# BEA WebLogic Commerce Server

## Webflow and Pipeline Management

BEA WebLogic Commerce Server 3.1
Document Edition 1.0
September 2000

## Copyright



## Restricted Rights Legend



## Trademarks or Service Marks



**Webflow and Pipeline Management**

| Document Edition | Date | Software Version |
|---|---|---|
| 1.0 | September 2000 | WebLogic Commerce Server 3.1 |

# Contents

## 3. Extending Webflow and Pipelines

## 4. Webflow and Pipeline JSP Tags



## Index

# About This Document

This document provides information about the Webflow and Pipeline mechanisms included in the BEA WebLogic Commerce Server. These mechanisms externalize the page flow and business logic that comprise any e-commerce Web site, and can be customized or extended to meet your business objectives.

This document includes the following topics:

- Chapter 1, “Overview of Webflow and Pipeline Management,” which describes the high-level architecture and categories for the Webflow and Pipeline mechanisms utilized in the BEA WebLogic Commerce Server product.
- Chapter 2, “Customizing Webflow and Pipelines,” which describes how a commerce engineer/JSP developer could customize the default Webflow and Pipeline mechanism to meet the requirements of their e-business.
- Chapter 3, “Extending Webflow and Pipelines,” which describes how a Java/EJB programmer can extend the default Webflow and Pipeline mechanisms to create new functionality for their e-business.
- Chapter 4, “Webflow and Pipeline JSP Tags,” which describes the specialized JSP tags that are used in the provided WebLogic Commerce Server Web application.

# What You Need to Know

This document is intended for the following audiences:

- The commerce engineer/JSP content developer, who uses JSP templates and tag libraries to implement interactive Web pages to meet business requirements. This user also maintains simple configuration files.
- The business analyst, who defines the company's business protocols (processes and rules) for a business-to-consumer Web site. This user may set pricing policies and discounts, and may plan promotional advertising.
- The site administrator, who uses Commerce and Personalization Server administration screens to configure the site's rules, portals, property sets, user profiles, content delivery, and product catalog.
- The Java/EJB programmer, who creates custom code to insert in the JSP files. This user may also handle complex configuration files.

# 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 WebLogic Commerce Server documentation Home page on the e-docs Web site (and also on the documentation CD). You can open the PDF in Adobe Acrobat Reader and print the entire document (or a portion of it) in book format. To access the PDFs, open the WebLogic Commerce 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 BEA WebLogic Commerce Server documents describe parts of an e-commerce application built upon the Webflow and Pipeline infrastructure:

- *BEA WebLogic Commerce Server Product Catalog Management*
- *BEA WebLogic Commerce Server Order Processing Package*
- *BEA WebLogic Commerce Server Registration and User Processing Package*

# Contact Us!

Your feedback on the BEA WebLogic Commerce 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 WebLogic Commerce Server documentation.

In your e-mail message, please indicate that you are using the documentation for the BEA WebLogic Commerce Server 3.1 release.

If you have any questions about this version of BEA WebLogic Commerce Server, or if you have problems installing and running BEA WebLogic Commerce 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. |

| Convention | Item |
|---|---|
| `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.<br>*Examples*:<br>`#include <iostream.h> void main ( ) the pointer psz`<br>`chmod u+w *`<br>`\tux\data\ap`<br>`.doc`<br>`tux.doc`<br>`BITMAP`<br>`float` |
| **`monospace boldface text`** | Identifies significant words in code.<br>*Example*:<br>`void` **`commit`** `( )` |
| *`monospace italic text`* | Identifies variables in code.<br>*Example*:<br>`String` *`expr`* |
| UPPERCASE TEXT | Indicates device names, environment variables, and logical operators.<br>*Example*s:<br>LPT1<br>SIGNON<br>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.<br>*Example*:<br>`buildobjclient [-v] [-o name ] [-f` *`file-list`*`]...`<br>`[-l` *`file-list`*`]...` |
| `\|` | Separates mutually exclusive choices in a syntax line. The symbol itself should never be typed. |

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

CHAPTER

# 1 Overview of Webflow and Pipeline Management

The Webflow and Pipeline are flexible mechanisms designed to help you manage both the presentation and business logic in your e-commerce Web site, without the need for advanced programming skills. This topic describes the high-level architecture of the Webflow and Pipeline, and provides preliminary information about how you can use these mechanisms to customize or extend the e-business site provided with the BEA WebLogic Commerce Server product.

This topic includes the following sections:

- High-level Architecture
- Development Roles
- Next Steps

# High-level Architecture

The BEA WebLogic Commerce Server design model separates presentation (such as HTML and JavaScript) from business logic (such as database updates and implementation of business rules). To create and maintain this separation, the Commerce Server makes use of the following six technologies:

- *HTML*: Standard HTML supported by Netscape Navigator or Microsoft Internet Explorer. Throughout this document, the term HTML refers to both HTML and JavaScript.
- *JSP Tags*: Customized tags used in the J2EE platform. The Commerce Server uses JavaServer Page (JSP) tags to add dynamic display to the HTML pages, such as displaying the name of a customer who is currently logged in.
- *Pipeline Components*: Discrete units of server-side business logic, such as calculating tax or commiting an order. Pipeline components can be combined into a Pipeline.
- *Pipeline Session*: Storage location for information about the current session (such as the current shopping cart) or more transient data (such as error messages about a customer's most recent input).
- *Input Processors*: Flexible mechanisms that handle form submission. Some may perform validation of customer data, but the primary role of an input processor is to store customer data into the Pipeline session for subsequent use by a Pipeline component.
- *Webflow*: Controls the flow of a customer's session through the pages displayed in a browser, and execution of specific pieces of business logic. Pages generate events (that is, which link or button the customer clicks) that result in the invocation of input processors and Pipelines. These in turn either succeed or generate exceptions, from which the Webflow decides which page to display or which piece of business logic to execute next.

This separation between presentation and business logic is beneficial for a number of reasons, but most importantly, it is helpful from a customization/maintenance standpoint. Different people within your organization may perform different tasks, and may specialize in a particular area. Keeping the user interface separate from the business processes and the Java programming allows your development team to

accomplish more in less time, and makes it easier for members of the team to focus on their areas of expertise or interest. For a description of typical roles, see “Development Roles” on page 1-5.

# Architecture Categories

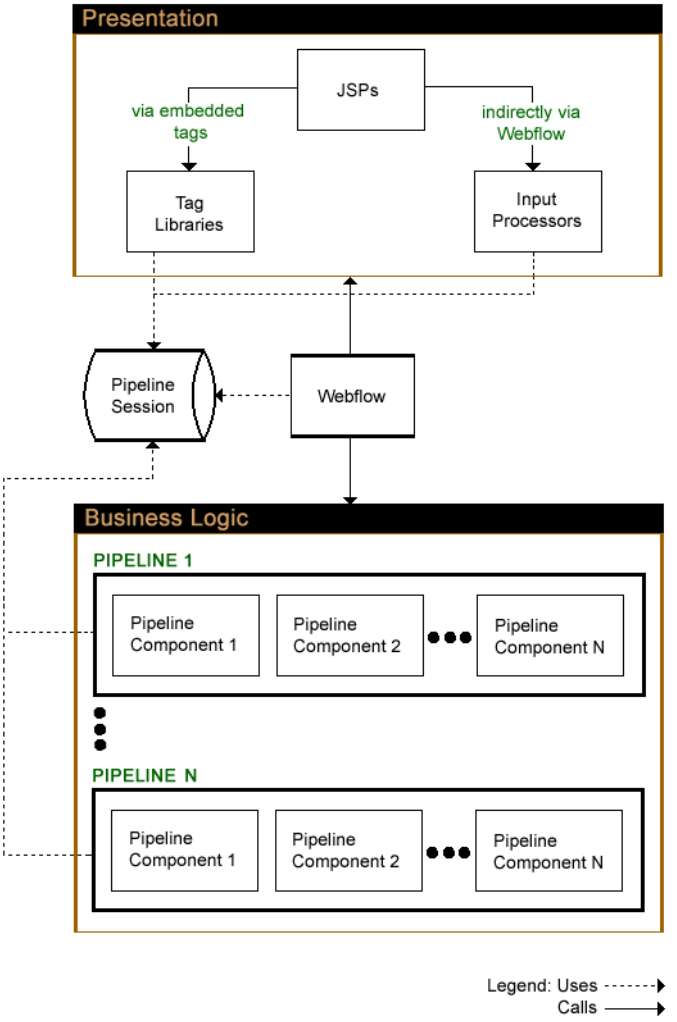
The six technologies previously described can best be understood as belonging to four categories: presentation, business logic, state maintenance, and flow of control.

