BEA WebLogic Portal™

Interaction Management Guide

Version 8.1 with Service Pack 5
Document Revised: October 2005

Copyright

Restricted Rights Legend

Trademarks or Service Marks

Contents

1. Interaction Management Overview

2. Setting up Content

3. Content Selectors

4. Placeholders

5. Campaigns

6. Events and Behavior Tracking

CHAPTER 1

Interaction Management Overview

WebLogic Portal provides a powerful set of tools and services you can use to enhance user interactions with your portals. For example, users can be dynamically shown images or links that are personalized just for them, or they can be dynamically guided through a process (such as signing up for employee benefits or shopping online) that takes them to different places in the process based on their personal preferences or characteristics. You can even record the paths users take through your portals to help gauge the effectiveness or your portal, its design, or your process flows, giving you valuable information that you can use in many ways, such as validating your strategies or making improvements and forecasts.

These tools and services are collectively called "Interaction Management."

This chapter describes the fundamental concepts and pieces of Interaction Management, shows the relationships among many of those pieces, and provides guidance on choosing and building the functionality you want. The remaining chapters in this guide provide details on implementing specific Interaction Management features.

This chapter includes the following sections:

Interaction Management Overview

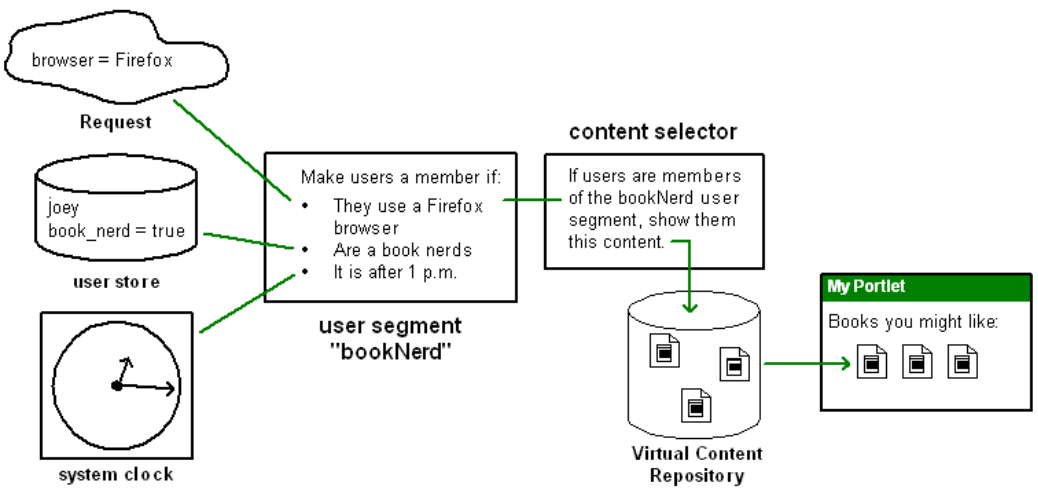
Interaction Management uses a wide variety of factors, or conditions, that help uniquely identify users and what they are doing. For example, following are some of the conditions that campaigns can use to figure out which users to target with personalized content:

- 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 occurring (such as something being clicked)
- Date/time factors
- Items or value of items in a shopping cart

When building Interaction Management functionality, you use different tools to: 1) Specify the exact characteristics that will identify the users you want to target (such as the conditions previously mentioned); and 2) Define the actions that will occur when users matching those conditions visit your portal.

Figure 1-1 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 1-1 Simple rendition of interaction management logic and flow



Underlying most Interaction Management features is a rules engine. The rules engine, which runs behind the scenes on a running server in a portal domain, reads all available conditions in memory, evaluates those conditions against the rules you have defined, and executes the actions you have defined if the conditions match your rules.

Tools and Services

Each stage in Figure 1-1 is enabled by using a specific Interaction Management tool in WebLogic Workshop:

- The Request Designer lets you identify the request properties you want to capture.
- The User Profile Designer lets you create the user profile properties you want to store for each user.
- The User Segment Designer lets you dynamically categorize users based on conditions such as the request and user profile properties you have created.
- The Content Selector Designer lets you specify that when a specific type of user is present, show them a specific set of content.

In addition to these types of convenient tools provided in WebLogic Workshop, Interaction Management also includes JSP tags, Java Controls, and a full API to implement the functionality in your code. Also, WebLogic Portal provides a Web application to deploy updated Interaction Management functionality to an application running in production.

Following is a list of Interaction Management features:

- **Property editors** – As mentioned previously, WebLogic Portal provides a set of editors in WebLogic Workshop that let you define many types of properties to use in defining conditions (uniquely identifying users). Some properties have other uses as well. For example, user profile properties can be used to define visitor entitlements and delegated administration.
- **User Segments** – User segments let you dynamically group users into identifiable units based on conditions you define; for example, you can define conditions that identify males, females, book nerds, and so on. Once identified and grouped into user segments, you can target groups with personalized actions through campaigns and content selectors.
- **Content Selectors** – Content selectors let you target specific users with specific content from BEA's Virtual Content Repository. For example, you can display a list of recommended movies to users identified as "movie buffs."

- **Campaigns** – Campaigns let you target specific users with a single piece of personalized content, send them a predefined e-mail automatically, and/or provide them with a discount in a commerce application. Campaigns run for a limited time frame that you determine using a variety of possible factors.
- **Placeholders** – Placeholders are used by campaigns to display personalized content on a JSP. Placeholders can also be used by themselves to display specific types of non-personalized content that isn't provided by a campaign.
- **Events Framework** – The events framework lets you create things that can happen in a user's interaction with your portal. Events can be used to trigger campaigns (such as a user clicking a button or registering in your portal), and they are used in tracking a user's path through a portal (behavior tracking).
- **Behavior Tracking** – With behavior tracking, events that occur in a user's path through your portal are logged to the database, letting you analyze the user behavior in your portals.
- **JSP Tags** – Interaction Management JSP tags are necessary for displaying personalized content to users.
- **Java Controls** – WebLogic Portal provides Java Controls—predefined Java functionality—that you can use in your Page Flows and Web Services to enable Interaction Management functionality. 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.
- **A full API** – WebLogic Portal provides a full API for developing Interaction Management functionality.
- **Rules Engine** – As mentioned previously, the rules engine is the backbone of most interaction management features. The rules engine is not a feature or tool you use directly. Rather, it runs behind the scenes, evaluating objects in memory against the conditions you define and executes the actions you have defined when the conditions are met.

Logic Decoupled From Code

One of the most important benefits provided by Interaction Management is the decoupling of logic 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 simply references those files. For example, campaigns show Web content using a JSP tag called a placeholder that looks like this: `<ph:placeholder name="myPlaceholder1"/>`. Add JSP placeholder tags (uniquely identified by the "name" attribute) anywhere you want in your portal's JSPs. Then

simply define your campaigns to use the existing placeholders, each of which can display content unique to the campaign and to the individual users. Campaigns can change and be added, but you never have to change your JSP code. The placeholders you need in the JSPs stay the same.

Choosing Which Type of Interaction Management to Develop

Use the following scenarios to help you determine which type of interaction management you want to develop.

Table 1-1 Choosing which type of Interaction Management to develop

If you want to	...do this
Display a binary property from a single content node from the Virtual Content Repository that can change each time a user visits your portal or clicks the browser refresh button. For example, each time an employee visits the Intranet portal, display a different picture from the company picnic.	For creating a generic rotation of content for all users, create a Placeholder and add a default query for the Placeholder that displays the range of content you want. For creating a targeted rotation of content for each user based on each user's characteristics, create 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 be used in the campaign. You can also use the <ad:adTarget> JSP tag as an alternative to a Placeholder to manually embed a content query in a JSP. See Placeholders in this guide.
Display a binary property from a single content node from the Virtual Content Repository that shows the same content node for each type of user. For example, if a user of type “manager” views the portal, always show the manager the “performance review reminder” graphic. If a user of type “regular employee” views the portal, always show the employee the “benefits open enrollment” graphic.	As in the previous scenario, you can create a Placeholder with a default query for all users or a Placeholder used by a campaign to target specific users differently. There are two keys to showing the same content node without content rotation: • 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 nodes. See Placeholders and Campaigns in this guide.

Table 1-1 Choosing which type of Interaction Management to develop

If you want to	...do this
Display multiple content nodes and properties from the Virtual Content Repository simultaneously. For example, show each user a unique list of recommended books based on the user's characteristics.	To show multiple content nodes from the Virtual Content Repository simultaneously, create content selectors and add them to your JSPs. See Content Selectors in this guide.
Display personalized content from an inline section of a JSP. For example, provide three different sections of HTML content in a JSP but show users only the sections that match their characteristics.	To display personalized inline JSP content, create user segments and use the <pz:div> JSP tag to wrap personalized content. 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): <cscm:default>, <cscm:not-default>, <cscm:recognized>, <cscm:not-recognized>, <cscm:when>, and <cscm:when-not>. See "Using <pz:div> Instead of a Content Selector" in the Content Selectors chapter of this guide. See "Multichannel JSP Tags" in the WebLogic Workshop help system at http://e-docs.bea.com/workshop/docs81/doc/en/portal/taglib/navMultichannel.html.
Send users automatic e-mails.	Create and store e-mail message files and create a campaign that uses an e-mail action. See Campaigns in this guide.

Table 1-1 Choosing which type of Interaction Management to develop

If you want to	...do this
Give users automatic discounts.	Perform the following tasks: • 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 surface catalog items from the Virtual Content Repository and identify them with "categories" and "SKU" numbers. • Create discounts and use the commerce API to surface the discounts in your shopping cart. If desired, use the API to surface the discount's description next to the discount amount displayed in the shopping cart. See "Building a Commerce Application" in the WebLogic Workshop help system at http://e-docs.bea.com/workshop/docs81/doc/en/portal/buildportals/navCommerce.html.
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 the guide to *Using Rules in Portal Applications* on edocs at http://edocs.bea.com/wlp/docs81/rules/index.html.

Adding Interaction Management to Your Applications

Interaction management development involves setting up interrelated pieces. The following sections describe the steps needed to implement interaction management functionality. Each section contains links to different implementation tasks depending on your needs. Use this topic as an overall roadmap for developing personalized applications.

Setting up Properties

As discussed previously, the properties you create in WebLogic Workshop are used in the conditions you define for your personalization logic.

For instructions on creating these properties, such as user profile, HTTP request and session, and event properties, see "Developing Personalized Applications" in the WebLogic Workshop help system at http://e-docs.bea.com/workshop/docs81/doc/en/portal/buildportals/P13NInteractionMgmt.html.

