returning a string that has the first character replaced by the second.	
starts-with	Returns a boolean indicating if the first string starts with the second string.	
substring	Accepts a string and two numbers, returning a string of the characters that fall within that number range from inside the given string.	
to-lower-case	Accepts a string, returning that same string converted to lower case.	
to-upper-case	Accepts a string, returning that same string converted to upper case.	
trim	Accepts a string, returning that same string but with any leading or trailing whitespace removed.	
actions	Instructions that are executed if the conditions are met. A group of zero or more action tags.	action
action	One instruction that is executed if the conditions are met.	action (nested), method (defined above), add-object

Table 10-7 Rules Control Elements for the Rules Engine

Rules Control Name	Description	All Possible Values
add-object	Adds an object of the type specified in the type-alias to working memory. Arguments are supplied as expressions, and the type of each argument is specified in the type-alias list in the arguments-signature. This is same functional definition as add instance above.	type-alias, arguments-signature, arguments

Creating Advanced Personalization with Rules

Part III Staging

Part III includes the following chapters:

[Chapter 11, “Modifying Property Set Values”](#)

[Chapter 12, “Modifying a User Segment”](#)

[Chapter 13, “Modifying a Content Selector”](#)

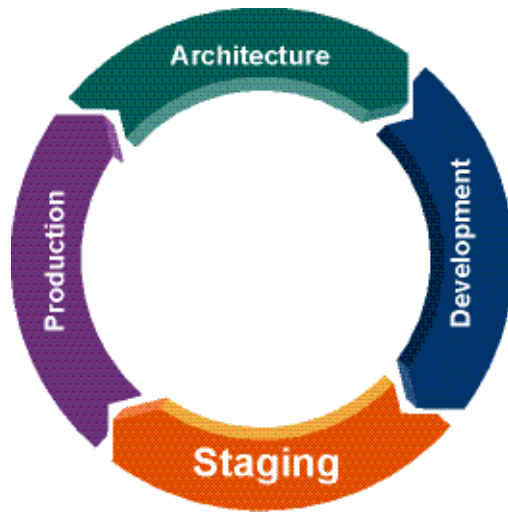
[Chapter 14, “Modifying a Placeholder”](#)

During the Staging phase, you test and modify the Property Sets, Content Selectors, User Segments, Placeholders, and Campaigns that you created in the [Development](#) phase. This staging environment simulates a production environment.

Consider setting up a common development environment for the Development phase and the [Staging](#) phase. You might move iteratively between these two phases, developing and then testing what you created.

When you move to a production environment in the [Production](#) phase, the Personalization files you created from the staging phase are moved there with the BEA Propagation Utility. The Propagation Utility moves Content Selectors, Placeholders, User Segments, and Campaigns. See the [Production Operations Guide](#) for more information on deployment and propagation.

For a description of the Staging phase of the portal life cycle, see the [WebLogic Portal Overview](#). The portal life cycle is shown in the following graphic:



Modifying Property Set Values

Developing user interaction that uses Personalization and Campaigns can involve setting up properties (such as User Profile or Session properties) that are used to define the conditions under which users will be targeted with personalized content.

User Profile property sets contain conditions that identify users. For example, you could classify all users who ordered more than five on-demand movies in the last 30 days. If visitors match the defined characteristics, they automatically become members of that User Segment and are shown specific web content with Content Selectors or they are targeted with Campaign actions.

This chapter describes how to change the values in your User Profile property sets. The properties are used in the conditions you define for your Personalization logic. Each user is dynamically served personalized web content, automatic e-mails, or discounts based on the logic conditions.

Developers used Workshop for WebLogic when they created property sets and properties (see [Chapter 4, “Creating a Property Set”](#) for instructions on creating property sets and properties). Portal administrators can use the WebLogic Portal Administration Console to update property values.

This chapter includes the following sections:

- [Editing a Property Value](#)
- [Deleting a Property Value](#)

For information on setting up and managing users that will experience Interaction Management features, see the [User Management Guide](#).

Editing a Property Value

Developers can edit property sets, properties, and conditions in Workshop for WebLogic. You can edit the property values in the WebLogic Portal Administration Console.

Editing Properties in Workshop for WebLogic

Developers can use Workshop for WebLogic to modify properties or conditions and their values for User Profiles, User Segments, HTTP session or request data, date and time conditions, or events. For instructions, see [“Modifying Properties and Conditions” on page 4-13](#).

When an attribute's value is requested for a particular user or group, and an attribute has not explicitly been assigned, then any default value assigned when the property set attribute was created is returned. Editing a value using the Administration Console or tags overrides that default value, allowing personalization for that user or group.

