



BEA WebLogic Portal™

Guide to Managing Presentation and Business Logic: Using Webflow and Pipeline

Version 4.0
Document Date: April 2002

Copyright

Copyright © 2002 BEA Systems, Inc. All Rights Reserved.

Restricted Rights Legend

This software and documentation is subject to and made available only pursuant to the terms of the BEA Systems License Agreement and may be used or copied only in accordance with the terms of that agreement. It is against the law to copy the software except as specifically allowed in the agreement. This document may not, in whole or in part, be copied photocopied, reproduced, translated, or reduced to any electronic medium or machine readable form without prior consent, in writing, from BEA Systems, Inc.

Use, duplication or disclosure by the U.S. Government is subject to restrictions set forth in the BEA Systems License Agreement and in subparagraph (c)(1) of the Commercial Computer Software-Restricted Rights Clause at FAR 52.227-19; subparagraph (c)(1)(ii) of the Rights in Technical Data and Computer Software clause at DFARS 252.227-7013, subparagraph (d) of the Commercial Computer Software--Licensing clause at NASA FAR supplement 16-52.227-86; or their equivalent.

Information in this document is subject to change without notice and does not represent a commitment on the part of BEA Systems. THE SOFTWARE AND DOCUMENTATION ARE PROVIDED "AS IS" WITHOUT WARRANTY OF ANY KIND INCLUDING WITHOUT LIMITATION, ANY WARRANTY OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. FURTHER, BEA Systems DOES NOT WARRANT, GUARANTEE, OR MAKE ANY REPRESENTATIONS REGARDING THE USE, OR THE RESULTS OF THE USE, OF THE SOFTWARE OR WRITTEN MATERIAL IN TERMS OF CORRECTNESS, ACCURACY, RELIABILITY, OR OTHERWISE.

Trademarks or Service Marks

BEA, WebLogic, Tuxedo, and Jolt are registered trademarks of BEA Systems, Inc. How Business Becomes E-Business, Operating System for the Internet, Liquid Data, BEA WebLogic E-Business Platform, BEA Builder, BEA Manager, BEA eLink, BEA WebLogic Commerce Server, BEA WebLogic Personalization Server, BEA WebLogic Process Integrator, BEA WebLogic Collaborate, BEA WebLogic Enterprise, BEA WebLogic Server, BEA WebLogic Integration, E-Business Control Center, BEA Campaign Manager for WebLogic, and Portal FrameWork are trademarks of BEA Systems, Inc.

All other trademarks are the property of their respective companies.

Guide to Managing Presentation and Business Logic: Using Webflow and Pipeline

Document Edition	Date	Software Version
4.0.2	April 2002	BEA WebLogic Portal 4.0

Contents

About This Document

What You Need to Know	xii
e-docs Web Site	xiii
How to Print the Document.....	xiii
Related Information.....	xiii
Contact Us!	xiv
Documentation Conventions	xv

1. Overview of Webflow

Introduction to Webflow	1-2
High-Level Architecture	1-2
Webflow Architecture	1-3
The Internals of the Webflow Mechanism.....	1-4
Understanding Webflow as a State Machine	1-6
Webflow and the MVC Design Pattern	1-7
The Relationship Between Webflows and Applications.....	1-7
Benefits of the Webflow Mechanism.....	1-8
The Webflow Configuration Files.....	1-9
What Files Will I Be Working With?.....	1-9
How Are These Files Different from the Webflow and Pipeline Properties Files?	1-10
Location of Webflow Configuration Files in the Directory Structure	1-10
Creating and Modifying Webflow Files.....	1-12
Who Should Create and Modify the Webflow Files?	1-12
How Do I Create and Modify Webflow Files?	1-12
Getting Started with Webflow.....	1-13
Understanding How Webflow Is Invoked from a URL	1-13

Setting Up Your Web Application's web.xml File	1-15
Specifying Webflow Context Parameters	1-15
Registering the WebflowServlet Servlet	1-17
Defining Your Web Application's Welcome File.....	1-18
Configuring Your Web Application to Use Webflow.....	1-19
Next Steps.....	1-20

2. Webflow Components and Concepts

Introduction to Webflow Components	2-3
Presentation and Processor Nodes	2-4
Input Processors and Pipelines	2-5
The Pipeline Session.....	2-6
Events	2-7
Namespaces	2-8
Special Webflow Components	2-9
The Begin Node.....	2-9
The Root Component Node.....	2-10
The Wildcard Nodes.....	2-10
The Configuration Error Page	2-11
Chaining and Branching with Processor Nodes	2-12
Using Webflow Components in Your Web Pages	2-12
Using Webflow Components with Portals	2-13
Webflow Execution Order.....	2-13
Presentation Nodes	2-14
Processor Nodes	2-15

3. Using the Webflow and Pipeline Editors

Introduction	3-3
Starting the Webflow and Pipeline Editors	3-3
Important Notes About Using the Webflow and Pipeline Editors	3-6
Next Steps.....	3-7
Learning to Use the Webflow and Pipeline Editors	3-8
Webflow and Pipeline Editor Essentials	3-9
Webflow Namespaces.....	3-9
Pipelines Versus Pipeline Namespaces	3-10

Information Displayed in the Editors' Title Bars.....	3-11
Webflow and Pipeline Files in Your Content Management System	3-12
Webflow Component Representations.....	3-13
Understanding the Webflow and Pipeline Editor Palettes	3-17
Tools in the Webflow Editor Palette.....	3-18
Tools in the Pipeline Editor Palette.....	3-19
Understanding the Webflow and Pipeline Editor Toolbars	3-20
Display and Behavior Buttons	3-20
Command Buttons.....	3-21
Organizing Webflow Components in an Editor Canvas	3-23
How to Select Webflow Components	3-23
How to Add Webflow Components.....	3-24
How to Edit a Webflow Component's Name (Label).....	3-24
How to Designate or Remove a Begin (Root) Node.....	3-25
How to Move a Webflow Component	3-25
How to Connect Nodes with Event or Exception Transitions	3-26
How to Reposition Connection Ports on a Node.....	3-27
How to Work with Elbows in Transitions.....	3-27
How to Move an Existing Elbow	3-28
How to Create a New Elbow.....	3-28
How to Delete a Elbow	3-28
Using the Webflow and Pipeline Editor Toolbars.....	3-29
How to Print a Webflow Namespace or Pipeline.....	3-29
How to Delete Webflow Components	3-30
How to Use the Zoomed Overview.....	3-31
How to Show/Hide the Grid.....	3-31
How to Snap Objects to the Grid	3-31
How to Enable and Disable Link Optimization	3-32
How to Show and Hide Exception Transitions	3-32
How to Validate the Selected Node	3-32
How to Validate All Nodes	3-33
How to Set the Configuration Error Page Name.....	3-33
How to Use the Pipeline Component Editor	3-34
How To View Pipeline Component Details.....	3-34
How to Add Pipeline Components.....	3-35

How to Edit Pipeline Components	3-36
How to Delete Pipeline Components	3-36
How to Make the Pipeline Transactional	3-37
How to Include the Pipeline Session in a Transaction	3-37
Using the Properties Editors	3-38
Viewing Component Properties	3-39
Description of Webflow Component Properties	3-39
Modifying Component Property Values	3-42
Migrating An Existing Webflow	3-43
Creating or Modifying a Webflow: Breadth-First Versus Depth-First	3-43
About the Webflow and Pipeline Editors' Validation Features	3-45
Validation Error Messages in a Properties Editor	3-46
What Do the Editors Validate?	3-47
Saving Invalid Webflows	3-47
Synchronizing Webflow Data for Your Application	3-48

4. Customizing and Extending Webflow

Pipeline Session Internals	4-3
Managing the Pipeline Session	4-3
Accessing the PipelineSession Interface	4-3
Setting and Getting Pipeline Session Properties	4-4
Using the Support Classes to Capture Exception Messages	4-6
Property Scoping	4-7
Request-Scoped Pipeline Session Properties	4-7
Pipeline Session-Scoped Pipeline Session Properties	4-8
Serializing Pipeline Session Properties	4-8
Error Handling	4-9
Non-Runtime and Runtime Processor Exceptions	4-9
Input Processor and Pipeline Component Exception Handling	4-11
Input Processor Exceptions	4-11
Pipeline Component Exceptions	4-11
JavaServer Page (JSP) Exception Handling	4-13
Accessing Exceptions and Exception Messages	4-13
Creating New Input Processor or Pipeline Component Exceptions	4-13
Configuring Pipeline Component Exception Fatality	4-14

Creating a New Input Processor	4-15
How to Create a New Input Processor	4-15
Input Processor Naming Conventions.....	4-16
Input Processors and Statelessness.....	4-17
Other Development Guidelines for Input Processors.....	4-17
Webflow Validators and Input Processors	4-17
The ValidatedValues Interface.....	4-18
Validation Example.....	4-20
Special Validation Exceptions	4-24
Creating a Custom Validator.....	4-25
Example of a Custom Validator.....	4-25
Creating a New Pipeline Component	4-28
How to Create a New Pipeline Component	4-28
Pipeline Component Naming Conventions	4-29
Implementation of Pipeline Components as Stateless Session EJBs or Java Objects	4-30
Stateful Versus Stateless Pipeline Components	4-31
Transactional Versus Nontransactional Pipelines	4-31
Including Pipeline Sessions in Transactions	4-32
Other Development Guidelines for Pipeline Components.....	4-32
Extending Webflow by Creating Extension Presentation and Processor Nodes.....	4-33
How to Create an Extension Presentation Node	4-33
How to Create an Extension Processor Node.....	4-34
Making Your Extension Presentation and Processor Nodes Available in the Webflow and Pipeline Editors	4-35
How To Register an Extension Presentation Node.....	4-35
How To Register an Extension Processor Node	4-36
Webflow Internationalization.....	4-38

5. Webflow JSP Tag Library Reference

Importing the Webflow Tag Library	5-2
URL Creation Tags.....	5-2
<webflow:createWebflowURL>.....	5-3
Example	5-5
<webflow:createResourceURL>.....	5-6

Example.....	5-6
Form Tags.....	5-6
<webflow:form>.....	5-7
Example.....	5-9
Validated Form Tags	5-10
<webflow:validatedForm>	5-10
<webflow:text>	5-13
<webflow:password>	5-14
<webflow:radio>	5-15
<webflow:checkbox>	5-15
<webflow:textarea>	5-16
<webflow:select>	5-17
<webflow:option>	5-17
Example.....	5-18
Pipeline Session Tags	5-19
<webflow:setProperty>	5-19
Example.....	5-20
<webflow:getProperty>.....	5-20
Example 1.....	5-21
Example 2.....	5-21
<webflow:setValidatedValue>.....	5-22
Example.....	5-22
<webflow:getValidatedValue>	5-23
Example 1.....	5-24
Example 2.....	5-24
<webflow:getException>	5-25
Example.....	5-25

6. An Example of Webflow: The Pet Flow Application

About the Pet Flow Sample Application	6-2
What Webflow Features Does the Pet Flow Sample Application Illustrate?	6-2
Accessing the Pet Flow Sample Application.....	6-3
Location of Pet Flow Files	6-3
Running Pet Flow in a Web Browser	6-4
Opening a Pet Flow Namespace in the Webflow Editor	6-4

Index



About This Document

This document provides information about the Webflow mechanism included in the BEA WebLogic Portal™ product. The Webflow mechanism externalizes a site's page flow and separates back-end processing activities from presentation. Both the flow of pages and the underlying business processing are configured in centralized XML files, making it easier than ever to maintain or modify the behavior of your Web site. Out-of-the-box, the Webflow mechanism comes with a number of components to get you started, but the Webflow can also be customized or extended to meet your specific objectives.

This document includes the following topics:

- Chapter 1, “Overview of Webflow,” which provides an introduction to the Webflow mechanism and describes its architecture. This topic also introduces you to the Webflow configuration files and describes the set up activities you will need to perform to leverage Webflow in your applications. For those who have used prior implementations of Webflow, this topic also briefly describes the differences in the current release and references documentation that will be helpful for migration.
- Chapter 2, “Webflow Components and Concepts,” contains the basic information you need to know to use Webflow in your Web applications. It describes each of the components that may be used within a Webflow, and indicates how you should go about creating JavaServer Pages (JSPs) that utilize Webflow. This topic also briefly addresses some important Webflow differences for those building portals, and includes helpful information about the Webflow execution order.
- Chapter 3, “Using the Webflow and Pipeline Editors,” provides you with information that will assist you in creating Webflows using the E-Business Control Center (EBCC) graphical Webflow and Pipeline Editors. These Editors are designed specifically to help you create, modify, and validate Webflow and Pipeline XML configuration files.

-
- Chapter 4, “Customizing and Extending Webflow,” explains ways you may want to customize or extend the Webflow mechanism that comes with WebLogic Portal. It includes detailed information about the Pipeline Session, guidelines for creating new exceptions, Input Processors, and Pipeline Components, as well as instructions for creating new presentation and processor nodes for use in your Webflows. This topic also contains information about the Webflow validators as they relate to Input Processors, as well as information about Webflow internationalization messages.
 - Chapter 5, “Webflow JSP Tag Library Reference,” describes a set of JSP tags designed to facilitate the development of JSPs using Webflow. This topic explains how to import the appropriate tag libraries into your Web pages, and describes the purpose of each tag.
 - Chapter 6, “An Example of Webflow: The Pet Flow Application,” provides you with information about the sample Pet Flow application that illustrates some capabilities of the Webflow mechanism.

What You Need to Know

This document is intended for Java/EJB developers who are responsible for modifying the XML files and extending the Webflow mechanism, and HTML/JSP developers, who use JSP tags to create interactive pages that meet business requirements. This document is not directed to business analysts, but naturally a Web site’s flow depends upon various business processes and rules, which need to be taken into account while developers are working with Webflow.

For more information about these roles, see the “Development Roles” section in the “Roadmap for Developing an E-Business Web Site” topic of the *Strategies for Developing E-Business Web Sites* documentation.

e-docs Web Site

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

How to Print the Document

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

A PDF version of this document is available on the WebLogic Portal 4.0 documentation Home page on the e-docs Web site. A PDF version of this document is also available in the documentation kit on the product CD. Or you can download the documentation kit from the WebLogic Portal portion of the BEA Download site. 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 Portal 4.0 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

If you have used prior implementations of Webflow, be sure to see the *Migration Guide* for some important information.

Additionally, the following WebLogic Portal documents describe product applications that are built upon the Webflow infrastructure:

- *Guide to Building a Product Catalog*

-
- *Guide to Managing Purchases and Processing Orders*
 - *Guide to Registering Customers and Managing Customer Services*

Contact Us!

Your feedback on the BEA WebLogic Portal 4.0 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 Portal 4.0 documentation.

In your e-mail message, please indicate that you are using the documentation for the WebLogic Portal 4.0 release.

If you have any questions about this version of BEA WebLogic Portal 4.0, or if you have problems installing and running BEA WebLogic Portal 4.0, 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.
<i>italics</i>	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. <i>Examples:</i> <pre>#include <iostream.h> void main () the pointer psz chmod u+w * \tux\data\ap .doc tux.doc BITMAP float</pre>
monospace boldface text	Identifies significant words in code. <i>Example:</i> <pre>void commit ()</pre>
<i>monospace italic text</i>	Identifies variables in code. <i>Example:</i> <pre>String <i>expr</i></pre>
UPPERCASE TEXT	Indicates device names, environment variables, and logical operators. <i>Examples:</i> <pre>LPT1 SIGNON OR</pre>

Convention	Item
{ }	Indicates a set of choices in a syntax line. The braces themselves should never be typed.
[]	Indicates optional items in a syntax line. The brackets themselves should never be typed. <i>Example:</i> 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: <ul style="list-style-type: none"> ■ 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. <i>Example:</i> 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 Webflow

Webflow is a mechanism designed to help you build Web applications that maintain the much-desired separation between presentation logic and underlying business processes. Because the Webflow's centralized XML configuration files specify the order in which pages are displayed to your Web site's visitors, use of the Webflow mechanism may reduce the amount of work necessary to create and modify the flow of your Web site. At appropriate times during a visitor's interaction, the Webflow may also invoke predefined, specialized components to validate data or to execute back-end business processes. Therefore, using the modular architecture Webflow provides may also make it faster and easier for your development team to complete modifications that require back-end programming.

This topic provides you with preliminary information about the architecture of the Webflow mechanism and describes some benefits of using Webflow in your applications. The relevant Webflow configuration files and their locations are introduced, along with a description of how you might go about modifying these files to meet your organizational requirements. For developers who may have worked with Webflow before, this topic includes a brief explanation of what has changed for this release and indicates where to look for migration information. In addition, this topic includes some instructions to get you started using Webflow.

This topic includes the following sections:

- Introduction to Webflow
 - High-Level Architecture
 - The Relationship Between Webflows and Applications
 - Benefits of the Webflow Mechanism
- The Webflow Configuration Files
 - What Files Will I Be Working With?

- How Are These Files Different from the Webflow and Pipeline Properties Files?
- Location of Webflow Configuration Files in the Directory Structure
- Creating and Modifying Webflow Files
- Getting Started with Webflow
 - Understanding How Webflow Is Invoked from a URL
 - Setting Up Your Web Application's web.xml File
 - Configuring Your Web Application to Use Webflow
- Next Steps

Introduction to Webflow

This section describes the Webflow architecture from a number of different viewpoints, then highlights some important benefits of using Webflow in your applications.

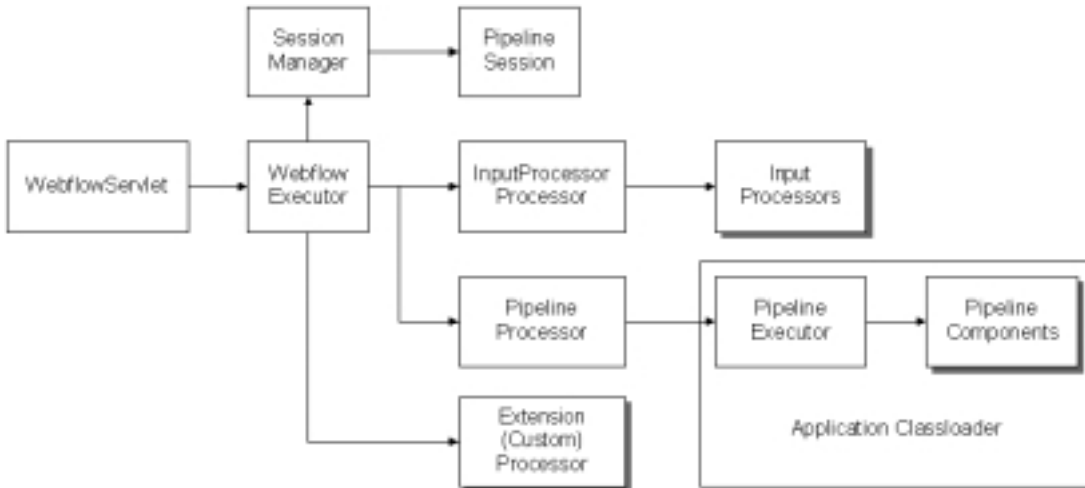
High-Level Architecture

This section provides a high-level description of the Webflow architecture as well as a simplified explanation of how the Webflow mechanism works. It also explains how you might understand Webflow as a state machine and as an enabler of the Model-View-Controller (MVC) design pattern.

Webflow Architecture

Figure 1-1 illustrates the basic architecture of the Webflow mechanism.

Figure 1-1 Webflow High-Level Architecture



The `WebflowServlet` is a servlet that receives all requests from a client and delegates to the `Webflow Executor`. You will register the `WebflowServlet` by defining it in your application's `WEB-INF/web.xml` deployment descriptor prior to using Webflow, as described in “Getting Started with Webflow” on page 1-13.

The `Webflow Executor` determines the Webflow execution order, executes all nodes that handle back-end processing, and then returns the final presentation node to the `WebflowServlet`. The `WebflowServlet` will then forward the client to that presentation node.

