BEA Campaign Manager for WebLogic
BEA WebLogic Commerce Server
BEA WebLogic Personalization Server

Guide to Events and Behavior Tracking

BEA Campaign Manager for WebLogic 1.1
BEA WebLogic Commerce Server 3.5
BEA WebLogic Personalization Server 3.5
Document Edition 3.5.1
June 2001

Guide to Events and Behavior Tracking

Document Edition	Date	Software Version
3.5.1	June 2001	Campaign Manager for WebLogic 1.1 WebLogic Commerce Server 3.5 WebLogic Personalization Server 3.5

Contents

About This Document

This document describes events and behavior tracking in BEA Campaign Manager for WebLogic™, BEA WebLogic Commerce Server™, and BEA WebLogic Personalization Server™.

This document includes the following topics:

- Chapter 1, "Overview of Events and Behavior Tracking," which describes the high-level architecture for events and behavior tracking. It also provides detailed information about each event type.
- Chapter 2, "Creating Custom Events," describes how to create custom events, custom behavior tracking events, custom event listeners, and custom behavior tracking listeners.
- Chapter 3, "Registering Custom Events," which describes how to register custom events with the BEA Business-Control Center.
- Chapter 4, "Persisting Behavioral Tracking Data," which describes how to record behavior tracking data and the database structure for behavior tracking.

What You Need to Know

This document is intended for the following audiences:

- The Commerce Business Engineer (CBE) or JSP content developer, who uses JSP templates to specify which products and Web site content trigger events.
- The business analyst, who defines the company's business protocols for its Web sites. This user may design scenario actions used in campaigns.

- The System Analyst or Database Administrator, who administers databases.
- The Java developer, who creates Java code for custom events.

e-docs Web Site

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

How to Print the Document

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

A PDF version of this document is available on the Campaign Manager for WebLogic, the WebLogic Commerce Server, and the WebLogic Personalization Server documentation Home page on the e-docs Web site (and also on the documentation CD). You can open the PDF in Adobe Acrobat Reader and print the entire document (or a portion of it) in book format. To access the PDFs, open the Campaign Manager for WebLogic, WebLogic Commerce Server, and WebLogic Personalization Server documentation Home page, click the PDF files button and select the document you want to print.

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

Related Information

The following Campaign Manager for WebLogic, WebLogic Commerce Server, and WebLogic Personalization Server documents contain information that is relevant to using events and behavior tracking.

- *Using the E-Business Control Center*.
- *Guide to Registering Customers and Managing Customer Services*.

Contact Us!

Your feedback on Campaign Manager for WebLogic, WebLogic Commerce Server, and WebLogic Personalization Server documentation is important to us. Send us e-mail at **docsupport@beasys.com** if you have questions or comments. Your comments will be reviewed directly by the BEA professionals who create and update the Campaign Manager for WebLogic, WebLogic Commerce Server, and WebLogic Personalization Server documentation.

In your e-mail message, please indicate that you are using the documentation for Campaign Manager for WebLogic, WebLogic Commerce Server, and WebLogic Personalization Server **Product Version:** release.

If you have any questions about this version of Campaign Manager for WebLogic, WebLogic Commerce Server, or WebLogic Personalization Server, or if you have problems installing and running Campaign Manager for WebLogic, WebLogic Commerce Server, or WebLogic Personalization Server, contact BEA Customer Support through BEA WebSUPPORT at **www.beasys.com**. You can also contact Customer Support by using the contact information provided on the Customer Support Card, which is included in the product package.

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

- Your name, e-mail address, phone number, and fax number
- Your company name and company address

- Your machine type and authorization codes
- The name and version of the product you are using
- A description of the problem and the content of pertinent error messages

Documentation Conventions

The following documentation conventions are used throughout this document.

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

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

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

Convention	Item
UPPERCASE TEXT	Indicates device names, environment variables, and logical operators. *Examples*: LPT1 SIGNON OR
`{ }`	Indicates a set of choices in a syntax line. The braces themselves should never be typed.

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

`|`	Separates mutually exclusive choices in a syntax line. The symbol itself should never be typed.

`...`	Indicates one of the following in a command line:
	■ That an argument can be repeated several times in a command line
	■ That the statement omits additional optional arguments
	■ That you can enter additional parameters, values, or other information
	The ellipsis itself should never be typed.
	Example:
	`buildobjclient [-v] [-o name ] [-f` *`file-list`*`]...`
	`[-l` *`file-list`*`]...`

. . .	Indicates the omission of items from a code example or from a syntax line. The vertical ellipsis itself should never be typed.

1 Overview of Events and Behavior Tracking

To help personalize campaigns and to effectively analyze customer interactions with a Web site, you need a comprehensive event tracking and logging system. To fulfill this requirement, BEA Campaign Manager for WebLogic, BEA WebLogic Commerce Server, and BEA WebLogic Personalization Server include an Event and Behavior Tracking system. Events identify how a customer is currently interacting with an e-commerce site and the Behavior Tracking system records the event information. With these systems you have the ability to specify, customize, and record selected information. Event data can be used by leading e-analytics and e-marketing systems to evaluate behavioral and transactional data from your online customers. With this analysis you can create and enhance personalization rules, customize product offers, and optimize interactive marketing campaigns. This topic introduces you to Events and Behavior Tracking and provides a general survey of the elements that make up this system.

This topic includes the following sections:

- What Are Events?
- Behavior Tracking
- Event Details
- Event Triggers
- Event Mechanism
- Event Sequence

What Are Events?

In general, an event is a notification that something has happened in a computer program. Campaign Manager for WebLogic, WebLogic Commerce Server, and WebLogic Personalization Serverprovide various points for triggering events. Events provide a detailed and comprehensive view of the entire customer life cycle across your e-commerce site. These points can be tailored for your applications.

You can use events with campaigns to enhance promotion of products and services. Additionally, you can use events to gather intelligence to evaluate the effectiveness of a campaign. Underlying campaigns are scenarios. Scenarios are executed in the context of a campaign. Scenarios are a set of rules, called scenario actions, that allow you to personalize customer experiences on your e-commerce site. For example, if a customer clicks a *Subscribe Me* link on your Web site, you may want to send that customer an e-mail confirming the subscription. Using events and scenarios, you can choreograph the interactions between customers and the site.

With regard to tracking visitor behavior for analysis, the primary interest is in what the customer saw and what the customer did. Inherent in this investigation is information about when customers came to the site and when they left it, plus knowledge about which rules were fired during their visit.

Behavior Tracking

The Event Service passes messages to Behavior Tracking. When configured, the Behavior Tracking data is recorded in a relational database. This information can then be used by data-mining systems to provide Web site customer information for e-marketing analysis. Behavior Tracking provides the following kinds of information:

- When did customers start, end, or login to their sessions?
- What content or products did customers see?
- What content or products did customers click on?
- What did customers put in their shopping cart?

- What did customers buy?
- What rules were triggered?

The information generated from these events allows various kinds of behavior analyses, such as the following:

- **Associations**: When one event can be correlated to another event.
- **Sequences**: When one event leads to another later event.
- **Classification**: The recognition of patterns and a resulting new organization of data.
- **Clustering**: Finding and visualizing groups of facts not previously known.
- **Forecasting**: Discovering patterns in the data that can lead to predictions about future customer behavior.

Event Details

This section provides information about the standard events provided by BEA. Specifically, a description of the event, the type of trigger, the class where triggering occurs, which product contains the event, and the elements of the event. Events elements comprise the data that is present within each event object.

Events are organized into categories. The following list presents each type of event category along with a brief description:

- **Session**: The start time, end time, and if executed, the login time of the customer’s session.
- **Registration:** The customer registers on the e-commerce site.
- **Product**: The customer is presented with a product or clicks (selects) the presented product.
- **Content**: The customer is presented some content, such as an ad, or clicks (selects) the presented content.

- **Cart**: An item is added, removed, or updated to the customer’s shopping cart. Also when an entire order is purchased.
- **Buy**: The customer completes the purchase of one or more items.
- **Rules**: The rules that are fired as a customer navigates a Web site.
- **Campaign**: The events generated within the context of a campaign.

Session Events

Session events fire at the start time, end time, and if executed, the login time of a customer’s session.

SessionBeginEvent

Description	Occurs when a customer begins interacting with a Web site.
Trigger Type	Fired internally
Class	Not applicable

Elements	`event_date`
	`event_type`
	`session_id`
	`user_id`

Products	Specific to WebLogic Personalization Server, available in Campaign Manager for WebLogic and WebLogic Commerce Server

SessionEndEvent

Description	Occurs when a customer leaves a Web site, or when the customer’s session has timed out.
Trigger Type	Fired internally

Class	Not applicable

Elements	`event_date`
	`event_type`
	`session_id`
	`user_id`

Products	Specific to WebLogic Personalization Server, available in Campaign Manager for WebLogic and WebLogic Commerce Server

SessionLoginEvent

Description	Occurs when a customer logs on a Web site.
Trigger Type	Fired internally using the Weblogic Server `j_security_hook`
Class	Not applicable

Elements	`event_date`
	`event_type`
	`session_id`
	`user_id`

Products	Specific to WebLogic Personalization Server, available in and WebLogic Commerce Server

Registration Event

Only one registration event exists. It is described in the following table.

UserRegistrationEvent

Description	Occurs when customer registers on a Web site.
Trigger Type	Input processor

Class	`com.beasys.commerce.ebusiness.customer.webflow.LoginCustomerIP`

Elements	`event_date`
	`event_type`
	`session_id`
	`user_id`

Products	Specific to WebLogic Personalization Server, available in Campaign Manager for WebLogic and WebLogic Commerce Server

Product Events

These events occur when customer is presented with a product or clicks (selects) the presented product.

ClickProductEvent

Description	Occurs when a customer clicks a product link.
Trigger Type	JSP Tag
Class	`com.bea.commerce.ebusiness.tracking.tags.ClickProductTag`

Elements	`event_date`
	`event_type`
	`session_id`
	`user_id`
	`document_type`
	`document_id`
	`sku`
	`category_id`
	`application_name`

Products	Specific to WebLogic Commerce Server, available in Campaign Manager for WebLogic

DisplayProductEvent

Description	Occurs when a product is displayed to the customer.
Trigger Type	JSP Tag
Class	`com.bea.commerce.ebusiness.tracking.tags.DisplayProductEventTag`

Elements	`event_date`
	`event_type`
	`session_id`
	`user_id`
	`document_type`
	`document_id`
	`sku`
	`category_id`
	`application_name`

Products	Specific to WebLogic Commerce Server, available in Campaign Manager for WebLogic

Content Events

These events occur when the customer is presented some content, such as an advertisement, or clicks the presented content.

ClickContentEvent

Description	Occurs when a customer clicks some Web site content, such as a link or banner.
Trigger Type	JSP Tag
Class	`com.bea.commerce.platform.tracking.tags.ClickContentTag`

Elements	`event_date`
	`event_type`
	`session_id`
	`user_id`
	`document_type`
	`document_id`

Products	Specific to WebLogic Personalization Server, available in Campaign Manager for WebLogic and WebLogic Commerce Server

DisplayContentEvent

Description	Occurs when content is presented to a customer, usually any content from a content management system.
Trigger Type	JSP Tag
Class	`com.bea.commerce.platform.tracking.tags.DisplayContentEventTag`

Elements	`event_date`
	`event_type`
	`session_id`
	`user_id`
	`document_type`
	`document_id`

Products	Specific to WebLogic Personalization Server, available in Campaign Manager for WebLogic and WebLogic Commerce Server

Cart Events

These events indicate that one or more items are added or removed from a customer's shopping cart.

AddToCartEvent

Description	Occurs when an item is added to a customer’s shopping cart.
Trigger Type	Pipeline component
Class	`com.bea.commerce.ebusiness.tracking.pipeline.AddToCartTrackerPC`

Elements	`event_date`
	`event_type`
	`session_id`
	`user_id`
	`sku`
	`quantity`
	`unit_list_price`
	`currency`
	`application_name`

Products	Specific to WebLogic Commerce Server, available in Campaign Manager for WebLogic

RemoveFromCartEvent

Description	Occurs when an item is removed from a customer’s shopping cart.
Trigger Type	Pipeline component
Class	`com.bea.commerce.ebusiness.tracking.pipeline.RemoveFromCartTrackerPC`

Elements	`event_date`
	`event_type`
	`session_id`
	`user_id`
	`sku`
	`quantity`
	`unit_price`
	`currency`
	`application_name`

Products	Specific to WebLogic Commerce Server, available in Campaign Manager for WebLogic

PurchaseCartEvent

Description	Occurs once for an entire order, unlike the `BuyEvent`, which occurs for each line item. This event is useful for campaigns. You can use it when writing scenario actions to know when your customer makes a purchase with specific characteristics, such as an order greater than $100 or the purchase of a particular product.
Trigger Type	Pipeline component
Class	`com.bea.commerce.ebusiness.tracking.pipeline.PurchaseTrackerPC`

Elements	`session_id`
	`user_id`
	`event_date`
	`event_type`
	`total_price`
	`order_id`
	`currency`
	`application_name`

Products	Specific to WebLogic Commerce Server, available in Campaign Manager for WebLogic

Buy Event

Only one buy event exists. It is described in the following table.