Creating User Segments

If you want to categorize groups of users dynamically for use in personalization, create user segments. See "Creating User Segments" in the WebLogic Workshop help system at http://e-docs.bea.com/workshop/docs81/doc/en/portal/buildportals/CreatingSegments.html.

Setting up Users

For information on setting up and working with the users for which you will develop Interaction Management, see the User Management Guide at http://edocs.bea.com/wlp/docs81/users/index.html.

Setting up Content

There are specific properties you can set on content items to enhance personalization in your applications. See Setting up Content in this guide. See also the *Content Personalization* white paper at http://edocs.bea.com/wlp/docs81/whitepapers/p13n/index.html.

For general information on content management, see the *Content Management Guide* on edocs at http://edocs.bea.com/wlp/docs81/cm/index.html.

Building Functionality

After you create all necessary personalization conditions and set up users and content, you can create interaction management functionality and surface it in your portals. 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 who are members of the user segment.

For instructions in creating content selectors, placeholders, and campaigns, see the chapters in this guide.

Administering and Updating Interaction Management

After you create user segments, content selectors, placeholders, and campaigns in WebLogic Workshop, you can modify the settings and queries for those components in the WebLogic Administration Portal. See the WebLogic Administration Portal help system for details.

If you need to add or modify these features beyond what you are allowed to modify in the WebLogic Administration Portal, such as creating or modifying properties or creating new content selectors, you must make the changes in WebLogic Workshop and use the Datasync Web

application to sych the changes with the database. For more information on the Datasync Web application, see the Production Operations User Guide.

CHAPTER 2

Setting up Content

Targeting users with content is one of the most important aspects of interaction management. Targeting content to users with placeholders and campaigns can be made even more powerful by adding specific properties to content items in your BEA-compatible content repository.

Note: For instructions on the many aspects of content management in WebLogic Portal, choose from the list of Content Management topics on the WebLogic Portal edocs home page at http://edocs.bea.com/wlp/docs81/index.html.

This chapter includes the following sections:

- Content Priority
- Other Content Properties
- Displaying Binary Content

Content Priority

A placeholder—the bucket (JSP tag) on 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 must decide which content item to display. Use a content property called `adWeight` to give content items a greater or lesser chance of being displayed in a placeholder when the content items are retrieved by a query.

Use Table 2-1 for guidance in adding the `adWeight` property to your content items.

Table 2-1 Properties for all content

Content Property	Property Type	Description
adWeight	Integer	The higher the adWeight number you assign to a content item, the higher the chance it will be displayed in the placeholder.

Other Content Properties

This section describes content properties you can use for image and Shockwave files. Use Table 2-2 and Table 2-3 for guidance in adding the image and Shockwave properties to your content items.

If you want to use Goal Setting to end campaigns based on the number of times content is clicked, you must use the `adTargetUrl`, `adTargetContent`, or `adMapName` property as described in Table 2-2.

Table 2-2 Properties for images

Content Property	Property Type	Description
adTargetUrl	String	Makes an image clickable and provides a target for the clickthrough, expressed as a URL. The Event Service records the clickthrough. Use either adTargetUrl, adTargetContent, or adMapName, depending on how you want to identify the destination of the ad clickthrough.
adTargetContent	String	Makes an image clickable and provides a target for the clickthrough, expressed as the content management system's content ID. The Event Service records the clickthrough. You can view a content item's unique ID by selecting the content item in the WebLogic Administration Portal and viewing the description in the Edit Content window. Use either adTargetUrl, adTargetContent, or adMapName, depending on how you want to identify the destination of the ad clickthrough.

Table 2-2 Properties for images (continued)

Content Property	Property Type	Description
adMapName	String	Makes an image clickable, using an image map to specify one or more targets. The value for this attribute is used in two locations: • In the anchor tag that makes the image clickable: <img> • In the map definition, <map name=*value*> If you specify a value for adMapName, you must also specify a value for adMap. Use either adTargetUrl, adTargetContent, or adMapName, depending on how you want to identify the destination of the ad clickthrough.
adMap	String	Supplies the XHTML definition of an image map. If you specify a value for adMap, you must also specify a value for adMapName.
adWinTarget	String	Displays the target in a new pop-up window, using JavaScript to define the pop-up window. The only value supported for this attribute is `newwindow`.
adWinClose	String	Specifies the name of a link that closes a pop-up window. The link appears at the end of the window content. For example, if you provide “Close this window” as the value for this attribute, then “Close this window” appears as a hyperlink in the last line of the pop-up window. If a customer clicks the link, the window closes.
adAltText	String	Specifies a text string for the alt attribute of the <img> tag. If you do not include this attribute, the <img> tag does not specify an alt attribute.
adBorder	Integer	Specifies the value for the border attribute of the <img> tag. If you do not include this attribute, the border attribute is given a value of “0”.

Table 2-3 shows useful properties you can set on Shockwave content items.

Table 2-3 Properties for Shockwave files

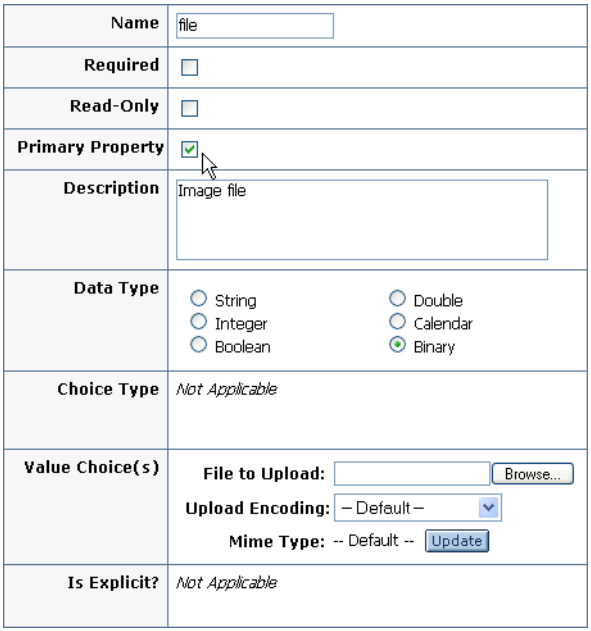
Content Property	Property Type	description
swfLoop	String	Specifies whether the movie repeats indefinitely (true) or stops when it reaches the last frame (false). Valid values are true or false. If you do not define this attribute, the default value is true.
swfQuality	String	Determines the quality of visual image. Lower qualities can result in faster playback times, depending on the client's Internet connection. Valid values are low, high, autolow, autohigh, best.
swfPlay	String	Specifies whether the movie begins playing immediately on loading in the browser. Valid values are true or false. If you do not define this attribute, the default value is true.
swfBGColor	String	Specifies the background color of the movie. This attribute does not affect the background color of the HTML page. Valid value syntax is #RRGGBB.
swfScale	String	Determines the dimensions of the movie in relation to the area that the HTML page defines for the movie. Valid values are showall, noborder, exact fit.
swfAlign	String	Determines whether the movie aligns with the center, left, top, right, or bottom of the browser window. If you do not specify a value, the movie is aligned in the center of the browser. Valid values are l, t, r, b.
swfSAlign	String	Determines the movie's alignment in relation to the browser window. Valid values are l, t, r, b, tl, tr, bl, br.
swfBase	String	Specifies the directory or URL used to resolve relative path names in the movie. Valid values are . (period), directory-name, URL.
swfMenu	String	Determines whether the movie player displays the full menu. Valid values are true or false.

Displaying Binary Content

In BEA's Virtual Content Repository, content items are made up of properties. When you are storing binary content such as images, you store those images using a binary property. In order for binary content to be displayed in your content selectors and placeholders, you must assign binary properties as the "Primary Property."

In Figure 2-1, a binary property is shown in the WebLogic Administration Portal. The property is binary, because the "Data Type" is set to "Binary." Notice the "Primary Property" option is selected.

Figure 2-1 A binary property set as the "Primary Property"



CHAPTER 3

Content Selectors

Content selectors are a powerful tool for personalization, letting you target specific groups of people with one or more specific content items from the BEA's Virtual Content Repository.

For example, if a user logs in who is identified in her user profile as a "book nerd," a content selector can display a list of recommended books.

A content selector is made up of two parts: a content selector file you create in WebLogic Workshop and a companion JSP tag that performs the processing.

In Figure 3-1, the content selector file is called "classic", and the JSP tag references the content selector file.

Figure 3-1 The two parts of a content selector: a content selector file and a JSP tag

This chapter includes the following sections:

- How Content Selectors Work
- Types of Content that Content Selectors Can Display

How Content Selectors Work

Users do not have to be authenticated (log in) to be targeted by content selectors. For example, the presence of specific HTTP request or session properties, or specific date/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 who are viewing 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 more accurately with personalized content. The “book nerd” example in the chapter introduction is an example of this.

The following sections describe the structure of a content selector in more detail: the content selector file and the JSP tag.

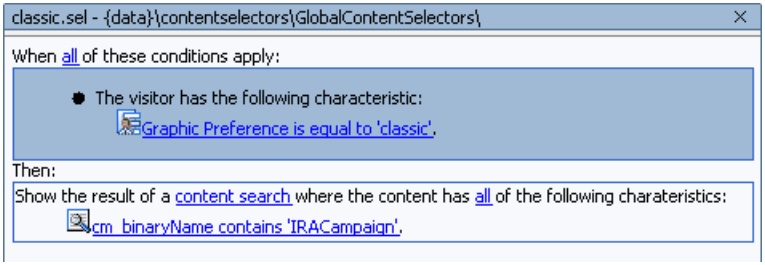
Content Selector File

A content selector file contains two parts:

- The 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/time conditions.
- A content query that retrieves a specific set of content properties.

Figure 3-2 shows a content selector file called `classic.sel`. The condition that triggers the content selector is a user whose user profile has a “Graphic Preference” property with a value of “classic”. The content that is retrieved is any content with a binary file name that contains “IRACampaign”.

Figure 3-2 Content selector file in WebLogic Workshop



<pz:contentSelector> JSP Tag

After you have created a content selector file, you must use the <pz:contentSelector> JSP tag to display the content. Following is an JSP tag for the "classic" content selector shown in Figure 3-2.

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

The JSP tag has two required attributes:

- `rule` – Contains the name of the content selector file (without the file extension). This tells the JSP tag which personalization rules and query to use.
- `id` – When a content selector runs its query and retrieves content from the Virtual Content Repository, the content is retrieved as an array of content properties. The `id` attribute, which is a String you enter, 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 Using Content Selectors in this chapter.

There are many other useful attributes you can set on the <pz:contentSelector> tag. See "<pz:contentSelector> Tag" in the WebLogic Workshop help system at http://edocs.bea.com/workshop/docs81/doc/en/portal/taglib/www.bea.com/servers/portal/tags/personalization/contentSelector.html

Types of Content that Content Selectors Can Display

Content selectors retrieve content properties, most of which are the text or numeric values. For example, say 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. All of that is accomplished through using only the text values of the content properties.