Note: More information about the Webflow execution order is included in “Webflow Execution Order” on page 2-13.

The `Session Manager` provides session and lifecycle management for the `Pipeline Session`. For more information about the `Pipeline Session`, see “The Pipeline Session” on page 2-6 and “Pipeline Session Internals” on page 4-3.

The Input Processor processor is a type of processor that delegates to user-defined classes that implement the `InputProcessor` interface. These Input Processor classes perform specific tasks, such as validating form data. Similarly, the Pipeline Processor is a type of processor that delegates to the Pipeline Executor.

The Pipeline Executor is a stateless session Enterprise JavaBean (EJB) that executes Pipeline Components, which execute the business logic associated with the application. The Pipeline Executor also handles transaction management and exception branch management within the Pipeline.

The Extension (Custom) Processor processors shown in Figure 1-1 represent those processors your development team may code to extend the out-of-the-box Webflow mechanism. For more information about creating a custom processor, see “Extending Webflow by Creating Extension Presentation and Processor Nodes” on page 4-33.

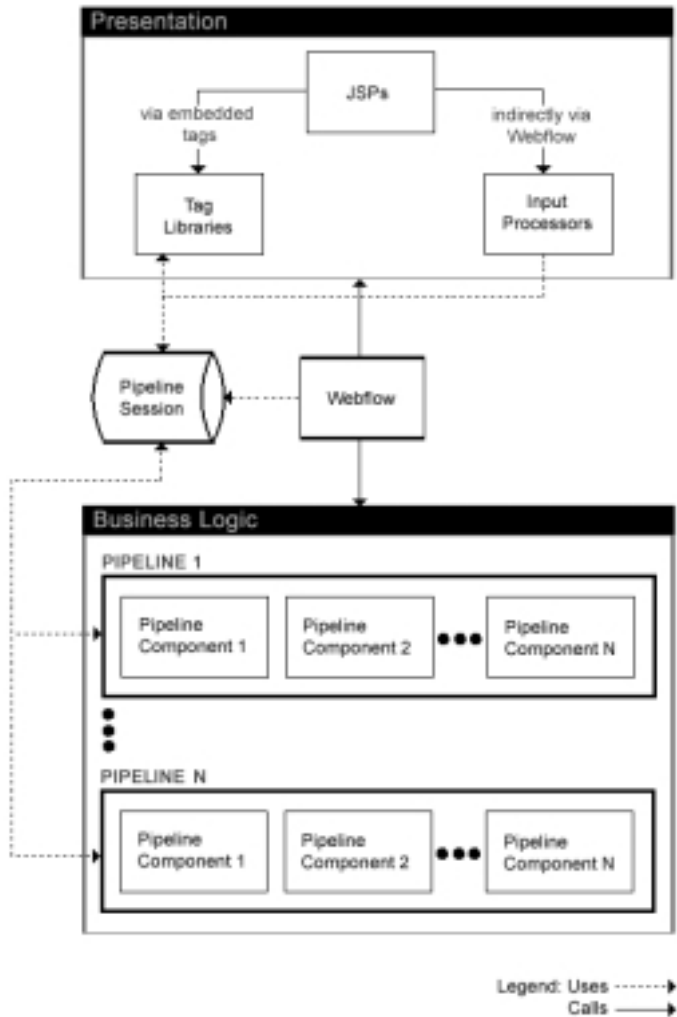
Note: The Webflow architecture used by portals is slightly different from that shown in Figure 1-1. It also includes a `PortalServlet` and Application Processor Executor, which are described in “Customizing Portals” in the *Getting Started with Portals and Portlets* documentation.

The Internals of the Webflow Mechanism

Figure 1-2 provides an example of how control typically flows from presentation and processor nodes (such as JSPs and Pipelines) when applications utilize the Webflow mechanism. Notice that the Pipeline Components containing pieces of business logic have no knowledge of HTML or any other presentation technology. Instead, the Pipeline Session maintains conversational state in the system. Similarly, the Webflow governs the flow of control.

Note: For more information about the Pipeline Session, see “The Pipeline Session” on page 2-6 and “Pipeline Session Internals” on page 4-3.

Figure 1-2 Flow of Control Using Webflow

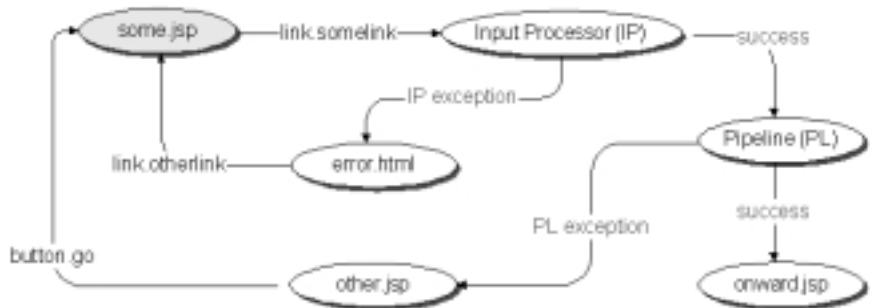


Understanding Webflow as a State Machine

As shown in Figure 1-3, Webflow can be thought of as a state machine. A Webflow begins at a node that handles presentation, such as a JavaServer Page (JSP). When a visitor interacts with a Web site that uses Webflow, their interaction causes information about the origin of the interaction and the event to be sent to the Webflow. Based on this information, the Webflow mechanism makes a decision about the next node it should transition to. If the Webflow decides to invoke a type of processor node (that is, an Input Processor or Pipeline) as a destination, the Webflow may temporarily store state information. This state information is stored only until the Webflow returns to another presentation node.

Using Figure 1-3 as an example, a visitor may be viewing a page called `some.jsp` that exists in your Web application. The visitor then clicks a link called `someLink`, which is available to them on `some.jsp`. In this case, the origin of the interaction is the current page (`some.jsp`) and the event is clicking on the link named `someLink`. Based on this information, the Webflow may decide to transition to an Input Processor. When invoked by the Webflow, this Input Processor may access and temporarily store information in a centralized location called a Pipeline Session. If the Input Processor throws an exception and transitions to the `error.html` page, that state information is no longer available.

Figure 1-3 Webflow as a State Machine



Notes: For more information about the different kinds of Webflow components (presentation and processor nodes, events, and so on), see Chapter 2, “Webflow Components and Concepts.” For more information about the Pipeline Session, see “The Pipeline Session” on page 2-6 and “Pipeline Session Internals” on page 4-3.

Webflow and the MVC Design Pattern

Webflow also enables the Model-View-Controller (MVC) design pattern, a useful methodology for developers looking to reuse Java code and decrease the amount of time required to develop Web applications. Processor nodes (such as Input Processors and Pipelines) represent the Model, or classes dealing with the underlying data structures necessary for a fully functional application. Presentation nodes (such as JavaServer Pages) represent the View, or classes that are concerned with the application’s interface. The Controller, represented as the `WebflowServlet` servlet, facilitates communication between the Model and the View.

Notes: For more information about presentation and processor nodes, see “Presentation and Processor Nodes” on page 2-4. For more information about the `WebflowServlet` servlet, see “Setting Up Your Web Application’s `web.xml` File” on page 1-15.

The Relationship Between Webflows and Applications

As in the previous release, Webflow is scoped to a Web application, and you can only have one Webflow per Web application. The Webflow for a Web application is configured in the application’s `WEB-INF/web.xml` deployment descriptor, as described in “Setting Up Your Web Application’s `web.xml` File” on page 1-15.

However, this release of Webflow does provide you with more flexibility. Much like you can have multiple enterprise applications per instance of the WebLogic Server, you can now have multiple Webflows per enterprise application. Pipelines are scoped to an enterprise application, meaning that multiple Webflows (or Web applications) may all invoke the same Pipeline. In addition, you can now divide an individual Web application’s Webflow into more manageable subflows, called namespaces. For more information about namespaces, see “Namespaces” on page 2-8.

Benefits of the Webflow Mechanism

Based on the information you have about Webflow, you may already see that using Webflow in your Web applications will be advantageous to your development processes. The following are just some of the benefits you will realize while using the Webflow mechanism:

- *Decoupled code*: The code structures required for most Web applications, such as those for presentation, form validation, and business logic, are all located in separate components and coordinated via centralized XML files.
- *Reusable code*: Because Web pages are not muddled with code that serves a number of different purposes and touches a variety of external systems, the components that handle presentation, form validation, and business logic can be reused in a variety of application contexts. These atomic components can be “plugged into” new contexts with little modification.
- *Maintainable code*: For the same reason why the components are reusable, the code for the entire Web application is also more easily maintained. It is easier to determine what is happening in each focused component, instead of following the logical (or perhaps illogical) structure that results from placing all the necessary code within a single page. Additionally, since components are configured in a centralized document, you can make modifications without having to touch a large number of files.
- *Increased developer productivity*: The decoupling of code structures may also result in increased developer productivity. Although each component has a specific function within the larger application, each component also exists as a separate entity. Developers can work independently on portions of the application without being overly concerned about how their modifications will affect other areas.

The Webflow Configuration Files

This section introduces the Webflow configuration files you will be working with for your applications, and includes a brief description of the information they contain. For those who have utilized previous releases of Webflow, this section also describes some important changes. The locations of the Webflow configuration files are provided, and methods for modifying these files are discussed.

What Files Will I Be Working With?

Recall that there is a one-to-one association between a Webflow and a Web application. However, because a Webflow contains configuration information for an entire Web application and because the main goal of Webflow is to preserve the separation between presentation and business logic, there are typically several configuration files per Webflow. All Webflow configuration files contain XML elements, which must conform to a XML Schema Definition (XSD) included in the product.

The primary files used for Webflow are `<namespace>.wf` and `<namespace>.pln` files. As suggested by the `<namespace>` portion of the filename, a Webflow is typically modularized into several namespaces. The `<namespace>.wf` and `<namespace>.pln` files are scoped to a Web application and enterprise application, respectively. The minimum number of Webflow files you can have for a Web application is one, but there is no limit to the maximum number of Webflow configuration files.

Notes: For more information the association between Webflows and applications, see “The Relationship Between Webflows and Applications” on page 1-7. For more information about using namespaces within a Webflow, see “Namespaces” on page 2-8.

As indicated by their extensions, a `<namespace>.wf` configuration file contains the primary information for a Webflow namespace, including the transitions between presentation and processor nodes. A `<namespace>.pln` file contains definitions for the Pipeline Components that may be invoked within a Pipeline in the namespace. Although some Webflow components are defined within different files (according to `<namespace>`), the flow of control passes seamlessly between them.

How Are These Files Different from the Webflow and Pipeline Properties Files?

If you have used previous releases of Webflow, the first change you will notice is that the `webflow.properties` and `pipeline.properties` files have been replaced with one or more `<namespace>.wf` and `<namespace>.pln` files. This new implementation uses the data synchronization model to make new or modified Webflows available to one or more applications, and to work with your organization's existing Content Management Systems. Instead of containing name-value pairs, the configuration files now contain XML elements, which must conform to a XML Schema Definition (XSD) included in the product.

You need to migrate your files to the new format in order to use the Webflow and Pipeline Editors included in the E-Business Control Center (EBCC). For more information about migrating your existing Webflows, see “Migrating An Existing Webflow” on page 3-43, or view the same topic from within the EBCC's online help. You should also refer to the *Migration Guide*.

Lastly, it is important that you continue reading this documentation to learn about other significant changes and improvements in the Webflow implementation.

Location of Webflow Configuration Files in the Directory Structure

Figure 1-4 illustrates where in the directory structure the Webflow and Pipeline configuration files reside. Note that there may be multiple `<namespace>.pln` files per enterprise application, but that `<namespace>.wf` files (and the `webflow-extensions.wfx` file used to register extension (custom) Webflow components for your Web application) exist in each individual Web application's directory.

Note: For more information about managing the `<namespace>.wf`, `<namespace>.pln`, and `webflow-extensions.wfx` files, see “Webflow and Pipeline Files in Your Content Management System” on page 3-12.

Figure 1-4 Webflow Configuration Files in a Sample Directory Structure



When you create a new Webflow or Pipeline for your application using the Webflow or Pipeline Editors, one or more `<namespace>.wf` and `<namespace>.pln` Webflow configuration files are automatically generated and appropriately placed in the directory structure. (For information on using the Webflow and Pipeline Editors, see Chapter 3, “Using the Webflow and Pipeline Editors.”)

The `webflow-extensions.wfx` file initially exists in the `sample_app` enterprise application’s `default/webflow` directory. When you create a new Web application using the E-Business Control Center (EBCC), this `webflow-extensions.wfx` file is copied over to your Web application’s directory. You can hand-edit any `webflow-extensions.wfx` file to include new Webflow components, but modifications made to the `webflow-extensions.wfx` file in the `sample_app` enterprise application’s `default/webflow` directory will affect all subsequent Web applications you create.

Notes: For more information about how the `webflow-extensions.wfx` file is copied when you create a new Web application, see “Creating an Application Structure for E-Business Control Center Data” and “Creating a Web Application Folder” in the *Guide to Using the E-Business Control Center* documentation.

Instructions for modifying the `webflow-extensions.wfx` file to register extension (custom) Webflow nodes are located in “Making Your Extension Presentation and Processor Nodes Available in the Webflow and Pipeline Editors” on page 4-35.

Creating and Modifying Webflow Files

This section includes important information about who should create and modify Webflow files for your organization’s applications, and explains the method for performing these tasks.

Who Should Create and Modify the Webflow Files?

It is expected that HTML/JSP developers or Java/EJB developers will be responsible for modifying the Webflow configuration files, and that only Java/EJB developers will customize or extend the Webflow mechanism. For more information about these roles, see the “Development Roles” section in the “Documentation Roadmap for WebLogic Portal” topic of the *Strategies for Developing E-Business Web Sites* documentation.

How Do I Create and Modify Webflow Files?

BEA strongly recommends that you utilize the Webflow and Pipeline Editors, accessible from the E-Business Control Center (EBCC), to edit your Webflow and Pipeline files. These graphical tools provide you with the ability to visualize your Webflows in smaller modules, and the Webflow Editor contains a built-in validation feature that works behind the scenes to ensure that your Webflows are valid. For information about using the Webflow and Pipeline Editors, see Chapter 3, “Using the Webflow and Pipeline Editors.” Or, select the “Using the Webflow and Pipeline Editors” topic from the EBCC online help.

Note: If you have used Webflow and Pipeline Editors from a previous release, you first need to migrate your Webflow and Pipeline files. For more information see “How Are These Files Different from the Webflow and Pipeline Properties Files?” on page 1-10 and the *Migration Guide*.

Getting Started with Webflow

In order to take advantage of Webflow in your applications, there are a few activities you must perform. This section provides you with the step-by-step instructions that will help you get started with Webflow. Namely:

- Setting Up Your Web Application’s web.xml File
- Configuring Your Web Application to Use Webflow

Note: Some of these instructions involve working with your Web application’s web.xml deployment descriptor. More information about this file can be found in the *Deployment Guide*.

Before you begin, however, you should have a high-level understanding of how the URL a visitor to your Web site enters in a browser eventually invokes the Webflow mechanism.

Understanding How Webflow Is Invoked from a URL

This section explains what typically happens when a visitor to your Webflow-enhanced Web site enters the URL `http://localhost:7001/mywebapp` in their Web browser.

In the preceding URL, `localhost` is a reference to the domain name of the server. The listen port of `7001` is defined in the `<domain_name>/config.xml` file. This can be any available port, and may be omitted if the application is running on port `80`. If you are using a proxy to front-end your Webflow, you may also need to set the `HTTP_PORT` and `HTTPS_PORT` context parameters in the web.xml file to the ports on the proxy. Lastly, `mywebapp` is the URI for the Web application, defined in the

META-INF/application.xml file. This can be omitted if the Web application is designated as the default Web application. More information about these settings can be found in the *Deployment Guide*.

Given this information, the Webflow sequence is as follows:

1. The browser forwards the request to the WebLogic Server running on localhost and port 7001.
2. Since the URI for the Web application is specified, the WebLogic Server directs the request to the specific Web application that corresponds to the mywebapp URI.
3. Since no additional path information is supplied on the URL, the Web application forwards the visitor to the first welcome file, as defined in the <welcome-file-list> element of the WEB-INF/web.xml file. Typically, the welcome file is something like index.jsp.
4. The only code contained within the index.jsp welcome file is <jsp:forward page="/application"/>. This forwards the request to the WebflowServlet, as defined in the web.xml file, with the URL pattern of application.

Since a namespace is not supplied via a query string, the WebflowServlet looks for the context parameter P13N_DEFAULT_NAMESPACE in the WEB-INF/web.xml file. This is the namespace the WebflowServlet will use. Typically, the default namespace is defined as main.

Note: Specifying a namespace with a query string would look like:
<jsp:forward page="/application?namespace=order"/>. If a namespace is specified, it will be used instead of the default namespace.

5. Since there is a one-to-one relationship between a namespace and a Webflow configuration file, the WebflowServlet looks for a main.wf file located in the <data-sync-directory>/webapps/mywebapp/ directory.
6. Since no origin and event are supplied as parameters, the Webflow engine will default to the begin origin. The begin origin is defined in the default namespace file (main.wf) as follows:

```
<begin-origin>  
  
<destination namespace="main" node-name="welcome"  
node-type="jsp"/>  
  
</begin-origin>
```

7. The begin origin's destination is a node named `welcome`, and is a JavaServer Page (JSP). Therefore, Webflow looks in the `main.wf` file for a node with the name of `welcome` and a type of `jsp`, which would be represented as:

```
<presentation-origin node-name="welcome" node-type="jsp">
<node-processor-info page-relative-path="/browse"
page-name="welcome.jsp"/>
</presentation-origin>
```

The `welcome` node has a page relative path of `/browse` and a page name of `welcome.jsp`. Therefore, Webflow forwards the visitor's request to `/browse/welcome.jsp`. This file needs to be located in the Web application's document root.

Setting Up Your Web Application's `web.xml` File

In order to set up the Webflow mechanism for use in your Web applications, you will first need to specify certain context parameters, register the `WebflowServlet` servlet, and define your welcome file in your application's `WEB-INF/web.xml` deployment descriptor file. This section describes the set up activities you will need to complete in the `web.xml` file.

Specifying Webflow Context Parameters

To make use of Webflow, you may need to specify values for five context parameters in the application's `web.xml` deployment descriptor file. These context parameters are:

- `P13N_DEFAULT_NAMESPACE`

The default namespace for the application, which Webflow will use when a namespace is not specified in the URL. This context parameter is required.

Note: Portals must have the `P13N_DEFAULT_NAMESPACE` context parameter set in a specific manner. Please see "Customizing Portals" in the *Getting Started with Portals and Portlets* documentation for more information.

- `P13N_APPLICATION_URL`

The application URL for the Webflow. This context parameter is required, and if not specified, will default to `/application`.

- P13N_URL_PREFIX

The prefix inserted into the URL by the `createWebflowURL()` method or JSP tags. (For more information about the `createWebflowURL()` method, see “Using Webflow Components in Your Web Pages” on page 2-12, or the *Javadoc*.) If not specified, a prefix is not added to the URL. You may want to specify a prefix when Apache or some other proxy is fronting Webflow. This context parameter is optional.

- P13N_STATIC_ROOT

The URL root used by the `createStaticResourceURL()` method to build URLs to static resources that exist on a proxy server. The `createStaticResourceURL()` method is an alternative to using the `<webflow:createResourceURL>` JSP tag, and is described in the *Javadoc*.

This context parameter is optional. If the `P13N_STATIC_ROOT` context parameter is commented out or left null, then the `<webflow:createResourceURL>` tag and the `createWebflowURL()` method will use the current domain name and port. If at a later time you want to house your GIF images to another server, just edit the `P13N_STATIC_ROOT` context parameter.