BuyEvent

Description	Occurs when a customer completes the purchase. A `BuyEvent` occurs for each line item. A purchase may consist of one or more line items. A line item may consist of one or more items. For example, although a particular line item may have quantity of four items, only one `BuyEvent` occurs.
Trigger Type	Pipeline component
Class	`com.bea.commerce.ebusiness.tracking.pipeline.PurchaseTrackerPC`

Elements	`event_date`
	`event_type`
	`session_id`
	`user_id`
	`sku`
	`quantity`
	`unit_price`
	`currency`
	`application_name`
	`order_line_id`

Products	Specific to WebLogic Commerce Server, available in Campaign Manager for WebLogic

Rules Event

Only one rule event exists. It is described in the following table.

RuleEvent

Description	Indicates the rules that were fired as a customer navigates a Web site.
Trigger Type	Fired internally from advislets
Class	Not applicable

Elements	`event_date`
	`event_type`
	`session_id`
	`user_id`
	`ruleset_name`
	`rule_name`

Products	Specific to WebLogic Personalization Server, available in Campaign Manager for WebLogic and WebLogic Commerce Server

Campaign Events

These events occur when a customer participates in a campaign.

CampaignUserActivityEvent

Description	Occurs when a customer participates in a campaign. Specifically, this event is fired whenever one or more scenario actions are true and the campaign service is activated. You can limit this event to a single occurrence for a particular scenario. This event is intended for use by analytic software.
Trigger Type	Fired internally from the campaign service
Class	Not applicable

Elements	`event_date`
	`event_type`
	`session_id`
	`user_id`
	`campaign_id`
	`scenario_id`

Products	Campaign Manager for WebLogic only

DisplayCampaignEvent

Description	Occurs when campaign content, such as an ad, is presented to the customer. Specifically, this event is fired whenever a campaign placeholder displays an ad placed in the ad bucket by a campaign. You can use this event to trigger another campaign. Analytic software uses this event to determine if a customer saw an ad as a result of a campaign.
Trigger Type	Fired internally from the campaign service
Class	Not applicable

Elements	`event_date`
	`event_type`
	`session_id`
	`user_id`
	`document_type`
	`document_id`
	`campaign_id`
	`scenario_id`
	`application_name`
	`placeholder_id`

Products	Campaign Manager for WebLogic only

ClickCampaignEvent

Description	Occurs when a campaign item, such as an ad, is clicked on by the customer. Specifically, this event is fired whenever a customer clicks a campaign ad that was placed in the ad bucket by a campaign. You can use this event to trigger another campaign. Analytic software uses this event to determine if a customer clicked on an ad as a result of a campaign.
Trigger Type	Fired internally from campaign service
Class	Not applicable

Elements	`event_date`
	`event_type`
	`session_id`
	`user_id`
	`document_type`
	`document_id`
	`campaign_id`
	`scenario_id`
	`application_name`
	`placeholder_id`

Products	Campaign Manager for WebLogic only

Event Triggers

The standard events supplied by BEA are triggered at important points in an e-commerce site. The components that enable events include Java APIs, JSP tags, JSP scriptlets, Webflow input processors, Pipeline components, the Flow Manager, content selectors, and classification advislets. You can add or customize triggers for each of the following events:

- `DisplayContentEvent`
- `DisplayProductEvent`
- `ClickContentEvent`

- `ClickProductEvent`

Note: `DisplayProductEvent` and `ClickContentEvent` are available in only Campaign Manager for WebLogic and WebLogic Commerce Server.

Each of these events are triggered by JSP tags. You can use the JSP tags that trigger these events to specify which products and what content triggers these events. For example, in the WLCS Web Application, the JSP tag for the `DisplayProductEvent` is located in the `details.jsp`.

The tag shown in Listing 1-1 triggers an event for any product displayed on a catalog detail page. If you wanted to trigger an event for one particular product, you could write a scriptlet that keys off the SKU for that product.

Listing 1-1 JSP Tag

```
<%-- once the product is displayed, fire off a displayProductEvent --%>
<productTracking:displayProductEvent documentId="<%= item.getName() %>"
        documentType="<%= DisplayProductEvent.ITEM_BROWSE %>"
                          sku="<%= item.getKey().getIdentifier() %>" />
```

When you add a JSP tag for an event, you should include a reference to the tag library descriptor, as shown below:

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

Notes: For more information about JSP tags, see Chapter 5, “JSP Tag Library Reference for Events and Behavior Tracking.”

The `details.jsp` is located at:

- `%WL_COMMERCE_HOME%\config\wlcsDomain\wlcsApp\wlcs\commerce\catalog\details.jsp` (Windows)
- `$WL_COMMERCE_HOME/config/wlcsDomain/wlcsApp/wlcs/commerce/catalog/details.jsp` (UNIX)

where `WL_COMMERCE_HOME` is the directory in which you installed Campaign Manager for WebLogic and WebLogic Commerce Server.

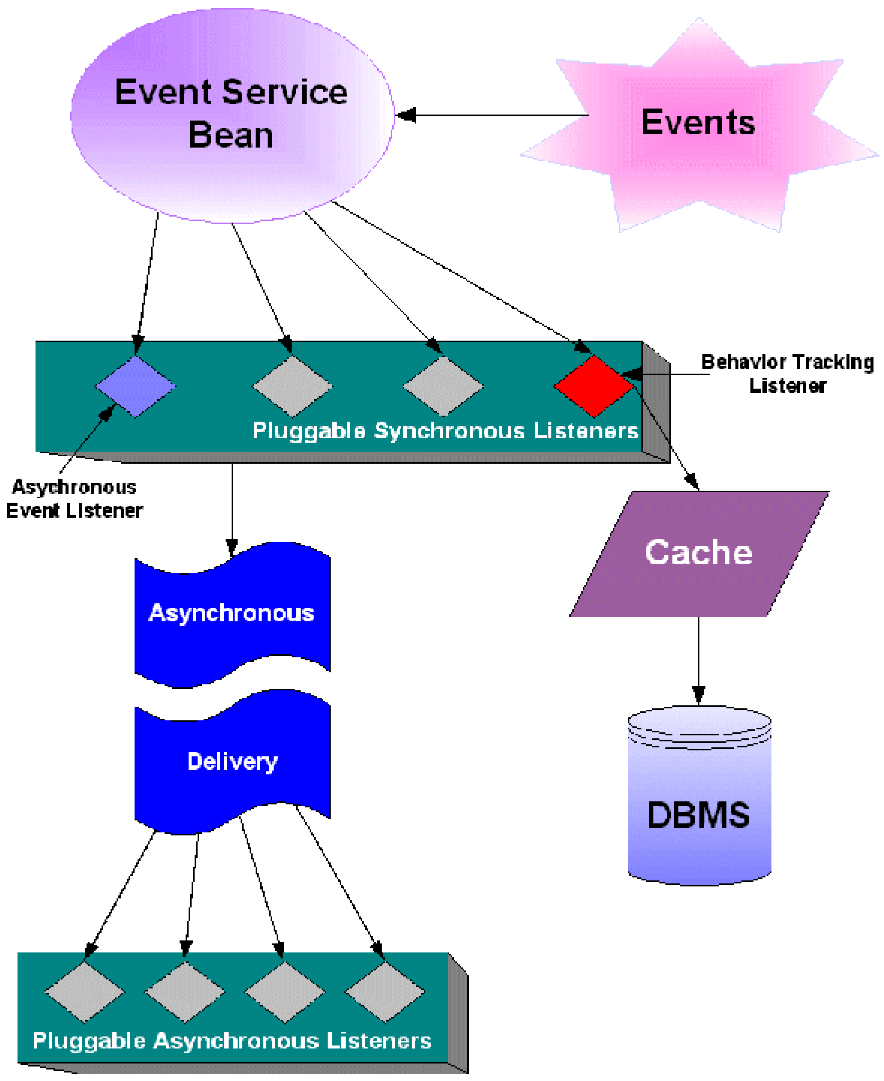
Event Mechanism

The Event service is an extensible, general purpose, event construction and propagation system. As shown in Figure 1-1, an event is generated by a trigger, such as a JSP tag, which creates the event object, locates the Event service bean, and passes the event object to the Event service. The Event service works with plug-in listeners that disseminate events to listeners interested in receiving the events. At creation time, each event listener returns the list of event types that it wants to receive. When the Event service receives an event, it checks the type of the event and sends the event to all listeners that are subscribed to receive that event’s type.

The Event service has two sets of listeners: those that respond to events synchronously and those that respond to events asynchronously. The synchronous listeners use the thread of execution that created and transmitted the event to perform actions in response to that event. The asynchronous listeners receive the event from the thread where it was created and some time later, handle the event in a different thread of execution. The asynchronous service exists so that long-running event handlers can execute without delaying the application from a Web site visitor’s perspective.

Whether a particular plug-in listener is installed on the synchronous or the asynchronous side of the Event service is based on the requirements of the application and is specified in the `weblogiccommerce.properties` file.

Figure 1-1 Event Mechanism


Event Service Bean
Events
Behavior Tracking Listener
Pluggable Synchronous Listeners
Asychronous Event Listener
Cache
Asynchronous
Delivery
DBMS
Pluggable Asynchronous Listeners

Event listeners implement the `com.bea.commerce.platform.events.EventListener` interface. The interface defines signatures for two public methods:

- `public String[] getTypes()`
- `public void handleEvent( Event theEvent )`

The first method returns a list of event types that the listener is interested in receiving from the Event service. For example, if a listener is designed to receive events of type *Foo*, the listener returns *Foo* as an item in the array returned from invoking `getTypes()` on the listener. The second method is invoked when an event is passed to the listener. A listener has no knowledge of whether it is synchronous or asynchronous.

If you wish to create a listener interested in only campaign events, you would list the listener's fully-qualified classname in the `weblogiccommerce.properties` file in either the `eventService.listeners` property or the `asynchronousHandler.listeners` property (for synchronous or asynchronous handling, respectively). The listener would implement the `EventListener` interface and return the following event types:

```
{“ClickCampaignEvent”,“DisplayCampaignEvent”,“CampaignUserActiv
ityEvent” }
```

when its `getTypes()` method is invoked.

After the listener is installed, events of one of these three types arrive through the listener's `handleEvent( Event theEvent )` interface.

The Asynchronous Delivery graphic in Figure 1-1 indicates that the asynchronous event handler receives events transmitted asynchronously from the synchronous side of the Event service. It then dispatches events to the pluggable asynchronous listeners based on the event types each listener is subscribed to receive.

Event Sequence

Figure 1-2 and Figure 1-3 provide a sample of the firing of events. These figures are intended to give you a sense of the order in which events fire, not a comprehensive examination of event sequencing.