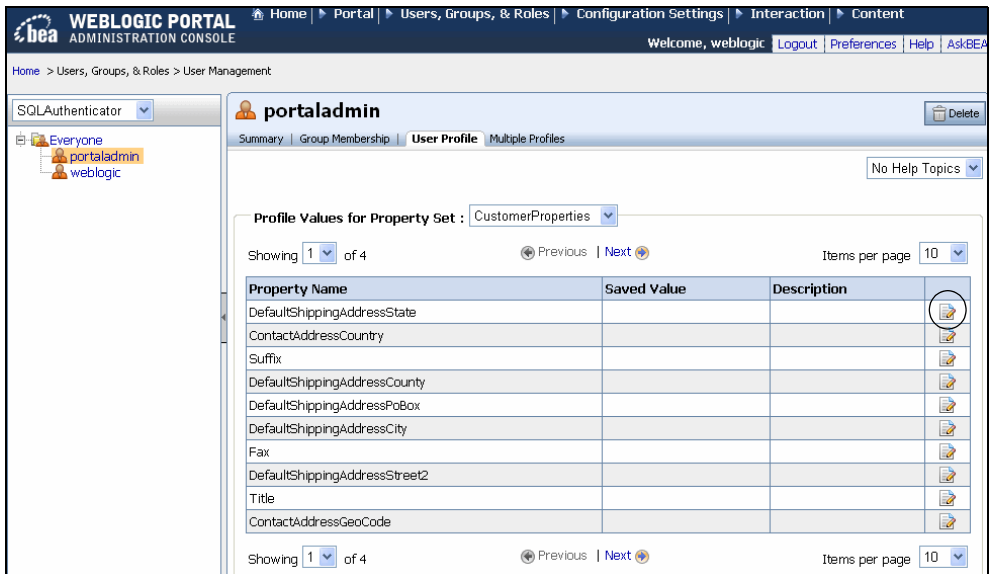
Developers can use the `<profile:setProperty>` JSP tag in JSPs or the Property control in a Page Flow to modify existing property values for users.

Editing Property Values in WebLogic Portal Administration Console

Perform the following steps to modify a property's value:

1. Start the Administration Console.
2. Choose **Users, Groups, & Roles > User Management**.
3. Select a user store from the drop-down list above the Resource Tree.
4. Select the user in the Resource Tree.
5. Select the **User Profile** tab and use the drop-down list to select the property set containing the value you want to edit.
6. In the **Property Name** field, locate the property value you want to edit and click **Edit**, as shown in [Figure 11-1](#).

Figure 11-1 Click Edit to Change the Property Value



7. Enter a new value in the **Update Saved Value** field and click **Update**. You can delete the value that is currently saved by clicking the **Delete Saved Value** check box. If you delete the saved value, a successor value is returned if one is defined. If a successor was not defined, the default value is returned. If there is no defined successor value or a default value, the property value is null.
8. Save the property set file.

Deleting a Property Value

You can use Workshop for WebLogic to delete individual properties from a property set, and you can delete an entire property set. See “[Deleting a Property or a Property Set](#)” on page 4-15 for instructions.

Perform the following steps to remove a property’s value in the WebLogic Portal Administration Console:

1. Start the Administration Console.
2. Choose **Users, Groups, & Roles > User Management**.
3. Select a user store from the drop-down list above the Resource Tree.

Modifying Property Set Values

4. Select the user in the Resource Tree.
5. Select the **User Profile** tab and use the drop-down list to select the property set that contains the value you want to remove.
6. In the **Property Name** field, locate the property value you want to remove and click **Edit**.
7. In the Edit Profile dialog box, select the **Delete Saved Value** check box to remove the value that is currently saved. (If you do not delete the saved value, a successor value is returned if one is defined. If a successor was not defined, the default value is returned. If there is no defined successor value or a default value, the property value is null.)
8. Click **Update** to save the property set file.

Modifying a User Segment

User Segments let you dynamically group users based on conditions you define. You can modify User Segments that you created in [Chapter 5, “Creating a User Segment”](#) by editing the segment’s characteristics, such as gender, browser type, date or time, and so on. If a user matches the characteristics of a *bookfan*, for example, the user automatically and dynamically becomes a member of the *bookfan* User Segment. You can then target this User Segment group with web content, automatic e-mails, and discounts based on the User Segment.

Developers use Workshop for WebLogic to create User Segments. Portal administrators can use the Administration Console to change the User Segment properties (conditions) to dynamically group users. Developer time is not required to update User Segment properties.

This chapter includes the following sections:

- [Modifying a User Segment](#)
- [Modifying a User Segment’s Properties](#)
- [Copying a User Segment](#)
- [Removing a User Segment](#)

Modifying a User Segment

Developers can use Workshop for WebLogic to change the conditions and characteristics that determine how users are categorized with User Segments. Portal administrators use the WebLogic Portal Administration Console to edit the properties for the User Segment.

Modifying a User Segment

Note: The steps in this chapter refer to the `data\src` folder in the Package Explorer View. Your `data` and `src` directories might be named differently.

Perform the following steps to edit a User Segment file in Workshop for WebLogic:

1. In the Portal Perspective in Workshop for WebLogic, double-click the User Segment `.seg` file in the `\data\src` folder in the Package Explorer View.
2. Enter the changes in the User Segment Editor.
3. Save the file by choosing **File > Save**.

The changes will appear in the Administration Console.