- P13N_URL_DOMAIN

If this context parameter is set, the `<webflow:createWebflowURL>`, `<webflow:form>`, and `<webflow:validatedForm>` JSP tags will use this domain (IP address) as opposed to the current domain name. You may want to do this when Webflow is being fronted by a proxy and that proxy exists on another machine. You can also specify a `domainName` property in the above named JSP tags to override the value of the `P13N_URL_DOMAIN` context parameter on a link-by-link basis. This context parameter is optional.

All of these context parameters are defined within `<context-param>` elements in your application’s `WEB-INF/web.xml` file, as shown in Listing 1-1.

Listing 1-1 Sample Context Parameters for Webflow in web.xml

```
<context-param>
  <param-name>P13N_DEFAULT_NAMESPACE</param-name>
  <param-value>main</param-value>
</context-param>
```



```
<context-param>
  <param-name>P13N_APPLICATION_URL</param-name>
  <param-value>/application</param-value>
</context-param>

<context-param>
  <param-name>P13N_URL_PREFIX</param-name>
  <param-value>/weblogic</param-value>
</context-param>

<context-param>
  <param-name>P13N_STATIC_ROOT</param-name>
  <param-value>http://localhost:7003/petflow</param-value>
</context-param>

<context-param>
  <param-name>P13N_URL_DOMAIN</param-name>
  <param-value>192.168.32.19</param-value>
</context-param>
```

Registering the WebflowServlet Servlet

Listing 1-2 illustrates the elements you will need to add to your application's WEB-INF/web.xml file to register the `webflowServlet` servlet. The `<servlet>` element defines the servlet's name and indicates its class name. The `<servlet-mapping>` element associates the `WebflowServlet` servlet with a URL pattern of `/application/*`.

Note: The `P13N_APPLICATION_URL` context parameter must also be set to the value shown in the `<url-pattern>` element to enable automatic URL generation in JavaServer Pages (JSPs). For more information, see “Specifying Webflow Context Parameters” on page 1-15.

Listing 1-2 Registering the Webflow Servlet in web.xml

```
<!-- Defines the Webflow servlet -->

<servlet>
  <servlet-name>webflow</servlet-name>
  <servlet-class>com.bea.p13n.appflow.webflow.servlets.internal.
    WebflowServlet</servlet-class>
</servlet>
```

```
<servlet-mapping>
  <servlet-name>webflow</servlet-name>
  <url-pattern>/application/*</url-pattern>
</servlet-mapping>
```

Note: Portal applications must also register the `PortalServlet` in the `WEB-INF/web.xml` file. Please see “Customizing Portals” in the *Getting Started with Portals and Portlets* documentation for more information.

Defining Your Web Application’s Welcome File

After you have registered the `WebflowServlet` servlet, you need to verify that you have defined a welcome file. Per the Java Servlet 2.2 specification, the welcome file is the first file displayed when a Web application is invoked. Listing 1-3 illustrates how to define a welcome file in your `web.xml` file.

Listing 1-3 Defining the Welcome File in `web.xml`

```
<welcome-file-list>
  <welcome-file>index.jsp</welcome-file>
</welcome-file-list>
```

The welcome file you list in the `web.xml` deployment descriptor must reside in the root directory of your Web application. This welcome file is the entry point from the Web application into the Webflow mechanism, and should contain the following code:

```
<jsp:forward page="/application?namespace=namespace_name">
```

where *application* is the name of your application (which matches that specified in the `P13N_APPLICATION_URL` context parameter) and *namespace_name* is the name of the initial namespace for your Web application. Specifying a namespace name is optional. If the namespace name is omitted, the namespace used will be that specified in the `P13N_DEFAULT_NAMESPACE` context parameter.

Note: For more information about the `P13N_APPLICATION_URL` and `P13N_DEFAULT_NAMESPACE` context parameters, see “Specifying Webflow Context Parameters” on page 1-15.

Configuring Your Web Application to Use Webflow

The welcome file specified in your WEB-INF/web.xml file (in this case, index.jsp) forwards Web site visitors to another page. Therefore, you need to specify the page visitors get forwarded to by creating a <namespace>.wf file (for this example, main.wf) in the E-Business Control Center (EBCC) Webflow Editor.

Note: For more information about creating a new <namespace>.wf file via the Webflow Editor, see “Creating Webflow/Pipeline Files” in the *Guide to Using the E-Business Control Center* documentation.

The main.wf file should contain a begin origin Webflow component, followed by a presentation node corresponding to the page specified by the begin origin. For this example, assume that the page has a name of helloworld and a node type of jsp. The XML code generated by the Webflow Editor when you save the main.wf file will be similar to that shown in Listing 1-4.

Listing 1-4 Example main.wf File

```
<?xml version="1.0" encoding="UTF-8"?>

<webflow-configuration
  xmlns="http://www.bea.com/servers/p13n/xsd/webflow/2.0"
  xmlns:xsi="http://www.w3.org/2000/10/XMLSchema-instance"
  xsi:schemaLocation="http://www.bea.com/servers/p13n/xsd/webflow/
  2.0 webflow.xsd">

  <namespace>main</namespace>

  <begin-origin>
    <destination namespace="main" node-name="helloworld"
      node-type="jsp"/>
  </begin-origin>

  <presentation-origin node-name="helloworld" node-type="jsp">
    <node-processor-info page-name="helloworld.jsp"/>
  </presentation-origin>

  ...

</webflow-configuration>
```

Note: For instructions on creating Webflow components, see Chapter 3, “Using the Webflow and Pipeline Editors.”

Next Steps

Subsequent topics in this *Guide to Managing Presentation and Business Logic: Using Webflow and Pipeline* documentation provide detailed information about Webflow. Depending on your goals, you may want to review the following:

- If you are entirely new to Webflow, continue to Chapter 2, “Webflow Components and Concepts.” This topic provides you with the basic information you need to know to use Webflow.
- If you would like to learn how to create and modify Webflow configuration files using the Webflow Editor, see Chapter 3, “Using the Webflow and Pipeline Editors,” or select this topic in the E-Business Control Center (EBCC) online help.
- If you are a Java/EJB developer and would like to learn about how to create new exceptions, Input Processors, Pipeline Components, and Extension (Custom) Presentation and Processor Nodes, see Chapter 4, “Customizing and Extending Webflow.”
- If you are an HTML/JSP developer and are interested in using Webflow JSP tags in your Web pages, see Chapter 5, “Webflow JSP Tag Library Reference.”
- If you would like to view a sample application that illustrates some capabilities of the Webflow mechanism, see Chapter 6, “An Example of Webflow: The Pet Flow Application.”

There are also several documents that address topics related to Webflow that may be of interest, including:

- *Strategies for Developing E-Business Web Sites*
- *Migration Guide*
- *Deployment Guide*
- *Performance Tuning Guide*

2 Webflow Components and Concepts

As described in “Overview of Webflow,” the Webflow mechanism controls the presentation of Web pages as visitors interact with your Web applications. To accomplish this, Webflow makes use of various components, some of which are designed to handle complex tasks like form validation or execution of back-end business processes. This topic first introduces you to all the components that may comprise a Webflow, and includes:

- Introduction to Webflow Components
- Presentation and Processor Nodes
- Input Processors and Pipelines
- The Pipeline Session
- Events
- Namespaces
- Special Webflow Components
 - The Begin Node
 - The Root Component Node
 - The Wildcard Nodes
 - The Configuration Error Page

Next, this topic discusses some additional concepts with respect to using the Webflow mechanism, including:

- Chaining and Branching with Processor Nodes

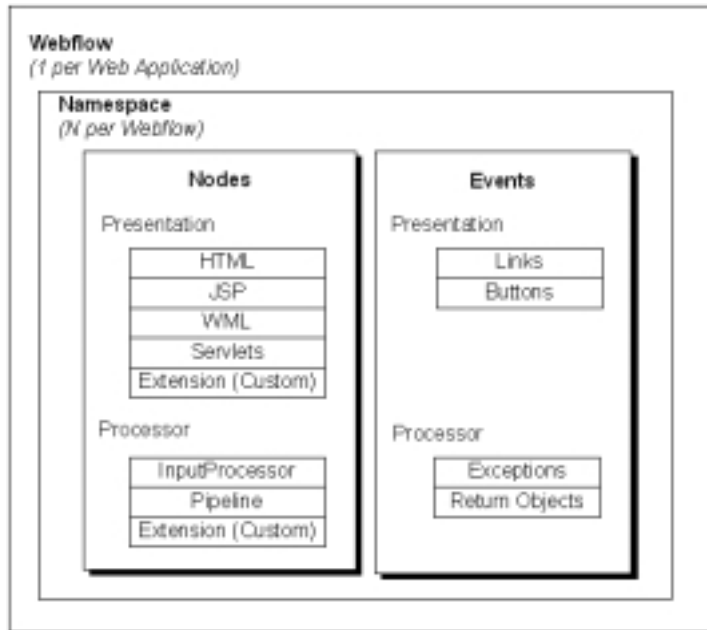
- Using Webflow Components in Your Web Pages
- Using Webflow Components with Portals
- Webflow Execution Order
 - Presentation Nodes
 - Processor Nodes

Notes: For information about using the Webflow and Pipeline Editors to work with the components described in this topic, see “Using the Webflow and Pipeline Editors.” For information about customizing and extending the Webflow mechanism, see “Customizing and Extending Webflow.”

Introduction to Webflow Components

Figure 2-1 illustrates the typical relationships among Webflow components. You can refer back to this illustration as you read more about each component.

Figure 2-1 Typical Relationship Among Webflow Components



Note: Webflow used in portal applications may respond to more events than are shown in Figure 2-1. For more information about portal applications and use of the Webflow mechanism, see “Customizing Portlets and Portals” in the *Getting Started with Portals and Portlets* documentation.

A Webflow can first be subdivided into nodes and events. A **node** represents a state in the Webflow. Depending on the node type, there are a number of predefined **events** that may occur (such as a visitor clicking a link on a Web page). When a particular event happens, the Webflow decides which subsequent node to invoke to continue the flow. This process is referred to as a **transition**, and is illustrated in Figure 2-2.

Figure 2-2 Generic Webflow Transition



As is also shown in Figure 2-2, nodes may be referred to as origin or destination nodes, depending on their location in a transition.

For more information about node types, see the next section, “Presentation and Processor Nodes.” For more information about the events associated with each node, see “Events” on page 2-7.

Presentation and Processor Nodes

There are two main types of nodes: presentation nodes and processor nodes. Each of the presentation and processor nodes can be used as origin or destination nodes within the Webflow.

As their name implies, **presentation nodes** represent states in which the Webflow presents or displays something to a person interacting with the Web application. A Webflow must always start and end with a presentation node. The form of the presentation can be:

- HTML
- JavaServer Page (JSP)
- Java servlets

You can also create extension (custom) presentation nodes for use in the Webflow. For more information about extension presentation nodes, see “How to Create an Extension Presentation Node.”

In contrast to presentation nodes, **processor nodes** represent states in which the Webflow invokes more specialized components to handle activities like form validation, or back-end business logic that drives the site’s presentation. The processor nodes available for use are:

- Input Processors

- Pipelines

“Input Processors and Pipelines” on page 2-5 discusses these processor nodes in more detail.

You can also create extension (custom) processor nodes for use in the Webflow. For more information about extension processor nodes, see “How to Create an Extension Processor Node.”

Note: For more information about creating a presentation or processor node in the Webflow Editor, see “How to Add Webflow Components.”

Input Processors and Pipelines

Input Processors and Pipelines are the two different types of processor nodes that come packaged with the Webflow implementation.

Input Processors are predefined, specialized Java classes that carry out more complex tasks when invoked by the Webflow mechanism. Input Processors are typically used to validate HTML form data, or to provide conditional branching within a Web page. For example, an Input Processor may contain code that verifies whether a date has been entered in the correct format, as opposed to embedding that code within the same JSP that displays the form fields. Input Processors contain logic that is specific to the Web application, and are therefore loaded by the Web application’s container.

Input Processors are embedded in Web pages using specialized JSP tags, which are discussed in “Webflow JSP Tag Library Reference.” Java/EJB developers report the status of processed form fields back to HTML/JSP developers via the `ValidatedValues` class, which is discussed in “Webflow Validators and Input Processors.”

A Pipeline is also a type of processor node that may be invoked by the Webflow. Pipelines initiate the execution of specific tasks related to your business process, and can be transactional or nontransactional. For example, if a visitor attempts to move to another page on your Web site but you want to persist the visitor’s information to a database first, you could use a Pipeline. Pipelines contain business logic that may apply to multiple Web applications within a larger enterprise application, and are therefore loaded by the Enterprise JavaBean (EJB) container.

Note: For more information about transactional versus nontransactional Pipelines, see “Transactional Versus Nontransactional Pipelines.”

All Pipelines are collections of individual Pipeline Components, which can be implemented as Java objects or stateless session Enterprise JavaBeans (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).

Notes: For an explanation of the different Pipeline Component implementations, see “Implementation of Pipeline Components as Stateless Session EJBs or Java Objects.”

For more information about creating an Input Processor or Pipeline in the Webflow Editor, see “How to Add Webflow Components.”

The Pipeline Session

It is often necessary to keep track of information gathered from your Web site visitors, or to share the data modified by Pipeline Components and Input Processors as a visitor moves through the site. You may also want to access data that is part of a larger enterprise application, or make a process transactional. To accomplish these tasks, the Webflow mechanism makes use of a Pipeline Session.

A Pipeline Session is an object that is created and stored within the `HttpServletRequest` object. The Pipeline Session provides a single point of communication for all Pipeline Components in a given Pipeline. Input Processors also read data from the `HttpServletRequest` and then use that data to create or update Java objects in the Pipeline Session. The Pipeline Session also provides central access and storage for external classes and also has transactional capabilities. For more information about including the Pipeline Session in transactions, see “Including Pipeline Sessions in Transactions.”

Note: If you have used a prior implementation of Webflow, you may recall that the Pipeline Session used to be stored in the `HTTPSession` rather than in the `HttpServletRequest`. This has been changed to enhance performance and facilitate clustering.

The Pipeline Session is comprised of many name/value pairs called attributes or **properties**. Pipeline Components, Input Processors, and external classes act on particular properties that exist within the Pipeline Session, and may also add new properties as necessary. All objects set in and retrieved from the Pipeline Session should be serializable. For more information about serializing objects in the Pipeline Session, see “Serializing Pipeline Session Properties.”

To get or set a Pipeline Session property within a Web page, you will use the Pipeline Session JSP tags, described in “Webflow JSP Tag Library Reference.” To work with the Pipeline Session from within Input Processor or Pipeline classes, you will use the provided support methods to access the Pipeline Session directly. For more detailed information about accessing the Pipeline Session using the support methods, see “Pipeline Session Internals.”

Events

Each node in a Webflow responds to events, which cause transitions (that is, movement from an origin node to a destination node). However, the types of events a node responds to depends on whether the node is a presentation node or a processor node.

As shown in Figure 2-1, presentation nodes respond to the following events:

- Links
- Buttons

In other words, when a visitor to the Web site clicks a link or a button, the Webflow responds to that event. A response might be to transition to another presentation node (such as a JSP) or to a processor node (such as an Input Processor to validate visitor-provided form data).

In contrast, processor nodes respond to the following events:

- Exceptions
- Return objects

Exceptions occur when an Input Processor or Pipeline does not execute properly, and indicates an error state. (More information about working with exceptions can be found in “Error Handling.”) Otherwise, these processor nodes return an object that the Webflow can use to continue.

Notes: Webflow used in portal applications may respond to more events than those described above. For more information about portal applications and use of the Webflow mechanism, see “Customizing Portlets and Portals” in the *Getting Started with Portals and Portlets* documentation.

For more information about connecting presentation and processor nodes with event and/or exception transitions in the Webflow Editor, see “How to Connect Nodes with Event or Exception Transitions.”

Namespaces

Although you will generally have only one Webflow per Web application, **namespaces** allow you to divide your Webflow into a number of smaller, more manageable modules. This modularity may make your development team more productive by allowing individual developers to simultaneously work with various portions of a Web application, without having to worry about naming collisions. For example, a Pipeline Component defined in one namespace can access a variable defined in another namespace, then redirect to a JSP defined in yet a third namespace.

You can have any number of namespaces within a Webflow, but namespaces can only be one level deep. In other words, you can not nest namespaces.

Note: Namespaces are best defined along functional lines, such as order management, user management, browsing, and so on. This arrangement enables developers to more easily collaborate, once separate portions of the site require integration. You would not typically define namespaces along horizontal lines (that is, Input Processors, JSPs, Pipeline Components, and so on) because this would make development roles/processes too restrictive.

When a Web application’s Webflow is divided into namespaces, the Pipeline Session is also subdivided accordingly. Therefore, a Pipeline Session property set in one namespace can be obtained from another namespace, and vice versa. For more detailed information about the Pipeline Session, see “Pipeline Session Internals.”

Note: You must specify namespaces in a specific way when using Webflow in portal application. For more information about portal applications and use of the Webflow mechanism, see “Customizing Portlets and Portals” in the *Getting Started with Portals and Portlets* documentation.

Special Webflow Components

In addition to the Webflow components previously described, there are four Webflow components that perform special functions in a Webflow. These components are described in detail in this section.

The Begin Node

Typically, a begin node is a node designated as the initial entry point or state of the Webflow, which automatically transitions to a presentation or processor node. The begin node is generally a presentation node in the form of a JavaServer Page (JSP).

If a URL does not specify an origin, namespace, or event, the Webflow mechanism looks for a begin node in the default namespace. If one is located, the begin node is used as the starting point for the visitor’s interaction with the application. Therefore, although the begin node is optional, we recommend that you have at least one defined in your default namespace.

Note: You specify a default namespace in the `P13N_DEFAULT_NAMESPACE` context parameter of your Web application’s `WEB-INF/web.xml` file. For more information about this context parameter, see “Setting Up Your Web Application’s web.xml File.” For more information about how an application’s Webflow gets invoked from a URL, see “Understanding How Webflow Is Invoked from a URL.”

You can have more than one begin node in a Webflow (one for each namespace). If a namespace specified in the URL has its own begin node, the Webflow will use that begin node before looking for a begin node in the default namespace. You can also use a begin node as a destination node in a Webflow transition.

Note: For more information about designating an existing node as a begin node in the Webflow Editor, see “How to Designate or Remove a Begin (Root) Node.”

The Root Component Node

A root component node is analogous to the begin node for a Webflow, only it is the entry point into or initial state of a Pipeline. You must have one (and only one) root component node per Pipeline.

Note: For more information about designating an existing node as a root component node in the Webflow Editor, see “How to Designate or Remove a Begin (Root) Node.”

The Wildcard Nodes

If the Webflow cannot locate a specific presentation or processor node to complete a transition, the Webflow will search for a wildcard presentation or processor node to use as the origin node. Therefore, wildcard presentation nodes and wildcard processor nodes implement default behavior for your Web application. Put another way, wildcard nodes allow you to abstract common functionality and to locate that functionality in a single place in your Webflow. Wildcard nodes are used only when destination nodes are not explicitly defined in the Webflow. You may have one wildcard presentation node and one wildcard processor node per namespace.

Note: If the Webflow must search for a wildcard node, there may be a slight impact on performance, as more processing is involved.

As an example, perhaps you want a link called Help (that is present on every page) to always direct to a JSP containing help information. To do so, you might use a wildcard presentation origin. Further, you might always want exceptions returned from processor nodes to transition to JSP containing detailed information about the error. This could be handled with a wildcard processor node.

Note: For more information about creating a wildcard node in the Webflow Editor (which is done the same way you create regular presentation and processor nodes), see “How to Add Webflow Components.”