HTML, JSP tags, and input processors consitute the presentation portion of the system. HTML is the display language understood by most browsers. JSP tags translate information from the Pipeline session to HTML, while input processors translate form data from HTML to the Pipeline session.

The Pipeline components containing pieces of business logic have no knowledge of HTML or any of the other presentation technologies. Instead, the Pipeline session maintains all of the conversational state in the system. Similarly, the Webflow governs the flow of control.

Figure 1-1 illustrates how the various technology categories interact to preserve the Commerce Server design model. Understanding this diagram is essential to understanding how to customize and extend the Webflow and Pipeline mechanisms.

**Figure 1-1 Webflow and Pipeline High-level Architecture**




As you learn more about the Webflow and Pipeline mechanisms, return to this architecture diagram. Each time you review the diagram, you will have a better understanding of the big picture.

# Development Roles

This document is intended for the following audiences:

- The commerce engineer/JSP content developer, who uses JSP templates and tag libraries to implement interactive Web pages to meet business requirements. This user also maintains simple configuration files.
- The business analyst, who defines the company's business protocols (processes and rules) for a business-to-consumer Web site. This user may set pricing policies and discounts, and may plan promotional advertising.
- The site administrator, who uses Commerce and Personalization Server administration screens to configure the site's rules, portals, property sets, user profiles, content delivery, and product catalog.
- The Java/EJB programmer, who creates custom code to insert in the JSP files. This user may also handle complex configuration files.

# Next Steps

The BEA WebLogic Commerce Server product ships with a working e-commerce site that can easily be modified to meet your specific business requirements. Many modifications, such as changes to page layout and presentation, can be completed without any Java coding. It is expected that these changes will be performed by a commerce engineer/JSP content developer or a site administrator, who consults with a business analyst about business strategies. For information about how to customize the Webflow and Pipeline, see Chapter 2, "Customizing Webflow and Pipelines."

Some of the more complex modifications, such as adding a Pipeline component to use a different credit card authorization system, will require Java coding. It is expected that these changes will be performed by a Java/EJB programmer. For information about how to extend the Webflow and Pipeline, see Chapter 3, "Extending Webflow and Pipelines."

The Commerce Server also contains a few JSP tags specifically designed to work with the Webflow and Pipeline mechanisms. You will use these JSP tags regardless of whether you are customizing or extending the Webflow and Pipeline. For detailed information about how to use the Webflow and Pipeline JSP Tags, see Chapter 4, “Webflow and Pipeline JSP Tags.”

# 2 Customizing Webflow and Pipelines

The most important benefit of the Webflow and Pipeline mechanisms is that they allow people with different levels of technical skill to customize both the presentation and business logic within an e-commerce site.

Commerce engineers/JSP content developers, site administrators, and business analysts can now divide Web site customization (and subsequent maintenance) activities based on their own expertise, interests, and job responsibilities. While they are working with the Webflow/Pipeline infrastructure, the Java/EJB programmers on the development team can be extending the BEA WebLogic Commerce Server packages to add functionality. Thus, some bottlenecks in the site development and maintenance process are greatly reduced.

If the packages that the BEA WebLogic Commerce Server product provides completely meet your requirements, all you may need to do to have a fully functioning e-business is to customize some aspects of the Webflow and/or Pipeline. This topic describes how to accomplish this.

This topic includes the following sections:

- Using Webflow
  - Customizing Webflow Using the webflow.properties File
  - Using Webflow in Your Web Pages
  - Webflow Search Order
- Using Input Processors with Webflow
  - Syntax of Input Processors in the webflow.properties File
  - Chaining Input Processors

# Using Webflow

Since every e-business is different, the BEA WebLogic Commerce Server product utilizes an external properties file to manage the sequence (flow) in which Web pages are displayed. The Commerce Server provides a default Webflow properties file to get you up and running quickly, and to provide you with a working example of this concept. You can modify this file to change the order of your pages, without having to edit each page individually.

This section provides information about the default Webflow and instructions for customizing it. This section also describes how to invoke the Webflow mechanism from your Web pages, and explains how missing transitions in the properties file are resolved.

## Customizing Webflow Using the webflow.properties File

The Webflow properties file (`webflow.properties`) controls the display of your site's Web pages and initiates execution of the business logic associated with these pages. The Webflow properties file contains one section for each JavaServer Page (JSP) and includes comments for increased readability.

Generically, each line in the `webflow.properties` file can be written as:

```
<origin>.[<event>][(<eventName>)]=<target>
```

Table 2-1 provides information about the elements shown above for which there are a limited number of valid values.

**Table 2-1 Valid Values for webflow.properties Elements**

| Element | Valid Values |
|---|---|
| <origin> | begin \| <page>.<extension> \| <inputprocessorName> \| <pipelineName> |
| <event> | event = link(<linkName>) \| button(<buttonName>) \| success \| exception(<exceptionName>) |
| <target> | <page>.<extension> \| <inputprocessorName> \| <pipelineName> |
| <extension> | jsp \| html \| htm \| inputprocessor \| pipeline |

**Notes:** Valid characters for <page>, <inputprocessorName> and <pipelineName> are limited to A-Z, a-z, and an underscore.

Lines in the webflow.properties file should never contain spaces. Including spaces can result in errors that are difficult to locate.

Text within the webflow.properties file is case sensitive.

## Syntax of the webflow.properties File

Each line in the webflow.properties file is comprised of a name/value pair, separated by an equal sign (=).

The name consists of the current state and a named event, and the value is a result state. In Listing 2-1, the current state is firstpage.jsp. The event is a button named Next, and the result state is nextpage.jsp.

**Listing 2-1 Webflow Properties Example**

```
firstpage.jsp.button(next)=nextpage.jsp
```

When a customer clicks the Next button from `firstpage.jsp`, the Webflow will load `nextpage.jsp`.

**Note:** The only exception to this syntax is the line in the `webflow.properties` file that defines the initial state for the Webflow, as follows:

```
begin=home.jsp
```

Web pages used as current or result states in the Webflow may be `.htm`, `.html`, or `.jsp` files. In addition to the `button` event shown in the previous example, there is also a `link` event associated with these file types.

## About Event Names

Events are given names because it is likely that a page has multiple events of the same type associated with it (that is, there are both previous and next buttons on `firstpage.jsp`, each requiring different result states). Event names are used to differentiate between these events, as shown in Listing 2-2.

**Listing 2-2 Event Names Example**

```
firstpage.jsp.button(previous)=previouspage.jsp
firstpage.jsp.button(next)=nextpage.jsp
```

**Note:** Although event names are arbitrarily selected, duplication of names within the `webflow.properties` file would defeat their purpose and should be avoided. Duplicate names will produce unpredictable results.

Although all the states in the previous examples are JSPs, both current and result states can also be input processors or Pipelines. For more information on input processors and Pipelines, see “Using Input Processors with Webflow” on page 2-10 and “Using Pipelines with Webflow” on page 2-13, respectively.

### Using the Wildcard Character

In cases where you want all your Web pages to reach a certain target page, you can substitute the wildcard character (*) for a specific page name in the current state. For example, if you want customers to be able to reach the home page from every page within your Web site, a line in the `webflow.properties` file would read:

```
*.jsp.link(home)=home.jsp
```

## Default Webflow

Listing 2-3 shows the portion of the default Webflow that handles category browsing. It can also be viewed in a simple text editor by opening `WL_COMMERCE_HOME/webflow.properties`, where `WL_COMMERCE_HOME` is the top-level directory where you installed WebLogic Commerce Server.