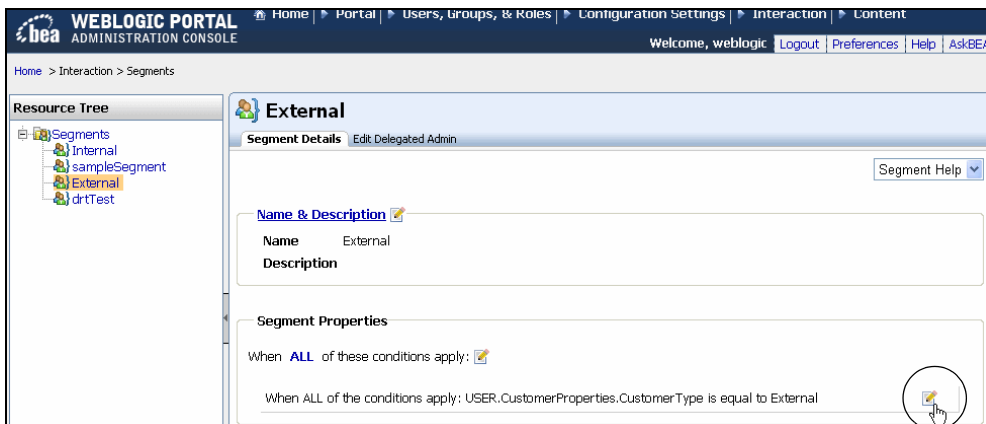
Modifying a User Segment's Properties

Portal administrators can edit individual properties in a User Segment with the Administration Console.

Perform the following steps to edit a User Segment's properties in the Administration Console:

1. Start the Administration Console.
2. Choose **Interaction > Segments**.
3. In the Resource Tree, select the User Segment you want to edit.
4. In the **Segment Details** tab, locate the segment property you want to change and click **Edit**, as shown in [Figure 12-2](#).

Figure 12-2 Click Edit to Change the User Segment Property



5. Enter your changes and click **Save**.

Copying a User Segment

Portal Administrators can save time and avoid errors by making a copy of a User Segment and then editing the properties in the new User Segment.

Perform the following steps in the Administration Console to duplicate a User Segment:

1. Start the WebLogic Portal Administration Console.
2. Choose **Interaction > Segments**.
3. Select a User Segment in the Resource Tree and click **Copy**.
4. Enter a name and description for the new User Segment, and click **OK**.

The new User Segment now appears in the User Segments Resource tree. You can now modify the User Segment's properties; see [“Modifying a User Segment” on page 12-5](#).

Tip: You can also duplicate a User Segment in Workshop for WebLogic.

Removing a User Segment

Deleting a User Segment removes it from Workshop for WebLogic and from the Administration Console.

Note: The steps in this chapter refer to the `data\src` folder in the Package Explorer View. Your `data` and `src` directories might be named differently.

Perform the following steps to remove a User Segment:

1. In the Portal Perspective in Workshop for WebLogic, select the User Segment `.seg` file in the `\data\src` folder in the Package Explorer View.
2. Right-click the segment and choose **Delete**.
3. Click **Yes** in the Confirmation dialog box.

Modifying a User Segment

Modifying a Content Selector

Developers can use Workshop for WebLogic to create Content Selectors and place them in a JSP. Portal administrators use the WebLogic Portal Administration Console to make changes to the Content Selectors that display content in the portal. Developer time is not required to update Content Selectors.

This chapter includes the following sections:

- [Modifying a Content Selector](#)
- [Deleting a Content Selector and Query](#)

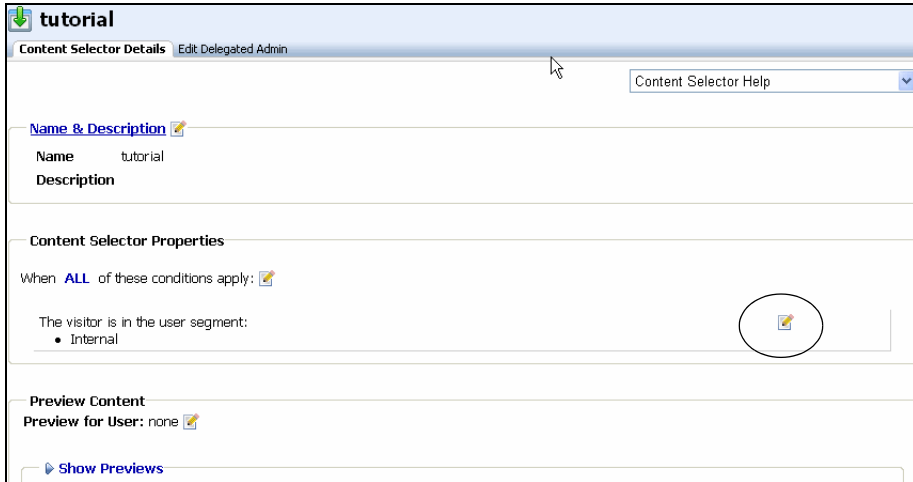
Modifying a Content Selector

Content Selectors allow you to define the content you want a particular type of visitor to see. Modifying a Content Selector property allows you to change the content that is displayed.

Perform the following steps to modify a Content Selector property:

1. Start the Administration Console.
2. Choose **Interaction > Content Selectors**.
3. In the Resource Tree, select the Content Selector you want to edit.
4. Locate the Content Selector property and click **Edit**, as shown in [Figure 13-1](#).