The Configuration Error Page

The configuration error page is essentially a presentation node whose primary purpose is to display debugging information when there is a configuration error in the Webflow. If the Webflow mechanism cannot locate the destination for a transition after exhausting a list of possible choices, the Webflow will instead display the configuration error page you defined. The configuration error page is useful during development, but should never be displayed to visitors of your Web site. That is, testing should ensure that the Webflow configuration is working before your Web application is put into production.

Note: For more information about how the Webflow searches for possible destination nodes, see “Webflow Execution Order” on page 2-13.

You can have more than one configuration error page in a Webflow (one per each namespace), but are not required to have any configuration error pages if you do not want to. If a namespace specified in the URL has its own configuration error page, the Webflow will use that configuration error page before looking for a configuration error page in the default namespace. Therefore, although the configuration error page is optional, we recommend that you have at least one defined in your default namespace.

Notes: You specify a default namespace in the `P13N_DEFAULT_NAMESPACE` context parameter of your Web application’s `WEB-INF/web.xml` file. For more information about this context parameter, see “Setting Up Your Web Application’s web.xml File.”

For more information about specifying a configuration error page name for your Webflow in the Webflow Editor, see “How to Set the Configuration Error Page Name.”

Chaining and Branching with Processor Nodes

Much like you can move from one presentation node (such as a JSP) to another, you can also move from one processor node (such as an Input Processor) to another processor node. In other words, you do not necessarily need to use processor nodes in between presentation nodes. For processor nodes, this movement is called **chaining**. In a chaining arrangement, the result state of one successfully executed processor node is another processor node.

Another useful technique regarding processor nodes is **branching**. You can branch on Input Processor nodes based on the value of the object the Input Processor returns. For example, you can code an Input Processor such that one response from a Web site visitor returns one value, and a different response returns a different value. Based on the value of the object returned from successful execution of the Input Processor, you can cause the Webflow to direct to different destination nodes. Similarly, you can branch on Pipelines based on the exceptions it returns. Based on the type of exception thrown, you can cause the Webflow to direct to different destination nodes. (You cannot branch on return values of objects in Pipelines because the Pipeline Components within them return Pipeline Session properties instead.)

Using Webflow Components in Your Web Pages

When you use the Webflow mechanism in your Web application, each of the URLs within that application should be dynamically generated. This is accomplished by using the Webflow JSP tags, described in “Webflow JSP Tag Library Reference.”

Note: If you used the prior implementation of Webflow, you may recall that dynamic URL generation was formerly accomplished using the `createWebflowURL()` method of the `WebflowJSPHelper` class. This method is still available, but it

is strongly suggested that you use the JSP tags instead. The only reason you may want to use the `WebflowJSPHelper` class instead of a JSP tag is in a servlet or wrapper class.

Using Webflow Components with Portals

With a few exceptions, Webflow works the same way in portal Web applications as it does within the rest of the platform. That is, you can use Webflow to coordinate different parts of your portal Web application, enabling code reuse and maintainability.

The Webflow associated with portals are generally not to be edited because it is fundamental to the application logic that makes up the portal platform. Individual portlets, however, can have individual Webflow files assigned to them, allowing practically all Webflow functionality to be applied to a portlet.

For more information, see “Customizing Portlets and Portals” in the *Getting Started with Portals and Portlets* documentation.

Webflow Execution Order

As described in “Understanding Webflow as a State Machine,” Webflow can be thought of as a state machine and as such, can only be in one state at a time. Webflow transitions from one state to another by interpreting URLs, return values, and exceptions. Therefore, to effectively work with Webflow, you need to understand the order in which Webflow searches for a transition. When searching for a transition, Webflow always attempts to resolve transitions with the exact criteria supplied, but if no matches are found, Webflow broadens its search using wildcards and inheritance hierarchies. The way Webflow searches for a valid transition differs slightly for Presentation Nodes and Processor Nodes.

Note: An important, related concept is that Webflow knows nothing of Pipelines; Pipelines are just Extension (Custom) Processors that BEA has created for you. Consequently, Webflow only governs the flow in and out of Pipelines but

not within the Pipeline itself. (The transitions that take place inside a Pipeline are governed by the Pipeline engine.) Therefore, the explanation provided in this section only applies to transitions to and from Presentation Nodes (JSP, HTML, servlets, Extension (Custom) Nodes) and Processor Nodes (Input Processors, Pipelines, Extension (Custom) Nodes) in a given Webflow.

Presentation Nodes

For Presentation Nodes, Webflow's search for valid transitions is done in the following manner:

1. Webflow searches to find the exact Presentation Node as supplied in the URL. Given the URL:

```
http://localhost:7501/myapp?namespace=order&origin=shoppingcart.jsp&event=link.help
```

Webflow tries to resolve to the `shoppingcart.jsp` Presentation Node, defined in the underlying `order.wf` Webflow namespace configuration file as:

```
<presentation-origin node-name="shoppingcart" node-type="jsp">
```

2. If the `shoppingcart.jsp` Presentation Node is not present, Webflow searches for the Wildcard Node of the same node type. (As previously shown in the URL, origins are in the form `<node-name>.<node-type>`.) Given the same URL, Webflow tries to resolve to a Wildcard Presentation Node, defined in the underlying `order.wf` Webflow namespace configuration file as:

```
<wildcard-presentation-origin node-type="jsp">
```

Note: For more information about the Wildcard Presentation and Wildcard Processor Nodes, see “The Wildcard Nodes” on page 2-10.

3. If the Webflow fails to locate the exact Presentation Node or a Wildcard Presentation Node, then a configuration error has occurred, and Webflow searches for the configuration error page.

Note: For more information about the configuration error page, see “The Configuration Error Page” on page 2-11.

Alternatively, if the Webflow does locate the exact Presentation Node or a Wildcard Presentation Node, then Webflow searches for the supplied event under the respective node. Therefore, given the same URL, Webflow searches for the

`link.help` event in the event list, defined in the underlying `order.wf` Webflow namespace configuration file as:

```
<event event-name="link.help">  
  <destination namespace="main" node-name="help"  
node-type="jsp" />  
</event>
```

4. If Webflow finds the Presentation Node but not the event, then Webflow searches under the Wildcard Presentation Node for that event. If no event is found, then a configuration error has occurred, and Webflow searches for the configuration error page.

Note: For more information about the configuration error page, see “The Configuration Error Page” on page 2-11.

Using the Webflow and Pipeline Editors to create Webflows can help identify potential problems with your Webflow and greatly reduce the likelihood of errors. This is because of the Webflow and Pipeline Editors’ built-in validation features. For more information, see “About the Webflow and Pipeline Editors’ Validation Features.”

Processor Nodes

Processor Nodes can transition in one of two ways: they can return a Java object or throw a Java exception.

- If the Processor Node returns a Java object, Webflow calls `toString()` on that object and searches for an event with a matching name in the event list. The same search patterns described in “Presentation Nodes” on page 2-14 apply here. Webflow first tries to look for the event under the exact Processor Node that returned the object, but if that Processor Node is not found, Webflow searches for the event under a Wildcard Processor Node.

Note: For more information about the Wildcard Presentation and Wildcard Processor Nodes, see “The Wildcard Nodes” on page 2-10.

- If the Processor Node throws an exception, Webflow searches the exception list associated with the current Processor Node. Webflow first searches for the exact exception thrown. If this exception is not found, Webflow walks up the exception’s inheritance hierarchy in an attempt to find a match. Webflow will walk all the way up the inheritance tree until it encounters the class `Exception`.

If still no match is found, Webflow searches the Wildcard Processor Node and performs the same exception walking activity.

- If all previous attempts fail, the Webflow will simply load the configuration error page defined for the current namespace. If the current namespace does not define a configuration error page, the Webflow will display the configuration error page defined for the default namespace.

Note: For more information about the configuration error page, see “The Configuration Error Page” on page 2-11.

- If for some reason the file for the configuration error page is missing, a standard 500 internal server error, which is outside the scope of the Webflow mechanism, would be displayed.

Using the Webflow and Pipeline Editors to create Webflows can help identify potential problems with your Webflow and greatly reduce the likelihood of errors. This is because of the Webflow and Pipeline Editors’ built-in validation features. For more information, see “About the Webflow and Pipeline Editors’ Validation Features.”

3 Using the Webflow and Pipeline Editors

The E-Business Control Center (EBCC) includes a Webflow Editor and a Pipeline Editor designed to help you create, modify, and validate Webflow and Pipeline XML configuration files. This topic provides an introduction to the Webflow and Pipeline Editors, describes their graphical interfaces, and provides some step-by-step instructions for their use. This topic also includes some best practices for creating, modifying, and migrating Webflow and Pipeline XML configuration files, and describes the Webflow Editor's validation feature. Lastly, this topic provides some information about synchronizing Webflow and Pipeline data for your applications.

This topic includes the following sections:

- Introduction
 - Starting the Webflow and Pipeline Editors
 - Important Notes About Using the Webflow and Pipeline Editors
 - Next Steps
- Learning to Use the Webflow and Pipeline Editors
 - Webflow and Pipeline Editor Essentials
 - Webflow Component Representations
 - Understanding the Webflow and Pipeline Editor Palettes
 - Understanding the Webflow and Pipeline Editor Toolbars
- Organizing Webflow Components in an Editor Canvas
 - How to Select Webflow Components

- How to Add Webflow Components
- How to Edit a Webflow Component's Name (Label)
- How to Designate or Remove a Begin (Root) Node
- How to Move a Webflow Component
- How to Connect Nodes with Event or Exception Transitions
- How to Reposition Connection Ports on a Node
- How to Work with Elbows in Transitions
- Using the Webflow and Pipeline Editor Toolbars
 - How to Print a Webflow Namespace or Pipeline
 - How to Delete Webflow Components
 - How to Use the Zoomed Overview
 - How to Show/Hide the Grid
 - How to Snap Objects to the Grid
 - How to Enable and Disable Link Optimization
 - How to Show and Hide Exception Transitions
 - How to Validate the Selected Node
 - How to Validate All Nodes
 - How to Set the Configuration Error Page Name
 - How to Use the Pipeline Component Editor
 - How to Make the Pipeline Transactional
 - How to Include the Pipeline Session in a Transaction
- Using the Properties Editors
 - Viewing Component Properties
 - Description of Webflow Component Properties
 - Modifying Component Property Values
- Migrating An Existing Webflow
- Creating or Modifying a Webflow: Breadth-First Versus Depth-First

- About the Webflow and Pipeline Editors' Validation Features
 - Validation Error Messages in a Properties Editor
 - What Do the Editors Validate?
 - Saving Invalid Webflows
- Synchronizing Webflow Data for Your Application

Note: This topic assumes that you are already familiar with basic Webflow concepts. If you are not, refer to “Webflow Components and Concepts.”

Introduction

This introductory section provides instructions on how to start the Webflow and Pipeline Editors, and lists some important notes about the Editors that you should take into account before using them to create or modify your Webflow and Pipeline XML configuration files.

Starting the Webflow and Pipeline Editors

To begin using the Editors, follow these steps:

1. Start the E-Business Control Center (EBCC). For detailed instructions on starting the EBCC, see “Starting the E-Business Control Center” in the *Guide to Using the E-Business Control Center* documentation.
2. Create a new Web application or open an existing Web application for which you will be working on a Webflow or Pipeline. For detailed instructions on performing these tasks, see “Creating an Application Structure for E-Business Control Center Data” or “Opening Application Data” in the *Guide to Using the E-Business Control Center* documentation.
3. Select the Site Infrastructure tab in the EBCC's Explorer window, then click the Webflows/Pipelines icon.

Note: You may have to use the scroll bar to locate the Webflows/Pipelines icon.

4. If the Webflows/Pipelines structure shown in the Explorer window is not expanded, click Webapps and then the Web application name to view the Webflow namespaces for that application, or click Pipeline Namespaces and then the namespace to view the Pipelines available within that namespace.

Figure 3-1 shows the Webflow namespaces and Pipelines in expanded form for the sample `petflow` Web application.

Figure 3-1 EBCC Explorer Window With Expanded Webflows/Pipelines List



5. If you want to modify an existing Webflow, double click any of the Web application's namespace files shown below the Web application name in the Webflows/Pipelines list. (Figure 3-1 shows the `main` namespace for the `petflow` Web application being selected.) This opens the Webflow Editor.

Note: Listing 3-1 also indicates that the main Webflow namespace is currently invalid, by the red and white square icon located to its left. For more information about Webflow validity, see “About the Webflow and Pipeline Editors’ Validation Features.” For more information about the status icons in the E-Business Control Center (EBCC), see “About the E-Business Control Center Interface” in the *Guide to Using the E-Business Control Center* documentation.

6. If you want to modify an existing Pipeline, double-click any of the Pipeline names (such as `viewShoppingCart` in Figure 3-1) shown in the Webflows/Pipelines list. This opens the Pipeline Editor.

Note: You can also open the Pipeline Editor by double clicking on the icon portion of a Pipeline Node that is currently displayed in the Webflow Editor canvas, as shown in Figure 3-2. For more information about Pipeline Nodes, see “Input Processors and Pipelines.”

Figure 3-2 Results of Double-Clicking Portions of Pipeline Node



7. If you would rather create a new Webflow or Pipeline for the Web application, see “Creating Webflow/Pipeline Files” in the *Guide to Using the E-Business Control Center* documentation.

Note: For general information about managing EBCC files, see “About the E-Business Control Center Interface” in the *Guide to Using the E-Business Control Center* documentation.

Important Notes About Using the Webflow and Pipeline Editors

The following list includes useful information for developers who are about to use the Webflow and Pipeline Editors:

- An application's Webflow is comprised of one or more Webflow namespace (`<namespace>.wf`) files, as well as zero or more Pipeline (`<namespace>.pln`) files that exist within a namespace. Webflow files specify the flow of control between presentation nodes (JSPs, HTML, and so on) and processor nodes (Input Processors and Pipelines) that exist in a Webflow namespace. Pipeline files contain information about the Pipeline Components that comprise a given Pipeline. Therefore, when you use the Webflow Editor, you are creating or editing a Webflow namespace; when you use the Pipeline Editor, you are creating or editing the Pipeline Component nodes for a Pipeline that exists *within* a given namespace. For more information, see "Webflow and Pipeline Editor Essentials" on page 3-9.
- The Webflow and Pipeline Editors are designed to help you create, modify, and validate `<namespace>.wf` and `<namespace>.pln` XML configuration files that are associated with a particular Web or enterprise application, respectively. The Editors do not support the editing of arbitrary files or other files in the WebLogic Portal product suite.
- The recommended editing method for the Webflow and Pipeline XML configuration files is via the Webflow and Pipeline Editors. If while hand-editing the XML configuration files in a text editor you create invalid entries, or if you fail to provide a namespace called `main` when using a `begin` node as the destination of an event in a new Webflow (as in `<destination node-name="begin"/>`), the behavior of the Editors may be unpredictable.
- It is expected that if Webflow and Pipeline XML configuration files for an application are stored in a Content Management System, they will be checked out by developers prior to modification in the Webflow and Pipeline Editors. If the appropriate files are not checked out, the Editors will open the files in read-only mode and the Editor palettes and toolbars that typically allow you to make modifications will not be shown. For more information, see "Webflow and Pipeline Files in Your Content Management System" on page 3-12.

- The Webflow and Pipeline Editors' use of namespace files allows different developers to work on portions of a single application's Webflow at the same time. Keep in mind, however, that references to nodes in other namespaces (called proxy nodes) are simply a snapshot of information that was available in your Content Management System at the time your files were checked out. Other developers with write permission for the application may be simultaneously modifying namespace modules within the application, causing the Webflow to enter an invalid state. It is important that developers on the team work together to ensure that changes are propagated to all affected namespaces and result in a valid Webflow for the entire application. The Webflow and Pipeline Editors do allow you to save a Webflow or Pipeline that is invalid, but you must ensure that an application's Webflow (as a whole) is valid prior to data synchronization. See "Synchronizing Application Data" in the *Deployment Guide* for more information.
- The E-Business Control Center (EBCC) does not support role-based security. Any user with access to the EBCC can also access the Webflow and Pipeline Editors.
- By default, changes made to Webflow and Pipeline XML configuration files do not require you to restart the server or perform any special deployment activities. Instead, the Webflow and Pipeline data for a particular application is synchronized along with other application data. See "Synchronizing Application Data" in the *Deployment Guide* for more information. If errors in a Webflow (or other data) are detected during the data synchronization process, an appropriate error message will be sent to the client. While such errors will not prevent application deployment, they may result in failures during execution.
- If you are using a cluster, the data synchronization service will update your application's Webflow and Pipeline information to all servers in the domain.

Next Steps

If you are new to this release of the Webflow and Pipeline Editors, it is strongly suggested that you proceed to the next section, "Learning to Use the Webflow and Pipeline Editors" on page 3-8. For step-by-step instructions on using the Webflow and Pipeline Editors to create Webflows, proceed to any of the following:

- Organizing Webflow Components in an Editor Canvas

- Using the Webflow and Pipeline Editor Toolbars
- Using the Properties Editors

Once you understand the behaviors and capabilities of the Webflow and Pipeline Editors, you may want to read some of the following sections, depending on your current goals:

- Migrating An Existing Webflow
- Creating or Modifying a Webflow: Breadth-First Versus Depth-First
- About the Webflow and Pipeline Editors' Validation Features

If you have already finished work on your application's Webflow, you may want to read "Synchronizing Webflow Data for Your Application" on page 3-48.

Learning to Use the Webflow and Pipeline Editors

This section provides you with general information about the Webflow and Pipeline Editor user interfaces, and includes a mapping of Webflow components to their graphical representations. This section also includes descriptions of tools in the Webflow and Pipeline Editor palettes and buttons in the Editors' toolbars. As such, this section includes the following:

- Webflow and Pipeline Editor Essentials
- Webflow Component Representations
- Understanding the Webflow and Pipeline Editor Palettes
- Understanding the Webflow and Pipeline Editor Toolbars

Webflow and Pipeline Editor Essentials

This section provides you with some basic information about the Webflow and Pipeline Editors. It includes an explanation of what the Webflow Editor displays and allows you to work with, introduces the Pipeline Editor, and describes some status indicators you should notice as you use the Webflow and Pipeline Editors. It also includes a brief discussion about using a Content Management System with your Webflow and Pipeline XML configuration files.

Webflow Namespaces

Although there is only one Webflow per Web application, BEA recognizes that different developers may be assigned to work in different functional areas of the same application. Therefore, a Webflow for a single Web application is comprised of numerous modules called **namespaces**. (See “Namespaces” for more information.)

The Webflow Editor displays and requires that you work with Webflow components within individual namespaces. In addition to allowing different developers to work in different areas of the same Webflow, displaying a Webflow by namespace in the Webflow Editor provides a much cleaner visualization, and thus may reduce the likelihood of error.

As such, the activities you perform in the Webflow Editor are really just operations on one or more editable namespace files. You can create, open, save, save as, or delete Webflow namespace files the same way you perform these tasks with other types of EBCC files. For detailed instructions on these general EBCC functions, see “Working with Files” in the *Guide to Using the E-Business Control Center* documentation.

The Webflow Editor does allow you to open more than one namespace file at a time. When you open more than one namespace, the tabs at the bottom of the Editor window will allow you to move between the namespace files that are currently available.

Note: For more information about the behavior of multiple files in an EBCC editor, see “About the E-Business Control Center Interface” in the *Guide to Using the E-Business Control Center* documentation.

Pipelines Versus Pipeline Namespaces

The Pipeline Editor displays and requires that you work with Webflow components within individual Pipelines, rather than within entire Pipeline namespaces. In other words, since you can define multiple Pipelines within one Pipeline namespace, the editable unit in the Pipeline Editor is a Pipeline. You can create, open, save, save as, or delete Pipelines the same way you perform these tasks with other types of EBCC files. For detailed instructions on these general EBCC functions, see “Working with Files” in the *Guide to Using the E-Business Control Center* documentation.