**Listing 2-3 Default Webflow for Category Browsing**

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

# Handle category browsing

##########################################

# Generic browse link gets browse parameters
*.jsp.link(browse)=BrowseCategory.inputprocessor

# Move intermediate results
BrowseCategory.inputprocessor.success=MoveSiblingResults.inputprocessor

# Get all category detail
MoveSiblingResults.inputprocessor.success=GetBrowseDetails.pipeline

# Display category detail
GetBrowseDetails.pipeline.success=commerce/catalog/browse.jsp

# Handle errors
BrowseCategory.inputprocessor.exception(ProcessingException)=commerce/catalog/
browse.jsp

MoveSiblingResults.inputprocessor.exception(ProcessingException)=commerce/
catalog/browse.jsp

GetBrowseDetails.pipeline.exception(PipelineFatalException)=commerce/catalog/
browse.jsp
```

```
# Define the input processor classes
BrowseCategory.inputprocessor=com.beasys.commerce.ebusiness.catalog.webflow.
GetCategoryIP

MoveSiblingResults.inputprocessor=com.beasys.commerce.ebusiness.catalog.webflow
.MoveAttributeIP
```

## Dynamically Modifying Your Site's Webflow

To dynamically modify your site's Webflow, consider the following:

- It is expected that a commerce engineer/JSP content developer (or someone with similar technical knowledge and abilities) will update the `webflow.properties` file.
- Be sure to modify the `webflow.properties` file in your development environment until you achieve the desired outcome. Then move your changes to a production environment.

To modify your site's Webflow, follow these steps:

1. Start a simple text editor like Notepad.
2. Open the default Webflow properties file, which can be found in `WL_COMMERCE_HOME/webflow.properties`, where `WL_COMMERCE_HOME` is the top-level directory where you installed WebLogic Commerce Server.
3. Modify the file as necessary, using the syntax described in the previous sections.
4. Save the modified file. You do not need to restart the server to view your changes if you have set the `webflow.hotdeploy.enable` property to `true` in the `weblogiccommerce.properties` file.

# Using Webflow in Your Web Pages

To utilize the Webflow mechanism, the URLs within your Web pages must include information that corresponds to a line in the `webflow.properties` file. Specifically, the URL must contain a page name and an event that match a current state in the `webflow.properties` file, as shown in Listing 2-4 and Listing 2-5.

**Listing 2-4 URL Within a <FORM> Tag in the Web Page**

```
<FORM method="post"
action="<%=WebflowJSPHelper.createWebflowURL(pageContext,
"login.jsp", button(createUser)", false)%>">
```

**Listing 2-5 Corresponding Line in the webflow.properties File**

```
login.jsp.button(createUser)=nextpage.jsp
```

These URLs are dynamically generated by a utility class called `WebflowJSPHelper`.

**Note:** In most cases, a Web page will use the `WebflowJSPHelper` class multiple times. Therefore, it is a good idea to import the class at the beginning of your Web page as shown in the following statement:

```
<%@ page import="com.beasys.commerce.webflow.WebflowJSPHelper" %>
```

As shown in Listing 2-4, the `WebflowJSPHelper` class has a `createWebflowURL()` method that takes four parameters: `pageContext`, the name of the current JSP with extension (origin), the event type and name, and a URL type. A URL type of `true` causes the returned string to include the origin and event parameters as query parameters. Using this information, the `createWebflowURL()` method returns an absolute URL.

Although the parameters of the `createWebflowURL()` method are always the same, the way you specify these parameters depends on whether you are generating the URL within a `<FORM>` tag or an `<A>` (anchor) tag.

To incorporate a URL using the <FORM> tag, use the action attribute to construct the URL as shown in Listing 2-4. To incorporate a URL using the <A> tag, call the createWebflowURL() method as shown in Listing 2-6.

**Listing 2-6 Dynamic URL Generation Within an <A> Tag**

```
<a href="<%=WebflowJSPHelper.createWebflowURL(pageContext,
"login.jsp", button(createUser)%>", false)>
```

In both cases, these statements are translated into login.jsp.button(createUser), which can be found in the webflow.properties file to the left of an equal sign, as shown in Listing 2-5. The value to the right of this equal sign will initiate the result state, and thus allow the Webflow mechanism to continue.

# Webflow Search Order

There may be times when a transition in the Webflow is missing (that is, no result state has been specified). To prevent any problem from being visible to your customer, the Webflow will attempt to resolve missing transitions by searching through several possibilities to locate an alternate flow. These search possiblities are examined by the Webflow mechanism in the following order:

- The Webflow substitutes the wildcard character for the specific page, input processor, or Pipeline.
- If wildcard subsitution fails, the Webflow produces a configuration exception relative to where it encountered the missing transition, and uses this as the result state.
- If contextual configuration exceptions do not allow the Webflow to continue, the Webflow combines the wildcard substitution with a generic exception, which it uses as the result state.
- If the previous attempts fail, the Webflow will simply load a configuration error page.

**Note:** The configuration error page can be configured in the webflow.properties file under the property configurationerrorpage.

In summary, the search order attempts to prevent a missing transition in the Webflow from interrupting a customer's experience on your Web site. Rather, in the very worst case, the Webflow would load the configuration error page. If for some reason this file was missing, a predefined system error page (`servererror.jsp`), which is also beyond the scope of the Webflow mechanism, would be used instead.

## Search Order Examples

Suppose the Webflow mechanism is attempting to locate the missing transition `login.jsp.link(home)` in the `webflow.properties` file. The following list illustrates the alternate transitions that may be used by the Webflow:

- `*.jsp.link(home)`
- `login.jsp.error(ConfigurationException)`
- `*.jsp.error(ConfigurationException)`
- `configurationerrorpage`

The Webflow search order will also be performed for input processors and Pipelines that are missing in the `webflow.properties` file. The following list illustrates the alternate transitions that may be used by the Webflow for the missing transition `ShoppingCartIP.inputprocessor.success`:

- `*.inputprocessor.success`
- `ShoppingCartIP.inputprocessor.error(ConfigurationException)`
- `*.inputprocessor.error(ConfigurationException)`
- `configurationerrorpage`

Similarly, the following list illustrates the alternate transitions that may be used by the Webflow for the missing transition `ShoppingCartPC.pipeline.success`:

- `*.pipeline.success`
- `ShoppingCartPC.pipeline.error(ConfigurationException)`
- `*.pipeline.error(ConfigurationException)`
- `configurationerrorpage`

# Using Input Processors with Webflow

States in the `webflow.properties` file are not restricted to other Web pages. Input processors are predefined classes that provide a way to indirectly carry out more complex tasks using the Webflow mechanism. Input processors reduce the need to incorporate complex Java code into your JSPs, and help maintain the separation between presentation and business logic.

The role of input processors is to read data from the `HTTPServletRequest` and use it to create or update Java objects in a Pipeline session. In addition to working with this data, some input processors may also validate information supplied by the customer.

**Note:** It is not required that you use input processors in your customized Webflow. If you do not wish to use input processors, simply do not specify any input processors in the `webflow.peoperties` file.

This section provides information about invoking input processors from theWebflow and about chaining input processors. This section also points you to additional information about extending or developing your own input processors.

## Syntax of Input Processors in the webflow.properties File

Input processors extend the syntax used for JSPs in the `webflow.properties` file. For example, if you want to verify that the customer filled in the required form fields for their address before sending the customer to the next page, you could use the `ValidateAddress` input processor as shown in Listing 2-7.

**Listing 2-7 Input Processor for Address Validation**

```
################################
# ValidateAddress input processor
################################
```

```
# Invoke the input processor
addaddress.jsp.button(continue)=ValidateAddressIP.inputprocessor

# Specify the fully qualified class name for the input processor
ValidateAddressIP.inputprocessor=com.beasys.commerce.ebusiness.
customer.webflow.ValidateAddressIP