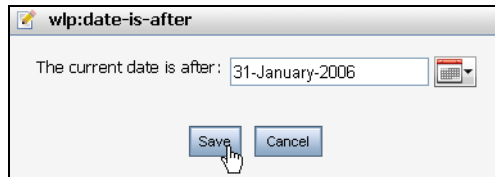
Figure 13-1 Edit a Content Selector Property



Depending on how the Content Selector has been defined in Workshop for WebLogic, you can specify the following properties:

- Whether any or all conditions apply
- If the visitor belongs to specific User Segments
- If the visitor has certain characteristics
- The visitor's HTTP session
- The visitor's HTTP request
- The current date
- The current date is after a particular date
- The current date and time is after a particular date and time
- The current time is between two particular times
- The current date is between two particular dates
- The current date and time is between two particular dates and times
- The result of the content search

5. Enter your change and click **Save**, as shown in [Figure 13-2](#).

Figure 13-2 Change a Content Selector Property

6. To view your selections, click **Show Previews**.

Tip: The Content Selector gets results from the Search cache, which might contain outdated Search results. Refreshing your browser does not clear the Search cache. You can flush the Search cache in the Administration Console by choosing **Configuration Settings > Service Administration**. In the Resource Tree, expand **Cache Manager**. Select **searchCache** (you might need to click **Next** to see all the caches) and click **Flush**.

Deleting a Content Selector and Query

Deleting a Content Selector or a Content Selector query removes the Content Selector or the query from Workshop for WebLogic and the Administration Console. See [“Deleting a Content Selector Query” on page 6-27](#) and [“Deleting a Content Selector” on page 6-28](#) for instructions.

Tip: You should also delete any `<pz:contentSelector>` tags in your JSPs that reference the Content Selector you deleted.

Modifying a Content Selector

Modifying a Placeholder

Placeholders display a single content item on a JSP. The content item is dynamically retrieved from BEA's Virtual Content Repository. A Placeholder retrieves content using predefined queries that are put in the Placeholder. Placeholders do not use conditions. Each query has a priority, or weight.

A Placeholder stores queries, runs one query at a time, and displays one of the content items retrieved by the query. A Placeholder is made up of two parts: a Placeholder file you create in Workshop for WebLogic and a companion JSP tag that performs the processing.

Developers can use Workshop for WebLogic to create Placeholders and place them in a JSP. Portal administrators use the WebLogic Portal Administration Console to manage the content that populates Placeholders by modifying the default Placeholder query or modifying a Campaign. Developer time is not required to update Placeholders.

This chapter includes the following sections:

- [Changing Content for a Placeholder](#)
- [Modifying a Placeholder](#)
- [Deleting a Query or a Placeholder](#)
- [Managing Placeholders for Optimal Performance](#)

Changing Content for a Placeholder

When a content query in a Placeholder (a default Placeholder query or a query put in the Placeholder by a Campaign) returns multiple possible content items to a Placeholder, the Placeholder determines which content item to display.

For more information on changing the content a Placeholder displays, see [Chapter 3, “Setting up Content”](#) and the *Content Management Guide*.

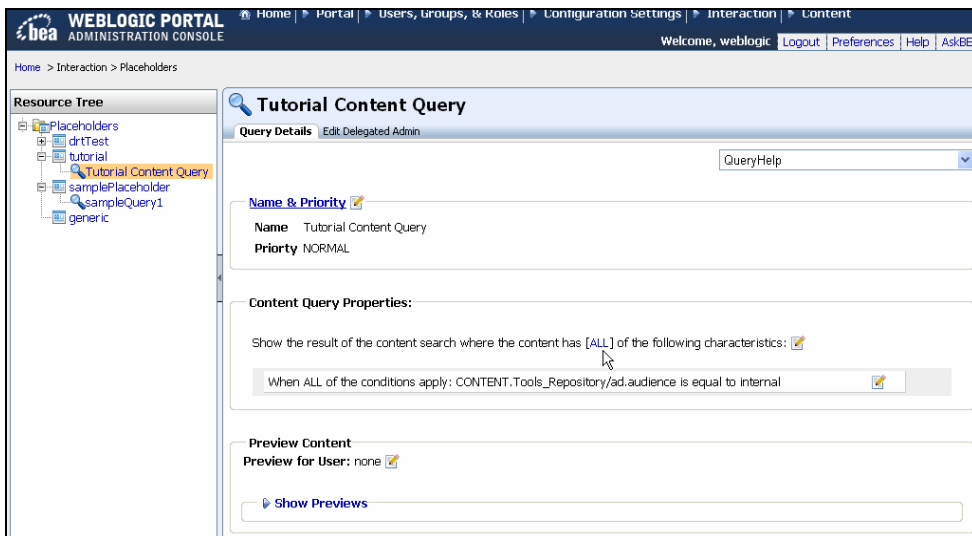
Modifying a Placeholder

Placeholders allow you to target content to a desired user (also called a visitor). You use the tools provided in the Administration Console to modify a Placeholder by editing the queries that determine the content displayed in a Placeholder on a JSP.

Perform the following steps to modify a Placeholder:

1. Start the Administration Console.
2. Choose **Interaction > Placeholders** and select the Placeholder and content search item in the Resource Tree, as shown in [Figure 14-3](#).

Figure 14-3 Select the Placeholder and Content Item



3. In the **Query Details** tab, change the descriptor in the **Content Query Properties** section by clicking the bracketed text, as shown in [Figure 14-4](#).