Because Pipelines are scoped to an enterprise application (instead of a Web application like Webflows), Pipelines do not appear within the Webapp folder in the E-Business Control Center Explorer window’s Webflows/Pipelines list. Rather, Pipelines appear under the Pipeline Namespaces folder. Thus, while a Pipeline is the editable unit within the Pipeline Editor, it is a *Pipeline namespace* (rather than the Pipeline itself) that corresponds to a generated `<namespace>.pln` file. You will need to check out the appropriate Pipeline namespace file from your Content Management System in order to work with a Pipeline in the Pipeline Editor. This is the only case where an object shown in the E-Business Control Center Explorer window does not have a one-to-one correspondence with a file.

The Pipeline Editor opens when you double click a Pipeline shown in the E-Business Control Center (EBCC) Explorer window, or when you double click on the icon portion of a Pipeline node that is currently displayed in the Webflow Editor canvas, as shown in Figure 3-3. The Pipeline Editor behaves exactly like the Webflow Editor, but operates with a different palette and toolbar, and on different node types (namely, Pipeline Components).

Figure 3-3 Results of Double-Clicking Portions of Pipeline Node



Notes: For more information about the differences between the Webflow and Pipeline Editors, see “Understanding the Webflow and Pipeline Editor Palettes” on page 3-17 and “Understanding the Webflow and Pipeline Editor Toolbars” on page 3-20. For more information about Pipelines and Pipeline Components, see “Input Processors and Pipelines.”

The Pipeline Editor allows you to open more than one Pipeline at a time. When you open more than one Pipeline, the tabs at the bottom of the Editor window will allow you to move between the Pipelines that are currently available. When you open multiple Pipelines, remember that these Pipelines may or may not be part of the same Pipeline namespace.

Note: For more information about the behavior of multiple files in an EBCC editor, see “About the E-Business Control Center Interface” in the *Guide to Using the E-Business Control Center* documentation.

Information Displayed in the Editors' Title Bars

In the Webflow Editor, the title bar displays the name of the Editor and the namespace for the Webflow you are currently creating or modifying, in brackets []. For example, the Webflow Editor's title bar may read: `Editor [Webflow: main]`, which means you are currently using the Webflow Editor to work with Webflow components in the namespace called `main`.

In the Pipeline Editor, the title bar displays the name of the Editor and the Pipeline you are currently creating or modifying, in brackets []. For example, the Pipeline Editor's title bar may read: `Editor [Pipeline: viewShoppingCart]`, which means you are currently using the Pipeline Editor to define Pipeline Components in the Pipeline called `viewShoppingCart`.

In both Editors, the title bar may also show the text “Read Only” in parentheses. If present, this text indicates that the opened file(s) are in read-only mode and must be checked out of your Content Management System prior to editing or saving. See “Webflow and Pipeline Files in Your Content Management System” on page 3-12 for more information.

Webflow and Pipeline Files in Your Content Management System

When you want to edit or save Webflow components within a Webflow namespace or a Pipeline using the Webflow and Pipeline Editors, you should verify that you have checked out all the necessary files from your Content Management System. Depending on the Webflow components you want to edit, you may or may not need all of the following files:

- `<namespace>.wf`

The Webflow configuration files contain XML elements that represent the Webflow components used in the Webflow namespace, as well as the relationships among them. The `<namespace>.wf` files are scoped to the Web application.

- `<namespace>.pln`

The Pipeline configuration files contain XML elements that represent the Pipeline Components within a Pipeline being used in the namespace, as well as the relationships among them. The `<namespace>.pln` files are scoped to the enterprise application.

- `<namespace>.wf.ui` and `<namespace>.pln.<pipeline_name>.ui`

These XML files contain information that is used by the Webflow and Pipeline Editors to display the Webflow components in the way you last organized them. They are generated when you save a Webflow or Pipeline, and are located in the appropriate directories under the `BEA_HOME\EBCC_HOME\applications\
<enterprise_app>\project-info\webapps\
<web_app>` directory.

- `webflow-extensions.wfx`

This XML file describes the Webflow components currently available for use in the Webflow Editor. Any extension (custom) presentation or processor nodes your development team creates must be registered in this file if you want them to appear in the Webflow Editor. The `webflow-extensions.wfx` file is scoped to a Web application, and should remain in read-only mode unless extensions need to be added. More information about the `webflow-extensions.wfx` file is located in “Making Your Extension Presentation and Processor Nodes Available in the Webflow Editor.”

Note: If you just want to view a Webflow namespace or Pipeline using the Editors, you do not need to check out files. The Webflow and Pipeline Editors will display the Webflow components in read-only mode. In read-only mode, the

Webflow and Pipeline Editors’ palettes and toolbars are not displayed. For more information about the Webflow and Pipeline Editors’ palettes and toolbars, see “Understanding the Webflow and Pipeline Editor Palettes” on page 3-17 and “Understanding the Webflow and Pipeline Editor Toolbars” on page 3-20, respectively.



Webflow Component Representations

Since the Webflow and Pipeline Editors are graphical tools for creating and modifying Webflows and Pipelines, each Webflow component has a graphical representation for display in an Editor. This section provides you with information that will help you associate each Webflow component with its graphical representation.

Note: In addition to associating Webflow components with their graphical representations, Table 3-1 and Table 3-2 also provide brief descriptions of each component. If you do not remember the details of a particular Webflow component, refer to “Webflow Components and Concepts” for more information.

Table 3-1 provides information about Webflow component representations in the Webflow Editor.

Table 3-1 Webflow Components in the Webflow Editor

Representation	Webflow Component	Description
	Presentation Node	A page displayed to visitors interacting with your Web application. The type of these pages can be HTML, JSP, Java servlet, or an Extension (Custom) Presentation Node.
 <i>(This representation also shows an automatically created connection port.)</i>	Wildcard Presentation Node	A single, global event contained on one or more Presentation Nodes, typically used to implement general functions. There may be several Wildcard Presentation Nodes within a Webflow namespace. The type property reflects the context in which the event occurs.

3 Using the Webflow and Pipeline Editors

Table 3-1 Webflow Components in the Webflow Editor (Continued)






Representation	Webflow Component	Description
<i>(Same as a Presentation Node.)</i>	Extension (Custom) Presentation Node	A node created by an experienced Java/EJB developer for use in a Webflow. To appear in the Webflow Editor, this node must have an equivalent server-side Java class representation and be registered in the <code>webflow-extensions.wfx</code> file.
	Input Processor Node	A processor Webflow component that handles data input, whether from Web site visitors or other processor nodes. Validation of data input is the typical use of an Input Processor.
 <i>(This representation also shows an automatically created connection port.)</i>	Pipeline Node	A processor Webflow component that encapsulates business logic for a Web application. Pipelines Nodes contain a series of Pipeline Component Nodes, the latter of which are editable in the Pipeline Editor.
 <i>(This representation also shows an automatically created connection port.)</i>	Wildcard Processor Node	A single, global event contained on one or more Processor Nodes, typically used to implement general functions. There may be several Wildcard Processor Nodes within a Webflow namespace. The type property reflects the context in which the event occurs.
	Extension (Custom) Processor Node	A node created by an experienced Java/EJB developer for use in a Webflow. To appear in the Webflow Editor, this node must have an equivalent server-side Java class representation and be registered in the <code>webflow-extensions.wfx</code> file.
 <i>(This image represents a Presentation Begin Node.)</i>	Begin Node	A node designated as the initial state of or entry point into the Webflow. There can be one Begin Node for each namespace in a Webflow. The Begin Node can also be used as a destination.

Table 3-1 Webflow Components in the Webflow Editor (Continued)





Representation	Webflow Component	Description
 <i>(This image represents a Pipeline Proxy Node.)</i>	Proxy Node	A special node that represents a node whose definition resides in a different namespace file. Nodes in the namespace being edited can refer to the Proxy Node as a destination, but the Proxy Node itself cannot be edited. Proxy Nodes can be Presentation Nodes, Processor Nodes, or Extension (Custom) Nodes.
 <i>(The second representation is of a self-referring event.)</i>	Event <i>(may be a link, button, or return object)</i>	An occurrence triggered by the interaction of a Web site visitor that results in some change in the state of the Webflow. In other words, events cause Webflow transitions (movement from an origin node to a destination node). There are different events for different node types: link and button are events for Presentation Nodes; return object is an event for a Processor Node that has executed successfully.
 <i>(The second representation is of a self-referring exception.)</i>	Exception	Representation of an exception, which occurs when Processor Nodes do not execute successfully.






Table 3-2 provides information about Webflow component representations in the Pipeline Editor.

Table 3-2 Webflow Components in the Pipeline Editor

Representation	Webflow Component	Description
 <i>(This representation also shows an automatically created connection port.)</i>	Pipeline Component Node	A specialized Webflow component that performs tasks related to the application's underlying business logic. Pipeline Components exist within Pipelines, and may be implemented as Java objects or stateless session EJBs.

3 Using the Webflow and Pipeline Editors

Table 3-2 Webflow Components in the Pipeline Editor (Continued)

Representation	Webflow Component	Description
	Root Node	A Pipeline Component that represents the initial entry point into the Pipeline. There can be one Root Node for each Pipeline. The Root Node can also be used as a destination.
	Abort Exception Node	A special node that represents the place where a Pipeline's exceptions connect to, because their destinations actually exist outside the Pipeline Editor (in the Webflow Editor). Note: For more information, see "Configuring Pipeline Component Exception Fatality."
 <i>(Self-referring event transitions are not allowed in the Pipeline Editor.)</i>	Event <i>(return object)</i>	An occurrence triggered by the successful execution of a Pipeline Component that results in some change in the state of the Webflow. In other words, events cause Webflow transitions (movement from an origin node to a destination node). There are different events for different node types: return object is an event for a Pipeline Component Node that has executed successfully.
  <i>(The second representation is of a self-referring exception.)</i>	Exception	Special event type triggered by a Pipeline Component Node when it does not execute successfully.

Understanding the Webflow and Pipeline Editor Palettes

When you first open a Webflow namespace in the Webflow Editor or a Pipeline in the Pipeline Editor, you will notice a palette on the left-hand side of the Editor window. This section describes each of the tools provided in the Webflow and Pipeline Editor palettes.

Notes: If you do not see a palette on the left-hand side of either the Webflow or Pipeline Editors when you open a Webflow namespace or a Pipeline, this means that the files you are opening are in read-only mode. To view the palettes, you will need to make sure the appropriate `.wf` and `.pln` files are writable. For more information about these files, see “Webflow and Pipeline Files in Your Content Management System” on page 3-12.

The Pipeline Editor opens when you double click a Pipeline shown in the E-Business Control Center (EBCC) Explorer window, or when you double click on the icon portion of a Pipeline node that is currently displayed in the Webflow Editor canvas, as shown in Figure 3-4. For more information about Pipelines, see “Input Processors and Pipelines.”

Figure 3-4 Results of Double-Clicking Portions of Pipeline Node



Tools in the Webflow Editor Palette

Table 3-3 describes each of the tools in the Webflow Editor palette.

Table 3-3 Description of the Webflow Editor Palette Tools












Tool	Function	Description
	Selection Tool	Allows you to select and move nodes, event transitions, and exception transitions. Also allows you to add elbows to transitions. This is the default tool for the Webflow Editor.
	Event Tool	Allows you to add an event transition between two nodes, or a self-referring event transition.
	Exception Tool	Allows you to add an exception transition between two nodes, or a self-referring exception transition.
	Begin Node	Allows you to designate one of the Presentation or Processor Nodes already on the Editor canvas as the Begin Node for the current Webflow namespace.
	Presentation Node	Allows you to add a new Presentation Node to the Editor canvas.
	Wildcard Presentation Node	Allows you to add a new Wildcard Presentation Node to the Editor canvas.
	Input Processor Node	Allows you to add a new Input Processor Node to the Editor canvas.
	Pipeline Node	Allows you to add a new Pipeline Node to the Editor canvas.
	Wildcard Processor Node	Allows you to add a new Wildcard Processor Node to the Editor canvas.






Table 3-3 Description of the Webflow Editor Palette Tools (Continued)

Tool	Function	Description
	Extension (Custom) Processor Node	Allows you to add a new Extension (Custom) Processor Node to the Editor canvas. Note: The Extension (Custom) Processor Node tool is disabled until the Webflow Editor detects a new node in the <code>webflow-extensions.wfx</code> file.
	Proxy Node	Allows you to add a Proxy Node to the Editor canvas. You should create a Proxy Node any time you want to refer to a node that is defined in another namespace.

Tools in the Pipeline Editor Palette

Table 3-4 describes each of the tools in the Pipeline Editor palette.

Table 3-4 Description of the Pipeline Editor Palette Tools

Tool	Function	Description
	Selection Tool	Allows you to select and move Pipeline Components, event transitions and exception transitions. This is the default tool for the Pipeline Editor.
	Event Tool	Allows you to add an event transition between two nodes.
	Exception Tool	Allows you to add an exception transition between two nodes, or a self-referring exception transition.
	Root Node	Allows you to designate one of the Pipeline Components already on the Editor canvas as the Root Node for the current Pipeline.
	Pipeline Component	Allows you to add new Pipeline Components to the Editor canvas.

Understanding the Webflow and Pipeline Editor Toolbars

You will find the Webflow and Pipeline Editor toolbars at the top of both the Webflow and the Pipeline Editors, respectively, but the buttons available from each Editor's toolbar may differ. This section describes which buttons are available from which Editor's toolbar, and describes their functions.

Display and Behavior Buttons

The toolbar buttons described in this section modify the display or behavior of Webflow components, and may not be available from both the Webflow and Pipeline Editors. These buttons act as toggles, meaning that you click the button once to select the tool, then again to deselect the tool.

Table 3-5 briefly describes the Display and Behavior buttons, and where appropriate, indicates which Editor they belong to.

Table 3-5 Description of the Display and Behavior Buttons







Tool	Function	Description
	Show/Hide Grid button	Allows you to show or hide a grid background in the Editor canvas.
	Snap to Grid button	Allows you to control whether or not Webflow components are automatically placed to the nearest grid point on the Editor canvas when you release the mouse button.
	Link Optimization button	Allows you to control whether or not the connectors on each node will be automatically moved around the perimeter of the node as the node is moved on the Editor canvas.
	Show/Hide Exceptions button	Allows you to show or hide exception transitions in the Editor canvas.

Table 3-5 Description of the Display and Behavior Buttons (Continued)




Tool	Function	Description
	Make This Pipeline Transactional button	Allows you to specify whether the Pipeline should be transactional. Note: Only available in the Pipeline Editor.
	Include Pipeline Session in Transaction button	Allows you to specify whether the Pipeline Session should be included in the transaction. Enabled only if the Pipeline Transaction button is on. Note: Only available in the Pipeline Editor.

Command Buttons

The toolbar buttons described in this section execute commands related to Webflow components, and may not be available from both the Webflow and Pipeline Editors. Many of these buttons also have keyboard shortcuts you may find helpful.





Table 3-6 briefly describes the Command buttons and where appropriate, indicates which Editor they belong to.

Table 3-6 Description of the Command Buttons

Tool	Function	Description	Keyboard Shortcut
	Print button	Allows you to print the entire Webflow namespace or Pipeline to a printer.	Ctrl+P
	Delete button	Deletes the selected Webflow component. The Delete button is disabled until a Webflow component is selected.	Delete
	Zoomed Overview button	Allows you to view the entire Webflow namespace or Pipeline at a glance.	Ctrl+Z

3 Using the Webflow and Pipeline Editors

Table 3-6 Description of the Command Buttons (Continued)

Tool	Function	Description	Keyboard Shortcut
	Validate the Selected Node button	Allows you to run the Webflow validation feature on the selected node. The Validate the Selected Node button is disabled until a Webflow component is selected.	Ctrl + V
	Validate All button	Allows you to run the Webflow Editor's validation feature on the entire Webflow namespace, or the Pipeline Editor's validation feature on the entire Pipeline.	Alt + V
	Set Up Configuration Error Page Name button	Allows you to specify the name and path to the configuration error page. Note: Only available in the Webflow Editor.	--
	Pipeline Component Editor button	Opens the Pipeline Component Editor, which allows you to manage Pipeline Components. Note: Only available in the Pipeline Editor.	--

Organizing Webflow Components in an Editor Canvas

This section provides you with detailed instructions about how to perform common tasks using the Webflow and Pipeline Editors. This section includes information about:

- How to Select Webflow Components
- How to Add Webflow Components
- How to Edit a Webflow Component's Name (Label)
- How to Designate or Remove a Begin (Root) Node
- How to Move a Webflow Component
- How to Connect Nodes with Event or Exception Transitions
- How to Reposition Connection Ports on a Node
- How to Work with Elbows in Transitions

How to Select Webflow Components

To select a Webflow component in either the Webflow or Pipeline Editor, click the Webflow component. For events and exceptions, this means clicking the transition itself or an associated connection port.

Note: When you select a Webflow component, detailed information about the component is displayed in the Properties Editor. For more information about the Properties Editor, see “Using the Properties Editors” on page 3-38.

How to Add Webflow Components

To add a new Webflow component (that is, a new Presentation Node, Wildcard Presentation Node, Input Processor Node, Pipeline Node, Wildcard Processor Node, Extension (Custom) Processor Node, Proxy Node, or Pipeline Component Node) in either the Webflow or Pipeline Editor, follow these steps:

1. Click the appropriate tool from the Editor's palette (see Table 3-3 or Table 3-4).
2. Position the cross-hairs on the Editor canvas.
3. Click the mouse to add the Webflow component to the Editor canvas in that location.

Notes: A default name (label) is provided for the added component. For information about editing component names, see “How to Edit a Webflow Component's Name (Label)” on page 3-24. In addition to the component's name, there are several other properties for a new component that you must provide. For more information, see “Using the Properties Editors” on page 3-38.

If you add a Proxy Node to the canvas, be sure to connect it to another node with an event or exception transition before saving your Webflow namespace. If you do not, the Proxy Node will not appear on the canvas following a reload. This is because the purpose of the Proxy Node is solely as a destination for transitions. If there are no transitions, there is no use for the Proxy Node. For more information about Proxy Nodes, see Table 3-1 and Table 3-3.

How to Edit a Webflow Component's Name (Label)

The Webflow and Pipeline Editors will typically give each component you add to the canvas a name (label). To edit the component's name, follow these steps:

1. Double-click the Webflow component's name to select it.
2. Type a new name for the component, and press Enter.

Note: You can also edit a Webflow component's name (label) using the Properties Editor. For Pipeline Component Nodes displayed in the Pipeline Editor, you can only edit the Pipeline Component's name by selecting a new one from the Properties Editor. For more information, see "Using the Properties Editors" on page 3-38.

How to Designate or Remove a Begin (Root) Node

To designate a Presentation or Processor Node as the Begin Node for the current Webflow namespace in the Webflow Editor, or a Pipeline Component Node as the Root Node for the current Pipeline in the Pipeline Editor, follow these steps:

1. Click the Begin Node or Root Node Tool (see Table 3-3 and Table 3-4).
2. Click a Presentation, Processor, or Pipeline Component Node that is already on the Editor canvas to designate it as the Begin (Root) Node. The node is marked by a green stripe to the right of the node name

Since the Begin and Root Node Tools work as toggles, you can also remove the Begin or Root designation from a node by following the same procedure. When you remove the begin (root) designation from a node, the green stripe is also removed.

Notes: Wildcard, Proxy, and Abort Exception nodes cannot be designated as begin or root nodes. For more information about the Begin Node, see "The Begin Node." For more information about the Root Node, see "The Root Component Node."

How to Move a Webflow Component

To move a Webflow component (that is, a Presentation Node, Wildcard Presentation Node, Input Processor Node, Pipeline Node, Wildcard Processor Node, Extension (Custom) Processor Node, Proxy Node, or Pipeline Component Node) to another location on the Webflow or Pipeline Editors' canvas, follow these steps:

1. Click and hold the mouse button down on the component while dragging it to the desired location.

Note: The node's corresponding event and exception components (that is, the associated transitions) will move with the node.