# Specify the result state for successful execution
ValidateAddressIP.inputprocessor.success=selectaddress.jsp

# Specify the result state for unsuccessful execution
ValidateAddressIP.inputprocessor.exception(ProcessingException)=
addaddress.jsp
```

In the first line, a customer who clicks the Continue button causes the flow to be turned over to the input processor called `ValidateAddressIP`. The second line defines the full class name of the `ValidateAddressIP` input processor, which will validate the form fields. The third and fourth lines make use of the event types defined for input processors: `success` and `exception`. If the validation is successful, the result state indicated by the success event is to load the `selectaddress.jsp` file. If the validation is not successful, the `ValidateAddressIP` input processor directs the customer back to `addaddress.jsp` to make corrections.

**Notes:** For the complete list of event types and more information on the syntax of input processors in the Webflow, see Table 2-1.

If execution of an input processor is not successful, you may specify different result states identified by more than one exception event.

# Chaining Input Processors

In addition to using input processors between JSPs and Pipelines, you can also use more than one input processor, or chain input processors. In a chaining arrangement, the result state of one successfully executed input processor will be another input processor, as shown in Listing 2-8.

**Listing 2-8 Example of Input Processor Chaining**

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

```
# Example of input processor chaining

####################################

# Invoke the first input processor
webpage.jsp.link(continue)=firstInputProcessor

# Specify the fully qualified path name for the first input
# processor
firstInputProcessor.inputprocessor=com.beasys.commerce.webflow.
firstInputProcessorIP

# Invoke the second input processor if the execution of the first
# input processor succeeds
firstInputProcessor.inputprocessor.success=secondInputProcessor

# Specify the fully qualified class name for the second input
# processor
secondInputProcessor.inputprocessor=com.beasys.commerce.webflow.
secondInputProcessorIP

# Specify the result state for successful execution of the second
# input processor
secondInputProcessor.inputprocessor.success=nextwebpage.jsp

# Specify the result state for unsuccessful execution of the first
# input processor
firstInputProcessor.inputprocessor.exception(ProcessingException)
=errorpage.jsp

# Specify the result state for successful execution of the second
# input processor
secondInputProcessor.inputprocessor.exception
(ProcessingException)=anothererrorpage.jsp
```

# Further Customization of Input Processors

If you would like to customize your site even further, you might choose to create and implement your own input processors or define your own exceptions for use with input processors. However, there are some important rules you need to follow to accomplish these tasks. For more information, see Chapter 3, "Extending Webflow and Pipelines."

**Note:** Only Java/EJB programmers (or someone with similar technical knowledge and abilities) should attempt to customize input processors.

# Using Pipelines with Webflow

Your site would not be considered an e-business if you simply displayed pages and performed some additional tasks with input processors. A customer's entire experience also relies upon the execution of back-end business processes that are related to where the customer is on your site and what the customer is trying to accomplish.

A Pipeline is an advanced mechanism invoked by the Webflow that initiates execution of specific tasks related to your business process. For example, if a customer attempts to move to another page on your site but you want to save the customer's identifying information to a database first, you could use a Pipeline.

All Pipelines are collections of individual Pipeline components, which can be either Java objects or stateless session EJBs. Pipeline components are the parts of a Pipeline that actually perform the tasks associated with the underlying business logic. When these tasks are complex, Pipeline components may also make calls to external services (other business objects). As in the case of input processors, the BEA Weblogic Commerce Server product provides predefined Pipeline components that you can use, or you can customize your site further by creating your own.

To successfully carry out business tasks, each Pipeline component must read attributes from a Pipeline session and if necessary, write modified versions of these attributes back to the Pipeline session. Pipeline session are available for the life of the HTTP session.

The Commerce Server provides a default Pipeline properties file to get you up and running quickly, and to provide you with a working example of this concept. You can modify this file to change the business logic associated with your Web pages, without having to edit each page individually.

**Note:** It is not required that you use Pipelines to execute business logic in your customized Webflow. If you do not wish to use Pipelines, simply do not specify any Pipelines in the `webflow.peoperties` file. However, eliminating Pipelines and Pipeline components results in a less scalable, 2-tier architecture instead of the 3-tier architecture provided by the Webflow/Pipeline infrastructure.

This section provides information about the default Pipeline and instructions for customizing it, and describes how to invoke Pipelines from the Webflow.

# Customizing Pipelines Using the pipeline.properties File

Much like the `webflow.properties` file specifies the flow of Web pages presented to a customer, the Pipeline properties file (`pipeline.properties`) specifies the flow of business logic as the customer moves through each page of the site. This properties file contains one section for each JavaServer Page (JSP) and includes comments for increased readability.

Generically, Pipeline definitions can be written as:

```
<pipelineName>.componentList
<pipelineName>.isTransactional=<true|false>
```

where `componentList` is a comma-separated list of Pipeline components to be executed in sequence.

Once all Pipeline definitions are complete, you must specify definitions for each Pipeline component in the Pipeline. Each Pipeline component definition consists of three properties: `className`, `jndiName`, and `isEJBSessionBean`.

Table 2-2 describes each of the Pipeline component properties in detail.

**Table 2-2 Pipeline Component Properties**

| Property | Description | Value |
|---|---|---|
| `className` | Name of the class that implements the Pipeline component, required if `isEJBSessionBean` is `false` | A fully qualified Java class name |
| `isEJBSessionBean` | Specifies whether or not the Pipeline component is a session bean, and always requires a value | `true` \| `false` |
| `jndiName` | JNDI name of the session bean that implements the Pipeline component, required only if `isEJBSessionBean=true` | A string |

**Notes:** Lines in the `pipeline.properties` file should never contain spaces. Including spaces can result in errors that are difficult to locate.

Text within the `pipeline.properties` file is case sensitive.

## Syntax of the pipeline.properties File

The top portion of the `pipeline.properties` file should contain only Pipeline definitions. Pipeline definitions include:

- A Pipeline name.
- A list of its associated Pipeline components in order of execution.
- A value for the `isTransactional` Pipeline property, indicating whether or not all the Pipeline components in the Pipeline will participate in a transaction.

Listing 2-9 is a Pipeline definition that might be used in the `pipline.properties` file.

**Listing 2-9 Pipeline Definition Example**

```
orderPipeline=CalculateTaxPC,CalculateDiscountPC,TotalCartCostPC
orderPipeline.isTransactional=true
```

In this example, the Pipeline called `orderPipeline` consists of three Pipeline components (`CalculateTaxPC, CalculateDiscountPC, TotalCartCostPC`). The `orderPipeline` is also transactional.

Listing 2-10 shows the corresponding Pipeline component definitions that might be used in the `pipeline.properties` file.

**Listing 2-10 Pipeline Component Definition Example**

```
CalculateTaxPC.classname=com.beasys.commerce.ebusiness.order.
pipeline.CalculateTaxPC
CalculateTaxPC.isEJBSessionBean=false
CalculateTaxPC.jndiName=

CalculateDiscountPC.classname=com.beasys.commerce.ebusiness.order
.pipeline.CalculateDiscountPC
CalculateDiscountPC.isEJBSessionBean=false
CalculateDiscountPC.jndiName=

TotalCartCostPC.classname=com.beasys.commerce.ebusiness.order
pipeline.TotalCartCostPC
```

```
TotalCartCostPC.isEJBSessionBean=true
TotalCartCostPC.jndiName=com.beasys.commerce.ebusiness.order.
pipeline.TotalCartCostPC
```

## Default Pipeline

Listing 2-11 shows portions of the default Pipeline property file that handle obtaining product categories (implemented as a Java object) and moving an item to a shopping cart (implemented as an EJB session bean). These can also be viewed in a simple text editor by opening `WL_COMMERCE_HOME/pipeline.properties`, where `WL_COMMERCE_HOME` is the top-level directory where you installed WebLogic Commerce Server.

**Listing 2-11 Default Pipelines for Product Categories and Shopping Cart**

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

# Java class Pipeline for obtaining product categories

############################################################

# GetTopCategories Pipeline definition
GetTopCategories.componentList=GetCategoryPC,GetSubcategoriesPC
GetTopCategories.isTransactional=false

# GetCategoryPC Pipeline component definition
GetCategoryPC.classname=com.beasys.commerce.ebusiness.catalog.
pipeline.GetCategoryPC
GetCategoryPC.jndiName=
GetCategoryPC.isEJBSessionBean=false

# GetSubcategoriesPC Pipeline component definition
GetSubcategoriesPC.classname=com.beasys.commerce.ebusiness.
catalog.pipeline.GetSubcategoriesPC
GetSubcategoriesPC.jndiName=
GetSubcategoriesPC.isEJBSessionBean=false

################################################################

# EJB session bean Pipeline for moving items to shopping cart

################################################################

# MoveProductItemToShoppingCart Pipeline definition
MoveProductItemToShoppingCart.componentList=
```

```
MoveProductItemToShoppingCartPC
MoveProductItemToShoppingCart.isTransactional=true

# MoveProductItemToShoppingCartPC Pipeline component definition
MoveProductItemToShoppingCartPC.classname=com.beasys.commerce.
ebusiness.shoppingcart.pipeline.MoveProductItemToShoppingCartPC
MoveProductItemToShoppingCartPC.jndiName=com.beasys.commerce.
ebusiness.shoppingcart.pipeline.MoveProductItemToShoppingCartPC
MoveProductItemToShoppingCartPC.isEJBSessionBean=true
```