Figure 14-4 Change a Content Query



4. In the pop-up window, enter the new value for the descriptor and click **Save**. This descriptor governs what content is selected for display in the Placeholder. The value is based on a property set definition; typically it is a User Profile property set.
5. To preview the modified content search for the Placeholder, click **Show Previews**.

The content search for the Placeholder has been modified.

Note: Developers might choose to modify a Placeholder in Workshop for WebLogic. Open the Placeholder file by double-clicking it in the Workshop Perspective, and select and modify the appropriate query.

Deleting a Query or a Placeholder

Removing a Query or a Placeholder removes it from Workshop for WebLogic and from the WebLogic Portal Administration Console.

Note: The steps in this chapter refer to the `data\src` folder in the Package Explorer View. Your `data` and `src` directories might be named differently.

Perform the following steps to delete a query in a Placeholder:

Modifying a Placeholder

1. In the Portal Perspective, double-click the Placeholder file in the `data\src\placeholders` folder in the Package Explorer View.
2. Select the query in the Editor window.
3. In the Content Search window, select the query and click **Remove**.
4. Click **OK**.

Perform the following steps to delete a Placeholder:

1. In the Portal Perspective, select-click the Placeholder file in the `data\src\placeholders` folder in the Package Explorer View.
2. Right-click the Placeholder file and choose **Delete**.
3. Click **Yes** to confirm the Placeholder deletion.

Tip: You should also delete any `<ph:placeholder>` tags in your JSPs for the deleted Placeholder.

Managing Placeholders for Optimal Performance

Placeholders can become crowded with many queries, especially if more than one Campaign uses a Placeholder to display content. Campaign queries can remain in Placeholders indefinitely unless something specific occurs to remove them.

Perform the following steps to remove unwanted queries and control the content that is displayed in a Placeholder:

1. When you create default queries in a Placeholder, you can designate that default queries do not run when a Placeholder contains Campaign queries:
 - a. In the Portal Perspective in Workshop for WebLogic, open a Placeholder file and select the default query.
 - b. In the **Properties** tab, set the **Mix Globals** property to `false`. This setting suppresses default queries when Campaign queries are present.
2. When you create a Content Action in a Campaign in Workshop for WebLogic, use the **Remove (all) existing content** option to minimize the number of queries held in a Placeholder at a given time.

3. For Campaign content actions, set the **Time to Live** (duration) field to an appropriate time so that the content action stops putting queries in the Placeholder when you want it to stop. Locate the **Time to Live** field in the Administration Console by choosing **Configuration Settings > Service Administration**. Then select **Personalization** in the Resource Tree and click **Cache Manager** in the **Service List** tab. Click a specific cache name to edit the **Time to Live** field.
4. To control the Campaign content that is displayed in a Placeholder, create a content action for each event that occurs. Use the previous recommendations to display a fresh rotation of content in the Placeholder.
5. To stop events from firing and placing content in a Placeholder, reset the existing content as described in [“Turning Off a Campaign” on page 8-32](#).

Modifying a Placeholder

Part IV Production

The Production phase allows you to update and change Personalization in your production environment.

Some of the Interaction Management tasks you can perform in the Production phase could include the following:

- Change a Campaign to revise its end date – See [“Modifying a Campaign” on page 15-1](#).
- Edit a Placeholder or Content Selector to change the content they display – See [“Modifying a Placeholder” on page 7-15](#) or [“Deleting a Content Selector Query” on page 6-27](#).
- Update property set properties to change your audience – See [“Modifying Properties and Conditions” on page 4-13](#).
- Fine tune Content Selectors to change how often content is refreshed – See [“Modifying a Content Selector” on page 13-1](#).
- Enable an audit trail of content changes to the virtual content repository – See [“Tracking Content Changes” on page 9-46](#).
- Add custom events, listeners, and property sets to a deployed application – These changes require application redeployment for the events and `CLASSPATH` updates, and you must run the BEA Propagation Utility to update the event properties in the database. See the [Production Operations Guide](#).

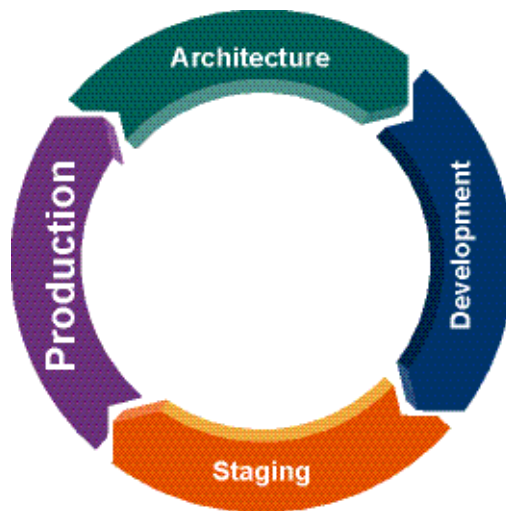
If you change Personalization features in your portal application in the Production phase, you should return to the Staging phase to test the functionality. This staging environment should

simulate a Production environment. The one exception to this process is using the rules files. During development, the rules files reload when they change (just like JSPs), so you can quickly develop with Content Selectors. However, when the server is in production mode, Content Selectors are loaded into the database (from the file-based definitions in the application) where they can be modified in the WebLogic Portal Administration Portal without redeploying the application or restarting the server.