But say book content also has a binary property that stores an image of the book's cover. The content selector can also display that binary property—show the actual image—using the ShowProperty servlet, as described in Using Content Selectors. The the type of binary content that content selectors can display is dependent upon the MIME types you have configured. By default, WebLogic Portal provides MIME support for displaying image, Shockwave, and XHTML files. To display content for other MIME types, see "Supporting Additional MIME Types" in the WebLogic Workshop help system at http://e-docs.bea.com/workshop/docs81/doc/en/portal/buildportals/mimetypes.html.

Setting up Content for Use in Content Selectors

For content selectors to display binary content such as images, the binary content property must be set to "Primary Property" in the Virtual Content Repository. See "Displaying Binary Content" in the Setting up Content chapter in this guide.

Creating Content Selectors

Creating a placeholder involves creating a content selector file in WebLogic Workshop and using companion <pz:contentSelector> JSP tags in any relevant JSPs. Each <pz:contentSelector> JSP tag must use its "rule" attribute to reference a content selector you have created in WebLogic Workshop, and it must use its "id" attribute to store the array of the content query results.

For steps on creating content selectors, see the following topics in the WebLogic Workshop Help system:

- "Creating Content Selectors" at http://edocs.bea.com/workshop/docs81/doc/en/portal/buildportals/createSelectors.html.
- "<pz:contentSelector> Tag" at http://edocs.bea.com/workshop/docs81/doc/en/portal/taglib/www.bea.com/servers/portal/tags/personalization/contentSelector.html.

Tutorial

The WebLogic Workshop help system includes a tutorial called "Showing Personalized Content in a Portlet" at http://e-docs.bea.com/workshop/docs81/doc/en/portal/tutorials/tutP13nPortletIntro.html. This tutorial involves creating content selectors and surfacing the content in a portlet.

Building Content Queries

WebLogic Portal provides advanced content query features for building powerful content queries. For more information, see "Building Content Queries with Expressions" in the WebLogic Workshop help system at http://e-docs.bea.com/workshop/docs81/doc/en/portal/buildportals/contentExpression.html.

Adding Content Selectors to JSPs

After you create a content selector file in WebLogic Workshop, you can add the content selector to a JSP in a number of ways in WebLogic Workshop:

- Drag a placeholder from the Application window onto a page of an open portal file. When you do this, three things happen:
  - The Portlet Wizard appears, letting you quickly create a portlet that will display the placeholder.
  - The resulting portlet is automatically added to the portal page.
  - 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:

```
<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 3-1 describes the additional tags.

Table 3-1 Tags to use in conjunction with content selectors

`<utility:notNull>`	Checks to see if the array actually contains content.
`<ul>`	Provides a container for listing content items in a bulleted list.
`<utility:forEachInArray>`	Iterates through the array and isolates each content item until all items in the array have been processed. Each item is given its own `<li>` bullet.
`<cm:getProperty>`	Takes each content item from `<utility:forEachInArray>` and designates which content property is going to be displayed.

- Open the JSP to which you want to add the content selector, and drag the content selector file from the Application window into the JSP in either Design View or Source View. The JSP tag is added automatically, the "rule" attribute is automatically populated with the name of the placeholder, the "id" attribute is included (without a value), and the tag library include statement is added automatically. See Figure 3-1.
- Drag the <pz:contentSelector> JSP tag from the Palette window (the Portal Personalization category) into an open JSP and populate the tag's attributes manually. The tag library include statement is added automatically.

Using Content Selectors

This section provides guidance on using 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 (content properties) 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 Table 3-1 are among the most useful tags for displaying retrieved content.
- To display binary content retrieved by content selectors, use the ShowProperty servlet. The following code example shows how:

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

<utility:notNull item="<%=nodes%>">

<ul>

<utility:forEachInArray array="<%=nodes%>" id="node"
type="com.bea.content.Node">

<img src="<%=request.getContextPath() + "/ShowProperty" +
node.getPath()%>">

</utility:forEachInArray>

</ul>

</utility:notNull>
```

The <img> 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 like this: `foo.getPath()`.