## Dynamically Modifying Your Site's Pipelines

To dynamically modify your site's Pipelines, consider the following:

- It is expected that a business analyst will work with the commerce engineer/JSP content developer (or someone with similar technical knowledge and abilities) to update the `pipeline.properties` file.
- Be sure to modify the `pipeline.properties` file in your development environment until you achieve the desired outcome. Then move your changes to a production environment.

To modify your site's Pipelines, follow these steps:

1. Start a simple text editor like Notepad.
2. Open the default Pipeline properties file, which can be found in `WL_COMMERCE_HOME/pipeline.properties`, where `WL_COMMERCE_HOME` is the top-level directory where you installed WebLogic Commerce Server.
3. Modify the file as necessary, using the syntax described in the previous sections.
4. Save the modified file. You do not need to restart the server to view your changes if you have set the `pipeline.hotdeploy.enable` property to `true` in the `weblogiccommerce.properties` file.

## Eliminating Pipeline Components

The `pipeline.properties` file provides an easy way to eliminate Pipeline components without the need for advanced programming skills. For example, you might want to eliminate a Pipeline component that performs your tax calculations in the `CommitOrder` Pipeline. The definition in the default `pipeline.properties` file for the `CommitOrder` Pipeline is shown in Listing 2-12.

**Listing 2-12 Default OrderCommit Pipeline**

```
# CommitOrder
CommitOrder.componentList=CommitOrderPC, AuthorizePaymentPC,
TaxCalculateAndCommitLineLevelPC
```

To eliminate a Pipeline component, follow these steps:

**Notes:** These instructions assume that the Commerce Server software is installed and has been started.

You do not need to restart the server to view your changes if you have set the `pipeline.hotdeploy.enable` property to `true` in the `weblogiccommerce.properties` file.

1. Open the `pipeline.properties` file and remove each reference to the Pipeline component you want to eliminate. For example, if tax calculations are not required, remove all tax calculation Pipeline components from the Pipeline definition, as shown in Listing 2-13. Be sure to save your changes.

**Listing 2-13 OrderCommit Pipeline Without Tax Calculation Component**

```
# CommitOrder
CommitOrder.componentList=CommitOrderPC, AuthorizePaymentPC
```

2. If necessary, edit the related JSPs to eliminate places in the user interface where the information is gathered. Be sure to save your changes.

**Note:** There is no JSP that collects tax information. In the `CommitOrder` Pipeline example, you would not need to make changes to the user interface.

3. If in step 2 you removed an entire JSP, you will need to open the `webflow.properties` file and change any reference(s) to bypass it. Be sure to save your changes.

## Reordering Pipeline Components

The `pipeline.properties` file also provides an easy way for you modify the sequence of Pipeline components, without the need for advanced programming skills.

To reorder a Pipeline component, all you need to do is open the `pipeline.properties` file and change the order that the Pipeline components are listed in the Pipeline definition. Be sure to save your changes.

Using the same `CommitOrder` Pipeline example, say you wanted to authorize the payment after calculating the tax instead of before it (as in the default `OrderCommit` Pipeline). Listing 2-14 shows how to do this.

**Listing 2-14 OrderCommit Pipeline with Tax Component Reordered**

```
# CommitOrder
CommitOrder.componentList=CommitOrderPC,
TaxCalculateAndCommitLineLevelPC, AuthorizePaymentPC
```

**Note:** You do not need to restart the server to view your changes if you have set the `pipeline.hotdeploy.enable` property to `true` in the `weblogiccommerce.properties` file.

# Using Pipelines in the Webflow

Pipelines are used in the `webflow.properties` file to initiate execution of the business logic required for a particular page. Each Pipeline must first be invoked by the Webflow, and then followed by a success and exception path. For example, if a customer were to submit their order for processing, the `orderPipeline` might be represented in the `webflow.properties` file as shown in Listing 2-15.

**Listing 2-15 Using a Pipeline in the Webflow**

```
shoppingcart.jsp.button(submit)=orderPipeline.pipeline

orderPipeline.pipeline.success=commitorder.jsp

orderPipeline.pipeline.exception(PipelineFatalException)=
shoppingcart.jsp
```

The first line indicates that when a customer clicks on the Submit button, the Webflow will turn control over to the Pipeline called `orderPipeline`. If the Pipeline executes successfully (that is, if each component in the Pipeline executes without error), the second line sends the customer to a page that allows the customer to commit the order. If the Pipeline does not execute successfully, the third line specifies the exception and directs the customer back to the shopping cart page.

# Further Customization of Pipelines

If you would like to customize your site even further, you might choose to create and implement your own Pipelines or define your own exceptions for use with Pipelines. However, there are some important rules you need to follow to accomplish these tasks. For a more information, see Chapter 3, “Extending Webflow and Pipelines.”

**Note:** Only Java/EJB programmers (or someone with similar technical knowledge and abilities) should attempt to customize Pipelines.

CHAPTER

# 3 Extending Webflow and Pipelines

Although the BEA WebLogic Commerce Server product provides default Webflow and Pipeline mechanisms that you can customize, the Webflow and Pipelines have also been designed for easy extensibility. For example, if your organizational requirements dictate the use of a new business process, the Java/EJB programmers on your development team can utilize the existing Webflow and Pipeline infrastructure to create and incorporate these components into the system. This topic describes how to accomplish this.

This topic includes the following sections:

- Pipeline Sessions
  - What Is a Pipeline Session?
  - Attribute Scoping
  - Managing the Pipeline Session
- Extending Input Processors
  - Using the InputProcessor Interface
  - Input Processor Exceptions
  - The CommerceInputProcessor Base Class
  - Input Processor Naming Conventions
  - Input Processors and Statelessness
  - Other Development Guidelines
- Extending Pipelines and Pipeline Components

- Using the PipelineComponent Interface
- Pipeline Component Exceptions
- The CommercePipelineComponent Base Class
- Pipeline Component Naming Conventions
- Implementation of Pipeline Components as Stateless Session EJBs or Java Objects
- Stateful Versus Stateless Pipeline Components
- Transactional Versus Non-transactional Pipelines
- Other Development Guidelines

■ Handling Session Timeouts

- Using the getPipelineSession() Method
- The InvalidSessionStateException Exception in webflow.properties
- PipelineComponent and Session Timeouts
- The InvalidPipelineSessionStateException Exception in webflow.properites
- About the sessiontimeout.jsp Template

# Pipeline Sessions

Although Pipelines and their components are reusable, they must relate to a particular customer's experience on your e-commerce site to make their execution relevant. For this reason, Pipeline components always operate on a Pipeline session. This section provides you with information about the Pipeline session, and provides instructions for configuring the Pipeline session to meet your own needs.

# What Is a Pipeline Session?

Clearly, it is necessary to keep track of information gathered from your customers and the data modified by Pipeline components as a customer moves through your site. To maintain this state of the business process, the BEA WebLogic Commerce Server product makes use of a Pipeline session. A Pipeline session is an object that is created and stored within the HTTP session, with the goal of providing a single point of communication for all Pipeline components in a given Pipeline. Additionally, Pipeline sessions provide central access and storage for all external classes that may also need to update the Pipeline session.

The Pipeline session is comprised of many name/value pairs called *attributes*. Pipeline components act on particular attributes that exist within the Pipeline session, and may also add new attributes as necessary.