2. Release the mouse button to place the component in the new position.

Caution: It is possible to drag a node far to the right or bottom of an Editor canvas, so that you cannot see the entire flow of the Webflow namespace or Pipeline at one glance. Verify that you have edited all the nodes you wanted to by using the scroll bars to the right and at the bottom of the Editor’s canvas. The Zoomed Overview tool, described in “How to Use the Zoomed Overview” on page 3-31, may be helpful here.

How to Connect Nodes with Event or Exception Transitions

Transitions from an origin node to a destination node result from events or exceptions thrown by the origin node. Therefore, you connect two different nodes with event and exception transitions, or you can create a self-referring transition (a transition that has the same origin and destination node).

Note: For more information about origin and destination nodes, see “Introduction to Webflow Components.”

To connect nodes with event or exception transitions, follow these steps:

1. Click the Event Tool or the Exception Tool (see Table 3-3 or Table 3-4).
2. Position the transition by moving the mouse to an edge of the origin node. A solid orange square indicates an acceptable connection location, and the cursor changes to indicate a transition addition.

Note: If the origin node already has a connection port (as shown in Figure 3-5), position the transition by moving the mouse over the existing connection port.

Figure 3-5 Pipeline Node with Automatically Created Connection Port



3. If you want to connect two different nodes with the transition, click, hold, and drag the mouse to the destination node. Release the mouse to connect the transition to the destination node.

Note: An outlined red square with an X in it indicates that you cannot place the transition at the current location.

If you want to create a self-referring transition, just single-click the mouse on the node to connect the transition to that same node.

How to Reposition Connection Ports on a Node

A **connection port** is a small graphical device on a node edge that represents where an event or exception is connected to that node. Connection ports accepting transitions are called input connection ports; connection ports where transitions originate are called output connection ports. In some cases, it may be helpful to move the node's connection port. To reposition the connection port on a node, follow these steps:

1. Click and hold the mouse button on the connection port, then drag the connection port to the desired location on the node.

Note: A solid orange square indicates an acceptable connection location; an outlined red square with an X in it indicates that you cannot place the transition at the current location.

2. Release the mouse button to place the connection port in the new location.

Notes: The connection port associated with a self-referring transition can only be moved along the same node edge.

When repositioning connection ports, you may find the Link Optimization feature helpful. For more information see “How to Enable and Disable Link Optimization” on page 3-32.

How to Work with Elbows in Transitions

In addition to the repositioning connection ports, you can also reposition transition lines on an Editor's canvas by moving, creating or deleting elbows. **Elbows** allow you to bend portions of the transition line.

How to Move an Existing Elbow

To move an existing elbow, follow these steps:

1. Single-click a transition to view the existing elbows, which appear as black squares.
2. Click and hold the mouse button on the elbow as you drag it to the desired position. The selected elbow appears as an orange square.
3. Release the mouse button to place the elbow in the new location.

How to Create a New Elbow

To create a new elbow, follow these steps:

1. Single-click a transition to view the existing elbows, which appear as black squares.
2. Click and hold the mouse button anywhere on the transition line (except on an existing elbow) and drag the mouse to add the new elbow. The selected elbow appears as an orange square.
3. Release the mouse button to create the elbow in that location.

How to Delete a Elbow

To delete an existing elbow, follow these steps:

1. Single-click a transition to view the existing elbows, which appear as black squares.
2. Click the elbow you want to delete to select it. The selected elbow appears as an orange square.
3. Click the Delete button on the Editor's toolbar, or press the Delete key.

Using the Webflow and Pipeline Editor Toolbars

This section provides you with detailed instructions about how to perform common tasks using the buttons in the Webflow and Pipeline Editor toolbars. This section includes information about:

- How to Print a Webflow Namespace or Pipeline
- How to Delete Webflow Components
- How to Use the Zoomed Overview
- How to Show/Hide the Grid
- How to Snap Objects to the Grid
- How to Enable and Disable Link Optimization
- How to Show and Hide Exception Transitions
- How to Validate the Selected Node
- How to Validate All Nodes
- How to Set the Configuration Error Page Name
- How to Use the Pipeline Component Editor
- How to Make the Pipeline Transactional
- How to Include the Pipeline Session in a Transaction

How to Print a Webflow Namespace or Pipeline

To print the contents of the Webflow Editor canvas or the Pipeline Editor canvas, click the Print button (see Table 3-6), or press Ctrl+P. A platform-specific Print dialog opens, which allows you to print as you would from any other application.

How to Delete Webflow Components

Caution: No confirmation dialog is presented before a Webflow component is deleted, and no undo feature is provided. If you delete a Webflow component by mistake, the only remedy is to close the Editor without saving. Be sure you want to delete a Webflow component before proceeding.

You may delete a Webflow component because you do not require it for your namespace, but keep in mind that other namespaces within the application's Webflow may still use the component. If this is the case, the other namespaces will become invalid. Be sure to communicate your changes to others on the development team who may be affected by the deletion.

To delete a Webflow component from an Editor's canvas, follow these steps:

1. Click the Webflow component you want to delete to select it.
2. Click the Delete button on the Editor's toolbar (see Table 3-6), or press the Delete key.

Notes: Deleting a transition or an input connection port associated with a node deletes the entire transition (that is, the input/output connection ports on the node, as well as the line between them). However, deleting an input connection port deletes only the input connection port and the line (that is, the output connection port is retained).

Deleting a transition line or a connection port on a Wildcard Node and saving the Webflow namespace or Pipeline file causes that Wildcard Node to disappear on reload, and no validation alert is produced. This is because the Wildcard Node has no meaning without an event or exception transition.

How to Use the Zoomed Overview

To use the zoomed overview feature, follow these steps:

1. Click the Zoomed Overview Tool (see Table 3-6) to see all the Webflow components in a Webflow namespace or a Pipeline at one glance.
2. Click and hold the mouse button on the green rectangle that appears at the top, left-hand corner of the Editor canvas.
3. Drag the mouse to the desired location. To return to the normal view, release the mouse button.

How to Show/Hide the Grid

When the Show/Hide Grid toggle button in an Editor's toolbar is selected, a rectangular grid appears in the background, behind your Webflow components. This grid may make manipulating components easier on your eyes and help you align components. When the Show/Hide Grid toggle button is deselected, the background is just a plain canvas. By default, the Editors open with the Show/Hide Grid toggle button in the deselected position (that is, no grid is shown).

How to Snap Objects to the Grid

When selected, the Snap To Grid toggle button in an Editor's toolbar can help you align Webflow components by placing them to the grid point closest to where you release the mouse button. When the Snap To Grid toggle button is deselected, you have complete control over placement of the other Webflow components. The Snap To Grid feature does not require the grid to be visible. By default, the Editors open with the Snap To Grid toggle button in the selected position (that is, Webflow components are snapped to the grid).

How to Enable and Disable Link Optimization

When the Link Optimization toggle button in an Editor’s toolbar is selected, link optimization is turned on. In other words, the connectors on each node will be automatically relocated as the node is moved on the Editor canvas, minimizing link length and enabling the best possible display. By default, the Editors open with the Link Optimization toggle button deselected (that is, connectors are not optimized).

How to Show and Hide Exception Transitions

When the Show/Hide Exceptions toggle button in an Editor’s toolbar is selected, exception transitions are not shown on the Editor canvas. This provides you with a cleaner picture of the other Webflow components. Alternatively, when the Show/Hide Exceptions toggle button is deselected, the exception transitions are shown in the Editor canvas. By default, the Editors open with the Show/Hide Exceptions toggle button in the deselected position (that is, exceptions are shown).

How to Validate the Selected Node

To validate a single node in the Webflow or Pipeline Editor, follow these steps:

1. Click the node you want to validate to select it.
2. Click the Validate the Selected Node button (see Table 3-6).

Note: The Validate the Selected Node button is disabled until you select a node as described in steps 1 and 2.

3. Click the Alerts tab of the Properties Editor to view any alerts for the selected node. If an alert exists, the severity of that alert is indicated by the traffic light icon.

Note: For more information about the validation feature, see “About the Webflow and Pipeline Editors’ Validation Features” on page 3-45.

How to Validate All Nodes

To validate all the nodes in the namespace using the Webflow Editor or all the nodes in the Pipeline using the Pipeline Editor, follow these steps:

1. Click the Validate All button (see Table 3-6).
2. Click the Alerts tab of the Properties Editor to view any alerts for the namespace or Pipeline. If an alert exists, the severity of that alert is indicated by the traffic light icon.

Note: As described in “About the Webflow and Pipeline Editors’ Validation Features” on page 3-45, the Alerts tab contains a running log of error messages as you work in the namespace or Pipeline. Therefore, using the Validate All button is useful when you have already cleared the Alerts and want a new, complete validation of your work.

How to Set the Configuration Error Page Name

You can set a configuration error page name using the Webflow Editor. To set the configuration error page name, follow these steps:

1. Click the Set Up Configuration Error Page Name button (see Table 3-6). The Configuration Error Page Dialog, shown in Figure 3-6, appears.

Note: The default configuration error page name that is used in the WebLogic Portal’s sample applications is `error/configurationerror.jsp`.

Figure 3-6 Configuration Error Page Dialog



2. Either type in the path and the file name for the configuration error page in the Page Name input field, or click the Browse button at the left of the input field to locate the Page Name with the Open window.

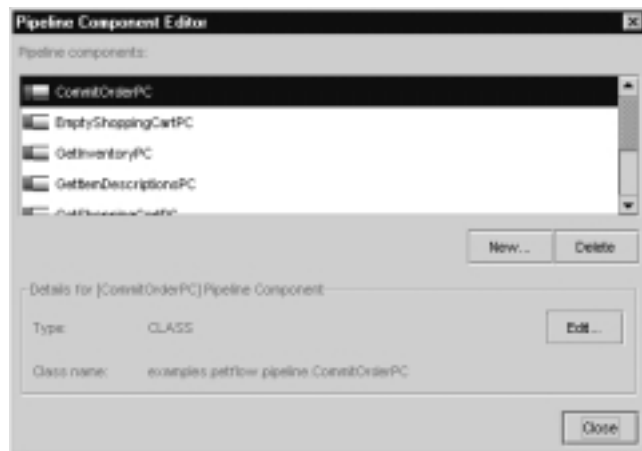
3. Click OK.

Note: For more information about the configuration error page, see “The Configuration Error Page.”

How to Use the Pipeline Component Editor

The Pipeline Component Editor is available from the Pipeline Editor. It allows you to view, add, edit, or delete the Pipeline Components that are available for selection in the Properties Editor (specifically, in the Component attribute). Figure 3-7 is a sample illustration of the Pipeline Component Editor.

Figure 3-7 Pipeline Component Editor



How To View Pipeline Component Details

To view the details of a Pipeline Component that is shown in the list, click the Pipeline Component’s name to select it. The details for that Pipeline Component are shown in the Details region of the Pipeline Component Editor.

How to Add Pipeline Components

To add a new Pipeline Component to the list, follow these steps:

1. Click New... The Pipeline Component Creator dialog opens, an example of which is shown in Figure 3-8.

Note: When you add a new Pipeline Component, a default name for the Pipeline Component is automatically shown in the Name input field.

Figure 3-8 Pipeline Component Creator Dialog



2. Type the name of the Pipeline Component in the Name input field.
3. Select the Type for the Pipeline Component (JNDI or Class) using the radio buttons. The default value for the Type is Class.

Note: If you change the Type to JNDI, the Class Name label will change to JNDI Name.
4. Type the class name or JNDI name for the Pipeline Component in the Class Name or JNDI Name input field.
5. Click OK. The Pipeline Component Creator Dialog closes, and the Pipeline Component you added appears in the Pipeline Components list.
6. Click Close. The Pipeline Component Editor closes, and the new Pipeline Component will now be available for selection as a Component attribute in the Properties Editor.

How to Edit Pipeline Components

To edit a Pipeline Component that is shown in the list, follow these steps:

1. Click the Pipeline Component's name shown in the list to select it.
2. Click Edit... The Pipeline Component Creator dialog opens, an example of which is shown in Figure 3-8.

Note: The existing values for the Pipeline Component are shown in the Pipeline Component Creator dialog.

3. If desired, type a new name for the Pipeline Component in the Name input field.
4. If desired, modify the Type for the Pipeline Component (JNDI or Class) using the radio buttons. The default value for the Type is Class.

Note: If you change the Type to JNDI, the Class Name label will change to JNDI Name.

5. If desired, type the new class name or JNDI name for the Pipeline Component in the Class Name or JNDI Name input field.
6. Click OK. The Pipeline Component Creator Dialog closes, and the new details for the Pipeline Component you edited appear in the Details region.
7. Click Close. The Pipeline Component Editor closes, and the modified Pipeline Component will now be available for selection as a Component attribute in the Properties Editor.

How to Delete Pipeline Components

Caution: No confirmation dialog is presented before a Pipeline Component is deleted, and no undo feature is provided. If you delete a Pipeline Component by mistake, the only remedy is to close the Editor without saving. Be sure you want to delete a Pipeline Component before proceeding.

To delete a Pipeline Component that is shown in the list, follow these steps:

1. Click the Pipeline Component's name shown in the list to select it.
2. Click Delete. The Pipeline Component is no longer shown in the Pipeline Components list.

3. Click Close. The Pipeline Component Editor closes, and the deleted Pipeline Component will no longer be available for selection as a Component attribute in the Properties Editor.

How to Make the Pipeline Transactional

When the Make This Pipeline Transactional toggle button in the Pipeline Editor's toolbar is selected, the Pipeline will be made transactional. Alternatively, when the Make This Pipeline Transactional toggle button is deselected, the Pipeline will not be transactional. By default, the Pipeline Editor opens with the Make This Pipeline Transactional toggle button in the deselected position (that is, the Pipeline will not be transactional).

Note: For more information about transactional Pipelines, see “Transactional Versus Nontransactional Pipelines.”

How to Include the Pipeline Session in a Transaction

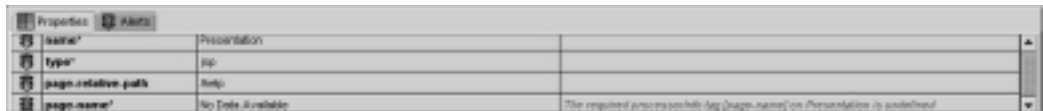
When the Include Pipeline Session in Transaction toggle button in the Pipeline Editor's toolbar is selected, the Pipeline Session will be included in the transaction. Alternatively, when the Include Pipeline Session in Transaction toggle button is deselected, the Pipeline Session will not be included in the transaction. The Include Pipeline Session in Transaction toggle button is not enabled unless the Make Pipeline Transactional toggle button is in the selected position.

Note: For more information about including Pipeline Sessions in transactions, see “Including Pipeline Sessions in Transactions.”

Using the Properties Editors

As shown in Figure 3-9, a Properties Editor is the portion of the Webflow or Pipeline Editor that is located directly below the canvas.

Figure 3-9 Example of a Properties Editor



The Properties tab in a Properties Editor contains information about the Webflow component currently selected. Properties for a component are displayed in two columns: one for the name and one for the value. To create a valid Webflow, you must fill in the required properties for each node in each namespace that comprises the Webflow. (Required properties are marked with an asterisk.) To help you accomplish this, this section includes information on the following:

- Viewing Component Properties
- Description of Webflow Component Properties
- Modifying Component Property Values

The third column in the Properties tab displays messages about the selected Webflow component if it is invalid, which should help you troubleshoot problems as you work. However, this column displays messages for the listed properties only. That is, the node may be invalid not because a property value is defined incorrectly, but because the node is missing a necessary transition. Adding that transition would cause another property to be displayed in the Properties Editor for the node. Therefore, to find a complete list of validation messages for the node, use the Validate the Selected Node button. For more information, see “How to Validate the Selected Node” on page 3-32.

Note: For more information about the Webflow and Pipeline Editors’ validation features, see “About the Webflow and Pipeline Editors’ Validation Features” on page 3-45.

Viewing Component Properties

To view the properties for a particular Webflow component, click the Selection Tool (see Table 3-3 and Table 3-4), then click the component. The properties for the selected Webflow component are displayed in the Properties Editor.

Description of Webflow Component Properties

The properties displayed in the Properties Editor depend on which Webflow component is selected. Table 3-7 lists the properties that should typically be specified for each Webflow component, and provides examples of property values.

Note: Required properties are marked with an asterisk (*).

Table 3-7 Webflow Component Properties

Component	Description of Properties	Example
Presentation Node <i>and</i>	Name* : The name (label) of the Presentation Node that is displayed in the Editor canvas. Initially defaults to the Page-name value.	Name: <code>item</code>
Extension (Custom) Presentation Node	Type* : The type of page, which can be HTML, JSP, Java servlet, or an Extension (Custom) Presentation Node. Page-relative-path* : The path to the Page-name, relative to the top of the Web application. The Page-relative-path property is only available when the type is JSP or HTML. For a servlet, this property is called Request-uri-path. Page-name* : The filename for the Presentation Node. Page-name is only a required property when the Type is JSP or HTML.	Type: <code>jsp</code> Page-relative-path: <code>/browse</code> Page-name: <code>item.jsp</code>
	Note: There will be an Event property for each transition originating from the node. Depending on how the Extension (Custom) Presentation Node is defined, there may be different properties.	

3 Using the Webflow and Pipeline Editors

Table 3-7 Webflow Component Properties (Continued)

Component	Description of Properties	Example
Wildcard Presentation Node	<p>Name*: The name (label) of the Wildcard Presentation Node that is displayed in the Editor canvas. Initially defaults to the Event value.</p> <p>Type*: The type of event, which can be HTML, JSP, a Java servlet, or an Extension (Custom) Presentation Node.</p> <p>Event*: The global event for the Wildcard Presentation Node. Initially defaults to the Name value.</p>	<p>Name: HelpWildcardLink</p> <p>Type: jsp</p> <p>Event: link.help</p>
Input Processor Node	<p>Name*: The name (label) of the Input Processor to be invoked by the Webflow.</p> <p>Class-name*: The package name for the Input Processor class, relative to the Web application's directory.</p> <p>Note: There will be Event or Exception properties from each transition originating from the node.</p>	<p>Name: itemIP</p> <p>Class-name: examples.petflow.ip.ItemIP</p> <p>Exception: com.bea.p13n.appflow.exception.ProcessingException</p> <p>Event: success</p>
Pipeline Node	<p>Name*: The name (label) of the Pipeline to be invoked by the Webflow.</p> <p>Pipeline-name*: The name of the Pipeline within the specified namespace.</p> <p>Pipeline-namespace: The name of the Pipeline namespace in which the Pipeline with Pipeline-name resides.</p> <p>Note: If dictated by the selected Pipeline-name, the Editor will automatically add connection ports to the node for events and exceptions. You must specify a Pipeline-namespace first, then the Pipeline-name.</p>	<p>Name: updateCart</p> <p>Pipeline-name: updateCart</p> <p>Pipeline-namespace: main</p> <p>Event: success</p> <p>Exception: com.bea.p13n.appflow.exception.PipelineException</p>

Table 3-7 Webflow Component Properties (Continued)

Component	Description of Properties	Example
Wildcard Processor Node	<p>Name*: The name (label) of the Wildcard Processor Node that is displayed in the Editor canvas. Initially defaults to the Exception value.</p> <p>Type*: The type of processor, which can be an Input Processor Node, a Pipeline Node, or an Extension (Custom) Processor Node.</p> <p>Exception*: The global exception for the Wildcard Processor Node. Initially defaults to the Name value.</p>	<p>Name: MyWildException</p> <p>Type: inputprocessor</p> <p>Exception: com.bea.pl3n.appflow.exception.ProcessingException</p>
Extension (Custom) Processor Node	<p>Name*: The name (label) of the Extension (Custom) Processor Node that is displayed in the Editor canvas.</p> <p>Type*: The type associated with the Extension (Custom) Processor Node.</p> <p>Note: There will be additional properties for the Extension (Custom) Processor Node, depending on how it is defined.</p>	<p>Name: customLayout</p> <p>Type: layoutManager</p>
Proxy Node	<p>Referent-namespace*: The name of the namespace in which the node actually exists.</p> <p>Entity Name*: The name (label) of the node entity to which the Proxy Node refers.</p>	<p>Referent-namespace: order</p> <p>Entity Name: checkoutShoppingCart</p>
Event	<p>Name*: The name of an event transition on a node. For Presentation Nodes, this can be <code>link</code> or <code>button</code>, and the name of the link or button, separated by dot notation. For Processor Nodes, the name is simply the return object.</p> <p>Note: The number of event properties you must specify depends on how many event transitions you add to a node.</p>	<p>Name: <code>link.logout</code> (Presentation Node); <code>success</code> (Processor Node)</p>
Exception	<p>Name*: The package name to the exception class, relative to the Web application's directory.</p>	<p>Name: com.bea.pl3n.appflow.exception.ProcessingException</p>
Pipeline Component	<p>Component*: The name of the Pipeline Component.</p> <p>Event*: The event transition on the node. For Pipeline Component Nodes, the event is simply the return object</p>	<p>Component: GetProductPC</p> <p>Event: success</p>

Table 3-7 Webflow Component Properties (Continued)

Component	Description of Properties	Example
Abort Exception Node	<p>Exception behavior: The behavior that is to be executed when there is a fatal exception in the Pipeline.</p> <p>Note: The value for this property cannot be modified if the Abort Exception Node is the destination of transitions. If no transitions are specified, the Pipeline’s default behavior can be specified by setting this property to Abort or Continue. Pipeline processing will either stop or continue if an exception is thrown by a Pipeline Component.</p>	Exception behavior: abort

Modifying Component Property Values

To modify the value of a Webflow component’s property in the Properties Editor, you must first be able to view the properties. To view a component’s properties, follow these steps:

1. Click the Selection Tool (see Table 3-3 and Table 3-4).
2. Click the component.
3. Click the Properties tab. The properties for the selected Webflow component will be displayed in the Properties Editor.