The Production phase can be ongoing; you can move iteratively between the Staging phase and the Production phase.

You might also modify other features, such as Delegated Administration and Visitor Entitlement roles and assignments, in the Production phase. See the [Security Guide](#) for more information.

For a description of the Production phase of the portal life cycle, see the [WebLogic Portal Overview](#). The portal life cycle is shown in the following graphic:



Managing a Campaign

Developers can use Workshop for WebLogic to create a Campaign and display it in a JSP. Portal administrators use the WebLogic Portal Administration Console to make changes to the Campaign, including the rules, start or stop date, the sponsor, and so on.

Developer time is not required to modify a Campaign. A Portal administrator can also duplicate an existing Campaign and then modify the new Campaign.

This chapter contains the following sections:

- [Modifying a Campaign](#)
- [Modifying a Rule](#)
- [Managing a Campaign for Optimal Performance](#)

Modifying a Campaign

Portal administrators can avoid creating new Campaigns by modifying existing Campaigns in the following ways:

- [Changing a Campaign's Description or Sponsor](#)
- [Changing a Campaign Start or Stop Date](#)
- [Activating and Deactivating a Campaign](#)
- [Turning Off a Campaign](#)
- [Resetting Campaign Settings](#)

- [Duplicating a Campaign](#)

Changing a Campaign's Description or Sponsor

You can change the text describing a Campaign; however, you cannot change the Campaign's name. You can also edit the Sponsor field, which describes the purpose of the Campaign or who it affects.

Perform the following steps to update a Campaign's description or sponsor:

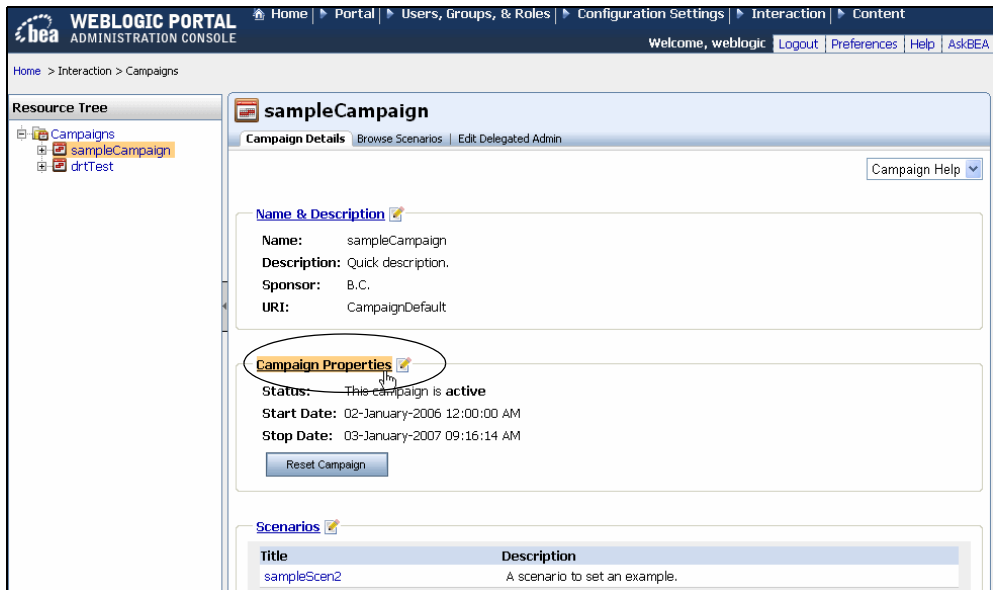
1. Start the Administration Console.
2. Choose **Interaction > Campaigns**.
3. In the Resource Tree, select the Campaign whose description you want to modify.
4. In the **Campaign Details** tab, click **Name & Description**.
5. Enter the new Description.
6. You can also change the Sponsor field, to describe the purpose of the Campaign or who it affects.
7. Click **Save**.

Changing a Campaign Start or Stop Date

Perform the following steps to change a Campaign start or stop date:

1. Start the Administration Console.
2. Choose **Interaction > Campaigns**.
3. In the Resource Tree, select the Campaign you want to modify.
4. In the **Campaign Details** tab, click **Campaign Properties**, as shown in [Figure 15-1](#).

Figure 15-1 Change a Campaign Property



5. Click the calendar icon next to the **Start Date** or **Stop Date** fields. Use the month and year drop-down menus to select the month and year. You can also use the left and right arrows at the top of the Calendar window to select the month and year. Select a number in the Date section to select a day. Use the up and down arrows in the Time section to select a specific time of day. You must click each field in the Time section before you can modify it and click **Done**.
6. Click **Save** in the Edit Campaign Properties window to save the new start or stop dates for the Campaign.
7. Click **Reset Campaign** for the new dates to take effect.

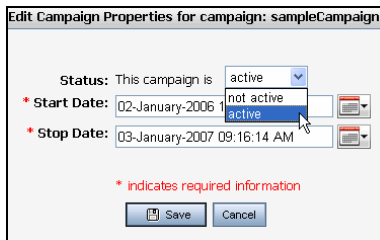
Activating and Deactivating a Campaign

Perform the following steps to start a Campaign:

1. Start the Administration Console.
2. Choose **Interaction > Campaigns**.
3. In the Resource Tree, select the Campaign you want to activate.