Figure 1-2 Event Sequence Sample—Part 1

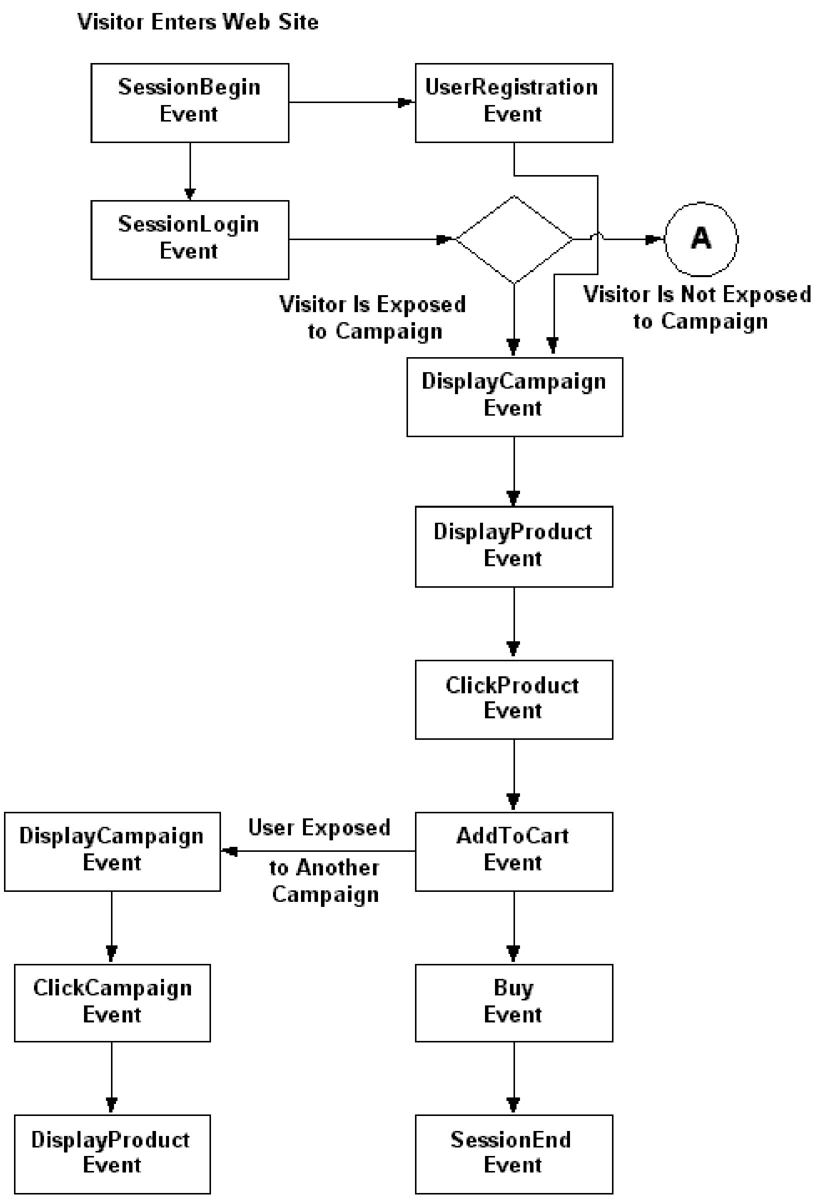
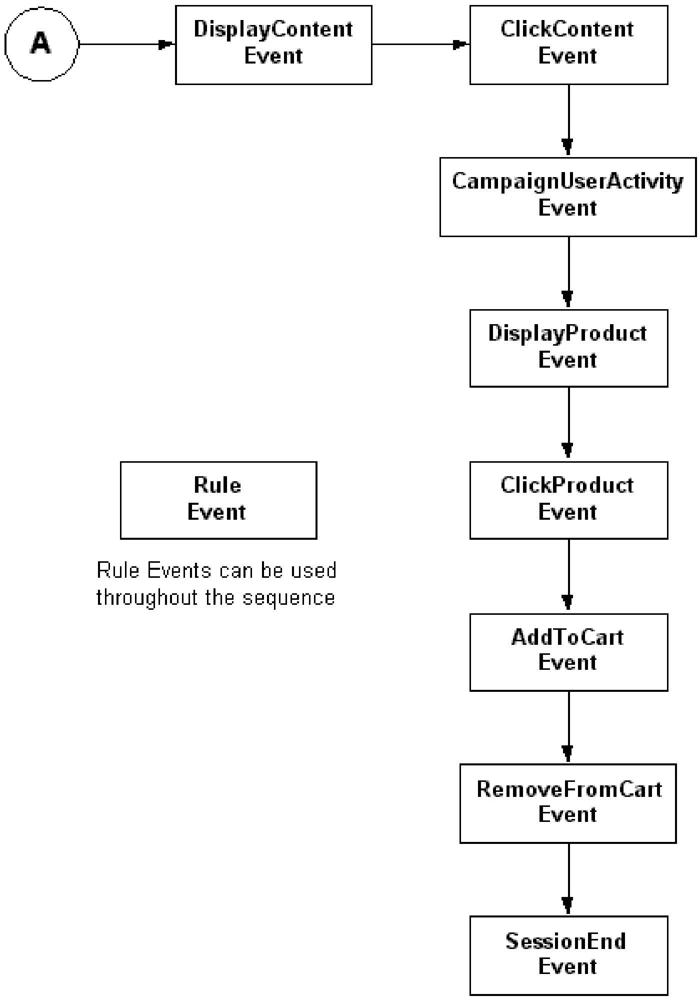


Figure 1-3 Event Sequent Sample—Part 2



2 Creating Custom Events

This topic provides the information necessary to write a custom event. You can create a custom event for anything you wish to track. For example, you could create an event that would tell you which pages are displayed for each customer. You could then use the information to determine how many pages are viewed on average per session and which pages are the most popular. Additionally, marketing professionals could use this event when developing scenario actions that are based on the display of particular pages. To demonstrate how to write a custom event, a simple example is provided. Each section references and expands the example.

This topic includes the following sections:

- Overview of Creating a Custom Event
- Writing a Custom Event Class
- Writing a Custom Event Listener
- Writing a Behavior Tracking Event Class
- Debugging the Event Service

Overview of Creating a Custom Event

The creation of a custom event is a multiple-step process. The following list provides an overview of the process and references the information not covered in this topic:

- Write the code that defines the event and event listener.
- Write the code to trigger the event with a JSP tag or an API call.
- Register the event. For more information, see Chapter 3, “Registering Custom Events.”
- To record the event data to the EVENT table, modify the weblogiccommerce.properties file to include the new event and create an entry for the event in the EVENT_TYPE table. For more information, see Chapter 4, “Persisting Behavioral Tracking Data.”

Writing a Custom Event Class

To create a custom event, you first write an event object. This object encapsulates all the necessary information for correctly interpreting and handling the event when it arrives at a listener. All custom events must subclass the com.bea.commerce.platform.events.Event class. This base class handles setting and retrieving an event’s timestamp and type and provided access to the custom event’s attributes. Two Event class methods set and retrieve attributes:

```
setAttribute( String theKey, Serializable theValue )
getAttribute( String theKey )
```

These methods can be called from the custom event’s constructor to set attributes specific to the new event. Keep in mind that all objects set as values in the Event object must be Java serializable. The getTimeStamp() method returns the date of the event’s creation in milliseconds. The type of an event is accessed using the Event class’s getType() method. The timestamp and type of an Event object instance can be set only at creation time in the Event constructor.

To illustrate the process of creating a custom event, a simple example is presented here, called TestEvent. The example is a basic demonstration of how to create an event subclass. An actual custom event would probably be more elaborate.

A custom event must first have a type. This type should be passed to the superclass constructor (for example, in the Event class); this type is returned at getType() invocations on custom-event object instances. For example:

```
/** Event Type */
public static final String TYPE = "TestEvent";
```

To properly initialize the `Event` base class of the custom event object, the value `TYPE` is passed to the event constructor. The type of all events must be a simple Java string object.

After defining the type, you must define the keys that access the attributes stored in the custom event. These attributes can be given values in the constructor. For example, the `TestEvent` class has two properties, `userPropertyOne` and `userPropertyTwo`; the type of the value associated with `userPropertyOne` is a `String` and `userPropertyTwo` is a `Double`. The keys are defined as follows:

```
/**
 * Event attribute key name for the first user defined property
 * Attribute value is a String
 */
public static final String USER_PROPERTY_ONE_KEY =
    "userPropertyOne";

/**
 * Event attribute key name for the second user defined property
 * Attribute value is a Double
 */
public static final String USER_PROPERTY_TWO_KEY =
      "userPropertyTwo";
```

Finally, a constructor brings the event type and the process of setting attributes together to create an event object. The constructor looks like:

```
/**
 * Create a new TestEvent
 *
 *
 * @param userPropertyOne some user defined property typed as
 * a String
 * @param userPropertyTwo some user defined property typed as
 * a Double
 */
 public TestEvent( String userPropertyOneValue,
                   Double userPropertyTwoValue )
 {
   /* calls the Event class constructor with this event’s type */
   super( TYPE );

   if( userPropertyOneValue != null )
       setAttribute( USER_PROPERTY_ONE_KEY,
                     userPropertyOneValue );
```

```
        if( userPropertyTwoValue != null )
            setAttribute( USER_PROPERTY_TWO_KEY,
                          userPropertyTwoValue );
    }
```

Putting all the parts together, the entire custom event class is shown in Listing 2-1.

Listing 2-1 TestEvent Class

```
/* Start TestEvent class */

public class TestEvent
  extends com.bea.commerce.platform.events.Event
{
  /** Event Type */
  public static final String TYPE = "TestEvent";

  /**
   * Event attribute key name for the first user defined property
   * Attribute value is a String
   */
   public static final String USER_PROPERTY_ONE_KEY = "userPropertyOne";

  /**
   * Event attribute key name for the second user defined property
   * Attribute value is a Double
   */
   public static final String USER_PROPERTY_TWO_KEY = "userPropertyTwo";

  /**
   * Crate a new TestEvent
   *
   *
   * @param userPropertyOne some user defined property typed as a String
   * @param userPropertyTwo some user defined property typed as a Double
   */
   public TestEvent( String userPropertyOneValue,
                     Double userPropertyTwoValue )
   {
     /* calls the Event class constructor with this event’s type */
     super( TYPE );

     if( userPropertyOneValue != null )
         setAttribute( USER_PROPERTY_ONE_KEY, userPropertyOneValue );

     if( userPropertyTwoValue != null )
```

```
         setAttribute( USER_PROPERTY_TWO_KEY, userPropertyTwoValue );
   }
}
/* End TestEvent class */
```

The example in Listing 2-1 shows you how to use the fundamental aspects of the `Event` base class and the event service. An actual custom event constructor would probably be more complex. For example, it might check for default values or disallow null attributes. Additionally, the custom-event object might have more methods or member data.

Writing a Custom Event Listener

In order to listen for an event, you must define an event listener. All event listeners must implement the `com.bea.commerce.platform.events.EventListener` interface and have a no arguments (default) constructor. This interface specifies two methods that are fundamental to transmitting events of a given type to interested listeners:

```
public String[] getTypes()
```

```
public void handleEvent( Event ev )
```

The first method returns the types, in a string array, that the listener is interested in receiving. The event service dispatches events of a given type to listeners that return the event's type in the types array. When the event service has determined that a given listener has registered to receive the type of the current event, an event of that type is dispatched to the listener using the `handleEvent( Event ev )` call.

When writing a custom event listener, both methods must be implemented from the `EventListener` interface. Continuing with the `TestEvent` example, the `TestEventListener` listens for instances of `TestEvent` that are sent through the event service. This can be specified as follows:

```
/** The types this listener is interested in */
private String[] eventTypes = {"TestEvent"};

/**
  The method invoked by the event service to determine the
```

```
    types to propagate to this listener.
   */
  public String[] getTypes()
  {
    return eventTypes;
  }
```

To handle the event, the `handleEvent( Event evt )` method is implemented as follows:

```
  /**
   * Handle events that are sent from the event service
   */
  public void handleEvent( Event ev )
  {
    System.out.println("TestListener::handleEvent " +
                       " -> received an event" +
                       " of type: " + ev.getType() );

    /* Do the work here */

    return;
  }
```

Putting all of these pieces together with a constructor, Listing 2-2 shows a simple event listener that registers to receive `TestEvent` objects.

Listing 2-2 Event Listener

```
import com.bea.commerce.platform.events.EventListener;
import com.bea.commerce.platform.events.Event;

/**
 * TestListener to demonstrate the ease with which listeners can be plugged
 * into the behavior tracking system.
 *
 * This class should be added to the property eventService.listeners
 * in order to receive events.  The fully qualified classname must be added
 * to this property; don’t forget to add the ",\" at the end of the previous
 * line or the properties parser will not find the new classname.
 *
 * The types of events that are heard are listed in the eventTypes
 * String array.  Add and remove strings of that type as necessary.
 *
 * @author Copyright (c) 2001 by BEA Systems, Inc. All Rights Reserved.
 */
public class TestListener
```

```
        implements EventListener
    {

    private String[] eventTypes = {"TestEvent"};

    public TestListener()
    {
    }

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

    public void handleEvent( Event ev )
    {
         System.out.println("TestListener::handleEvent -> received an event" +
                          " of type: " + ev.getType() );

         return;
    }
}
```

As with writing a simple event, writing a simple `EventListener` is also straightforward. Any event listener's internals should be generic; the same `TestEventListener` instance may not handle all `TestEvent` objects. Therefore `TestEventListener` should be entirely stateless and should operate on data that is contained in the event object or stored externally (that is, in a database).

Note: Multiple instances of any listener may execute concurrently.

Installing a Listener Class in the Event Service

After you have created a custom event and listener, you must install them in the `weblogiccommerce.properties` file. Installation of a listener allows the listener to receive events when the server is started. As discussed in "Event Mechanism" on page 1-16, there are two sides to the event service: synchronous and asynchronous. Each side has a list of listeners residing in the `weblogiccommerce.properties` file. The lists are `eventService.listeners` and `asynchronousHandler.listeners`, respectively.

To install a listener, the fully-qualified class name must be entered into the appropriate list of listeners. Additionally, to continue the property to the next line, the listener list for both sides must be comma delimited and have a backslash (,\) at the end of each line. As shown below, installing the `TestEventListener` is done by adding the class name to the existing list:

```
eventService.listeners=\
    <path to the class>.TestEventListener
```

To install the listener on the asynchronous side, add the class name as follows:

```
asynchronousHandler.listeners=\
    <path to the class>.TestEventListener
```

Although not shown, other listeners may be installed at either of these properties.