Using <pz:div> Instead of a Content Selector

With the <pz:div> JSP tag, you can provide in-line HTML personalization. You can populate a JSP with sets of inline 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 inline content. For example, say you have created a user segment called `booknerdUserSegment.seg` in WebLogic Workshop that makes anyone a member who has a "bookNerd" user profile property set to "true". You could do the following:

```
<pz:div rule="booknerdUserSegment">

       <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 as content selectors. The difference is that content selectors retrieve their content from the Virtual Content Repository, and <pz:div> encloses its content inline in the JSP.

CHAPTER 4

Placeholders

Placeholders display a single content item on a JSP—a content item dynamically retrieved from BEA's Virtual Content Repository. A placeholder retrieves content using predefined queries that are put in the placeholder.

Think of a placeholder as a bucket that holds 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 WebLogic Workshop and a companion JSP tag that performs the processing.

In Figure 4-1, the placeholder file is called "foo", and the JSP tag references the placeholder file.

Figure 4-1 The two parts of a placeholder: a placeholder file and a JSP tag

This chapter describes how queries are put in placeholders and how the placeholder decides which query to run and which content item to display.

This chapter includes the following sections:

- Queries that Placeholders Run

Queries that Placeholders Run

Placeholders can run two types of queries: default queries and campaign queries. Both types of queries have the exact same structure but come from different places.

- Default queries – When you create a placeholder file in WebLogic Workshop, you can also add one or more queries to that placeholder file that always stay with the placeholder (unless you change or remove the queries in WebLogic Workshop). These are default queries. They return content from the Virtual Content Repository regardless of who is viewing the portlet or JSP in your portal. Creating a default query for a placeholder is a best practice, because it ensures that no matter what, a content item will always appear in the placeholder.

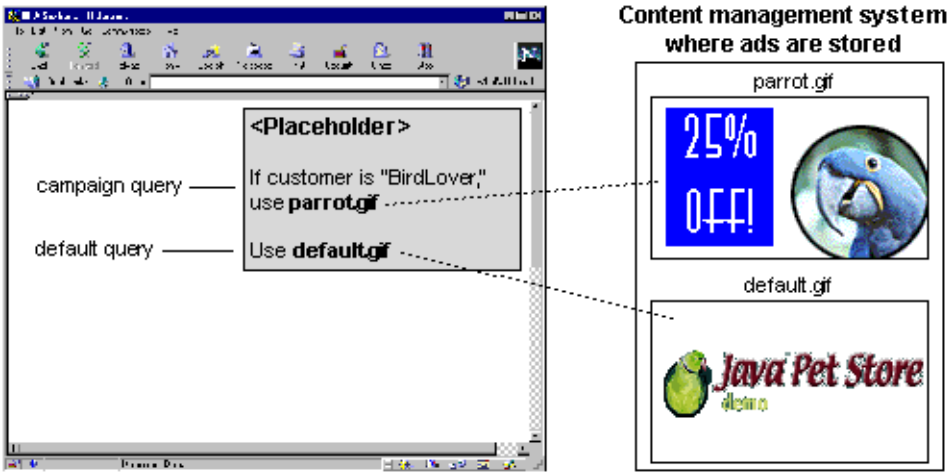
  A placeholder can run a default query each time a user loads a page that includes the placeholder. For example, you define a header for a particular shell that contains a JSP file with a placeholder. If that placeholder contains a default query, a user is likely to see different content in the header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

- Campaign queries – When you create a campaign in WebLogic Workshop, one of the things you may most often 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 a certain actions (events) occur, such as a user logging in to the desktop who belongs to a certain user segment.

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

Figure 4-2 shows the location on a page where the placeholder will display 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, first, which of its queries to run, and then which of the retrieved content items to display.

Figure 4-2 Placeholders can display default content and campaign content



Types of Content Placeholders Can Display

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 – For this type of document, a placeholder generates an <img> tag with attributes that the browser needs to display the image. If you want images to be clickable, you must specify the target URL and other link-related information as ad attributes in your content management system.
- Shockwave files – For this type of document, a placeholder generates the <OBJECT> tag, which Microsoft Internet Explorer on Windows uses to display the file, and the <EMBED> tag, which browsers that support the Netscape-compatible plug-in used to display the file. In your content management system, you can specify attributes for the <OBJECT> and <EMBED> tags.
- XHTML (a fragment or an entire document). For this type of document, a placeholder passes the text directly to the JSP.

If you want to use additional MIME types in placeholders, see “Supporting Additional MIME Types” in the WebLogic Workshop help system at http://e-docs.bea.com/workshop/docs81/doc/en/portal/buildportals/mimetypes.html.

Setting up Content for Use in Placeholders

There are specific properties you can add to content items to better support the use of content in placeholders. See Setting up Content in this guide.

Creating Placeholders

Creating a placeholder involves creating a placeholder file in WebLogic Workshop and using companion <ph:placeholder> JSP tags in any relevant JSPs. Each <ph:placeholder> JSP tag must use its "name" attribute to reference a placeholder you have created in WebLogic Workshop.

For steps on creating placeholders, see the following topics in the WebLogic Workshop Help system:

- "Creating Placeholders" at http://edocs.bea.com/workshop/docs81/doc/en/portal/buildportals/createPlaceholders.html.
- "<ph:placeholder> Tag" at http://edocs.bea.com/workshop/docs81/doc/en/portal/taglib/www.bea.com/servers/portal/tags/placeholder/placeholder.html.

Building Content Queries

WebLogic Portal provides advanced content query features for building powerful content queries. For more information, see "Building Content Queries with Expressions" in the WebLogic Workshop help system at http://e-docs.bea.com/workshop/docs81/doc/en/portal/buildportals/contentExpression.html.

Adding Placeholders to JSPs

After you create a placeholder file in WebLogic Workshop, you can add the placeholder to a JSP in a number of ways in WebLogic Workshop:

- Drag a placeholder from the Application window onto a page of an open portal file. When you do this, three things happen:
  - The Portlet Wizard appears, letting you quickly create a portlet that will display the placeholder.
  - The resulting portlet is automatically added to the portal page.

  - 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 to which you want to add the placeholder, and drag the placeholder file from the Application window 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 added automatically. See Figure 4-1.
- Drag the <ph:placeholder> JSP tag from the Palette window (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.

How Placeholders Choose Which Query and Which Content

Placeholders go through a two-step process in choosing which content to display. This section describes:

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

1. How a Placeholder Chooses a Query

When you add a default query to a placeholder or create a campaign query in WebLogic Workshop, 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 more information, see Combining Default Queries and Campaign Queries in Placeholders.

2. How a Placeholder Chooses 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 Setting up Content in this guide.

The higher the adWeight number you assign to a content item, the higher the likelihood that the content item will be displayed in a placeholder when the content item is retrieved by a query.

Combining Default Queries and Campaign Queries in Placeholders

Placeholders can often 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. This section describes techniques you can use to clear out unwanted queries and better control the content that is displayed in a placeholder.

- When you create default queries in a placeholder, you can designate that default queries do not run when a placeholder contains campaign queries. In WebLogic Workshop, open a placeholder file and select the default query. In the Property Editor window, set the Mix Globals property "false". This suppresses default queries when campaign queries are present.
- When you create a content action in a campaign, use the "Remove (all) existing content" option to minimize the number of queries held in a placeholder at a given time.
- For campaign content actions, set the duration (time to live) to an appropriate time so that the content action stops putting queries in the placeholder when you want it to stop.
- To more accurately 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.
- To clear any content that has been put in a placeholder, reset the previously placed content / ad buckets using the reset feature as described in "Resetting Campaigns" in the Campaigns chapter of this guide.

Using <ad:adTarget> 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.

For information on building queries, see “Building Content Queries with Expressions” in the WebLogic Workshop help system at http://e-docs.bea.com/workshop/docs81/doc/en/portal/buildportals/contentExpression.html.

See “<ad:adTarget> Tag” in the WebLogic Workshop help system at http://e-docs.bea.com/workshop/docs81/doc/en/portal/taglib/www.bea.com/servers/portal/tags/ad/adTarget.html.

CHAPTER 5

Campaigns

Campaigns are powerful tools for personalization, letting you target users with specific Web content, e-mails, and discounts based on specific conditions and fine-grained rules.

The following topics guide you through the campaign creation process:

Overview of Campaigns

Campaigns let you provide the following for individual users:

- **Personalized Web content** – When you use a campaign to display personalized content, the content (such as an image) is retrieved from BEA's Virtual Content Repository through a query and displayed in a placeholder on a JSP, whether the JSP is in a portlet or in a desktop header region.
- **Predefined e-mails sent automatically** – The campaign service reads user profile properties to obtain a user's e-mail address, then sends a predefined e-mail to that user.

- **Personalized discounts in a commerce application** – If you have built a commerce application with the necessary catalog and shopping cart functionality, you can provide a variety of discount types to specific users.

Campaigns are extremely 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 looks like this: `<ph:placeholder name="myPlaceholder1"/>`. Add JSP placeholder tags (uniquely identified by the "name" attribute) anywhere you want in your portal's JSPs. Then simply define your campaigns to use the existing placeholders, each of which can display content unique to the campaign and to the individual users. Campaigns can change and be added, but you never have to change your JSP code. The placeholders you need stay the same.

In addition to providing this high level of flexibility and personalization, campaigns, as the name implies, begin at a specific time and end when their purpose has been fulfilled, whether specific goals have been achieved or a time deadline has been reached. Campaigns can even be set up to run only once for each visitor.

Structurally, a campaign contains one or more scenarios, and each scenario contains one or more actions that show personalized content, send an automatic e-mail, and/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 (though you are not required to assign user segments to scenarios). For example, you could create a campaign with two scenarios: One that will target its actions only to males and the other that will target its actions only to females. The logic would work like this in a scenario for females:

Are you a member of the "female" user segment?

- **Yes**. Your user profile says you are. If this campaign is triggered, you will be targeted with any of the actions in this scenario that apply to you.
- **No**. Your user profile says you are a male. If this campaign is triggered, you will *not* be targeted with any of the actions in this scenario.

Campaign Examples

The following scenarios provide examples of the versatility and power provided by 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 to 30, during which time the campaign displays an Open Enrollment graphic in the portal header region and, when employees make changes to their benefits and click Submit, they are automatically sent a confirmation e-mail.

- 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.

Campaigns and behavior tracking are not currently supported for anonymous, non-trackable users. See "Anonymous Users" in the *User Management Guide* at http://edocs.bea.com/wlp/docs81/users/anonymous.html.

How Campaigns are Triggered

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 do not trigger campaigns are DisplayContentEvent, DisplayProductEvent, BuyEvent, SessionBeginEvent, and SessionEndEvent, as listed in `<PortalApplication>/wps.jar/com/bea/campaign/internal/listeners.properties`.)

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.

When campaign actions are not triggered as expected using session, request, and event properties, one or more of the following is usually to blame:

- No event was fired that the campaign service was listening for.
- The session, request, or event properties contained in the campaign rule were not part of the Request object snapshot taken when the event was fired.
- 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.

Consider the following Campaign action rules as created in WebLogic Workshop:

When all of these conditions apply:

an HTTP request has the following properties:

RequestPropertyOne is equal to “success”

any of the following events has occurred:

SessionLoginEvent

Do the following [content action, e-mail action, or discount action].

The rule will be evaluated only if an event for which the campaign service is listening occurs. (This event need not 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.

Here is how the previous Campaign action rules would be evaluated:

- Is there a request property called RequestPropertyOne with a value of “success”?
- Am I 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 will not be 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 would have fired if the request property evaluated to true.

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 to learn how to send batch mode emails. See Creating Campaigns for instructions on how to change the mailing behavior to real-time.

Preparing to Use Campaigns

Before you create campaigns, ensure that you have performed the following appropriate steps.

Note: Campaigns and behavior tracking are not currently supported for anonymous, non-trackable users. See “Anonymous Users” in the *BEA WebLogic User Management Guide* at http://edocs.bea.com/wlp/docs81/users/anonymous.html.

1. Create a Portal Application

See Creating a Portal Application and Portal Web Project in the WebLogic Workshop help system.

2. Set up Content

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.

For more information on setting up content for use in interaction management, see Setting up Content in this guide.

For instructions on using content management in WebLogic Portal, choose from the list of Content Management topics on the e-docs home page at http://edocs.bea.com/wlp/docs81/index.html.

Goal Setting Performance

If you are using Goal Setting in campaigns to end a campaign based on the number of content items displayed and/or clicked:

- You can determine how frequently the campaign service checks to see if goals have been met. In the WebLogic Administration Portal, select **Service Administration**, select **Campaign Service** in the resource tree, and set the **Goal Check Time** to the frequency you want. The default is 300000 milliseconds (5 minutes). Less-frequent goal checks improve performance, but the campaign service is slower to find out whether goals have been met. For testing, set the value to 0 so that there are no delays in checking for goals.
- You can determine how many impressions/clickthroughs occur before that number is written to the database. In the database, the campaign service compares the current count to the impressions/clickthroughs goal you set in the campaign. In the WebLogic Administration Portal, select **Service Administration**, select **Ad Service** in the resource tree, and set the **Display Flush Size** to the number you want. The default is 10. A larger flush size improves performance, but the campaign service is slower to find out whether goals have been met. For testing, set flush size to a small number to ensure campaigns end right when your goals have been met.

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

3. Create Placeholders

Campaigns use placeholders to display personalized Web content. If you will display personalized content through campaigns, create the placeholders that will hold your campaign queries and display the Web content.

For more information on placeholders, see Placeholders in this guide. In particular, see the section on "Combining Default Queries and Campaign Queries in Placeholders."

4. Create User Segments

If you want to trigger campaign scenarios based on users who are grouped dynamically based on specific characteristics, create user segments. For more information on user segments, see the Interaction Management Overview in this guide and "Creating User Segments" in the WebLogic Workshop help system at http://e-docs.bea.com/workshop/docs81/doc/en/portal/buildportals/CreatingSegments.html.

5. Create Property Sets

If you will trigger campaigns based on user, event, HTTP session, or HTTP request properties, perform any of the following relevant procedures found in the WebLogic Workshop help system:

- Creating User Profile Properties
- Registering Custom Events
- Creating Session Properties
- Creating Request Properties

For more information on how these properties are used in interaction management, see the Interaction Management Overview in this guide.

6. Set up E-Mail Messages

If you will send automatic e-mails in campaigns, make sure you have done the following:

1. Define an e-mail address property – Create a property in a user profile property set that will 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 for sending automatic e-mails to users. In the WebLogic Administration Portal, choose **Service Administration --> Campaign Service**. Enter the property set name and the name of the e-mail property you set up in the previous step.

   You can also tell the campaign service to send automatic e-mails only if users have specified that they want to receive them. To achieve this, define a user profile property with the single, restricted values of true and false. Then, in the Campaign Service configuration, use the `Opt In` fields to enter the property set and name of the property containing that preference. E-mails will not be sent to users who have their property value set to false.

3. Set SMTP for outgoing mail – In the WebLogic Administration Portal Server Administration tools, choose **Service Administration --> Mail Service**. In the SMTP Host Name field, set the host name for your e-mail server's outgoing mail.

4. Create e-mail messages – Create your predefined e-mail messages. E-mail messages can be in any of the following formats: TXT, HTML, JSP, or XML (with style sheets). Store them in the following location: `<PortalWebProject>/campaigns/emails`. If you want to use a different directory for storing e-mail files, follow the instructions in the next step.

   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, e-mails are not sent. See Setting Up Bulk E-Mail Messages to learn how to change the mailing behavior to real-time

5. Set e-mail security – Use the following information to prevent unauthorized access to e-mail messages:

   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 `<PortalWebProject>/campaigns/emails`. Use the following information if you need to change the default e-mail security configuration.

   Email files are secured by the following deployment descriptors:

   – `<PortalApplication>/wps.jar/META-INF/weblogic-ejb-jar.xml` - 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 Server Administration Console at the server level. In a portal domain

created with the Configuration Wizard, the groups “Administrators” and “PortalSystemAdministrators” that are provided by default are configured to be members of the global PortalSystemAdministrator role. Because the user “portaladmin” (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.

If you need to use a different user for e-mail security: 1) Back up wps.jar; 2) Un-jar wps.jar and change the name of the user in weblogic-ejb-jar.xml; 3) Make sure the user exists; 4) Make sure 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 wps.jar.

– `<PortalWebProject>/WEB-INF/web.xml` – 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>
```

If you need to use a different directory for storing e-mail files:

1) Change the <url-pattern> path in web.xml to ensure the files in the new directory are secured.

2) In the WebLogic Administration Portal, choose Service Administration > Campaign Service, and change the directory in the Base Directory for Email Browsing field.

3) Redeploy the application.

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 `/campaigns/emails/q1`, the url-pattern `/campaigns/emails/*` does not protect the e-mail files in the `/q1` directory. To protect those e-mail files, the url-pattern must be `/campaigns/emails/q1/*`.

After making changes to the Campaign Service settings in the WebLogic Administration Portal, redeploy the application (or restart the server during development in WebLogic Workshop) to apply your changes.