4. In the **Campaign Details** tab, click **Campaign Properties**.
5. In the Edit Campaign Properties window, click the **Status** drop-down list, select **Active**, and click **Save**, as shown in [Figure 15-2](#).

Figure 15-2 Select Active to Start a Campaign



6. In the **Campaign Details** tab, click **Reset Campaign** for the new dates to take effect. The Campaign is now active.

Perform the following steps to stop a Campaign:

1. Start the Administration Console.
2. Choose **Interaction > Campaigns**
3. In the Resource Tree, select the Campaign you want to deactivate.
4. In the **Campaign Details** tab, click **Campaign Properties**.
5. In the Edit Campaign Properties window, click the **Status** drop-down list, select **Not Active**, and click **Save**.
6. In the **Campaign Details** tab, click **Reset Campaign** for the new dates to take effect. The Campaign is no longer active. The targeted user will see default content rather than specific Campaign content in the Campaign's Placeholder.

Turning Off a Campaign

To turn off a Campaign so that it does not fire Campaign events, you can remove the `CampaignEventListener`.

Tip: The following steps in the WebLogic Portal Administration Console work for a portal that is deployed as an exploded EAR file. If your portal is a compressed EAR file, you will need to do these steps manually and then re-build and deploy the EAR file.

Perform the following steps to turn off a Campaign:

1. Start the Administration Console.
2. Choose **Configuration > Service Administration**.
3. In the Resource Tree, expand the **Personalization** folder and select **Event Service**.
4. In the **Browse** tab, select the **Delete** check box next to the `com.bea.campaign.internal.CampaignEventListener` class and click **Delete**. Campaign events will no longer be fired, but if you set up other Behavior Tracking or other event listeners, those events will continue to fire.

Note: If you want to activate the Campaign later, add the `CampaignEventListener` as a Synchronous Listener in the **Browse** tab.

Resetting Campaign Settings

You can reset specific Campaign components.

1. Start the Administration Console.
2. Choose **Interaction > Campaigns**.
3. In the Resource Tree, select the Campaign you want to reset.
4. In the **Campaign Details** tab, click **Reset Campaign**.
5. In the pop-up window, click **OK** to reset all of the Campaign components.

Duplicating a Campaign

You can save time and reduce errors by creating a copy of a Campaign and then making changes to it.

Perform the following steps to duplicate a Campaign:

1. Start the Administration Console.
2. Choose **Interaction > Campaigns**

3. In the Resource Tree, select the Campaign you want to copy.
4. Right-click the Campaign, and choose **Copy**.
5. In the pop-up window, enter the name for the new Campaign and click **OK**.

The new Campaign appears in the Resource Tree.

Modifying a Rule

As an administrator, you can modify a content action within a Campaign to change the content query, change the Placeholder, change the conditions, and so on.

Campaigns can include the following actions:

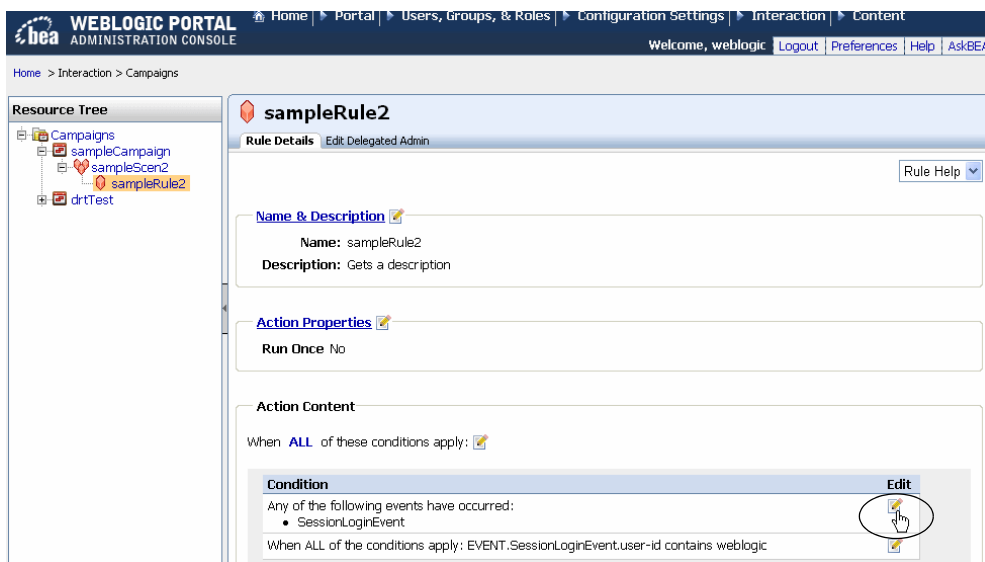
- [Modifying a Content Action](#)
- [Modifying an E-Mail Action](#)
- [Modifying a Discount Action](#)
- [Previewing a Modified Campaign Action](#)

Modifying a Content Action

Perform the following steps to modify a Content Action in a Campaign:

1. Start the Administration Console.
2. Choose **Interaction > Campaigns**.
3. In the Resource Tree, select the Campaign, the scenario, and the rule you want to modify.
4. In the **Rule Details** tab, click **Edit** next to the condition you want to change, as shown in [Figure 15-3](#).