Note: The server must be restarted after any changes to this list.

Writing a Behavior Tracking Event Class

A Behavior Tracking event is a special type of event that tracks a customer's interactions with an e-commerce site. E-analysis systems use the data gathered from Behavior Tracking events to evaluate customer behavior. The evaluation is primarily used for campaign development and optimizing customer experience on a Web site.

A Behavior Tracking event and its listeners are created in much the same way as the `TestEvent` class and `TestEventListener` examples. A simple example is also presented here. The example tracking event is called `TestTrackingEvent`. All Behavior Tracking events persisted (recorded) to a database for use with BEA Behavior Tracking are handled by the `com.bea.commerce.platform.listeners.BehaviorTrackingListener`. The `BehaviorTrackingListener` extends the `com.bea.commerce.platform.events.EventListener` class.

The `BehaviorTrackingListener` receives and persists Behavior Tracking events from the event service when it is plugged into one of the listener's properties in the `weblogiccommerce.properties` file.

Note: For scalability reasons, you should plug the `BehaviorTrackingListener` into the `eventService.listeners` property.

This listener receives events from the event service and adds them to a cache that is intermittently persisted to the Behavior Tracking tables in the database. The frequency of the sweeping of events from the cache is controlled by the following properties in the `weblogiccommerce.properties` file:

- `behaviorTracking.cache.maxSize`
- `behaviorTracking.cache.checkIntervalSec`
- `behaviorTracking.cache.maxAgeSec`

You should tune these properties to optimize performance. A cache sweep should be performed often enough that writing to the database is not too time consuming but not so frequent that the operation is wasteful.

Facilitating OffLine Processing

For facilitating offline processing of customer interactions with a Web site, Behavior Tracking events are designed to be persisted to a table in the database, called the `EVENT` table. Part of the process of recording data from Behavior Tracking events is creating an XML representation of the data, which is stored in the `xml_definition` column of the `EVENT` table. You can persist events in an alternate location and table structure as requirements dictate. This discussion assumes that you are planning to use the BEA Behavior Tracking event persistence mechanism. Therefore, to persist events in the provided `EVENT` table, your custom event must conform to the descriptions in this section so that it is created and persisted properly.

To formally specify the data comprising a Behavior Tracking event, you need to develop an XML-XSD schema for the new event. While XSDs are not used internally to verify the creation of XML, the XML that is created represents the event's data in the database. If the event class is properly developed and used, it will conform to the XML-XSD schema. With an XSD document, development of the constructor and attribute keys for a Behavior Tracking event follows easily.

To correctly turn a Behavior Tracking event into an XML representation, the Behavior Tracking event must have several pieces of member data that fully describe an XML instance document for the schema associated with the event type. This data describes the namespace and XSD file associated with the event. It also lists the keys and their order for creating an XML instance document from an event object. The structure of an XSD document and details on XML namespaces can be found at http://www.w3.org/XML/Schema. Several XSD schemas for BEA Behavior Tracking events can be found in:

- `%WL_COMMERCE_HOME%\lib\dtd\tracking` (Windows)
- `$WL_COMMERCE_HOME/lib/dtd/tracking` (UNIX)

where `WL_COMMERCE_HOME` is the directory in which you installed BEA Campaign Manager for WebLogic, BEA WebLogic Commerce Server, and/or BEA WebLogic Personalization Server.

The namespace and schema are specified as:

```
/**
  The XML namespace for this event
 */
private static final String XML_NAMESPACE=
    "http://<your URI>/testtracking";

/**
  The XSD file containing the schema for this event
 */
private static final String XSD_FILE="TestTrackingEvent.xsd";
```

Note: These values are used when creating an instance document to populate the fields.

The `schemaKeys` are a list of strings which are keys to the event class's `getAttribute` and `setAttribute` methods. These keys are used to extract the data that populate elements in the XML instance document which represent the Behavior Tracking event. The keys should be listed in an array that consists of string-typed objects. Their order specifies the order in which they appear in the XML instance document. In the XSD files that the Behavior Tracking system generates, the order of the elements is important; an XML file will not validate with an XSD file if elements are out of order. Elements can be omitted by using the XML `numOccurs` keyword and setting the value to zero. For examples of how this is done, see the XSD schemas for BEA Behavior Tracking events in

- `%WL_COMMERCE_HOME%\lib\dtd\tracking` (Windows)
- `$WL_COMMERCE_HOME/lib/dtd/tracking` (UNIX)

where `WL_COMMERCE_HOME` is the directory in which you installed Campaign Manager for WebLogic, WebLogic Commerce Server, and/or WebLogic Personalization Server.

An example array for the Behavior Tracking version of the `TestEvent` described above might appear as:

```
/**
 These are the keys and their order for elements that
 will be present in the XML representing this object.
 */
private static final String localSchemaKeys[] =
{
    SESSION_ID, USER_ID, USER_PROPERTY_ONE_KEY,
        USER_PROPERTY_TWO_KEY
};
```

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` is the WebLogic Server session ID that is created for every session object. (For more information, see the WebLogic Server 6.0 Documentation Center.) The `USER_ID` field (which may be null) is the username of the Web site customer associated with the session from which the event was generated. For some events, a user may not be associated with an event; as previously mentioned, the `numOccurs` for the `USER_ID` field in an XSD file should be zero. To persist events in the EVENT table, the `SESSION_ID` must be non-null.

All Behavior Tracking events must extend the `com.bea.commerce.tracking.events.TrackingEvent` class. This class defines three keys that are useful for setting attributes for all tracking events, as follows:

- `TrackingEvent.SESSION_ID`
- `TrackingEvent.USER_ID`
- `TrackingEvent.REQUEST`.

These keys are used in `setAttribute` calls made in the `TrackingEvent` constructor when setting the `SESSION_ID`, `USER_ID`, and `REQUEST` (an `HttPServletRequest` object), respectively. They should also be used to retrieve values associated with each key when invoking `Event.getAttribute` (`String Key`) on event objects that extend `TrackingEvent`.

TrackingEvent Base Class Constructor

The `TrackingEvent` base class has a constructor that is more complicated than the `Event` class's constructor. The `Event` constructor is invoked by the `super( String eventType )` call in the `TrackingEvent` constructor. The `TrackingEvent` constructors are shown in Listing 2-3 and Listing 2-4.

Listing 2-3 Tracking Event Constructor—Example 1

```
/**
 * Create a new TrackingEvent.
 *
 * @param theEventType the event’s type
 * @param theSessionId from HttpSession.getId()
 * @param theUserId from HttpServletRequest.getRemoteUser() or equivalent
 * (null if unknown)
 * @param theXMLNamespace the namespace for an XML representation of this event
 * type
 * @param theXSDFile the file that contains the schema which specifies and
 * enforces typing on the data in the XML file
 * @param theSchemaKeys the list of keys (in their order in the XSD schema)
 * representing the data to be persisted in this event’s XML
      */
public TrackingEvent( String theEventType,
                      String theSessionId,
                      String theUserId,
                      String theXMLNamespace,
                      String theXSDFile,
                      String[] theSchemaKeys )
```

The `TrackingEvent` constructor shown in Listing 2-4 takes an `HttpServletRequest` object.

Listing 2-4 Tracking Event Constructor—Example 2

```
/**
 * Create a new TrackingEvent.
 *
 * @param theEventType the event’s type
 * @param theSessionId from HttpSession.getId()
 * @param theUserId from HttpServletRequest.getRemoteUser() or equivalent
 * (null if unknown)
 * @param theXMLNamespace the namespace for an XML representation of this event
 * type
 * @param theXSDFile the file that contains the schema which specifies and
 * enforces typing on the data in the XML file
 * @param theSchemaKeys the list of keys (in their order in the XSD schema)
 * representing the data to be persisted in this event’s XML
 * @param theRequest the http servlet request object
 */
public TrackingEvent( String theEventType,
                      String theSessionId,
                      String theUserId,
                      String theXMLNamespace,
                      String theXSDFile,
                      String[] theSchemaKeys,
                      HttpServletRequest theRequest )
```

In the first constructor, shown in Listing 2-3, the only data that is optional (that is, that can be null) is `theUerId`; all other data is required so that the tracking event is correctly persisted to the `EVENT` table. In the second constructor, shown in Listing 2-4, the `HttpServletRequest` object can be passed in from triggering locations where the `HttpServletRequest` object is available. This object provides the data needed to fire rules against event instances.

Note: In order to fire rules on a custom Behavior Tracking event, the `HttpServletRequest` and the `USER_ID` must be non-null. Generally, a non-null `USER_ID` means that a customer is logged into a Web site. Rules cannot be fired on an event with a null-user.

The `TestTrackingEvent` constructor is shown in Listing 2-5.

Listing 2-5 TestTrackingEvent Constructor

```
/**
 * Create a new TestTrackingEvent
 *
 * @param theSessionId from HttpSession.getId()
 * @param theUserId from HttpServletRequest.getRemoteUser() or equivalent
 * (null if unknown)
 * @param userPropertyOne some user defined property typed as a String
 * @param userPropertyTwo another user defined property typed as a Double
 */
```

```
public TestTrackingEvent( String theSessionId,
                          String theUserId,
                          String userPropertyOneValue,
                          Double userPropertyTwoValue )
{
    super( TYPE, theSessionId, theUserId, XML_NAMESPACE, XSD_FILE,
           localSchemaKeys );

    if( userPropertyOneValue != null )
        setAttribute( USER_PROPERTY_ONE_KEY, userPropertyOneValue );

    if( userPropertyTwoValue != null )
        setAttribute( USER_PROPERTY_TWO_KEY, userPropertyTwoValue );

}
```

This constructor calls the `TrackingEvent` constructor to populate the required values and then sets the attributes necessary for this particular Behavior Tracking event type.

The entire `TestTrackingEvent` is shown in Listing 2-6.

Listing 2-6 TestTracking Event

```
import com.bea.commerce.platform.tracking.events.TrackingEvent;

/**
 * Test, user-defined behavior tracking event.
 *
 * This event can be persisted to the database.
 *
*/
public class TestTrackingEvent
    extends TrackingEvent
{

    /** Event type */
    public static final String TYPE = "TestTrackingEvent";

    /**
      The XML namespace for this event
     */
    private static final String XML_NAMESPACE="http://<your URI>/testtracking";

    /**
```

```
       The XSD file containing the schema for this event
      */
     private static final String XSD_FILE="TestTrackingEvent.xsd";

     /**
      * Event attribute key name for the first user defined property
      * Attribute value is a String
      */
     public static final String USER_PROPERTY_ONE_KEY = "userPropertyOne";

     /**
      * Event attribute key name for the second user defined property
      * Attribute value is a Double
      */
     public static final String USER_PROPERTY_TWO_KEY = "userPropertyTwo";

     /**
       These are the keys and their order for elements that
       will be present in the XML representing ths object.
      */
     private static final String localSchemaKeys[] =
     {
         SESSION_ID, USER_ID, USER_PROPERTY_ONE_KEY, USER_PROPERTY_TWO_KEY
     };

     /**
      * Create a new TestTrackingEvent
      *
      * @param theSessionId from HttpSession.getId()
      * @param theUserId from HttpServletRequest.getRemoteUser() or equivalent
      * (null if unknown)
      * @param userPropertyOne some user defined property typed as a String
      * @param userPropertyTwo another user defined property typed as a Double
      */
     public TestTrackingEvent( String theSessionId,
                               String theUserId,
                               String userPropertyOneValue,
                               Double userPropertyTwoValue )
     {
         super( TYPE, theSessionId, theUserId, XML_NAMESPACE, XSD_FILE,
                localSchemaKeys );

         if( userPropertyOneValue != null )
             setAttribute( USER_PROPERTY_ONE_KEY, userPropertyOneValue );

        if( userPropertyTwoValue != null )
            setAttribute( USER_PROPERTY_TWO_KEY, userPropertyTwoValue );
     }
 }
```

The `TestTrackingEvent`, shown in Listing 2-6, correctly sets its own attributes and sets the attributes in its instantiation of `TrackingEvent`. This enables correct population of the XML instance document at the time of its creation. Recall that the XML instance document represents the `TestTrackingEvent` in the database's `EVENT` table.

If you want the custom Behavior Tracking event type to be persisted in the database, the event must be added to the `behaviorTracking.persistToDatabase` property list. (If you are not persisting the event, you do not need to add the event type to the property.) This list is already present in the `weblogiccommerce.properties` file. It is a list of types that are listened for by the `BehaviorTrackingListener` and added to the cache of events that are persisted in the database when listened-for event types are received. Accordingly, if you wanted to add the `TestTrackingEvent` to the list, you would add the type `TestTrackingEvent` to the list and add a comma and a backslash (`,\`) to the end of the item on the previous row.