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 cron 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`.

Note: When you send campaign e-mails, the default is set to `true` to use a batch mode operation. Batch mode requires you to run the `mailmanager.bat` file from the `<root>/weblogic81/portal/bin` directory to send the e-mails through the SMTP server. For WebLogic Workshop 8.1.x, you can request a patch to modify the `batch flag` to change it to `false` to send the campaign e-mails in real-time.

This section contains the following topics:

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)
   or
   `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.p13n.mail.MailManager` class.

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 5-1 contains a list of the command arguments.s

Table 5-1 Mailmanager Command Arguments

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

Table 5-2 contains examples of `mailmanager` commands.s

Table 5-2 Examples of Mailmanager Commands

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

Sending Bulk E-Mail

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.
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 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. You must specify the name of a batch when you use the `mailmanager` command to send mail, so 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 after you send them (see Sending Bulk E-Mail).

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
```

7. Set up Commerce for Discounts

If you are going to create discount campaign actions, perform the following steps. For details on these steps, see "Building a Commerce Application" in the WebLogic Workshop help system at http://e-docs.bea.com/workshop/docs81/doc/en/portal/buildportals/navCommerce.html.

- 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 surface catalog items from the Virtual Content Repository and identify them with "categories" and "sku" numbers.
- Create discounts and use the commerce API to surface the discounts in your shopping cart. If desired, use the API to surface the discount's description next to the discount amount displayed in the shopping cart.

8. Trigger the 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 Events in Campaigns" on page 6-39 and "Registering Events for Campaigns" on page 6-40 for instructions. See also "How Campaigns are Triggered" on page 5-3.

Creating Campaigns

For instructions on using all the previously mentioned pieces to create a campaign in WebLogic Workshop, see "Creating Campaigns" in the WebLogic Workshop help system at http://e-docs.bea.com/workshop/docs81/doc/en/portal/buildportals/createCampaigns.html.

Use the following guidelines for creating campaigns:

To use session or request properties to trigger campaign actions, make sure you do the following:

- 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/set any event properties you want to use.
- If you want to use session properties to trigger campaign actions, make sure the firing event is in the same session containing the session properties you want to use.

Note: When you send Campaign emails, the default is set to `true` to use a batch mode operation. Batch mode requires you to run the `mailmanager.bat` file from the `<root>/weblogic81/portal/bin` directory to send the e-mails through the SMTP server. For WebLogic Workshop 8.1.x, you can request a patch to modify the `batch flag` to change it to `false` to send the Campaign e-mails in real-time.

Resetting Campaigns

You can reset different aspects 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 so that the users who were supposed to see personalized content one time 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.

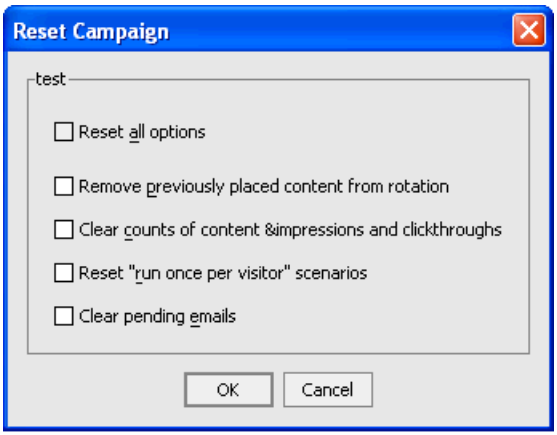
You can reset campaigns either in the development environment (for testing) or in the production environment.

Resetting a Campaign in the Development Environment

With a campaign file open, choose **Portal --> Reset Campaign**. The Reset Campaign dialog box appears, as shown in Figure 5-1.

For details on using this feature for testing, see Testing Campaigns in this chapter.

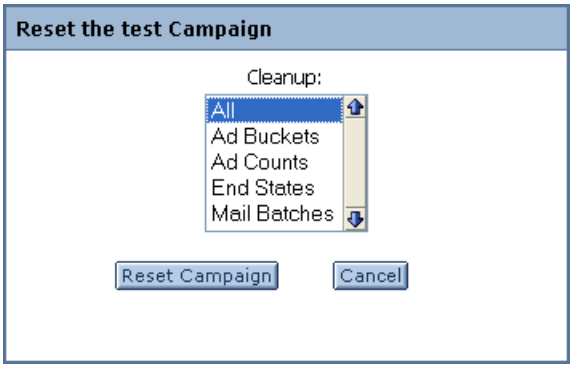
Figure 5-1 Resetting a campaign in the development environment



Resetting a Campaign in the Production Environment

To reset a campaign in the production environment, select the campaign in the WebLogic Administration Portal and click the **Reset Campaign** button. The Reset dialog box appears, as shown in Figure 5-2.

Figure 5-2 Resetting a campaign in a production environment



Testing Campaigns

Use the following guidelines for testing campaigns in your development environment on your development server:

- Make sure the campaign is complete. The entire campaign, each scenario, and each action have specific conditions for completeness. When you select each, the "Is Complete" property in the Property Editor window displays a read-only value of "true" or "false". If "Is Complete" 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 for completeness.
- Make sure the campaign is active. With the campaign selected (not a scenario or action), make sure the Active property in the Property Editor window is set to "true".
- Just below the Campaign Designer window, if you see the text "Campaign is currently stopped", you must change the Start Date and/or Stop Date properties so that the current date falls between the two. Once the current date is within the campaign date range, the "Campaign is currently stopped" text disappears.
- If your campaign uses Goal Setting to end a campaign based on content impressions or clickthroughs, modify the following Service Administration settings in the WebLogic Administration Portal:

  a. Select **Campaign Service**, 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.

  b. Select **Ad Service**, and set the Display Flush Size to 1. This writes each impression or clickthrough to the database each time it occurs and ends the campaign on the exact number of impression/clickthrough counts you have established. If, for example, you wanted to end a campaign on 5 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 which point the campaign service would see that the 5 impressions had already been met, effectively ending the campaign after 10 impressions rather than 5.

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

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

- 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 WebLogic Workshop, choose **Portal --> Reset Campaign** and reset any aspects of the campaign.
- Testing "run once" content actions – Follow this procedure to test content actions that you want to run only once per user:

  a. With the campaign open in WebLogic Workshop, choose **Portal --> Reset Campaign**.

b. In the Reset Campaign dialog box, select “Reset all options” and click OK.

c. View the portal and run the content action by kicking off an event that the campaign service is listening for (such as logging in). Make sure the campaign content is displayed.

d. Log out or click the browser’s Back button to return to the place where you can launch the event again.

e. Go back to the campaign in WebLogic Workshop and choose Portal --> Reset Campaign, and reset the “Remove previously placed content from rotation” option.

f. Go back to the portal and kick off the event again. The campaign content should not appear again.

- Other placeholder issues – To troubleshoot campaign content in placeholders, make sure you 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 Placeholders in this guide.

- Managing content caches (performance testing) – From WebLogic Workshop, you can disable, enable, and flush caches that are used for Web content. With a campaign open in WebLogic Workshop, the Portal menu provides those cache options. The following caches are affected (which you can view in the WebLogic Administration Portal by selecting Service Administration and expanding the Cache Manager).
  - adBucketServiceCache – (Reserved for future use.)
  - searchCache – Caches the results of content searches for the virtual content repository.
  - documentMetadataCache – Caches the results of document searches for the DocumentManager. It is not used by the content repositories.
  - binaryCache.BEA Repository – Caches binary property values for the BEA Repository.
  - documentContentCache – Caches the document bytes for the DocumentManager. It is not used by the content repositories.
  - nodeCache.BEA Repository – Caches Nodes for the BEA Repository.
  - documentIdCache – Caches the results of document searches (ids only) for the DocumentManager. It is not used by the content repositories.
  - adServiceCache – Used by the ad service to cache the results of searches for content rendering.

Note: For optimal performance, be sure to enable these caches in your production environment.

Optimizing Campaign Performance

See Tuning for Campaigns in the *Performance Tuning Guide*.

CHAPTER 6

Events and Behavior Tracking

As 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, events are generated.

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 programmatic functionality when events occur.

Note: Interaction management events are distinct from portlet events, which provide a framework for inter-portlet communication.

Following are examples of 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 (such as manager or regular employee) logs in to your human resources Intranet most often. 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.

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; and 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 are 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 hit. 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.

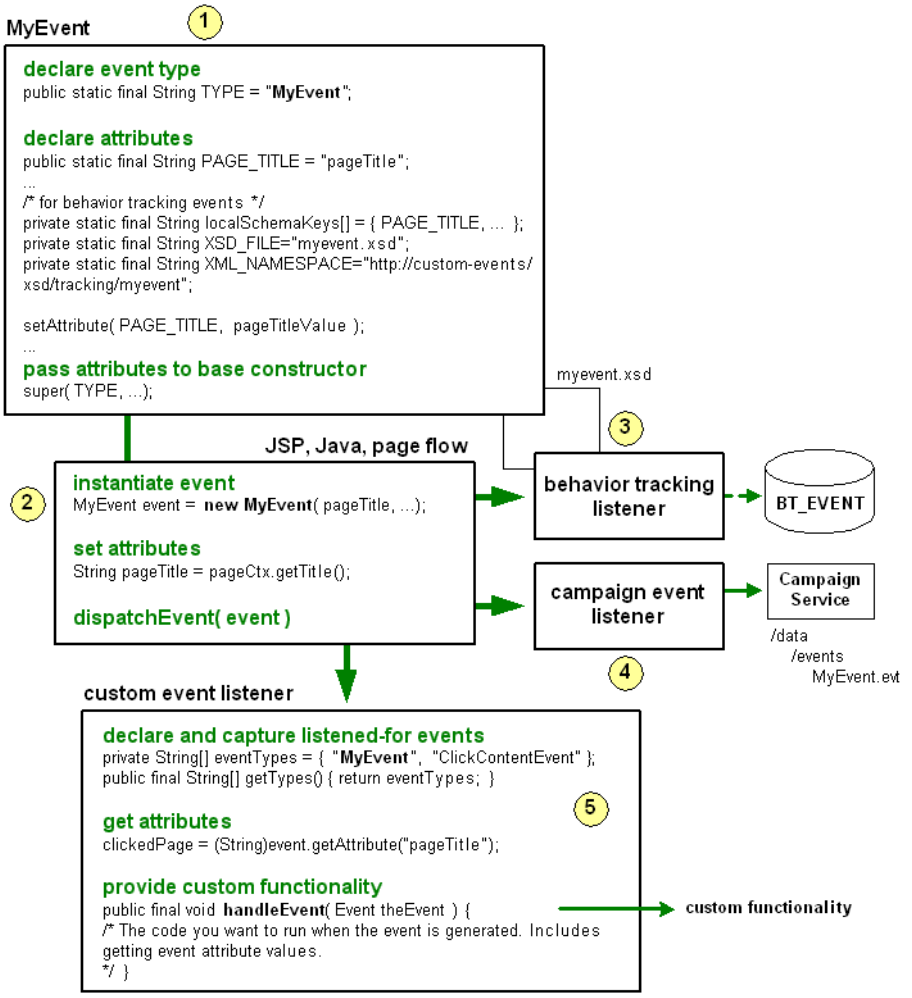
This chapter describes the components of the event framework, helps you plan an event strategy by explaining the purpose and use of each piece of the framework, describes WebLogic Portal's predefined events, and provides guidance and instructions on using events in your applications.

This chapter includes the following sections:

Overview of the Event Framework

The event framework, Figure 6-1, gives you the flexibility to handle events in many ways. The table following the figure describes the pieces of the framework.

Figure 6-1 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 any types 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 hit most often, or to trigger a campaign when a specific JSP is viewed; you can capture information about content retrieved from the virtual content repository; 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 and Configuring Behavior Tracking" on page 6-23.

4 The campaign event listener listens for and handles all events, except those listed for exclusion 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 ever 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/events` directory in WebLogic Workshop). An event property set contains the exact names of the attributes you are setting in your event. The campaign designer interface uses the event property set in drop-down fields that you use to create the campaign definition.

   For information on campaigns, see the Campaigns chapter in this guide.

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 it is interested in—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.

Planning an Event Strategy

WebLogic Portal's event framework provides many options for generating and handling events, as described in the previous section. This section provides guidelines to help you determine the pieces of the event framework you want to use to implement the functionality you need.