Figure 15-3 Click Edit to Change a Content Action



You can change the following items:

- Whether any or all conditions apply
- If the visitor has predefined characteristics
- If the visitor is in a particular User Segment
- The visitor's HTTP session
- The visitor's HTTP request
- If any of the following events occurred:
 - UserRegistrationEvent
 - AddToCartEvent
 - ClickContentEvent
 - DisplayCampaignEvent
 - PurchaseCartEvent
 - RemoveFromCartEvent
 - SessionLoginEvent

- ClickProductEvent
 - ClickCampaignEvent
 - ContentConfigurationEvent
 - ContentCreateEvent
- The event characteristics
 - The current date
 - The current date is after a particular date
 - The current date and time is after a particular date and time
 - The current time is between two particular times
 - The current date is between two particular dates
 - The current date and time is between two particular dates and times
 - A random number between one and 100 that falls between one and 100 (this defines 100 percent of the qualifying visitors)
5. Make the changes and click **Save**.
 6. After you modify the Content Action characteristics, you can modify the Content Action further by specifying the specific content to retrieve for the visitor and the specific Placeholder for it. To modify the retrieved content, click **Edit** next to the property in the Action Properties section in the **Rule Details** tab.
 7. Make the changes and click **Save**.

Modifying an E-Mail Action

As an administrator, you can modify an E-mail Action within a Campaign to change the action conditions, descriptions, User Segments, and so on.

Perform the following steps to modify an e-mail action:

1. Start the Administration Console.
2. Choose **Interaction > Campaigns**.
3. In the Resource Tree, select the Campaign, scenario, and e-mail action rule you want to modify.
4. In the **Rule Details** tab, click **ANY** or **ALL** to determine if any or all conditions apply.

5. Click **Edit** next to a condition.
6. Make the changes and click **Save**.
7. Specify the e-mail action to perform by clicking **Edit** next to the action.
8. You can change the e-mail location, title, and sent by address and click **Save**.

Modifying a Discount Action

As an administrator, you can modify a Discount Action rule in a Campaign to change the action conditions, descriptions, User Segments, and so on.

Perform the following steps to modify a Discount Action:

1. Start the Administration Console.
2. Choose **Interaction > Campaigns**.
3. In the Resource Tree, select the Campaign, scenario, and discount action rule you want to modify
4. In the **Rule Details** tab, click **ANY** or **ALL** to determine if any or all conditions apply.
5. Click **Edit** next to a condition.
6. Make the changes and click **Save**.
7. Specify the discount action to perform by clicking **Edit** next to the property in the Action Properties section.
8. Make the changes and click **Save**.

Previewing a Modified Campaign Action

Portal administrators can use the Administration Console to preview a modified Content Action, E-mail Action, or Discount Action in a Campaign.

You can modify any of these actions in the Administration Console by changing the conditions or content queries of these actions. After changing an action, you can preview it to verify the change is correct.

Perform the following steps to preview a modified Campaign Action:

1. Start the Administration Console.

2. Choose **Interaction > Campaigns**.
3. In the Resource Tree, select the Campaign, scenario and action rule that you want to modify.
4. Modify the conditions and actions for the Discount, E-mail, or Content Action and save your changes.
5. In the Resource Tree, select the modified rule to display in the **Rule Details** tab.
6. Click **Show Previews**. The modified content does not show in the preview until you refresh the tree.
7. Close the Resource Tree and then expand it to refresh it.
8. In the Resource Tree, select the Campaign and subsequent action you want to preview.
9. Click **Show Previews** to see the modified content.

Managing a Campaign for Optimal Performance

Placeholders can become crowded with many queries, especially if more than one Campaign uses a Placeholder to display content. Campaign queries can remain in Placeholders indefinitely unless something specific occurs to remove them.

Perform the following steps to remove unwanted queries and control the content that is displayed in a Placeholder:

1. When you create default queries in a Placeholder, perform the following steps to designate that default queries do not run when a Placeholder contains Campaign queries:
 - a. In the Portal Perspective in Workshop for WebLogic, open the Placeholder file in the `\data\src\placeholders` folder in the Package Explorer View.
 - b. Select the default query by selecting the **Placeholder Editor** tab.
 - c. In the **Properties** tab, set the **Mix Globals** property to `false`. This setting suppresses default queries when Campaign queries are present.
2. When you create a Content Action in a Campaign in Workshop for WebLogic, use the **Remove (all) existing content** option to minimize the number of queries held in a Placeholder at a given time.
3. For Campaign content actions, set the **Time to Live** (duration) field to an appropriate time so that the content action stops putting queries in the Placeholder when you want it to stop. Locate the **Time to Live** field in the Administration Console by choosing **Configuration**

Settings > Service Administration. Select **Cache Manager** in the Resource Tree and click a specific cache name to edit the **Time to Live** field.

4. To control the Campaign content that is displayed in a Placeholder, create a Content Action for each event that occurs. Use the previous recommendations to display a fresh rotation of content in the Placeholder.
5. To clear any content that has been put in a Placeholder, reset the previously placed content using the reset feature as described in [“Resetting Campaign Settings” on page 15-5](#).
6. Enable Campaign content caches by following the steps in [“Setting Campaign Content Caches” on page 8-34](#).

Managing a Campaign