# Attribute Scoping

The Pipeline session provides an API that allows you to add Pipeline session attributes. All attributes in the Pipeline session can have one of two scopes: Pipeline Session scope or Request scope. The method signature for creating Pipeline session attributes is:

```
public void setAttribute(String key, Object attribute, int scope);
```

where `scope` is either `PipelineConstants.PIPELINE_SESSION_SCOPE` or `PipelineConstants.REQUEST_SCOPE`.

In the Pipeline Session scope, the attribute exists in the Pipeline session until the end of the current HTTP session. Pipeline Session scope is the default scope for Pipeline session attributes, and will be used if the third parameter to the `setAttribute()` method is not specified. In the Request scope, the attributes are made available in the `HTTPServletRequest`, and these attributes should be accessed via the `getPipelineProperty` JSP tag (that is, the attributes exist only for the life of an HTTP request).

Basically, Pipeline Session and Request scoping differ by how long the attribute is retained. When an attribute is specified with the Request scope, it is available from the time it is set, up to and including the display of the next JSP. The attribute is automatically deleted when a new request starts. Therefore, Request scope is useful for temporary objects that will only be needed for one page. For example, search results

from the product catalog are stored as Request-scoped attributes. Attributes that must be longer lived should be specified as Pipeline Session scope, which will cause them to be retained throughout the customer's session. If you know that a Pipeline session attribute is only required for the current request, use the Request scope.

**Note:** All attributes added to the Pipeline session should be serializable. If they are not, the server will generate an error when trying to serialize the Pipeline session, and thus no Pipelines will be executed. To assist in debugging, set the `pipelineSession.debug` property in the `weblogiccommerce.properties` file to `true`. Then, when a Pipeline session `setAttribute()` method is called, the server console will indicate whether the attribute is serializable or not.

# Managing the Pipeline Session

It is important that the Pipeline and HTTP sessions are associated with each other and that they are updated in parallel. This section contains information about accessing and storing the Pipeline session that is important to maintaining this relationship.

## Accessing the Pipeline Session

It is highly recommended that clients requiring access to the Pipeline session use one of the APIs of the `CommerceInputProcessor` base class. The `CommerceInputProcessor` class is responsible for creating a new Pipeline session (if required), and for associating the Pipeline session with the HTTP session.

**Note:** For more information about `CommerceInputProcessor`, see "The CommerceInputProcessor Base Class" on page 3-7.

## Storing the Pipeline Session in the HTTP Session

Each time the Pipeline session is updated, the HTTP session also needs to be updated so that the Pipeline session is replicated across all the nodes in a cluster. Because the Webflow infrastructure is responsible for setting the Pipeline session to the HTTP session at appropriate times., it is highly recommended that none of the `InputProcessors` directly store the Pipeline session in the HTTP session.

**Note:** For more information about `InputProcessors`, see “Using the InputProcessor Interface” on page 3-6.

# Extending Input Processors

In addition to using the input processors provided in the BEA WebLogic Commerce Server product, you can create your own input processors. This section describes the conventions you must follow when creating new input processors.

**Note:** It is expected that a Java/EJB programmer (or someone with similar technical knowledge and abilities) will develop new input processors.

## Using the InputProcessor Interface

New input processors must implement the `InputProcessor` interface and must supply an implementation for the `process` method. The `process` method accepts an `HTTPServletRequest` object as a parameter and returns a string (such as success) if execution is successful, as shown in the following method signature:

```
public String process (HTTPServletRequest request) throws ProcessingException
```

**Notes:** For more information about the `InputProcessor` interface, see the *Javadoc*.

For information about how the `InputProcessor` interface can be used to handle session time outs, see "Handling Session Timeouts" on page 3-14.

## Input Processor Exceptions

All input processors must throw the `ProcessingException` exception, or one of its subclasses. To obtain the `ProcessingException` exception's exception message, use the scriptlet shown in Listing 3-1.

**Listing 3-1 Obtaining the ProcessingException Exception Message**

```
<% String errorMsg =
(String)request.getAttribute(HttpRequestConstants.PIPELINE_MESSAGE); %>
```

**Note:** For more information about the `ProcessingException` exception, see the *Javadoc*.

## The CommerceInputProcessor Base Class

`CommerceInputProcessor` is the abstract base class for all the BEA WebLogic Commerce Server classes that implement the `InputProcessor` interface. This class has a number of utility methods for all the derived classes to use, some of which allow you to:

- get the `PipelineSession` object associated with the current session.
- get the `PipelineSession` object only if the `HttpSession` is a valid session.

**Note:** For more information on the `CommerceInputProcessor` base class, see the *Javadoc*.

## Input Processor Naming Conventions

The name of an input processor should end with the suffix `IP`. For example, an input processor that is responsible for deleting a shipping address might be called `DeleteShippingAddressIP`.

## Input Processors and Statelessness

Because the Webflow controls the life cycle of input processors, the Webflow may create and destroy input processors without regard for the data that may be contained within them. Therefore, input processors should always be stateless, and it is recommended that you do not define any instance variables in an input processor.

# Other Development Guidelines

Execution of business (application) logic should not be done within input processors. Specifically, input processors should not call EJBs or attempt to access a database. All such logic should be implemented in Pipeline components. Although it is possible to execute this logic within an input processor, doing so would defeat the purpose of the Webflow/Pipeline infrastructure and would not easily lend itself to modification.

By separating business logic from the presentation logic, your e-commerce site is inherently flexible in nature. Modifying or adding functionality can be as simple as creating and plugging in new Pipelines and/or input processors.

- For more information about input processors, see "Using Input Processors with Webflow" on page 2-10.
- For more information about Pipeline components, see "Using Pipelines with Webflow" on page 2-13.

# Extending Pipelines and Pipeline Components

In addition to using the Pipelines and Pipeline components provided in the BEA WebLogic Commerce Server product, you can create your own Pipelines and Pipeline components. This section describes the conventions you must follow when creating new Pipelines and Pipeline components.

**Note:** It is expected that a Java/EJB programmer (or someone with similar technical knowledge and abilities) will develop new Pipelines and Pipeline components.

## Using the PipelineComponent Interface

New Pipeline components must implement the `PipelineComponent` interface and must supply an implementation for the `process` method. The `process` method accepts a `PipelineSession` object as a parameter, and returns updated `PipelineSession` objects if the execution is successful, as shown in the following method signature:

```
public PipelineSession process(PipelineSession session) throws RemoteException,
PipelineNonFatalException, PipelineFatalException
```

**Notes:** For more information about the `PipelineComponent` interface, see the *Javadoc*.

For information about how the `PipelineComponent` interface can be used to handle session time outs, see "Handling Session Timeouts" on page 3-14.

## Pipeline Component Exceptions

Pipeline components may throw a `PipelineFatalException` to signify that the component has failed. When this occurs, no further Pipeline components are executed and if the Pipeline is transactional, the transaction will be rolled back.

To obtain the `PipelineFatalException` exception's exception message, use the scriptlet shown in Listing 3-2.

**Listing 3-2 Obtaining the PipelineFatalException Exception Message**

```
<% String errorMsg =
(String)request.getAttribute(HttpRequestConstants.PIPELINE_MESSAGE); %>
```

**Note:** For more information about fatal Pipeline exceptions in a transactional Pipeline, see "Transactional Versus Non-transactional Pipelines" on page 3-12.

Pipeline components may also throw a `PipelineNonFatalException` to indicate that the component has failed, but that subsequent Pipeline components should be executed. Lastly, a Pipeline component may throw a `RemoteException`.

The Webflow integrates with these exceptions as follows:

- `PipelineFatalException`: If any component in a Pipeline throws a `PipelineFatalException` or a class derived from `PipelineFatalException`, besides aborting the Pipeline and the transaction, the Webflow will perform an exception search on the exception thrown.

**Note:** For a detailed description about how Webflow searches transitions, see "Webflow Search Order" on page 2-8.

- `RemoteException`: If the Pipeline throws a `RemoteException`, it is treated as a server error and the `servererror.jsp` is displayed.