When to Use WebLogic Portal's Predefined Events

WebLogic Portal provides many predefined behavior tracking events you can use in your applications, described in "Predefined Events Provided by WebLogic Portal" on page 6-8. Each event collects specific attributes and structures those attributes as XML, and the behavior tracking listener puts the XML in a buffer for insertion into BT_EVENT database table.

Most of the predefined events also have predefined event property sets in WebLogic Workshop, stored in the portal application's `data/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.

Use WebLogic Portal's predefined events when:

- The predefined events capture the attributes you want.
- You want to store event data as XML in the BT_EVENT table; and/or
- The events capture the attributes you want, but you want to handle the events in a customized way by creating your own event listener; and/or
- You want to use the events in your campaigns.

When to Create a Custom Event

If none of WebLogic Portal's predefined events captures 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.

Behavior Tracking Events

The only reason to create a custom behavior tracking event is when none of WebLogic Portal's predefined events captures the event attributes you want and you want 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, there is no reason to create a custom Behavior Tracking event.

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

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.

Create a custom regular event when:

- You want to capture a specific set of attributes with an event and use that event to trigger campaigns; and/or
- 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).

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 by way of 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.

Deploying 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 Datasync Web Application to update the event

properties in the database. For information on deployment and the Datasync Web application, see the Production Operations User Guide.

Predefined Events Provided by WebLogic Portal

This section provides details on using the predefined behavior tracking events provided by WebLogic Portal. 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. Behavior tracking must be enabled to persist the event attributes (as described in "Enabling and Configuring Behavior Tracking" on page 6-23). You can also use these events to trigger campaigns.

A few predefined events are provided for compatibility with legacy WebLogic Portal commerce applications: AddToCartEvent, PurchaseCartEvent, and RemoveFromCartEvent. These events were dispatched by pipeline components that were part of the flow of a commerce application. In previous versions of WebLogic Portal, application flows and catalog management used different frameworks for retrieving event property values and dispatching the events. 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 (such as from 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 6-35.

SessionLoginEvent

Use the SessionLoginEvent to dispatch an event when a user logs in to a portal (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 6-3.

Properties	session-id user-id See Table 6-1 for details on how these properties are set.
How to dispatch	Use any of the following: **Page Flow**: Session Login Event control **API**: Create an instance of the event in your code and dispatch it with `com.bea.p13n.tracking.TrackingEventHelper.dispatchSessionLoginEvent()` or Use the `PortalServletFilter` to check for user logins (`com.bea.p13n.servlets.PortalServletFilter`, a registered filter in a portal Web project `web.xml` file that has the `fireSessionLoginEvent` parameter set to true by default).
Javadoc	`com.bea.p13n.tracking.events.SessionLoginEvent` `com.bea.p13n.servlets.PortalServletFilter`

SessionBeginEvent and SessionEndEvent

SessionBeginEvent and 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 6-3.

SessionBeginEvent and 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, see How Campaigns are Triggered in the "Campaign" chapter of this guide.

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; for example, with a registration portlet).

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 6-3.

Properties	session-id user-id See Table 6-1 for details on how these properties are set.
How to dispatch	**Page Flow**: User Registration Event control **API**: Create an instance of the event in your code and dispatch it with `com.bea.p13n.tracking.TrackingEventHelper.dispatchUserRegistrationEvent()`
Javadoc	`com.bea.p13n.tracking.events.UserRegistrationEvent`

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 6-3.

Properties	application-name currency quantity session-id sku unit-list-price user-id See Table 6-1 for details on how these properties are set.

How to dispatch	See "Dispatching Events" on page 6-37.
Javadoc	`com.bea.commerce.ebusiness.tracking.events.AddToCartEvent`

RemoveFromCartEvent

Use RemoveFromCartEvent to generate 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 6-3.

Properties	application-name currency quantity session-id sku unit-price user-id See Table 6-1 for details on how these properties are set.
How to dispatch	See "Dispatching Events" on page 6-37.
Javadoc	`com.bea.commerce.ebusiness.tracking.events.RemoveFromCartEvent`

PurchaseCartEvent

Use PurchaseCartEvent to dispatch 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 6-3.

Properties	application-name currency order-id session-id total-price user-id See Table 6-1 for details on how these properties are set.
How to dispatch	See "Dispatching Events" on page 6-37.
Javadoc	`com.bea.commerce.ebusiness.tracking.events.PurchaseCartEvent`

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 6-3.

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 Registering Custom Events in the WebLogic Workshop help system.

For instructions on using rules, see the Rule Event Control and the guide to Using Rules in Portal Applications.

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 6-3.

Properties	application-name campaign-id document-id document-type placeholder-id scenario-id session-id user-id
How to dispatch	This event is generated automatically whenever a campaign puts a content item in a placeholder.
Javadoc	`com.bea.campaign.tracking.events.DisplayCampaignEvent`

Display Content Events

WebLogic Portal provides a Display Content Event control and a <BehaviorTracking:displayContentEvent/> JSP tag that let you generate a behavior tracking event when you display a piece of content in a JSP.

See Table 6-1 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 6-3.

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 How Campaigns are Triggered in the "Campaign" chapter of this guide.

Display Product Events

WebLogic Portal provides a <productTracking:displayProductEvent/> JSP tag that lets you generate a behavior tracking event when you display a product from your catalog.

See Table 6-1 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 6-3.

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 How Campaigns are Triggered in the "Campaign" chapter of this guide.

ClickCampaignEvent

Use ClickCampaignEvent in conjunction with the `ClickThroughEventFilter` to generate an event when a user clicks a content item displayed by a campaign.

To enable content clicking, you must do the following:

- 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 6-17.
- Configure your content items with specific properties, described in the Setting up Content chapter of this guide.

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 6-3.

Properties	application-name campaign-id document-id document-type placeholder-id scenario-id session-id user-id See Table 6-1 for details on how these properties are set.
How to dispatch	Follow the instructions in "Generating Events for Content Clicks" on page 6-17 for using the ClickThroughEventFilter, and configure the appropriate content properties in the virtual content repository, as described in described in the Setting up Content chapter of this guide.
Javadoc	`com.bea.campaign.tracking.events.ClickCampaignEvent`

ClickProductEvent

Use ClickProductEvent in conjunction with the `ClickThroughEventFilter` to generate an event when a user clicks a "product" content item. Product content items are typically those in a catalog or shopping cart. 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 6-17.

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 6-3.

Properties	application-name category-id document-id document-type session-id sku user-id See Table 6-1 for details on how these properties are set.
How to dispatch	Follow the instructions in "Generating Events for Content Clicks" on page 6-17 for using the ClickThroughEventFilter. Use the <productTracking:clickProductEvent> JSP Tag to set the event attributes that are passed as parameters to the `ClickThroughEventFilter`, and set up the appropriate <a href> in your JSP. For sample code, see "Use the ClickThroughEventFilter" on page 6-17.
Javadoc	`com.bea.commerce.ebusiness.tracking.events.ClickProductEvent`

ClickContentEvent

Use the ClickContentEvent in conjunction with the `ClickThroughEventFilter` to generate an event when a user clicks any content item in a portal 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 6-17.

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 6-3.

Properties	document-id document-type session-id user-id See Table 6-1 for details on how these properties are set.
How to dispatch	Follow the instructions in Generating Events for Content Clicks for using the ClickThroughEventFilter. Use the <BehaviorTracking:clickContentEvent> JSP Tag or the Click Content Event control in a page flow to set the event attributes that are passed as parameters to the `ClickThroughEventFilter`. Set up the appropriate <a href> in your JSP. For sample code, see "Use the ClickThroughEventFilter" on page 6-17.
Javadoc	`com.bea.p13n.tracking.events.ClickContentEvent`

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 following JSP tags enable content click events: <BehaviorTracking:clickContentEvent> and <productTracking:clickProductEvent>. The ClickCampaignEvent also generates content click events. To enable content to be clicked so that an event is dispatched to the Event Service, follow the instructions in these sections:

- Use the ClickThroughEventFilter
- Ensure the Event Service Is Registered in web.xml and weblogic.xml
- Campaign Clickthroughs

Use the ClickThroughEventFilter

Use the `ClickThroughEventFilter` whenever you use /ShowBinary 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 in conjunction with a click event JSP tag. Then, when a user clicks the content, a click content event is generated by the `ClickThroughEventFilter`.

Note: Using /ShowBinary as the URL pattern is the most logical choice for showing binary content. However, you are not limited to using /ShowBinary.

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.p13n.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 %>"><img src="<%=request.getContextPath() + "/ShowBinary"
+ node.getPath()%>" ></a>
```

Ensure the Event Service Is Registered in web.xml and weblogic.xml

The `ClickThroughEventFilter`, described in the previous section, references the Event Service, so you must make sure the Event Service is included in your portal Web project `web.xml` and `weblogic.xml` files.