XML Creation of Behavior Tracking Events

When persisting Behavior Tracking events to the `EVENT` table, the bulk of the data must be converted to XML. The XML document should conform to an XML XSD schema that you create which specifies the order of the XML elements in the XML instance document. Additionally, the schema must include the types of elements and their cardinalities. The process of creating XML from an event object is handled by a helper class that utilizes variables and constants in a Behavior Tracking event's class file. All schema documents use the namespace: “http://www.w3.org/1999/XMLSchema” and all instances of Behavior Tracking schemas use the namespace: “http://www.w3.org/1999/XMLSchema-instance”. The XML created in Listing 2-7 will conform to the XSD schema.

Listing 2-7 XSD Document Example

```
<?xml version="1.0"?>
<schema
targetNamespace="http://<your URI>/testtracking"
xmlns="http://www.w3.org/1999/XMLSchema"
xmlns:bt="http://<your URI>/testtracking"
```

```
>
<element name="TestTrackingEvent">
<complexType>
<sequence>
<element ref="bt:event_date"/>
<element ref="bt:event_type"/>
<element ref="bt:session_id"/>
<!--
minOccurs defaults to 1. Setting it to 0 makes
the element optional.
-->
<element ref="bt:user_id" minOccurs="0" />
<element ref="bt:userPropertyOne"/>
<element ref="bt:userPropertyTwo"/>
</sequence>
</complexType>
</element>
<element name="event_date" type="timeInstant"/>
<element name="event_type" type="string"/>
<element name="session_id" type="string"/>
<element name="user_id" type="string"/>
<element name="userPropertyOne" type="string"/>
<element name="userPropertyTwo" type="double"/>
</schema>
```

Creation of an event's representation in XML takes place generically relative to the event's type. Consequently, to create an accurate XML instance document, each event must specify the namespace, event type, elements, and order of its elements. Using the `TestTrackingEvent` example, the XML representing an instance of the `TestTrackingEvent` is constructed as follows:

Note: Assume that `testTrackingEvent` is a well-formed instance of a `TestTrackingEvent`.

1. Get the event's type with the `testTrackingEvent.getType()` call.
2. Get the event's namespace with the `((TrackingEvent)testTrackingEvent).getXMLNamespace()`call.
3. Get the event's XSD filename with the `((TrackingEvent)testTrackingEvent).getXSDFile()` call.

Using the schema keys from the `TestTrackingEvent` class, values are inserted into the XML document. Schema key/attribute value pairs correspond to XML elements in this way:

<schema Key>value</schema Key>

The helper class that creates XML for Behavior Tracking assumes that the elements inserted into an XML instance document are not deeply nested. Additionally, the `toString()` method is used to create a representation of the value object that is retrieved through the `Event` classes's `getAttribute( String Key )` call. The contents of the string returned by invoking `toString()` on the value object must match the type specified in the event's schema document. The `TestTrackingEvent` retrieves values using the following keys in the order specified in the `schemaKeys` array:

- `SESSION_ID`
- `USER_ID`
- `USER_PROPERTY_ONE_KEY`
- `USER_PROPERTY_TWO_KEY`

The values for these keys are retrieved using the `testTrackingEvent.getAttribute( <schema Key> )` call. The order in which the XML formatted key/value pairs are inserted into the instance document is specified by the constant `schemaKeys` array, which is defined and populated in the `TestTrackingEvent` class.

The steps assembled to create an XML instance document for the `TestTrackingEvent` are presented in Listing 2-8.

Listing 2-8 XML Instance Document Example

```
<TestTrackingEvent
 xmlns="http://<your URI>/testtracking"
 xmlns:xsi="http://www.w3.org/1999/XMLSchema-instance"
 xsi:schemaLocation="http://<your URI>/testtracking
TestTrackingEvent.xsd"
 >
<event_date>XML time instant formatted event date</event_date>
<event_type>TestTrackingEvent</event_type>
 <session_id>theSessionIdValue</session_id>
 <user_id>theUserIdValue</user_id>
 <userPropertyOne>userPropertyOneValue</userPropertyOne>
 <userPropertyTwo>userPropertyTwoValue</userPropertyTwo>
</TestTrackingEvent>
```

The XML creation is performed automatically when events arrive at the `BehaviorTrackingListener`, which enables Behavior Tracking in Campaign Manager for WebLogic, WebLogic Commerce Server, and/or WebLogic Personalization Server. The Behavior Tracking listener is installed by adding it to the `eventService.listeners` property in the `weblogiccommerce.properties` file.

You must be careful when defining the namespaces, XSD documents, and schema keys variables in custom Behavior Tracking event classes, especially if they will be persisted to the `EVENT` table. The method for creating and storing XML presented in this discussion exactly follows the variables and constants specified in the event class. You are free to develop other ways of creating and storing XML; this section is directed only at the process of persisting XML Behavior Tracking representations in the BEA `EVENT` table.

Note: The Event's date is retrieved using the `Event` class's `getTimeStamp()` call, which returns a Java primitive `long` typed value. That `long` must be converted into the type specified for the `event_date` element in the XSD schema document. The type in this case is time instant. Event date and event type the first two elements in all XML instance documents created through the `BehaviorTrackingListener`.

Custom Behavior Tracking Event Listeners

To create a custom Behavior Tracking listener, in addition to or instead of the default `BehaviorTrackingListener`, follow the example presented in "Writing a Custom Event Listener" on page 2-5. Add the new event types to the custom listener's `eventTypes` array (for example, `TestTrackingEvent`). A given listener can listen for any number of event types that may or may not be Behavior Tracking events. The custom Behavior Tracking listener can be installed on either the synchronous or asynchronous side of the event service, whichever is appropriate.

Writing Custom Event Triggers

Once events are created, you must set up a mechanism for triggering events in the application. Events may be generated from pipeline components, input processors, JSP scriptlets, or JSP tags. Some Behavior Tracking events are triggered from within Campaign Manager for WebLogic, WebLogic Commerce Server, or WebLogic Personalization Server software.

After determining the mechanism for triggering events, tracking events can be sent to the event system using the `com.bea.commerce.platform.tracking.TrackingEventHelperImpl` class. This class defines helper methods that pass events to the event service. Listing 2-9 shows an example of passing the `TestTrackingEvent`.

Listing 2-9 Dispatching an Event

```
/*
 * Create the event
 */
Event theEvent = new TestTrackingEvent( "<some session id>",
                                        "<some user id> ",
                                        new String("userPropertyOneValue"),
                                        new Double( 3.14 ) );

/*
 * Dispatch the event
 */
TrackingEventHelperImpl.getInstance().dispatchEvent( theEvent );
```

To dispatch a `TestEvent` to the event service, the event service name can be looked up in the JNDI, and an instance of the `EventService` bean can be obtained by invoking the `create()` method on an `EventServiceHome` instance. The JNDI name of the `EventServiceHome` interface is the classname of the `EventServiceHome` class (`com.bea.commerce.platform.events.EventServiceHome`). Listing 2-10 shows an example.

Listing 2-10 JNDI Example

```
import com.beasys.commerce.axiom.util.helper.JNDIHelper;
import com.bea.commerce.platform.events.Event;
import com.bea.commerce.platform.events.EventServiceHome;
import com.bea.commerce.platform.events.EventService;

import javax.ejb.CreateException;
import javax.rmi.PortableRemoteObject;

/* code here */

    public void demonstrateEventDispatch()
    {
        Event event = <some event instance>;

        try
        {
            EventServiceHome home = (EventServiceHome)
            PortableRemoteObject.narrow( JNDIHelper.lookup
                ( "com.bea.commerce.platform.events.EventServiceHome" ),
                EventServiceHome.class );

            EventService eventService = home.create();

            eventService.dispatchEvent( event );
        }
        catch( Exception e )
        {
           /*
            Do exception handling here
           */
        }
    }
/* more code here */
```

Debugging the Event Service

The `DebugEventListener` listener logs events to the `weblogic.log` file. The `DebugEventListener`, as with all event listeners, can be plugged in on the synchronous or asynchronous sides of the event service and can be used to make sure that custom events are firing correctly.

Note: The `weblogic.log` file gets extremely large very quickly. Therefore, this listener should not be included in a production environment.

To use the listener, add the class `com.bea.commerce.platform.events.listeners.DebugEventListener` to either the `eventService.listeners` or the `asynchronousHandler.listeners` properties in the `weblogiccommerce.properties` file.

3 Registering Custom Events

This topic contains basic information about registering custom events. This information includes background information about custom events, how to register events using the Events Editor in the BEA E-Business Control Center, and what you need to do when you make changes to custom events.

This topic contains the following sections:

- Overview of Creating a Custom Event
- Why Register an Event?
- Registering a Custom Event
- Updating a Registered Custom Event

Note: You cannot change any of the standard events supplied with BEA Campaign Manager for WebLogic, BEA WebLogic Commerce Server, or BEA WebLogic Personalization Server.

Overview of Creating a Custom Event

The creation of a custom event is a multiple-step process. The following list provides an overview of the process:

- Create the code that defines the event and event listener.

- Create the code to trigger the event with a JSP tag or an API call.
- Register the event using the instructions in this topic.
- To record the event data for Behavior Tracking analysis, modify the `weblogiccommerce.properties` file to include the new event and create an entry for the event in the `EVENT_TYPE` table.

Note: For information about defining an event and defining a trigger, see Chapter 2, "Creating Custom Events." For information about modifying the `weblogiccommerce.properties` file, and creating an entry in the `EVENT_TYPE` table, see Chapter 4, "Persisting Behavioral Tracking Data."

Why Register an Event?

When you create a custom event, you must register the event. Registering a custom event lets the E-Business Control Center know that the custom event exists. Registering permits campaign developers using the E-Business Control Center to create scenario actions that refer to the event. Registering also identifies the event's properties.

Whenever you change the event code, you must update the event registration. Conversely, whenever you change the event registration, you must also update the event code. A possible ramification of event modification is that the scenario actions that refer to the event's properties may need to be modified.

Registering a Custom Event

The Event Editor in the E-Business Control Center allows you to easily register a custom event. For the purpose of registering an event, you can consider an event property as a name-value pair. During the registration of a custom event, you specify the event's name, description, and one or more properties. Each property has a range, type of permissible value, and default value. The information you need to register for an event should be available from your Commerce Business Engineer (CBE) or Java developer.

The properties for a custom event includes the following information:

- **Data type**: Specifies the data type for your property. The possible values are Text, Numeric, Floating Point Number, Boolean, and Date/Time.
- **Selection mode**: Specifies whether an property has a single default value or a collection of default values.
- **Value range**: Specifies whether the defaults are restricted to one specific value, one or more specific values, or any value.

Note: When you set property values, you are not guaranteed that the property will adhere to these restrictions at run time. Events are not checked by the `SchemaManager` for adherence to a property schema. Therefore, you need to keep the event type definition and the event registration synchronized.

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

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

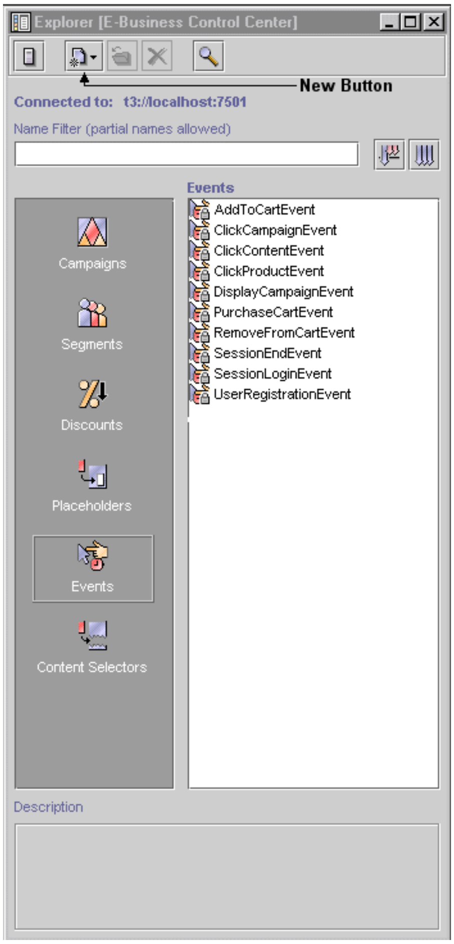
Instructions for Registering a Custom Event

To register a custom event, complete the following steps:

1. Start the E-Business Control Center and connect it to a server. The Explorer window opens as shown in Figure 3-1.

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

Figure 3-1   E-Business Control Center Window



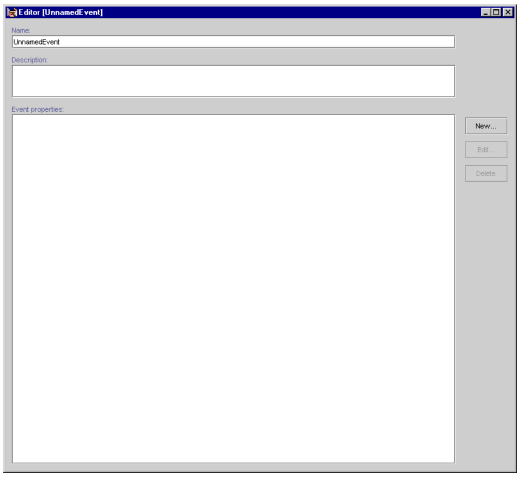
2. Open the Event Editor as follows:

   Note: You cannot edit the standard events.