When an exception search is performed, the Webflow looks for the exact exception found as the event. If this exception is not found, the Webflow will begin looking through the search order, as decribed in "Webflow Search Order" on page 2-8.

# The CommercePipelineComponent Base Class

`CommercePipelineComponent` is an abstract base class for all the BEA WebLogic Commerce Server classes that implement the `PipelineComponent` interface. This class provides a utility method that allows you to obtain database connections from the `commercePool` (configured in the `weblogic.properties` file).

**Note:** For more information about the `CommercePipelineComponent` base class, see the *Javadoc*.

# Pipeline Component Naming Conventions

The name of a Pipeline component should end with the suffix `PC`. For example, a Pipeline component that is responsible for saving a shopping cart might be called `SaveCartPC`.

# Implementation of Pipeline Components as Stateless Session EJBs or Java Objects

Pipeline components can be implemented as either stateless session EJBs or as Java objects. Table 3-1 describes the differences between the two implementations.

**Table 3-1 Comparison of Pipeline Component Implementations**

| Stateless Session EJBs | Java Objects |
|---|---|
| Heavier in weight and more complex to implement due to EJB overhead. | Lightweight, low overhead. |
| Server-provided instance caching. | No instance caching, possibly degrading performance. |
| Server-provided load balancing. | No load balancing, always executes on the node in the cluster where the Pipeline started execution. |

**Table 3-1 Comparison of Pipeline Component Implementations**

| Stateless Session EJBs | Java Objects |
|---|---|
| Can use ACL-based security according to EJB specification. | Must manage security. |

An implementing class that is a stateless session EJB must meet the following requirements:

- It must declare and implement a `create()` method in the bean's `Home` interface that takes no arguments and returns the appropriate `Remote` interface.
- It must declare and implement the `process()` method as part of its `Remote` interface.

# Stateful Versus Stateless Pipeline Components

Whether Pipeline components are implemented as stateless session EJBs or as Java objects, Pipeline components themselves should be stateless. The business logic implemented in Pipeline components should only depend upon the `PipelineSession` object, the database, and other external resources. Should you define any instance variables, static variables, or static initializers within a Pipeline component, the results may be unpredictable.

# Transactional Versus Non-transactional Pipelines

If all Pipeline components within the Pipeline will be invoked under one transaction, the respective Pipeline's `isTransactional` property should be set to `true` in the Pipeline definition (within `pipeline.properties` file). Transactional Pipelines provide support for rolling back the database transaction and for making changes to the Pipeline session. If a transactional Pipeline fails, any database operations made by each of its Pipeline components are rolled back.

If a Pipeline component in a transactional Pipeline is implemented as a stateless session EJB, then its transaction attribute must be `Required`. Also, be sure that each of the Pipeline components in a transactional Pipeline has the correct transaction flag. Transaction flags indicate whether or not each bean will participate in the transaction.

If the Pipeline's `isTransactional` property is `true` and the participating Pipeline components (beans) have their transaction flag set to `never`, the Pipeline will fail to execute. Similarly, if the Pipeline's `isTransactional` property is `false` and the Pipeline components have the transaction flag set to `mandatory`, the Pipeline will also fail to execute.

If a Pipeline component in a transactional Pipeline is implemented as a simple Java object, then for all database operations, the Pipeline component must use the Transactional DataSource associated with the connection pool, as defined in the `weblogic.properties` file. A transactional Pipeline containing Pipeline components implemented as simple Java objects commits the transaction upon success, and rolls back the transaction upon failure.

# Other Development Guidelines

All server-side coding guidelines apply for development of new Pipeline components. Specifically:

- Avoid using threads.
- Avoid accessing the file system, since these operations are not thread-safe.
- Program all Pipeline components that are implemented as Java objects to be thread-safe.

# Handling Session Timeouts

In any Web application, the `HttpSession` is usually short-lived. Therefore, every time the `HttpSession` is accessed, it must be evaluated to determine whether the session is new or whether the client has joined the current session. If the session is new and an attempt is made to access the `PipelineSession` from the `HttpSession`, then a null value will be returned unless it is recreated. This section describes in more detail how to handle session timeouts using the `InputProcessor` base classes.

## Using the getPipelineSession() Method

The `CommerceInputProcessor` provides an overloaded `getPipelineSession()` method to help you handle session timeouts.

The first version of the `getPipelineSession()` method attempts to get the `PipelineSession` from the `HttpSession`. If the method is not able to locate the `PipelineSession`, then it will create a new instance and return a reference to the `PipelineSession`, as shown in the following method signature:

```
public PipelineSession getPipelineSession(HttpServletRequest request)
```

The second version of the `getPipelineSession()` method has an extra parameter, `checkValidity`, as shown in the following method signature. If `checkValidity` is `true` and the `HttpSession` is new, then the `getPipelineSession()` method throws an `InvalidSessionStateException` exception.

```
public PipelineSession getPipelineSession(HttpServletRequest
request, Boolean checkValidity) throws InvalidSessionStateException
```

**Note:** For more information about the `InvalidSessionStateException` exception, see "The InvalidSessionStateException Exception in webflow.properties" below.

# The InvalidSessionStateException Exception in webflow.properties

The `InvalidSessionStateException` exception can be used for input processors. In the `webflow.properties` file, you can either provide the input processor name (as shown in the first line of Listing 3-3), or use the wildcard character (as shown in the second line of Listing 3-3).

**Listing 3-3 Using InvalidSessionStateException in webflow.properties**

```
InputprocessorName.inputprocessor.exception
(InvalidSessionStateException)= sessiontimeout.jsp

*.inputprocessor.exception(InvalidSessionStateException)=
sessiontimeout.jsp
```

The second option indicates that all input processors experiencing session timeout (throwing an `InvalidSessionStateException`) should load the `sessiontimeout.jsp` file.

**Note:** For more information about using the wildcard character in the `webflow.properties` file, see "Using the Wildcard Character" on page 2-5.

# PipelineComponent and Session Timeouts

As part of handling session timeouts, each class that implements the `PipelineComponent` interface should determine whether or not a required attribute exists in the `PipelineSession` object. If the attribute does not exist, the subclass should throw an `InvalidPipelineSessionStateException` exception.

**Note:** For more information about the `InvalidPipelineSessionStateException` exception, see "The InvalidPipelineSessionStateException Exception in webflow.properites" below.

# The InvalidPipelineSessionStateException Exception in webflow.properites

The `InvalidPipelineSessionStateException` exception can be used for Pipelines. In the `webflow.properties` file, you can either provide the Pipeline name (as shown in the first line of Listing 3-4), or use the wildcard character (as shown in the second line of Listing 3-4).

**Listing 3-4 Using InvalidPipelineSessionStateException in webflow.properties**

```
PipelineName.pipeline.exception
(InvalidPipelineSessionStateException)= sessiontimeout.jsp

*.pipeline.exception(InvalidPipelineSessionStateException)=
sessiontimeout.jsp
```

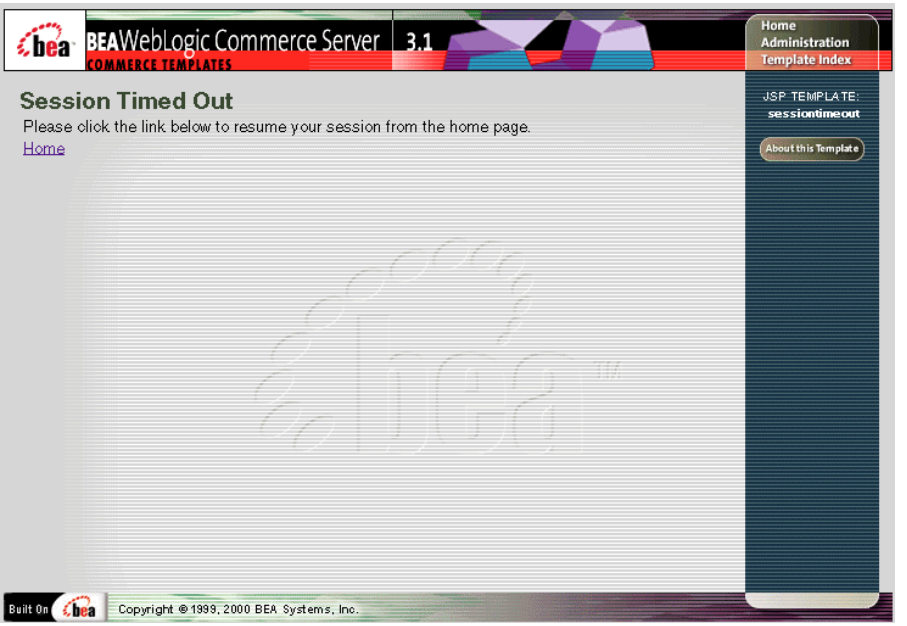
The second option indicates that all Pipelines experiencing session timeout (throwing an `InvalidPipelineSessionStateException`) should load the `sessiontimeout.jsp` file.