When you create a new portal Web project, the following entries are included in `web.xml` and `weblogic.xml` by default. However, check to make sure your files contain these entries:

web.xml

```
<ejb-ref>
    <description>Event Service</description>
    <ejb-ref-name>ejb/EventService</ejb-ref-name>
    <ejb-ref-type>Session</ejb-ref-type>
    <home>com.bea.p13n.events.EventServiceHome</home>
    <remote>com.bea.p13n.events.EventService</remote>
</ejb-ref>
```

weblogic.xml

```
<ejb-reference-description>
    <ejb-ref-name>ejb/EventService</ejb-ref-name>
    <jndi-name>${APP_NAME}.BEA_personalization.EventService</jndi-name>
</ejb-reference-description>
```

Note: `${APPNAME}` is a variable. Do not change it to a hard-coded application name.

Campaign Clickthroughs

To enable campaign clickthroughs that trigger the predefined ClickCampaignEvent, you must configure your content items with specific properties, as described in the Setting up Content chapter of this guide.

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 6-1 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, among others, an attribute called placeholder-id. 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 6-1 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: `com.bea.p13n.events.Event.getApplication()`
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: `com.bea.campaign.CampaignInfo.getUniqueId()`
category-id	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: `com.beasys.commerce.ebusiness.catalog.service.category.CategoryManager.getItemCategories()` If your catalog is built in any other way, get the category-id in the appropriate way.

Table 6-1 Getting attributes for predefined events

Event Attribute	How to get the attribute
currency	Gets the type of currency on an item. You must set this attribute manually. To get the currency in the following way, your commerce application must be built on the WebLogic Portal commerce API: `com.beasys.commerce.axiom.units.Money.getCurrency()` If your currency is set in any other way, get the currency in the appropriate way.
document-id	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. `com.bea.content.Node.getId()` or `getName()` or Use the `<cm:getProperty>` JSP tag to get the `cm_uid` or `cm_nodeName` property.
document-type	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 `com.bea.content.Node.getType()` to retrieve the content's type.
order-id	The unique ID of a customer's order. To get the order-id in the following way, your commerce application must use the WebLogic Portal order framework. Use the following method: `com.beasys.commerce.ebusiness.order.Order.getIdentifier()` If your commerce application is built in any other way, get the order-id in the appropriate way.
placeholder-id	The unique ID of the content placeholder displaying the content. The predefined events that use this attribute set it automatically.

Table 6-1 Getting attributes for predefined events

Event Attribute	How to get the attribute
quantity	The quantity of an item in a shopping cart. To get the quantity in the following way, your commerce application must use the WebLogic Portal shopping cart framework. Use the following method: `com.beasys.commerce.ebusiness.shoppingcart.ShoppingCartLine.getQuantity()` 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: `com.bea.campaign.action.Action.getScenarioId()`
session-id	The unique ID of the current session. This is retrieved automatically by the predefined events, which extend the `TrackingEvent` class, which uses `javax.servlet.http.HttpSession.getId()` and assigns the return value to the session-id property.
sku	The sku number of the catalog item. To get the sku in the following way, you must use the commerce framework to set sku numbers. Use the following method: `com.bea.commerce.ebusiness.tracking.tags.ProductEventTag.getSku()` 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 in the following way, your commerce application must use the WebLogic Portal shopping cart framework. Use the following method: `com.beasys.commerce.ebusiness.shoppingcart.ShoppingCart.getTotalPrice()` If your shopping cart is built in any other way, get the total-price in the appropriate way.

Table 6-1 Getting attributes for predefined events

Event Attribute	How to get the attribute
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: `com.beasys.commerce.ebusiness.shoppingcart.ShoppingCartLine.getUnitPrice()` 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 `TrackingEvent` 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: `com.bea.p13n.usermgmt.SessionHelper.getUserId(request)`

Enabling and Configuring 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.

To activate behavior tracking, register the BehaviorTrackingListener class with the Event Service using the following steps.

Note: This procedure assumes 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/application-config.xml` file. That file is read-only in an EAR.

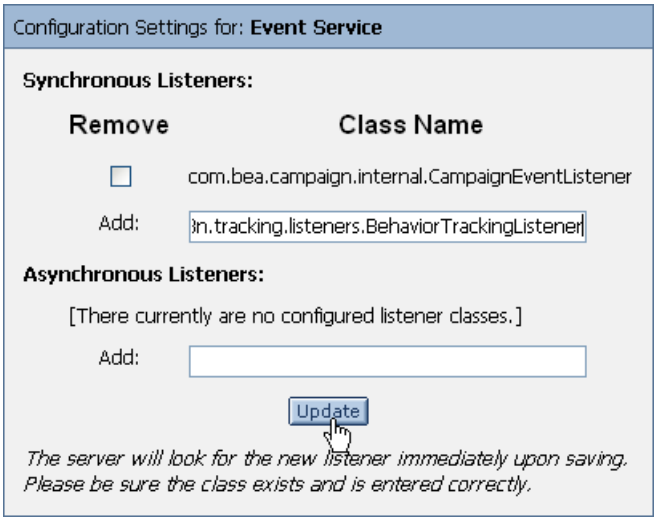
1. Start and log in to the WebLogic Administration Portal as a system administrator, and select **Service Administration > Event Service**.
2. In the **Add** field for Synchronous Listeners, as shown in Figure 6-2, enter the following class:

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

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

3. Click **Update**. The class appears in the Class Name list, and the `application-config.xml` file is updated. Behavior tracking is activated. No server restart or redeployment is required.

Figure 6-2 Activating behavior tracking



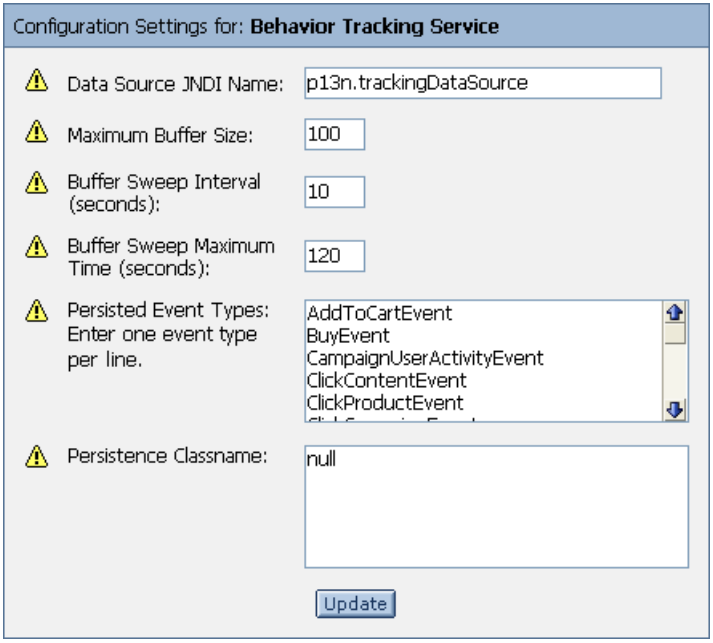
Configuring Behavior Tracking

By default, behavior tracking data is not written to the database the instant 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.

To configure behavior tracking select **Service Administration > Behavior Tracking Service** in the WebLogic Administration Portal, as shown in Figure 6-3, and modify the settings, described in Table 6-2.

You must restart the server for your changes to take effect.

Figure 6-3 Configuring behavior tracking



Use the following field information for guidance.

Table 6-2 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. Default value: 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. Default value: 10 seconds.
Buffer Sweep Maximum	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). Default value: 120 seconds.

Leave the default values for Data Source JNDI Name (p13n.trackingDataSource) and Persistence Classname (null). These 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.

For information on the Persisted Event Types field, see "Creating a Behavior Tracking Event Class" on page 6-29.

Performance Tuning

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.

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 6-35.

Creating a Separate Database for Behavior Tracking Events

For improved performance, if you are heavily using behavior tracking, you can store behavior tracking data in a separate database. In the WebLogic Portal Database Administration Guide, there are explicit instructions for creating a separate behavior tracking database the chapters for each type of database.

Creating Custom Events

If none of WebLogic Portal's predefined events meets your needs, you can create your own custom events. For guidance on when to create custom events and what type to create, see "Planning an Event Strategy" on page 6-5.

Creating a custom event involves the following steps:

- Creating the Event Class
- Creating an XML Schema (Behavior Tracking Only)

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 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. You can trigger campaigns with custom regular events and perform your own event handling if you create a custom event listener.

The following procedure shows you how to create an custom event class.

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.

1. Create a custom event:
   - Create an event class that extends the `com.bea.p13n.events.Event` class. For example:

     ```
     public class MyEvent
       extends com.bea.p13n.events.Event
     ```

   - Declare the type (name) of the event as a String. For example:

     ```
     public static final String TYPE = "MyEvent";
     ```

   - 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";
     ```

– Create the event object:

```
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:

```
event.setAttribute("request", new Request(request, true));
```

– Call the `Event` class constructor and pass the event type back to it:

```
super( TYPE );
```

– Declare the event attributes.

```
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).
```

2. Compile the event class. When you run javac to compile, use the -classpath option to put `p13n_ejb.jar` in the classpath. For example:

```
javac -classpath D:\bea\weblogic\<portalApp>\p13n_ejb.jar MyEvent.java
```

3. Put the compiled class in the server classpath (or in a JAR, and put the JAR in the classpath).

   In WebLogic Workshop with your enterprise application open, select **Tools > Application Properties**. In the Application Properties window, select **WebLogic Server**, and scroll to the bottom of the window to add your path to the **Server classpath additions** list.

Now you can dispatch the event from a JSP, Java code, or a page flow, as described in "Dispatching Events" on page 6-37.

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 6-40.

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 6-35.

Creating a Behavior Tracking Event Class

The only reason to create a custom behavior tracking event is when none of WebLogic Portal's predefined events captures the event attributes you want and you want 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, there is no reason to create a custom Behavior Tracking event. If you do not want to use the Behavior Tracking service, create a custom regular event.

Your behavior tracking event works in tandem 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 following procedure shows you how to create an custom behavior tracking event class.

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.

1. Create a custom behavior tracking event:
   - Create an event class that extends the `com.bea.p13n.tracking.events.TrackingEvent` class. For example:

     ```
     public class MyEventBT
       extends com.bea.p13n.tracking.events.TrackingEvent
     ```

   - Declare the type (name) of the event as a String. For example:

     ```
     public static final String TYPE = "MyEventBT";
     ```

   - 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";
     ```

   - Declare the name of the XML schema file. For example:

     ```
     private static final String XSD_FILE = "mytrackingschema.xsd";
     ```

   - 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";
```

– 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.