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

   b. Click the New Button, and then select Event. The Event Editor window appears as shown in Figure 3-2.

Figure 3-2 Event Editor Window



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

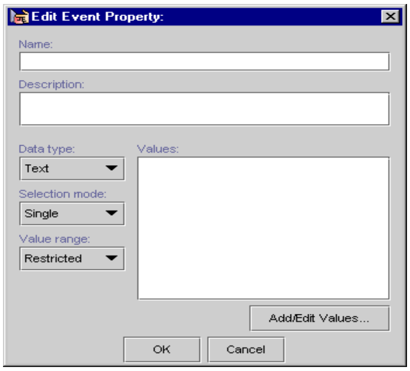
   a. In the Name field, enter a unique name for the event no longer than 100 characters (required).

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

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

   d. To create properties for the event, click the New button. The Edit Event Property window opens, as shown in Figure 3-3.

Figure 3-3 Edit Event Property Window



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

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

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

   c. In the Data type list, select the data type.

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

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

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

   f. Click the Add/Edit Values button.

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

Entering Property Values and Setting the Default Value

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

- Entering Properties with Boolean or a Single Value and Single Default.
- Entering Properties with Multiple Values and Single, Multiple, or All Defaults

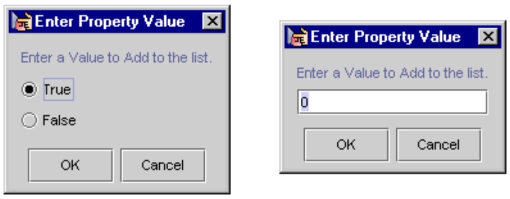
- Entering Properties with Date and Time Values

Entering Properties with Boolean or a Single Value and Single Default

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

1. In the applicable Enter Property Value window (Figure 3-4), perform one of the following:
   - For a Boolean property, select either True or False.
   - For a Single Value, Single Default property, enter a value.

Figure 3-4 Enter Property Values Window—Boolean or Single Value, Single Default



Enter a Value to Add to the list.

Enter a Value to Add to the list.

0

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

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

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

1. In the applicable Enter Property Values window (Figure 3-5, Figure 3-6 or Figure 3-7), enter a value, and then click the Add button.

Figure 3-5 Enter Property Values—Multiple Values, Single Default

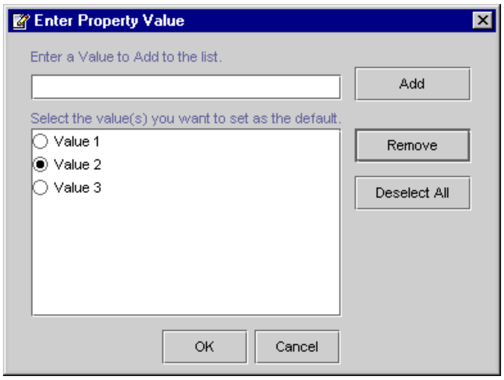

Enter Property Value
Enter a Value to Add to the list.
Add
Select the value(s) you want to set as the default.
Value 1
Value 2
Value 3
Remove
Deselect All
OK
Cancel

Figure 3-6 Enter Property Values—Multiple Values, Multiple Restricted Defaults

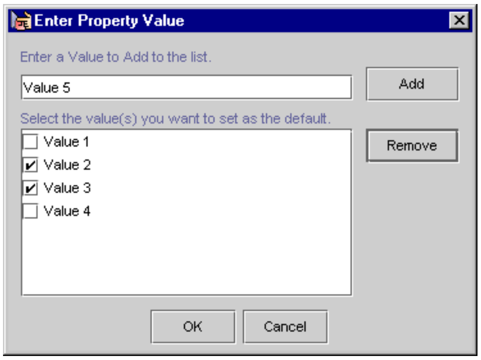
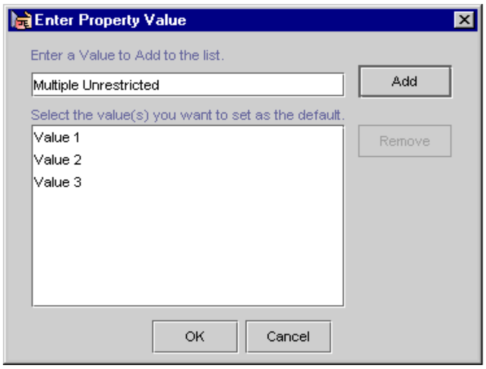

Enter Property Value
Enter a Value to Add to the list.
Value 5
Add
Select the value(s) you want to set as the default.
Value 1
Value 2
Value 3
Value 4
Remove
OK
Cancel

Figure 3-7 Enter Property Values—Multiple Values, Multiple Unrestricted Defaults



Enter a Value to Add to the list.

Multiple Unrestricted

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

Value 1
Value 2
Value 3

2. Repeat the previous step until you have entered all values.

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

   - If you do not want to select a default, go to next numbered step.
   - For multiple values with a single default, select the value (radio button) that you want to set as the default, and then click the OK button.

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

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

   Note: For multiple values without restrictions (that is, the Value range is Unrestricted), you do not need to select any defaults.

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

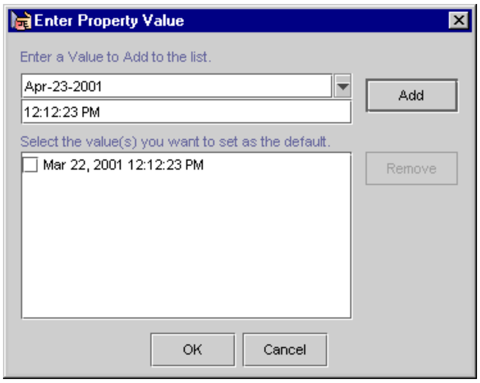
Entering Properties with Date and Time Values

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

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

1. In the Enter Property Values window shown in Figure 3-8, click the drop-down arrow in the Date list. A calendar appears.

Figure 3-8 Enter Date/Time Values


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

Apr-23-2001

12:12:23 PM

Mar 22, 2001 12:12:23 PM

2. Select a date from the calendar.

3. In the Time field, enter a time.

4. Click the Add button.

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

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

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

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

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

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

Updating a Registered Custom Event

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

To update a custom event, complete the following steps.

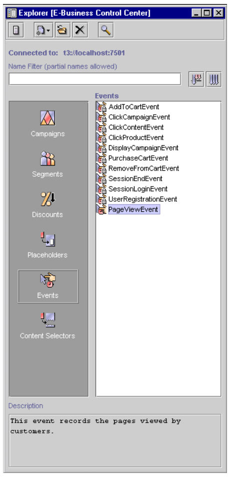
1. Start the E-Business Control Center and connect it to a Web server. The Explorer window opens.

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

2. In the Explorer window, select the Event icon. A list of events appears in the Events field as shown in Figure 3-9.

   Note: You cannot edit standard events.

Figure 3-9 Explorer Window



Connected to: t3://localhost:7501

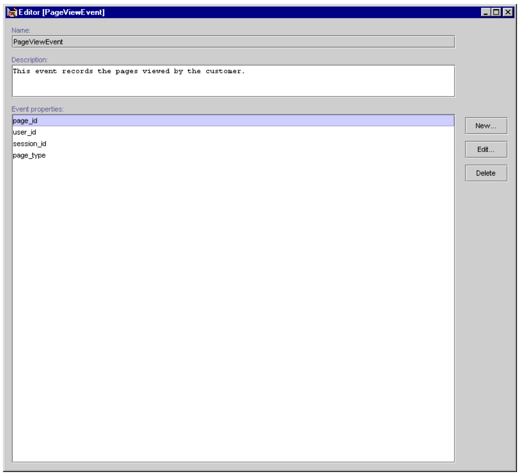
Events:
- AddToCartEvent
- ClickCampaignEvent
- ClickContentEvent
- ClickProductEvent
- DisplayCampaignEvent
- PurchaseCartEvent
- RemoveFromCartEvent
- SessionEndEvent
- SessionLoginEvent
- UserRegistrationEvent
- PageViewEvent

Description:

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

3. Double-click the custom event that you wish to edit. The Event Editor window opens as shown in Figure 3-10. The Event properties field displays a list of existing properties.

Figure 3-10 Event Editor Window



Name:
PageViewEvent

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

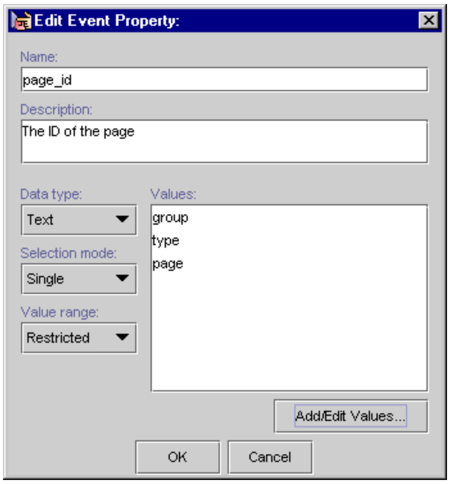
Event properties:
page_id
user_id
session_id
page_type

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

   Note: For more information about setting custom event properties, see “Entering Property Values and Setting the Default Value” on page 3-7.

5. Click the Edit button. The Edit Event Editor window opens as shown in Figure 3-11.

Figure 3-11 Edit Event Property Window



Name:
page_id

Description:
The ID of the page

Data type:
Text

Selection mode:
Single

Value range:
Restricted

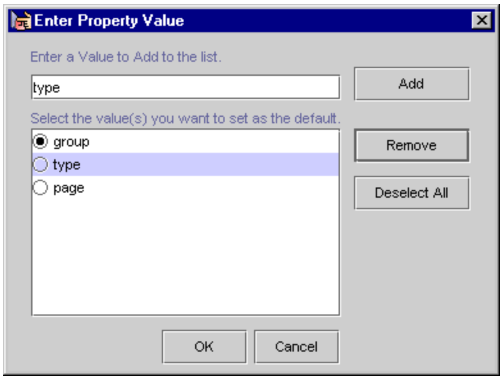
Values:
group
type
page

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

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

7. To add or change values, click the Add/Edit values button. The Enter Property Value window opens as shown in Figure 3-12.

Figure 3-12 Enter Property Value Window


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

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

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

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

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

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

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

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

4 Persisting Behavioral Tracking Data

To record how online customers are interacting with your e-commerce site, you can record event information to a database. These kinds of events are called Behavior Tracking events. E-analytics and e-marketing systems can then analyze these events offline to evaluate customer behavior and transactional data. You can use the knowledge gained from analysis to create and optimize personalization rules, set up product offers, and develop interactive marketing campaigns. This section describes the requirements and database schema needed to log event data for analytical use.

This topic includes the following sections:

- Activating Behavior Tracking
- Data Storage
- Constraints and Indexes
- Scripts

Activating Behavior Tracking

Before Behavior Tracking events can be recorded to a database, you must enable the Behavior Tracking listener. This is accomplished by adding a class to the `weblogiccommerce.properties` file.

The `weblogiccommerce.properties` file contains the `eventService.listeners` property, which is a list of listeners that hear events transmitted through the event service. Each listener contains a list of one or more event types that the listener can receive from the event service. To enable Behavior Tracking, add the following class to the list:

```
com.bea.commerce.platform.tracking.listeners.BehaviorTrackingLi
stener
```

Note: You must configure your database before activating Behavior Tracking. For information on how to do this, see “Production Environment Scenario” on page 4-12.

Event Properties in the weblogiccommerce.properties File

This section describes Behavior Tracking properties more fully and details the mechanism that persists Behavior Tracking event data to the database.

As previously mentioned, Behavior Tracking events are persisted to a database and then analyzed offline. The `behaviorTracking.persistToDatabase` property lists the events that are persisted to the database. The types in this list must match the type specified in the event; for example, the `SessionBeginEvent` has as its type the string “`SessionBeginEvent`”.