**Note:** For more information about using the wildcard character in the `webflow.properties` file, see "Using the Wildcard Character" on page 2-5.

# About the sessiontimeout.jsp Template

The `sessiontimeout.jsp` template (shown in Figure 3-1) that ships with the BEA WebLogic Commerce Server product contains a link that calls the Webflow with the initial state, thereby giving the user a chance to start all over again.

**Figure 3-1 The sessiontimeout.jsp Template**



# 4 Webflow and Pipeline JSP Tags

The BEA WebLogic Commerce Server product provides JSP tags specifically related to the Webflow and Pipeline mechanisms. This topic explains how to import each set of tags into your Web pages, and describes what each of these tags can do.

This topic includes the following sections:

- Webflow JSP Tags
  - getValidatedValue Tag
- Pipeline JSP Tags
  - getPipelineProperty Tag
  - setPipelineProperty Tag

## Webflow JSP Tags

The Webflow JSP tags are utility tags that simplify the implementation of JSPs that utilize the Webflow mechanism. To import the Webflow JSP tags, use the following code:

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

**Note:** For more information about the Webflow mechanism, see "Overview of Webflow and Pipeline Management" on page 1-1.

# getValidatedValue Tag

The `getValidatedValue` tag is used in a JSP to display the fields in a form that a customer must correct. Table 4-1 describes the attributes that can be used with the `getValidatedValues` JSP tag.

**Table 4-1 getValidatedValues Tag Attributes**

| Attribute | Description | Required? |
|---|---|---|
| `fieldName` | The name of the field for which the status is desired. | Yes |
| `fieldValue` | The current value of the field as set by a previous invocation of the page. | No |
| `fieldStatus` | The processing status of the field, which may be unspecified, invalid, or valid. | No |

These fields are determined and marked by an input processor after performing its validation activities. All `InputProcessors` use a `ValidatedValues` object to communicate which fields were successfully processed as well as those that were determined to be invalid.

## About the ValidatedValues Java Class

The `ValidatedValues` class allows a Java/EJB programmer who writes an `InputProcessor` to report the status of processed form fields back to the commerce engineer/JSP content developer.

The constructor for the `ValidatedValues` class takes an `HTTPSession` as a parameter, as shown in the following method signature:

```
public ValidatedValues (javax.servlet.http.HttpSession s)
```

The public methods used to convey the status of the validation through the `getValidatedValue` JSP tag are shown in Table 4-2.

**Table 4-2 ValidatedValues Public Methods**

| Method Signature | Description |
|---|---|
| `public String getStatus (String name)` | Retrieves the status for the specified field, which may be unspecified, invalid, or valid. |
| `public void setStatus (String name, String value)` | Sets the status for the specified field. |
| `public String getValue (String name)` | Retrieves the current value for the specified field. |
| `public void setValue (String name, String value)` | Sets the value for the specified field. |

## Example

Listing 4-1 provides an example of how you might use the `getValidatedValue` tag. When used in a JSP, this sample code will obtain the current value and processing status of the `<field_name>` form field.

**Listing 4-1 Example Usage of the getValidatedValue Tag**

```
<webflow:getValidatedValue fieldName="<field_name>"
fieldValue="<field_value>" fieldStatus="status" />
```

# Pipeline JSP Tags

The Pipeline JSP tags are used to store and retrieve attributes in a Pipeline session. To import the Pipeline JSP tags, use the following code:

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

**Note:** For more information about the Pipeline mechanism, see "Overview of Webflow and Pipeline Management" on page 1-1.

## getPipelineProperty Tag

The `getPipelineProperty` JSP tag retrieves a named attribute (property) from the Pipeline session object, from a property of one of the objects that has been retrieved from the Pipeline session, or from the request. Objects that are stored as attributes of the Pipeline session must conform to the standard bean interface and implement a `get<Attribute>()` method for each publicly accessible attribute.

Table 4-3 describes the attributes that can be used with the `getPipelineProperty` JSP tag.

**Table 4-3 getPipelineProperty Tag Attributes**

| Attribute | Description | Required? |
|---|---|---|
| `attributeScope` | Scope of the attribute to locate. (`PipelineConstants.PIPELINE_SESSION_SCOPE` or `PipelineConstants.REQUEST.SCOPE`) | No |
| `pipelineObject` | Name of the object in the Pipeline session in which to locate the specified attribute. | No |
| `propertyName` | Name of the attribute to locate. If omitted, the Pipeline session itself is returned. | No |
| `returnName` | Name of the variable that will contain the value of the attribute. If omitted, then the tag is treated as inline (that is, the variable is not returned but the `toString()` method is called and the results are displayed to the browser). | No |

**Table 4-3 getPipelineProperty Tag Attributes**

| Attribute | Description | Required? |
|---|---|---|
| `returnType` | A valid type for the returned attribute. | No |

### Example

Listing 4-2 provides an example of how you might use the `getPipelineProperty` tag. When used in a JSP, this sample code will retrieve an attribute named `<property_name>` and store it in a variable named `<return_name>`. The type of this variable will be `<return_type>`, and the scope to which this attribute belongs is specified by `<attribute_scope>`.

**Note:** For more information about the scope of Pipeline session attributes, see "Attribute Scoping" on page 3-3.

**Listing 4-2 Example Usage of the getPipelineProperty Tag**

```
<pipeline:getPipelineProperty propertyName="<property_name>"
returnName="<return_name>" returnType="<return_type>"
attributeScope="<%=<attribute_scope>%>"/>
```

## setPipelineProperty Tag

The `setPipelineProperty` JSP tag sets a named attribute (property) to the Pipeline session object or to a property of one of the objects that has been retrieved from the Pipeline session. Objects that are stored as attributes of the Pipeline session must conform to the standard bean interface and implement a `set<propertyName>()` method for each publicly accessible attribute.

Table 4-4 describes the attributes that can be used with the `setPipelineProperty` JSP tag.

**Table 4-4 setPipelineProperty Tag Attributes**

| Attribute | Description | Required? |
|---|---|---|
| `pipelineObject` | Name of the object in the Pipeline session in which to set the specified attribute. | No |
| `propertyName` | Name of the attribute to set. | Yes |
| `propertyValue` | Value of the attribute to set. | Yes |

If `pipelineObject` is not specified, then the given property and its value will be set to the Pipeline session. If the `pipelineObject` is specified, then the object must implement the `set<PropertyName>()` method, which takes two parameters: a property name (`String`) and a property value (`Object`), as shown in the following method signature:

```
public void set<PropertyName>(String propertyName,java.lang.Object
propertyValue);
```

**Note:** If the `set<PropertyName>()`method is not implemented, an exception will be thrown during the processing of the JSP that has the `setPipelineProperty` tag in it.

## Example

Listing 4-3 provides an example of how you might use the `setPipelineProperty` tag. When used in a JSP, this sample code will set the property named `<property_name>` of the `<pipeline_object_name>` with the value specified in `<property_value>`.

**Listing 4-3 Example Usage of the setPipelineProperty Tag**

```
<pipeline:setPipelineProperty propertyName="<property_name>"
propertyValue="<property_value>"
pipelineObject="<pipeline_object_name>"/>
```

**Note:** The `<pipeline_object_name>` must be a fully qualified class name.

# Index

## D



## E

## F



## G



## H

## I



## J



## L



## M

## N



## O



## P

## R



## S

## T



## U



## V



## W