The way you modify the value of a Webflow component’s property depends on the property itself. A single-click of the mouse button in an property value field may:

- Allow you to type in a new value. If this is the case, type the new value and then press Enter.
- Allow you to select a predefined value from a drop-down list. If this is the case, just click the new value to select it.

Note: Be sure to save the changes to your Webflow or Pipeline using the E-Business Control Center (EBCC). For more information about saving files in the EBCC, see “Saving Files” under “Working with Files” in the *Guide to Using the E-Business Control Center* documentation.

Migrating An Existing Webflow

If you used a previous release of Webflow, you will notice that the format of the Webflow and Pipeline configuration files has changed. That is, the `webflow.properties` and `pipeline.properties` files have been converted to XML files. The new Webflow and Pipeline XML configuration files can be created and modified using the E-Business Control Center (EBCC)'s graphical Webflow and Pipeline Editors.

To use the Webflow and Pipeline Editors, you will first need to migrate your existing Webflow and Pipeline properties files to the new XML format. To do this, use the Data Migrator Tool as described in the *Migration Guide*.

The migration of your `webflow.properties` and `pipeline.properties` files will result in two XML files: `main.wf` and `main.pln`, where `main` is the default namespace name assigned by the Migrator Tool. You will find these files in the root of the destination directory, which is defined by the `data_dst_dir` variable in the `migration_install.properties` file. (See the *Migration Guide* for details.)

Once you copy these files to the appropriate place (as explained in the *Migration Guide*), you can use the Webflow and Pipeline Editors to divide your Webflow or Pipelines into multiple namespaces. (See "Opening Files" in the *Guide to Using the E-Business Control Center* documentation for detailed instructions.)

Note: Because there is no prior graphical layout for the Webflow components, the Webflow and Pipeline Editors automatically display the components in a way that preserves their relationships. You may also want to clean up the layout.

Creating or Modifying a Webflow: Breadth-First Versus Depth-First

There are two different approaches you can take to create or modify a Webflow. These approaches may depend on how you prefer to work, or the amount of information you currently have about your Web application. These approaches are:

- **Breadth-first:** Create and lay out all Presentation and Processor Nodes on the Webflow Editor canvas, using the default names (labels). Once all Presentation and Processor Nodes are on the canvas, go back and fill in the properties for each node (except for Pipeline Nodes), using the Properties Editor. Then connect the nodes with event and/or exception transitions.

Create and lay out Pipeline Component Nodes on the Pipeline Editor canvas, using the default names (labels). Once all Pipeline Components are on the canvas, go back and fill in the properties for each node, using the Properties Editor. (You may want to launch the Pipeline Component Editor to create or modify individual Pipeline Components.) Then connect the nodes with event and/or exception transitions.

Return to the Webflow Editor to specify which Pipeline your Pipeline Nodes reference, using the Properties Editor.

- **Depth-first:** Create a Presentation or Processor Node and fill in all the properties for that node, using the appropriate combination of Editors (that is, Properties Editor, Pipeline Editor, and/or Pipeline Component Editor). Then, do the same for another node. Once you have two complete nodes, connect them with event and/or exception transitions. Continue to create complete nodes (that is, nodes for which all properties are filled in), and connect the completed nodes to the existing Webflow with transitions.

A breadth-first approach will temporarily cause individual Webflow components in your namespace be marked as invalid. Invalid nodes are marked with a red and white square status icon, which serves as a reminder that you must fill in the properties of each node in the namespace or Pipeline for your Webflow to be valid, regardless of when you decide to do it. All nodes placed on an Editor canvas are initially invalid.

Notes: Although your Webflow namespaces and Pipelines may be invalid, you may still save them and continue working on them later. For more information, see “Saving Invalid Webflows” on page 3-47. For more information about troubleshooting invalid Webflows, see “About the Webflow and Pipeline Editors' Validation Features” on page 3-45.

About the Webflow and Pipeline Editors' Validation Features

In addition to making an application's Webflow easier to visualize, the Webflow and Pipeline Editors are also beneficial because of their built-in validation features. When you first start the Webflow and Pipeline Editors, the Webflow namespace or Pipeline is automatically validated and any messages are shown in the Properties Editor. Additionally, the validators continually work behind the scenes as you create or modify Webflow namespaces and Pipelines. This section explains more about the Editors' validation features, and includes:

- Validation Error Messages in a Properties Editor
- What Do the Editors Validate?
- Saving Invalid Webflows

Caution: If the underlying XML in your Webflow or Pipeline configuration files has been corrupted (perhaps as the result of hand-editing), the Webflow and Pipeline Editors may not be able to complete the validation process.

Validation Error Messages in a Properties Editor

The Alerts section of a Properties Editor (shown in Figure 3-10) is the primary location of validation error messages for your Webflow namespace or Pipeline. Since validation is automatically run on a Webflow namespace or Pipeline when you open an Editor, you can get a sense of any problems before you start working. If there are validation error messages, the traffic light icon on the Alerts tab will appear red. If there are validation warnings, the traffic light icon will appear yellow. (Warnings typically occur when you redefine an event or exception on a wildcard node.) Click the Alerts tab to see the detailed messages. Click the Alerts tab to see the detailed messages.

Figure 3-10 Alerts in a Properties Editor



As you are working, the validation feature will continue to run, causing the Alerts section to collect a running log of validation error messages. To clear the alerts that have collected, click Clear.

After you clear alerts, you may choose to validate a particular node or the entire Webflow namespace or Pipeline again. For instructions on how to accomplish these tasks, see “How to Validate the Selected Node” on page 3-32 or “How to Validate All Nodes” on page 3-33.

Notes: In addition to the validation error messages shown in the Alerts section, messages for individual Webflow component properties may also appear in the third column of a Properties Editor’s Properties section, as explained in “Using the Properties Editors” on page 3-38.

When there is a validation error, the Webflow namespace or Pipeline will also be marked as invalid in the E-Business Control Center (EBCC) Explorer window. Validation warnings, however, will not be marked as invalid. For more information about the EBCC Explorer’s status icons, see “About the E-Business Control Center Interface” in the *Guide to Using the E-Business Control Center* documentation

What Do the Editors Validate?

The Webflow and Pipeline Editor's validation features check:

- The properties of components within the Webflow. All required properties must be filled in using the Properties Editor and must contain valid values. In other words, the validation feature verifies that the grammar in the Editor-generated XML files is in accordance with the provided XML Schema Definition (XSD).
- That the entities (for example, JSPs or Java classes) represented by the Presentation and Processor Nodes in a Webflow actually exist within the Web or enterprise application and are deployed on the server.
- That all event and exception transitions have valid destination nodes.
- That a Pipeline used in a Web application's Webflow exists in the associated enterprise application (that is, outside of the Web application).

The Webflow and Pipeline Editor's validation features do not verify:

- That the entities (for example, JSPs or Java classes) represented by the Presentation and Processor Nodes can actually handle the particular events or can throw the exceptions specified in the Webflow.
- That all the events or exceptions that can be generated by the entities (for example, JSPs or Java classes) are specified in the Webflow.

Saving Invalid Webflows

You can save the files associated with a Webflow namespace or a Pipeline using the E-Business Control Center (EBCC), even if they are not yet in a valid state. (See "Working with Files" in the *Guide to Using the E-Business Control Center* documentation for more information.) However, in order for your application's Webflow data to be synchronized, the entire Webflow (that is, all the Webflow namespaces and Pipelines that comprise the Webflow) should be valid.

Notes: The EBCC's Explorer window indicates when a Webflow namespace or Pipeline is complete by removing the red and white square icon associated with the object. For more information about the EBCC Explorer's status icons, see "About the E-Business Control Center Interface" in the *Guide to Using the E-Business Control Center* documentation.

For more information about Webflow data and synchronization, see "Synchronizing Webflow Data for Your Application" on page 3-48 or the *Deployment Guide*.

Synchronizing Webflow Data for Your Application

The Webflow and Pipeline XML configuration files you create using the Webflow and Pipeline Editors represent just one piece of application data that must eventually be synchronized. You synchronize all application data at once, using the E-Business Control Center (EBCC). Thus, new or modified Webflow and Pipeline XML configuration files will be synchronized along with other application data. For instructions on how to synchronize application data using the EBCC, see "Synchronizing Application Data" in the *Deployment Guide*.

Warning: If you and other developers concurrently synchronize data to a single enterprise application, it is possible to overwrite each others work or to create sets of changes that are incompatible and difficult to debug. To prevent this possibility, synchronize to separate instances of your application. For more information on how to set up your development environment, see "Milestone 4: Set Up a Development Site" in the "Workflow for Developing an E-Business Web Site" topic of the *Strategies for Developing E-Business Web Sites* documentation.

4 Customizing and Extending Webflow

Although the applications in the WebLogic Portal product suite build upon a solid Webflow implementation that you can use when building your own applications, the Webflow mechanism has also been designed for easy customization and extensibility. For example, if your organization dictates the use of a new business process, the Java/EJB developers on your project team can utilize the existing Webflow infrastructure to create and incorporate these components into the system. Once created, the new Webflow components can be reused like any of the prebuilt components BEA provides. This topic describes some concepts that will allow you to customize and extend Webflow to meet your specific requirements.

Note: The diagrams in this topic are used primarily to illustrate dependencies in the classes. Please be sure to visit the *Javadoc* for the most up-to-date information.

This topic includes the following sections:

- Pipeline Session Internals
 - Managing the Pipeline Session
 - Property Scoping
 - Serializing Pipeline Session Properties
- Error Handling
 - Non-Runtime and Runtime Processor Exceptions
 - Input Processor and Pipeline Component Exception Handling
 - JavaServer Page (JSP) Exception Handling

- Accessing Exceptions and Exception Messages
- Creating New Input Processor or Pipeline Component Exceptions
- Configuring Pipeline Component Exception Fatality
- Creating a New Input Processor
 - How to Create a New Input Processor
 - Input Processor Naming Conventions
 - Input Processors and Statelessness
 - Other Development Guidelines for Input Processors
- Webflow Validators and Input Processors
 - The ValidatedValues Interface
 - Special Validation Exceptions
 - Creating a Custom Validator
- Creating a New Pipeline Component
 - How to Create a New Pipeline Component
 - Pipeline Component Naming Conventions
 - Implementation of Pipeline Components as Stateless Session EJBs or Java Objects
 - Stateful Versus Stateless Pipeline Components
 - Transactional Versus Nontransactional Pipelines
 - Including Pipeline Sessions in Transactions
 - Other Development Guidelines for Pipeline Components
- Extending Webflow by Creating Extension Presentation and Processor Nodes
 - How to Create an Extension Presentation Node
 - How to Create an Extension Processor Node
 - Making Your Extension Presentation and Processor Nodes Available in the Webflow and Pipeline Editors
- Webflow Internationalization

Pipeline Session Internals

Although Pipelines and Pipeline Components are reusable, they must relate to a particular visitor's experience on your Web site to make their execution relevant. For this reason, Pipeline Components always operate on a Pipeline Session. This section provides you with more detailed information about the Pipeline Session, and provides instructions for configuring the Pipeline Session to meet your specific requirements. Specifically, this section includes information about:

- Managing the Pipeline Session
- Property Scoping
- Serializing Pipeline Session Properties

Note: It is assumed that you have already read “The Pipeline Session” on page 2-6. If you have not, it is strongly recommended that you do so before continuing with this section.

Managing the Pipeline Session

The Pipeline Session is an interface containing a number of helpful methods. For detailed information about the Pipeline Session interface, see the *Javadoc*.

Accessing the PipelineSession Interface

To access the `PipelineSession` interface from within a JavaServer Page (JSP), use the Pipeline Session JSP tags described in “Pipeline Session Tags” on page 5-19.

If you need to access the `PipelineSession` interface from within an `InputProcessor` class (and your `InputProcessor` class extends the `InputProcessorSupport` class), you can use the static helper methods on the `InputProcessorSupport` class to access the Pipeline Session. Similarly, if you need to access the `PipelineSession` interface from within a Pipeline Component class (and your Pipeline Component class extends the `PipelineComponentSupport` class), you can use the static helper methods on the

`PipelineComponentSupport` class to access the Pipeline Session. See “Overview of the `InputProcessorSupport` Helper Methods” on page 4-5 or “Overview of the `PipelineComponentSupport` Helper Methods” on page 4-6 for more information.

Notes: For more information about the static helper methods on the `InputProcessorSupport` class, see “Creating a New Input Processor” on page 4-15 or the *Javadoc*. For more information about the static helper methods on the `PipelineComponentSupport` class, see “Creating a New Pipeline Component” on page 4-28 or the *Javadoc*.

If your Input Processor class does not extend the `InputProcessorSupport` class, you can also use the `SessionManagerFactory` class to access the `PipelineSession` interface, as shown in the following code fragment:

```
public Object process(HttpServletRequest req, Object
requestContext) throws ProcessingException
{
    PipelineSession pSession = null;

    pSession = SessionManagerFactory.getSessionManager(),
    getPipelineSession(req);

    ...
}
```

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

Setting and Getting Pipeline Session Properties

To set or get properties in the Pipeline Session from within a JavaServer Page (JSP), use the Pipeline Session JSP tags described in “Pipeline Session Tags” on page 5-19.

If you need to set or get properties in the Pipeline Session from within an Input Processor or Pipeline Component class (and your class extends the `InputProcessorSupport` or `PipelineComponentSupport` class, respectively), you can use the static helper methods provided by these support classes. An alternative is to call the Pipeline Session directly using the getter/setter methods, but if you do so, you will need to handle the `InvalidArgumentException` exceptions that the Pipeline Session throws. The following sections briefly describe the methods that are made available by the support classes.

Overview of the InputProcessorSupport Helper Methods

- The method signature for setting Request-scoped properties (attributes) in the Pipeline Session is:

```
public static void setRequestAttribute(String key, Object obj,
String namespace, Object reqContext, PipelineSession pSession)
throws ProcessingException
```

- The method signature for setting Pipeline Session-scoped properties (attributes) in the Pipeline Session is:

```
public static void setSessionAttribute(String key, Object obj,
String namespace, PipelineSession pSession) throws
ProcessingException
```

- The method signature for getting Request-scoped properties (attributes) from the Pipeline Session is:

```
public static Object getRequestAttribute(String key, String
namespace, Object reqContext, PipelineSession pSession) throws
ProcessingException
```

- The method signature for getting Pipeline Session-scoped properties (attributes) from the Pipeline Session is:

```
public static Object getSessionAttribute(String key, String
namespace, PipelineSession pSession) throws ProcessingException
```

Notes: More information about these methods can be located in the *Javadoc*.

The scope of Pipeline Session properties is described in detail in “Property Scoping” on page 4-7.

Overview of the PipelineComponentSupport Helper Methods

- The method signature for setting Request-scoped properties (attributes) in the Pipeline Session is:

```
public static void setRequestAttribute(String key, Object obj,
String namespace, Object reqContext, PipelineSession pSession)
throws PipelineException
```

- The method signature for setting Pipeline Session-scoped properties (attributes) in the Pipeline Session is:

```
public static void setSessionAttribute(String key, Object obj,
String namespace, PipelineSession pSession) throws
PipelineException
```

- The method signature for getting Request-scoped properties (attributes) from the Pipeline Session is:

```
public static Object getRequestAttribute(String key, String
namespace, Object reqContext, PipelineSession pSession) throws
PipelineException
```

- The method signature for getting Pipeline Session-scoped properties (attributes) from the Pipeline Session is:

```
public static Object getSessionAttribute(String key, String
namespace, PipelineSession pSession) throws PipelineException
```

Notes: More information about these methods can be located in the *Javadoc*.

The scope of Pipeline Session properties is described in detail in “Property Scoping” on page 4-7.

Using the Support Classes to Capture Exception Messages

If your Input Processor or Pipeline Component class extends its associated support class, you may also want to use the static helper methods on the `InputProcessorSupport` and `PipelineComponentSupport` classes to capture exception messages from the Pipeline Session and throw an exception from your Input Processor or Pipeline Component with the message set accordingly. An example of how to do this is shown in the following code fragment (taken from an Input Processor class):

```
public Object process(HttpServletRequest req, Object
requestContext) throws ProcessingException
{
    ...

    // Put the category ID in the Pipeline Session
    setRequestAttribute(CATEGORY_ID, category, namespace,
requestContext, pSession);

    ...
}
```

Property Scoping

All properties in the Pipeline Session can have one of two scopes: Pipeline Session scope or Request scope. Pipeline Session and Request scoping differ by how long the property is retained.

Request-Scoped Pipeline Session Properties

When properties are Request-scoped, they are made available in the `HttpServletRequest` and exist only for the life of an HTTP request. In other words, Request-scoped properties are available from the time they are set, up to and including the display of the next JSP. The property is automatically deleted when a new request starts. Therefore, Request scope is useful for temporary objects that will only be needed for one page, and is less expensive than Pipeline Session scope. For example, search results from the Product Catalog are stored as Request-scoped properties. Request-scoped properties should be accessed via the `<webflow:getProperty>` JSP tag or via the appropriate helper method from one of the support classes discussed in “Setting and Getting Pipeline Session Properties” on page 4-4.

Note: More information about the `<webflow:getProperty>` JSP tag can be found in “Pipeline Session Tags” on page 5-19.

In the current release, the Pipeline Session also supports contexting for Request-scoped properties. This is particularly helpful if your Web site uses frames and a visitor navigates quickly. Contexting ensures that the correct Request-scoped property is set during such an interaction.

Pipeline Session-Scoped Pipeline Session Properties

Properties that must live longer than the HTTP request should be specified as Pipeline Session scope, which will cause them to be retained throughout the Web site visitor's HTTP session, but will be a more expensive operation. If you know that a Pipeline Session property is only required for the current request, use the Request scope. Pipeline Session-scoped properties should be accessed via the `<webflow:getProperty>` JSP tag or via the appropriate helper method from one of the support classes discussed in "Setting and Getting Pipeline Session Properties" on page 4-4.