– Create the event object with the properties you declared, along with the user-id. For example:

```
public MyEventBT(
    String pageLabel,
    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:

```
event.setAttribute("request", new Request(request, true));
```

– Call the `TrackingEvent` constructor and pass the required arguments back to it in the required order:

```
{
super(
    TYPE,
    session,
    XML_NAMESPACE,
    XSD_FILE,
    localSchemaKeys,
    request );
```

– Declare the event attributes:

```
setAttribute( PAGE_LABEL,  pageLabelValue );
setAttribute( SESSION_ID,  session_id );

if( user_id != null )
    setAttribute( USER_ID, user_id );
else
    setAttribute( USER_ID, "unknown" );
}
```

where `pageLabelValue` is the variable storing the value you retrieved in your code (not shown here).

2. Compile the event class. When you run javac to compile the event class, use the -classpath option to put `p13n_ejb.jar` in the classpath. For example:

```
javac -classpath D:\bea\weblogic\<portalApp>\p13n_ejb.jar MyEventBT.java
```

3. Put the compiled class in the server classpath (or in a JAR, and put the JAR in the classpath).

   In WebLogic Workshop with your enterprise application open, select **Tools > Application Properties**. In the Application Properties window, select **WebLogic Server**, and scroll to the bottom of the window to add your path to the **Server classpath additions** list.

4. Register the behavior tracking event with the Behavior Tracking Service in the WebLogic Administration Portal. This tells the behavior tracking listener to handle this type of event.

   a. In the WebLogic Administration Portal, select **Service Administration > Behavior Tracking Service**.

   b. In the Behavior Tracking Service window (Figure 6-3), enter the name of your behavior tracking class in the **Persisted Event Types** list (such as MyEventBT), and click **Update**.

5. Create an XML schema that determines the structure of the XML document generated by the behavior tracking event. See "Creating an XML Schema (Behavior Tracking Only)" on page 6-33.

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 6-40.

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 6-35.

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. For example, use this technique in a JSP that has a form that can supply values to event properties.

To create an event with a scriptlet:

1. Create an event property set in WebLogic Workshop. See "Registering Events for Campaigns" on page 6-40.
2. After you have created the event property set, open the JSP in which you want to create the event, and select the Source View tab.
3. In the WebLogic Workshop Application Window, drag the event property set file into the the location of 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.p13n.events.Event event = new
com.bea.p13n.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);

%>
```

Notice that 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`, such as from the value of a form field.

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 6-35.

Creating an XML Schema (Behavior Tracking Only)

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:

  ```
  private static final String localSchemaKeys[] =
  {
  SESSION_ID, USER_ID, PAGE_LABEL_KEY
  };
  ```

  your XSD file must list those properties in the same order.

You can view the sample XSD file `ResourceDisplayedEventBT.xsd` in the following file: http://edocs.bea.com/wlp/docs81/interm/src/sample_events.zip.

The XSD is mostly boilerplate. Simply change 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 at the following location: `weblogic81\p13n\lib\p13n_ejb.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_ejb.jar` file using the following steps.

1. Back up your `p13n_ejb.jar` file by creating a copy of it and naming it something like `p13n_ejb.orig`.
2. In your application directory, temporarily add the `lib/schema/<yourschema>.xsd` directory/file.
3. Add the schema to `p13n_ejb.jar`. 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_ejb.jar lib\schema\<yourschema>.xsd
   ```

The schema is added to the JAR file. Redeploy `p13n_ejb.jar`.

Creating Custom Event Listeners

An event listener serves one purpose: When an event occurs that it is listening for, 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 `wps.jar`) and calls the Campaign Service to evaluate and trigger campaign actions.
- BehaviorTrackingListener – Listens for all events registered with the Behavior Tracking service (Figure 6-3) and puts data from behavior tracking events in a buffer, where the data is later moved into the BT_EVENT database table. (You must manually register this listener to activate behavior tracking, as described in "Enabling and Configuring Behavior Tracking" on page 6-23.)

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; or 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:

- `getTypes()` – Tells the Event Service which types of events the listener is interested in.
- `handleEvent()` – Lets you insert the custom functionality you want to perform when the listener receives an event it is interested in.

This section shows you how to write a custom listener for one or more events. The listener can listen for regular events, behavior tracking events, or both.

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.

1. Your custom event listener must implement `EventListener`. For example:

```
public class MyEventListener
    implements EventListener
{
```

2. Define the event(s) that the listener will listen for. For example:

```
private String[] eventTypes = {"MyEvent", "ClickContentEvent"};
```

3. Create the listener:

```
public MyEventListener()
{
}
```

4. Pass the type of events listened for back to the base constructor. This tells the Event Service which events to send to this listener when the events occur:

```
    public String[] getTypes()
    {
        return eventTypes;
    }
```

5. Override the `handleEvent()` method to provide the programmatic functionality you want the listener to perform when the listened-for events occur:

```
    public void handleEvent( Event ev )
    {
        //Put your custom code here.
        //This code is executed when the events occur.
    }
}
```

6. Compile the event class. When you run javac to compile, use the -classpath option to put `p13n_ejb.jar` in the classpath. For example:

```
javac -classpath D:\bea\weblogic\<portalApp>\p13n_ejb.jar
MyEventListener.java
```

7. Put the compiled class in the server classpath (or in a JAR, and put the JAR in the classpath).

   In WebLogic Workshop with your enterprise application open, select **Tools > Application Properties**. In the Application Properties window, select **WebLogic Server**, and scroll to the bottom of the window to add your path to the **Server classpath additions** list.

8. Register the listener with the Event Service.

   a. In the WebLogic Administration Portal, select **Service Administration > Event Service**.

b. In the Event Service window, enter the fully qualified class name in either the Synchronous or Asynchronous field. For example:

```
com.bea.p13n.events.custom.listeners.MyEventListener
```

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

c. Click **Update**. The listener is registered with the Event Service. No server restart is required.

Dispatching Events

With events and listeners in place, you can generate, or dispatch those events in your JSPs, Java code, and page flows. Dispatching an event means having the Event Service send 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 http://edocs.bea.com/wlp/docs81/interm/src/sample_events.zip, a sample event, 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.

Inserted in each of these files is the code shown in Listing 6-1. This example uses code from `page.jsp`. The code dispatches a ResourceDisplayEventBT event when portlets are viewed on a page.

Listing 6-1 Dispatching an event from page.jsp

```
<%@ page import="com.bea.p13n.tracking.TrackingEventHelper,
                 examples.events.ResourceDisplayedEventBT" %>
...
ResourceDisplayedEventBT rde;
...
rde = new ResourceDisplayedEventBT( ppc.getLabel(),  // resourceId,
                                    portletTitle,    // resourceLabel
                                    "portlet",       // resourceType
                                    sessionId,
                                    userId,
                                    "true",  // portlet is being displayed
                                    request,
                                    session );
...
TrackingEventHelper.dispatchEvent( rde );
```

- The file imports the custom event class and the `TrackingEventHelper`, which is used to dispatch the event.
- The event class is assigned to the variable `rde`.
- 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 6-1 illustrates the event life cycle, and in this example, the skeleton JSP is item 2 in the diagram.

The section "Creating an Event With a Scriptlet" on page 6-32 also describes how to dispatch an event for which you have created no event class.

Dispatching events when content is clicked requires special instructions, described in "Generating Events for Content Clicks" on page 6-17.

Some predefined events have their own dispatch methods. See "Predefined Events Provided by WebLogic Portal" on page 6-8.

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.

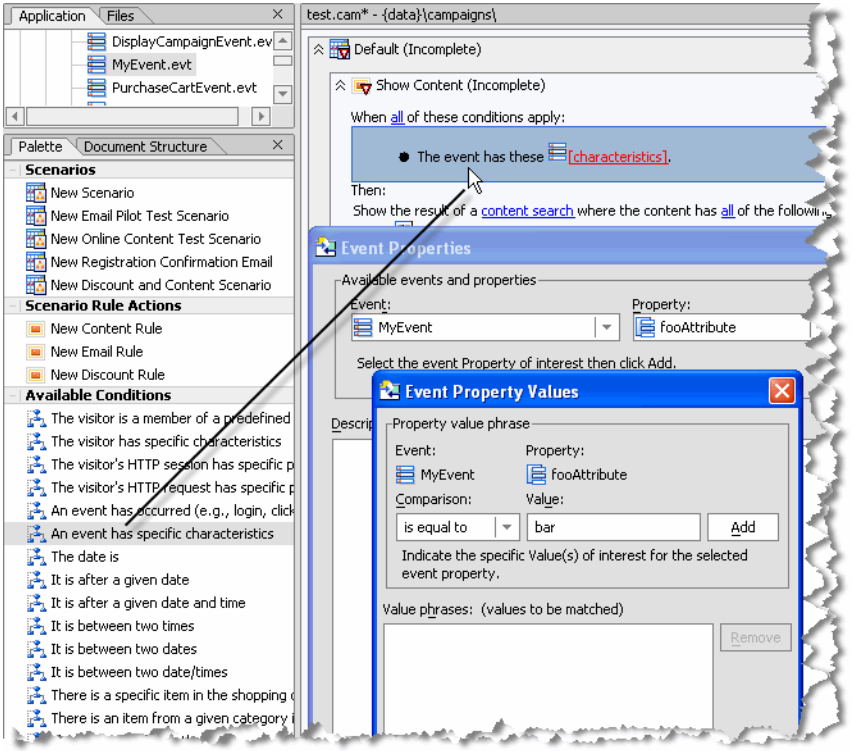
To use an event in a campaign, you do not have to explicitly tell the Campaign Service about your events. The campaign listener listens for all events that are not explicitly excluded in the `listeners.properties` file in `wps.jar`.

When an event occurs that the campaign listener is listening for, 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 6-40.

An example of using an event in a campaign definition is illustrated in Figure 6-4, where a campaign scenario is going to be 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 6-4 Using an event to trigger a campaign scenario



To exercise the most control with your campaign, especially when you want precise control over the personalized content that is displayed, create events that trigger campaign actions at the key places in your application where you want the precise control. For more information on controlling personalized content, see Combining Default Queries and Campaign Queries in Placeholders in the "Placeholders" chapter of this guide.

For information on creating campaigns, see the Campaigns chapter in this guide.

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 have to exactly match the event attribute names.

For instructions on creating event property sets, see Registering Custom Events in the WebLogic Workshop help system.

Changing Event Properties

If you create, modify, or delete event property sets after an application has already been deployed, you must update those property set definitions in the database using the WebLogic Portal Datasync Web application. For more information on the Datasync Web application, see the Production Operations User Guide.

Debugging the Event Service

To debug the Event Service, create the following file:
`weblogic81\portal\debug.properties`.

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.BehaviorTrackingListerner: on
com.bea.p13n.tracking.listeners.BehaviorTrackingListerner: on
com.bea.p13n.tracking.SessionEventListerner: on
```