Behavior tracking events are stored in a cache. The cache is intermittently swept into the database. The frequency of the sweeping of events from the cache is controlled by the following properties:

- `behaviorTracking.cache.maxCount`
- `behaviorTracking.cache.checkIntervalSec`
- `behaviorTracking.cache.maxAgeSec`

The sweeping is done as follows: Using the value of the `checkIntervalSec` property, a check is made to see if the size of the cache is greater than the value of the `behaviorTracking.cache.maxCount` property. If the size of the cache is greater than the value of the property, all Behavior Tracking events present in the cache are swept into the database in a single transaction. If the size criteria is not met after the `checkIntervalSec` interval has passed, the check is made again in `checkIntervalSec` seconds. Once the total amount of time since the last cache sweep is greater than `maxAgeSec` value, a cache sweep is performed regardless of the number of events in the cache.

The `behaviorTracking.database.connectionPool` property is the pool of database connections used when Behavior Tracking events in the cache are swept into the database. You can use a different connection pool. To do this, an additional pool must be set up in the WebLogic Server console, and its name substituted for the `behaviorTracking.database.connectionPool` property. For more information about creating a connection pool, see the WebLogic Server 6.0 Documentation Center.

Data Storage

This section provides an overview of relational databases and the database schemas and tables that are required for recording Behavior Tracking events.

Relational Databases

Relational databases have both logical and physical structures. Logically you may define one or more databases. Each database may contain one or more tables and indexes, and each table may have multiple columns and rows. The logical structure of databases is quite similar between vendors. However, the physical structure of a database is very vendor-specific. Essentially, the physical structure defines areas on disk drives where the data is stored. Each database environment uses its own terminology and implementation for storing data at the operating system level. For example, Oracle uses the term *tablespace* and the Microsoft SQL Server uses the term *filegroup*.

When a database structure is defined by a database administrator, attention must be paid to the location of specific tables. Some tables are static in that they do not change much; some tables are dynamic in that many rows are being added and deleted; and some tables are read frequently and some rarely. Depending on their behavior, tables should be placed on different physical locations. Some of the most highly-used tables in Campaign Manager for WebLogic, WebLogic Commerce Server, and WebLogic Personalization Server are used for Behavior Tracking. The activity of a single customer moving around your site may generate multiple table entries. Therefore, it is recommended that you place these tables on the fastest drives in the computer. Experienced database administrators are aware of many techniques for monitoring and configuring a database installation for optimal performance. If you do not have a database administrator working with your installation and you have a lot of activity on your site, you should bring in a well-qualified database administer for regular maintenance of your system.

Database Directory Paths

The default database directory paths are:

- `%WL_COMMERCE_HOME%\db\<db vendor>\<db version>\...` (Windows)
- `$WL_COMMERCE_HOME/db/<db vendor>/<db version>/...` (UNIX)

where `WL_COMMERCE_HOME` is the directory in which you installed Campaign Manager for WebLogic, WebLogic Commerce Server, and/or WebLogic Personalization Server.

For example, if you are using Oracle 8.16 on UNIX, the location would be `$WL_COMMERCE_HOME/db/oracle/8.16/....`

BEA provides scripts to help set up the database schema needed for recording Behavior Tracking events, as well as the schema needed for recording data associated with Campaign Manager for WebLogic, WebLogic Commerce Server, and WebLogic Personalization Server. This data includes information from orders, catalogs, products, portals, and portlets.

For Oracle databases, the tablespaces created for Campaign Manager for WebLogic, WebLogic Commerce Server, and WebLogic Personalization Server data are the `WLCS_DATA` and `WLCS_INDEX`.

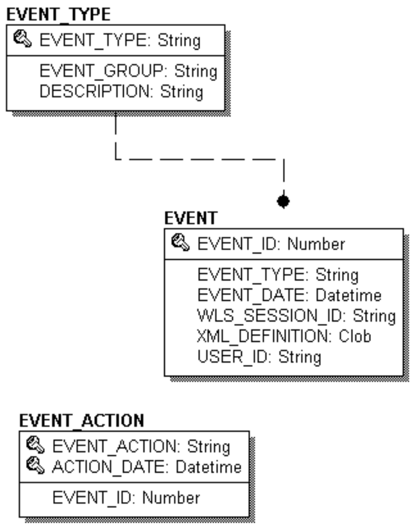
Note: `WLCS_DATA` and `WLCS_INDEX` are tablespace names created by BEA scripts. If you use a particular naming convention, you can rename them.

Behavior tracking uses a tablespace called `WLCS_EVENT_DATA`. This tablespace stores all Behavior Tracking tables, indexes, and constraints. Because of the potential for high volumes of data, this tablespace should be monitored closely.

Behavior Tracking Database Schema

Three tables are provided for the Behavior Tracking data. The `EVENT` table stores all event data. The `EVENT_ACTION` table logs actions used by third-party vendors against the recorded event data, and the `EVENT_TYPE` table references event types and categories in the `EVENT` table. Figure 4-1 shows a logical entity-relation diagram for the Behavior Tracking Database.

Figure 4-1 Entity-Relation Diagram for the Behavior Tracking Database



The EVENT Database Table

Table 4-1 describes the metadata for the EVENT table. This table stores all Behavior Tracking event data. It is an extremely active table.

Table 4-1 The EVENT Table Metadata

Column Name	Data Type	Description and Recommendations
EVENT_ID	NUMBER	A unique, system-generated number used as the record ID. This field is the table's primary key.
EVENT_TYPE	VARCHAR(30)	A string identifier that shows which event was fired.
EVENT_DATE	DATE	The date and time of the event.
WLS_SESSION_ID	NUMBER	A unique, WebLogic Server-generated number assigned to the session.
XML_DEFINITION	CLOB	An XML document that contains pertinent event information. It is stored as a CLOB (Character Large Object).
USER_ID	VARCHAR(50)	The user ID associated with the session and event. If the user has not logged in this column will be null.

As shown in Table 4-1, the EVENT table has six columns; each column corresponds to a specific event element. Five of the EVENT table's columns contain data common to every event type. The XML_DEFINITION column contains all information from these five columns plus event data that is unique to each event type. An XML document is created specifically for each event type. The data elements corresponding to each event type are captured in the XML_DEFINITION column of the EVENT table. These elements are listed in Table 4-2.

Table 4-2 XML_DEFINITION Data Elements

Event	Data Element

AddToCartEvent	`event_date`
	`event_type`
	`session_id`
	`user_id`
	`sku`
	`quantity`
	`unit_list_price`
	`currency`

BuyEvent	`event_date`
	`event_type`
	`session_id`
	`user_id`
	`sku`
	`quantity`
	`unit_price`
	`currency`
	`application_name`

CampaignUserActivityEvent	`event_date`
	`event_type`
	`session_id`
	`user_id`
	`campaign_id`

ClickCampaignEvent	`event_date`
	`event_type`
	`session_id`
	`user_id`
	`document_type`
	`document_id`
	`campaign_id`
	`scenario_id`
	`application_name`

ClickContentEvent	`event_date`
	`event_type`
	`session_id`
	`user_id`
	`document_type`

Table 4-2 XML_DEFINITION Data Elements (Continued)

Event	Data Element

ClickProductEvent	`event_date`
	`event_type`
	`session_id`
	`user_id`
	`document_type`
	`document_id`
	`sku`
	`category_id`
	`application_name`

DisplayCampaignEvent	`event_date`
	`event_type`
	`session_id`
	`user_id`
	`document_type`
	`document_id`
	`campaign_id`
	`scenario_id`
	`application_name`

DisplayContentEvent	`event_date`
	`event_type`
	`session_id`
	`user_id`
	`document_type`
	`document_id`

DisplayProductEvent	`event_date`
	`event_type`
	`session_id`
	`user_id`
	`document_type`
	`document_id`
	`sku`
	`category_id`
	`application_name`

Table 4-2 XML_DEFINITION Data Elements (Continued)

Event	Data Element

PurchaseCartEvent	`session_id`
	`user_id`
	`event_date`
	`event_type`
	`total_price`
	`order_id`
	`currency`
	`application_name`

RemoveFromCartEvent	`event_date`
	`event_type`
	`session_id`
	`user_id`
	`sku`
	`quantity`
	`unit_price`
	`currency`
	`application_name`

RuleEvent	`event_date`
	`event_type`
	`session_id`
	`user_id`
	`ruleset_name`
	`rule_name`

SessionBeginEvent	`event_date`
	`event_type`
	`session_id`
	`user_id`

SessionEndEvent	`event_date`
	`event_type`
	`session_id`
	`user_id`

SessionLoginEvent	`event_date`
	`event_type`
	`session_id`
	`user_id`

Table 4-2 XML_DEFINITION Data Elements (Continued)

Event	Data Element

UserRegistrationEvent	`event_date`
	`event_type`
	`session_id`
	`user_id`

The EVENT_ACTION Database Table

Table 4-3 describes the metadata for the `EVENT_ACTION` table. This table logs actions used by third-party vendors against the recorded event data. It is a fairly static. It has two primary keys.

Table 4-3 EVENT_ACTION Table Metadata

Column Name	Data Type	Description and Recommendations
`EVENT_ACTION`	`VARCHAR(30)`	The event action taken such as `BEGIN EXPORT` or `END EXPORT`. This field is one of the table's primary keys.
`EVENT_DATE`	`DATE`	The date and time of the event. This field is one of the table's primary keys.
`EVENT_ID`	`NUMBER`	The ID of the event that corresponds with the event action taken.

The EVENT_TYPE Database Table

Table 4-4 describes the metadata for the `EVENT_Type` table. This table references event types and categories in the `EVENT` table. This table is static.

Table 4-4 EVENT_TYPE Table Metadata

Column Name	Data Type	Description and Recommendations
`EVENT_TYPE`	`VARCHAR(30)`	A unique, system-generated number used as the record ID. This field is the table's primary key.
`EVENT_GROUP`	`VARCHAR(10)`	The event category group associated with the event type.
`DESCRIPTION`	`VARCHAR(50)`	A description of the `EVENT_TYPE`.

Note: To record custom events, you must create an entry in this table. If a custom event does not have a record in this table, you cannot persist it to the `EVENT` table.

Constraints and Indexes

There is a single foreign key constraint between the `EVENT_TYPE` columns in the `EVENT` and `EVENT_TYPE` tables. As previously mentioned, if a custom event does not have a record in the `EVENT_TYPE` table, it cannot be persisted to the `EVENT` table.

Other than Primary Keys on each of the tables, there are only two indexes on the `EVENT` table. One index is on the `EVENT.EVENT_DATE` column and the other index is comprised of the `EVENT.EVENT_TYPE and EVENT.EVENT_DATE` columns.

Scripts

BEA provides scripts to create the Behavior Tracking database schema and tables for Oracle databases. This section provides information about the structures used in both a development and a production environment.

Development Environment Scenario

In a development environment, you may not want or need separate databases or tablespaces for recording Behavior Tracking events from the databases or tablespaces used for Campaign Manager for WebLogic, WebLogic Commerce Server, and WebLogic Personalization Server. Accordingly, you can include the Behavior Tracking database objects along side the database objects of these products. The easiest way to accomplish this is to execute the `create_all` script found in the `event` directory of your database installation.

Log into Oracle using SQL*Plus and execute the `create_all.sql` script in this location:

```
%WL_COMMERCE_HOME%/db/oracle/8.1.6/event/create_all.sql
```

where `WL_COMMERCE_HOME` is the directory in which you installed Campaign Manager for WebLogic, WebLogic Commerce Server, and/or WebLogic Personalization Server.

The `create_all` scripts in the `event` subdirectory executes the following scripts:

- **`drop_event.sql`**: Drops all the Behavior Tracking database objects.
- **`create_event.sql`**: Creates all the Behavior Tracking database objects.
- **`insert_event_type.sql`**: Populates the `EVENT_TYPE` table with base data.

Production Environment Scenario

This scenario is intended for use in an Oracle production environment where multiple tablespaces and their corresponding elements, such as tables and indexes, can reside in separate tablespaces and potentially on a different database server than Campaign Manager for WebLogic, WebLogic Commerce Server, or WebLogic Personalization Server database objects.

Before enabling the Behavior Tracking events, complete the following steps:

1. Identify the server and database used for recording Behavior Tracking events.

2. In the `WL_COMMERCE_HOME/db/oracle/8.1.6/event` directory where `WL_COMMERCE_HOME` is the directory in which you installed the Campaign Manager for WebLogic, WebLogic Commerce Server, and/or WebLogic Personalization Server:

   a. Edit the `create_event_tablespaces.sql` script to properly define the tablespace path and data filenames.

   b. Execute the `create_event_tablespaces.sql` to create the tablespaces.

   c. Edit the `create_event_users.sql` to ensure the correct user account will be created when this script is executed (the account name by default is `WLCS_EVENT`).

   d. Execute the `create_event_users.sql`.

3. Using SQL*Plus, connect as the user defined in `create_event_users.sql` and execute the script `create_all.sql`. This script will call `drop_event.sql`, `create_event.sql`, and `insert_event_type.sql`.

4. Change your JDBC connection pool information to point to this host, database instance, and user account. For more information, see "Event Properties in the weblogiccommerce.properties File" on page 4-2.

Description of Each Script

The Oracle scripts are described in the following list:

- `WL_COMMERCE_HOME/db/oracle/8.1.6/event/create_all.sql`

  Executes the following scripts: `drop_event.sql`, `create_event.sql`, and `insert_event_type.sql`.

- `WL_COMMERCE_HOME/db/oracle/8.1.6/event/create_event.sql`

  Creates the tables, indexes, and constraints associated with Behavior Tracking events.

- `WL_COMMERCE_HOME/db/oracle/8.1.6/event/create_event_tablespaces.sql`

  Creates tablespaces for storage of Behavior Tracking events information.

- `WL_COMMERCE_HOME/db/oracle/8.1.6/event/create_event_users.sql`

Creates the `WLCS_EVENT` database user and grants the appropriate privileges for working with the Behavior Tracking event tables.

- `WL_COMMERCE_HOME/db/oracle/8.1.6/event/drop_event.sql`

  Drops the Behavior Tracking event tables.

- `WL_COMMERCE_HOME/db/oracle/8.1.6/event/insert_event_type.sql`

  Populates the `EVENT_TYPE` table with base data.

CHAPTER

5 JSP Tag Library Reference for Events and Behavior Tracking

This tag library contains several tag extensions used in the BEA WebLogic Personalization Server. Tags in this library are specifically used in the Events and Behavior Tracking component of the server.

The Events and Behavior Tracking tags allow you specify user behavior that you are interested in monitoring as users navigate across your site pages. These tags cause events to be generated which may be subsequently analyzed by third-party analytical tools, or which may be processed immediately in support of a campaign scenario.

The Events and Behavior Tracking tags are divided into three general areas: content tracking, product tracking, and campaign tracking. Content and product tracking tags can be used in any personalization or commerce application. The campaign tag generates events that feed into active campaign scenarios.

Note: To use the campaign features, you must have the BEA Campaign Manager for WebLogic installed on your system.

This topic includes the following sections:

- Content
  <tr:clickContentEvent>
  <tr:displayContentEvent>
- Product
  <trp:clickProductEvent>
  <trp:displayProductEvent>
- Campaign
  <trc:clickCampaignEvent>

Use the following code to import the content events tag library:
```
<%@ taglib uri="tracking.tld" prefix="tr" %>
```

Use the following code to import the product events tag library:
```
<%@ taglib uri="productTracking.tld" prefix="trp" %>
```

Use the following code to import the campaign events tag library:
```
<%@ taglib uri="campaignTracking.tld" prefix="trc" %>
```

Note: The `<tr:>` prefix means “track.”
The `<trp:>` prefix means “track-product.”
The `<trc:>` prefix means “track-campaign.”

<tr:clickContentEvent>

The `<tr:clickContentEvent>` tag (Table 5-1) is used to generate a behavior event when a user has clicked (through) on an ad impression. This tag will return a URL query string containing event parameters. It is then used when forming the complete URL that hyperlinks the content.

Use the following code to import the content events tag library:

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

Table 5-1 <tr:clickContentEvent>

Tag Attribute	Req'd	Type	Description	R/C
`documentId`	No	String	ID of the item that is displayed, if applicable (that is, an image URL or banner ad ID).	R
`documentType`	No	String	Type or category of the item that is displayed (if applicable).	R
`id`	No	String	Page variable which will hold the output of this tag.	C
`redirectURL`	Yes	String	Where the server should redirect the client to after the document is clicked on and after the server processes the clickthrough event.	R
`userId`	No	String	Name of the user that content was retrieved for. If the optional value is not provided, it will be set to the value of the `request.getRemoteUser()`.	R

Example

The example below demonstrates a clickthrough example going to the flow manager servlet. This link will cause a clickthrough content event to be generated and also display the indicated content. If you wish to redirect the client to an external site after recording the clickthrough event, specify a `redirectURL` attribute and target the `clickThroughServlet` instead of the flow manager servlet.

```
<%@ taglib uri="tracking.tld" prefix="tr" %>
.
.
.
<%-- Note: example code is from the news_index.jsp servlet --%>
<es:forEachInArray id="nextRow" array="<%=headlines%>"
  type="com.beasys.commerce.axiom.content.Content">
  <es:notNull item="<%=nextRow%>">
  <tr:clickContentEvent
     id="url"
     documentId="<%=nextRow.getIdentifier()%>"
     documentType="<%=headingProp%>"
     userId="<%=request.getRemoteUser()%>"
     redirectURL="http://netscape.com" />
  <a href="<%=response.encodeURL(createURL(request,
getHomePage(request), (url + "&contentselected=" +
java.net.URLEncoder.encode
(nextRow.getIdentifier()))))%>">
<cm:printProperty id="nextRow" name="title" encode="html" /></a>
    <%--
clickthrough example going to clickthrough servlet. This link will
cause a clickthrough content event to be generated and then redi-
rect the client.
    --%>
    <LI>
    <a href="<%=response.encodeURL(request.getContextPath()
    + getClickThruPage() + "?" + url)%>">
    <cm:printProperty id="nextRow" name="title" encode="html" />
    </a>
</es:notNull>
</es:forEachInArray>
```

<tr:displayContentEvent>

The `<tr:displayContentEvent>` tag (Table 5-2) is used to generate a behavior event when a user has received (viewed) an ad impression, (typically a gif image).

Use the following code to import the content events tag library:
`<%@ taglib uri="tracking.tld" prefix="tr" %>`

Table 5-2 <tr:displayContentEvent>

Tag Attribute	Req'd	Type	Description	R/C
`documentId`	No	String	ID of the item that is displayed, if applicable (that is, an image URL or banner ad ID).	R
`documentType`	No	String	Type or category of the item that is displayed (if applicable).	R

Example

The example below shows a code snippet of processing that would follow a `<cm:select>` call. For each document returned but not displayed in this example, the `<tr:displayContentEvent>` tag generates an event and passes the document's ID and type.

```
<%@ taglib uri="tracking.tld" prefix="tr" %>
.
.
.
<es:forEachInArray id="nextRow" array="<%=headlines%>"
  type="com.beasys.commerce.axiom.content.Content">
  <es:notNull item="<%=nextRow%>">
    <tr:displayContentEvent
      documentId="<%=nextRow.getIdentifier()%>"
      documentType="<%=headingProp%>"/>
  </es:notNull>
</es:forEachInArray>
```

<trp:clickProductEvent>

The `<trp:clickProductEvent>` tag (Table 5-3) is used to generate a behavior event when a user has clicked (through) on a product impression. This tag will return a URL query string containing event parameters. It is then used when forming the complete URL that hyperlinks the content.

At least one of `sku, categoryId,` or `documentId` is required.

Use the following code to import the product events tag library:
`<%@ taglib uri="productTracking.tld" prefix="trp" %>`

Table 5-3 <trp:clickProductEvent>

Tag Attribute	Req'd	Type	Description	R/C
`applicationName`	No	String	The webApp or application name, if applicable. Can be used to separate data when multiple storefronts are hosted on the same server (or persisted to the same database).	R
`categoryId`	No	String or Category object	Category of the product associated with the content displayed, if applicable.	R
`documentId`	Yes	String	Name of the item that is displayed, if applicable (that is, an image URL or banner ad ID).	R
`documentType`	No	String	Type or category of the item that is displayed (if applicable).	R
`redirectURL`	No	String	Where the server should redirect the client to after the document is clicked on and after the server processes the clickthrough event.	R
`sku`	No	String or ProductItem object	ID of the product associated with the content item that is displayed, if applicable.	R
`userId`	No	String	Name of the user that content was retrieved for. If the optional value is not provided, it will be set to the value of the `request.getRemoteUser()`.	R

Example

The example below demonstrates a clickthrough example going to the flow manager servlet. This link will cause a clickthrough product event to be generated and also display the indicated content. If you wish to redirect the client to an external site after recording the clickthrough event, specify a `redirectURL` attribute and target the `clickThroughServlet` instead of the flow manager servlet.

```
<%@ taglib uri="productTracking.tld" prefix="trp" %>
.
.
.
<%
detailsUrl = WebflowJSPHelper.createWebflowURL(pageContext,
"itemsummary.jsp", "link(" + detailsLink + ")",
"&" + HttpRequestConstants.CATALOG_ITEM_SKU + "=" +
productItem.getKey().getIdentifier() + "&" +
HttpRequestConstants.CATALOG_CATEGORY_ID + "=" +
category.getKey().getIdentifier() + "&" +
HttpRequestConstants.DOCUMENT_TYPE + "=" + detailsLink, true);
%>

<trp:clickProductEvent
   id="url"
   documentId="<%= productItem.getName() %>"
   sku="<%= productItem.getKey().getIdentifier() %>" />

<%
detailsUrl = detailsUrl + "&" + url;
%>

<a href="<%= detailsUrl %>">
```

<trp:displayProductEvent>

The `<trp:displayProductEvent>` tag (Table 5-4) is used to generate a behavior event when a user has received (viewed) a product impression, (typically a gif image).

At least one of `sku, categoryId,` or `documentId` is required.

Use the following code to import the product events tag library:
`<%@ taglib uri="productTracking.tld" prefix="trp" %>`

Table 5-4 <trp:displayProductEvent>

Tag Attribute	Req'd	Type	Description	R/C
`applicationName`	No	String	The webApp or application name, if applicable. Can be used to separate data when multiple storefronts are hosted on the same server (or persisted to the same database).	R
`categoryId`	No	String or Category object	Category of the product associated with the content displayed, if applicable.	R
`documentId`	No	String	Name of the item that is displayed, if applicable (that is, an image URL or banner ad ID).	R

`documentType`	No	String	Type or category of the item that is displayed (if applicable).
			`Suggestions:`
			`DisplayProductEvent.CATEGORY_BROWSE`
			`DisplayProductEvent.ITEM_BROWSE`
			`DisplayProductEvent.CATEGORY_VIEW`
			`DisplayProductEvent.BANNER_AD_PROMOTION`
				R

`sku`	No	String or ProductItem object	ID of the product associated with the content item that is displayed, if applicable.	R

Example

The example below shows a code snippet of processing that would follow the retrieval of a catalog item. The `<tr:displayProductEvent>` tag generates an event and passes the document's ID, type and SKU number of the product item.

```
<%@ taglib uri="productTracking.tld" prefix="trp" %>
.
.
.
<trp:displayProductEvent
    documentId="<%= item.getName() %>"
    documentType="<%= DisplayProductEvent.ITEM_BROWSE %>"
    sku="<%= item.getKey().getIdentifier() %>" />
```

<trc:clickCampaignEvent>

The `<trc:clickCampaignEvent>` tag (Table 5-5) is used to explicitly generate a clickthrough event relevant to a campaign. A *clickthrough* is when a user clicks on an advertisement's content. This tag will return a URL query string containing event parameters. It is then used when forming the complete URL that hyperlinks the content.

Note: The `<ph:placeholder>` tag is the principal means used to generate campaign click and display events on ads, which it does implicitly.

Use the following code to import the campaign events tag library:

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

Table 5-5 <trc:clickCampaignEvent>

Tag Attribute	Req'd	Type	Description	R/C
`campaignId`	No	String	ID of the associated campaign.	R
`applicationName`	No	String	The webApp or application name, if applicable. Can be used to separate data when multiple storefronts are hosted on the same server (or persisted to the same database).	R
`documentId`	Yes	String	Name of the item that is displayed, if applicable (that is, an image URL or banner ad ID).	R
`documentType`	No	String	Type or category of the item that is displayed, if applicable.	R
`id`	No	String	Page variable which will hold the output of this tag.	C
`placeholderId`	No	String	Name of the placeholder.	R
`redirectURL`	Yes	String	Where the server should redirect the client to after the document is clicked on and after the server processes the clickthrough event.	R
`scenarioId`	No	String	ID of the scenario assoicated with a campaign.	R

Table 5-5 <trc:clickCampaignEvent> (Continued)

Tag Attribute	Req'd	Type	Description	R/C
`userId`	No	String	Name of the user that content was retrieved for. If the optional value is not provided, it will be set to the value of the `request.getRemoteUser()`.	R

Example

The example below shows a code snippet of processing that generates the query string to be included in a campaign item hyperlink. Processing of the `redirectURL` attribute is analagous to the use of the `<tr:clickContentEvent>` tag.

```
<%@ taglib uri="campaignTracking.tld" prefix="trc" %>
.
.
.
<trc:clickCampaignEvent
  id="url"
  campaignId="<%=campaignId%>"
  placeholderId="<%=placeholderId%>"
  applicationName="myAppName"
  userId="<%=request.getRemoteUser()%>"/>
```

Index