Note: More information about the `<webflow:getProperty>` JSP tag can be found in "Pipeline Session Tags" on page 5-19.

Pipeline Session scope is the default scope for Pipeline Session properties, and will be used if the optional `scope` attribute to the Pipeline Session JSP tags is not specified. Unlike Request-scoped properties, there is no support for contexting of Pipeline Session-scoped properties.

Note: For more information about the support classes' static helper methods for setting and getting Pipeline Session properties, see "Setting and Getting Pipeline Session Properties" on page 4-4.

Serializing Pipeline Session Properties

All properties 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. This is especially relevant in clustered and distributed environments. To assist in debugging, uncomment the `com.bea.p13n.appflow.pipeline.internal.PipelineExecutorImpl`: on line in the `debug.properties` file by removing the # sign. Then, when one of the static helper methods for setting a Pipeline Session property (attribute) is called, the server console will indicate whether that property is serializable or not.

Note: For more information about the support classes' static helper methods for setting and getting Pipeline Session properties, see "Setting and Getting Pipeline Session Properties" on page 4-4.

Error Handling

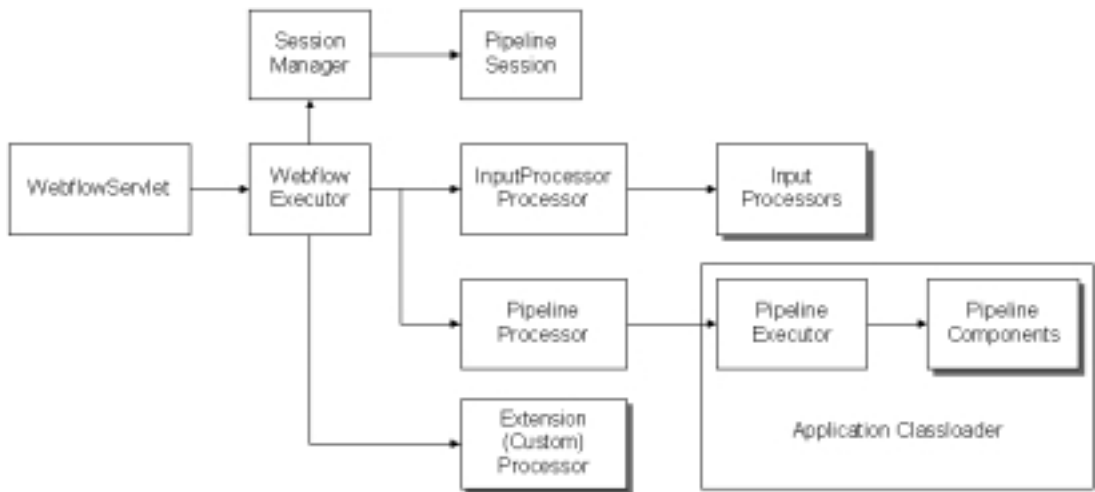
You can configure Webflow to handle any type of exception (as well as superclasses of that exception) by creating one or more `<namespace>.wf` files using the Webflow Editor. To assist you in error handling, this section includes information about:

- Non-Runtime and Runtime Processor Exceptions
- Input Processor and Pipeline Component Exception Handling
- JavaServer Page (JSP) Exception Handling
- Accessing Exceptions and Exception Messages
- Creating New Input Processor or Pipeline Component Exceptions
- Configuring Pipeline Component Exception Fatality

Non-Runtime and Runtime Processor Exceptions

It is important to understand that runtime exceptions are treated slightly different from non-runtime exceptions. Throughout this explanation, it may be helpful to refer to Figure 4-1, which was first introduced in “High-Level Architecture” on page 1-2.

Figure 4-1 Webflow High-Level Architecture



When a processor (Input Processor processor, Pipeline Processor, or Extension (Custom) Processor) throws an exception, then the Webflow mechanism performs the following steps (in order):

1. If the exception is a non-runtime exception, Webflow logs the exception at the INFO level, with no stack trace. If the exception is a runtime exception, then Webflow logs the exception at the ERROR level with a stack trace.
2. Searches for the specific exception in the exception list of the Processor Node. If found, the Webflow invokes that destination.
3. Searches for a superclass of the exception in the exception list of the Processor Node. If not found, Webflow walks all the way up to the `Exception` exception in an attempt to find and invoke a destination.
4. Searches for the specific exception in the exception list of the Wildcard Processor Node for that node type. If found, the Webflow invokes that destination.
5. Searches for a superclass of the exception in the exception list of the Wildcard Processor Node for that node type. Again, if no destination is found, Webflow walks all the way up to the `Exception` exception.

6. Searches for a configuration error page defined in the current namespace. If no configuration error page is defined in the current namespace, Webflow searches for one in the default namespace (as defined in the `WEB-INF/web.xml` file). If no configuration error page is defined, then Webflow has no choice but to 500 the user.

Input Processor and Pipeline Component Exception Handling

The Webflow mechanism knows nothing about Input Processors or Pipelines, as these are just Extension (Custom) Processors. This section contains information about how the Input Processor processor and the Pipeline Processor handle specific exceptions for Input Processors and Pipeline Components.

Input Processor Exceptions

When an Input Processor throws an exception, the Input Processor processor performs the following checks for the exception (in this order):

- If the exception is of type `InstantiationException`, `ClassNotFoundException`, or `ClassCastException`, log the exception at the `ERROR` level with a stack trace, and throw the exception back to the Webflow.
- If the exception is of type `RuntimeException`, throw the exception back to the Webflow because this exception can be handled by the normal Webflow configuration. `RuntimeException` exceptions will be logged with a stack trace.
- If the exception is of type `Exception`, throw the exception back to the Webflow because this exception can be handled by the normal Webflow configuration. `Exception` exceptions will be logged at the `INFO` level, without a stack trace.

Pipeline Component Exceptions

When a Pipeline Component throws an exception, the Pipeline Executor performs the following checks for the exception (in this order):

- If the exception is of type `PipelineException`, or a subclass of `PipelineException`:

- If this exact exception (not superclass) is defined in the abort exception list for this component, or the default exception behavior is to abort, roll back the transaction (if transactional) and throw the same exception back to the Webflow to be handled by the normal configuration.
- If the exception is defined as a branch component, log the exception at the `INFO` level without a stack trace, and branch to that component.
- If the exception is of type `RemoteException`, log the exception at the `ERROR` level with a stack trace, roll back the transaction (if transactional), and throw the exception back to the Webflow.

Note: For Pipeline Components that are implemented as EJBs, any `RuntimeException` exceptions thrown by the Pipeline Component will also cause the container to throw a `RemoteException`.

- If the exception is of type `ConfigurationException` (thrown internally if there is a bad Pipeline configuration), log the exception at the `ERROR` level without a stack trace, roll back the transaction (if transactional), and throw the exception back to the Webflow.
- If the exception is of type `PipelineSystemException` (a runtime exception thrown internally or by a Pipeline Component not implemented as an EJB), log the exception at the `ERROR` level with a stack trace, roll back the transaction (if transactional), and throw the exception back to the Webflow.
- If the exception is of type `IllegalAccessException` (caused by a security violation on a Pipeline Component implemented as an EJB), throw a `PipelineSystemException` exception back to the Webflow with an embedded `IllegalAccessException` exception, and roll back the transaction (if transactional).
- If the exception is of type `IllegalArgumentException`, log the exception at the `ERROR` level with a stack trace, roll back the transaction (if transactional), and throw the exception back to the Webflow.
- If the exception is of type `InstantiationException`, throw a `PipelineSystemException` exception back to the Webflow with an embedded `IllegalAccessException` exception and roll back the transaction (if transactional).
- If the exception is of type `ClassNotFoundException`, throw a `PipelineSystemException` back to the Webflow with an embedded

`IllegalAccessException` exception and roll back the transaction (if transactional).

- If the exception is of type `ClassCastException`, log the exception at the `ERROR` level with a stack trace, roll back the transaction (if transactional), and throw the exception back to the Webflow.

JavaServer Page (JSP) Exception Handling

If the destination JavaServer Page (JSP) throws an exception while being rendered, there is nothing Webflow can do. This must be handled through normal JSP exception processing, with a JSP error page, or caught within the JSP itself.

Accessing Exceptions and Exception Messages

You can retrieve the original exception and/or the exception's message using the `<webflow:getException>` JSP tag. This tag can be inlined or the `id` attribute can be supplied to return the actual exception.

Note: For more information about the `<webflow:getException>` JSP tag, see “`<webflow:getException>`” on page 5-25.

Creating New Input Processor or Pipeline Component Exceptions

All Input Processors must throw the `ProcessingException` exception, one of its subclasses, or any other exception that Webflow is configured to handle.

Pipeline Components may throw a `PipelineException` exception, one of its subclasses, or any other exception that Webflow is configured to handle to signify that execution of the component has failed. Depending on whether or not the exception is configured as fatal or nonfatal, it is possible that no further Pipeline Components may be executed. If the Pipeline is transactional, the transaction will be rolled back.

Notes: See the related sections “Configuring Pipeline Component Exception Fatality” on page 4-14 and “Transactional Versus Nontransactional Pipelines” on page 4-31. For more information about the `ProcessingException` or `PipelineException` exception, see the Javadoc.

Configuring Pipeline Component Exception Fatality

For those who used prior implementations of Webflow, you may recall that Pipeline Components could throw `PipelineFatalException` and `PipelineNonFatalException` exceptions. In this release, there is a general `PipelineException` exception that you can use or extend. Fatality of Pipeline exceptions depends upon how they are handled. Fatal Pipeline exceptions (also called **abort exceptions**) are those that are part of the abort exception list and therefore must be handled in the Webflow. Nonfatal Pipeline exceptions (called **branch exceptions**) are those that are handled inside the Pipeline by rerouting through another Pipeline Component.

In the Pipeline Editor, branch exceptions are shown as transitions between Pipeline Component Nodes, while abort exceptions are shown as transitions between a Pipeline Component Node and the Abort Exception Node.

If there are no exception transitions from any Pipeline Component Nodes to the Abort Exception Node, a default exception will be used when any Pipeline Component in that Pipeline throws an exception that is not handled as a branch exception. In this case, you can set the default exception behavior for the Abort Exception Node to either continue or abort the Pipeline using the Properties Editor.

A default abort exception of Continue is handled by ignoring the exception, continuing on to the next Pipeline Component, and passing a success event to the Pipeline Node in the Webflow namespace. A default abort exception of Abort is handled by throwing the unhandled exception to the Pipeline Node in the Webflow namespace. If the Pipeline is transactional, the transaction is rolled back to what it was before the Pipeline was started and then the exception is thrown.

Note: For more information about the Pipeline Editor, see Chapter 3, “Using the Webflow and Pipeline Editors.”

Creating a New Input Processor

As you may recall from “Input Processors and Pipelines” on page 2-5, an Input Processor is one type of processor node that is typically used in a Webflow for form validation. BEA has developed a number of Input Processors that are packaged with the WebLogic Portal product suite, which you may want to reuse in your own applications. However, you may also want to create your own Input Processors for use in your applications’ Webflows.

How to Create a New Input Processor

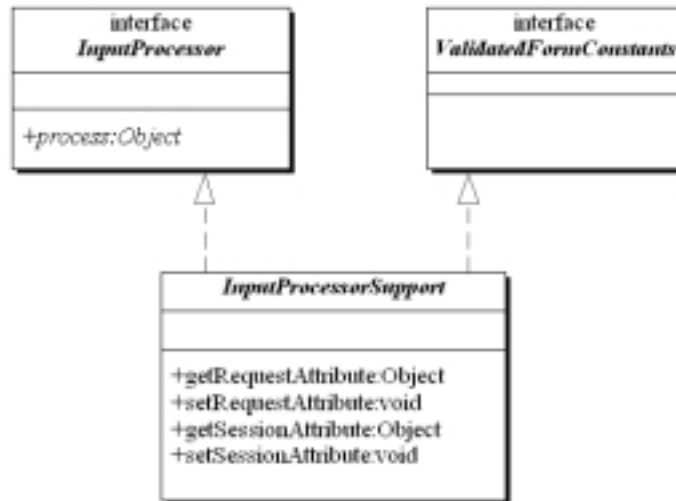
To create a new Input Processor, you must implement the `com.bea.p13n.appflow.webflow.InputProcessor` interface by providing the details of the `process()` method, which has the following method signature:

```
public Object process(HttpServletRequest req, Object  
requestContext) throws ProcessingException
```

Note: All classes located in `PORTAL_HOME\applications\petflowApp\petFlow\WEB-INF\src\examples\petflow\ip` implement the `InputProcessor` interface, and can be viewed as examples.

Alternately, your new Input Processor can extend the `com.bea.p13n.appflow.webflow.InputProcessorSupport` class, as shown in Figure 4-2. As its name implies, this abstract class allows you to use static helper methods that provide additional support for an Input Processor. If your new Input Processor class must extend some other class, however, you will not be able to take advantage of the `InputProcessorSupport` class.

Figure 4-2 Relationship of the InputProcessorSupport Class



Note: For more information about the `InputProcessor` interface, the `InputProcessorSupport` class, or the `ValidatedFormConstants` interface, see the Javadoc.

When you are using the Webflow Editor to specify the properties for an Input Processor node you placed on the canvas, simply include the class name of your newly created Input Processor in the appropriate field. There are no additional activities you need to perform to make your Input Processor work with the existing Webflow mechanism.

Note: For more information about using the Webflow Editor, see Chapter 3, “Using the Webflow and Pipeline Editors.”

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`. This naming convention should help you keep track of Input Processors more easily.

Input Processors and Statelessness

Like Pipeline Components, Input Processors are multi-threaded. To ensure that your Input Processors are thread safe, 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 for Input Processors

Execution of business (application) logic should typically not be done within Input Processors. Specifically, Input Processors should not call Enterprise JavaBeans (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, such logic could not be transactional, and would defeat a primary purpose of the Webflow architecture. By separating business logic from the presentation logic, your Web 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.

Webflow Validators and Input Processors

An Input Processor is essentially a Webflow component that works to validate data entered by a Web site visitor in an HTML form. Although you are not required to use Input Processors in your Webflows, use of Input Processors may increase developer productivity and will automatically preserve information the visitor originally entered while clearly identifying fields that require attention. Behind-the-scenes, Input Processors make use of a `ValidatedValues` interface to accomplish these tasks.

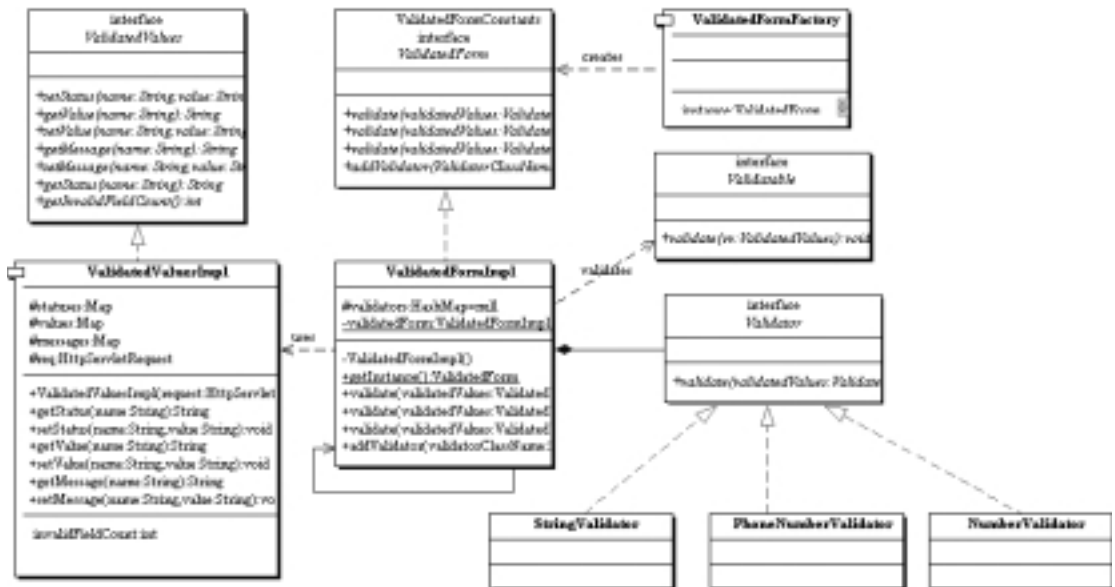
This section includes information about the following:

- The `ValidatedValues` Interface
- Special Validation Exceptions
- Creating a Custom Validator

The ValidatedValues Interface

The ValidatedValues interface (shown in Figure 4-3) allows a Java/EJB developer who writes an Input Processor to report the status of processed form fields back to a HTML/JSP developer. The HTML/JSP developer receives the status of each form field in their Web pages via the `<webflow:validatedForm>` JSP tag. For more information about the `<webflow:validatedForm>` JSP tags, see Chapter 5, “Webflow JSP Tag Library Reference.”

Figure 4-3 Validation Class Diagram



The ValidatedValues interface uses the imported `javax.servlet.http.HttpServletRequest`. The public methods used to convey the status of the validation through the `<webflow:getValidatedValue>` and `<webflow:setValidatedValue>` JSP tags are shown in Table 4-1.

Table 4-1 ValidatedValues Public Methods

Method Signature	Description
<code>public String getStatus (String name)</code>	Retrieves the status for the specified field, which may be unspecified, invalid, or valid.
<code>public void setStatus (String name, String value)</code>	Sets the status for the specified field.
<code>public Object getValue (String name)</code>	Retrieves the current value for the specified field.
<code>public String getValueAsString (String name)</code>	Retrieves the current value of the field as a string. This method will return an empty string as opposed to null.
<code>public void setValue (String name, String value)</code>	Sets the value for the specified field.
<code>public String getMessage (String name)</code>	Retrieves the message for the field.
<code>public void setMessage (String name, String value)</code>	Sets the message for the field.
<code>public int getInvalidFieldCount ()</code>	Return the number of HTML form fields that did not validate.
<code>public void setInvalidFieldCount(int count)</code>	Set the number of HTML form fields that did not validate.
<code>public void incInvalidFieldCount()</code>	Increment the number of HTML form fields that did not validate.
<code>public HttpServletRequest getHttpServletRequest()</code>	Obtain a reference to the <code>HttpServletRequest</code> .

Note: For more information about the `<webflow:getValidatedValue>` and `<webflow:setValidatedValue>` JSP tags, see Chapter 5, “Webflow JSP Tag Library Reference.” Please also be sure to visit the *Javadoc* for the most up-to-date information about the `ValidatedValues` class.

Validation Example

Listing 4-1 uses each of the validated form tags in an HTML form page that gathers some information about a Web site's visitors, which we will later validate using an Input Processor class and the `ValidateValues` interface.

Listing 4-1 The JSP Form Fields Requiring Validation

```
<%@ page import="com.bea.pl3n.appflow.webflow.WebflowJSPHelper" %>
<%@ page import="java.util.Map" %>

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

<html>
  <head>
    <title>Webflow Demo</title>
  </head>

  <% String validStyle="background: white; color: black; font-family: Arial"; %>
  <% String invalidStyle="background: white; color: red; font-style: italic"; %>

  <!-- If there was an InvalidFormDataException thrown display the message -->

  <font size="5" color="green"><webflow:getException/></font>
  <br>

  <webflow:validatedForm event="button.go" applyStyle="message"
    messageAlign="right" validStyle="<%=validStyle%>"
    invalidStyle="<%=invalidStyle%>" unspecifiedStyle="<%=validStyle%>" >

    <p>Username:
    <webflow:text name="username" value="start" size="15" maxlength="30"
      htmlAttributes="onMouseOver=\\\"self.status='User ID';return true\\\"" />

    <br>Password:
    <webflow:password name="password" size="15" retainValue="true" maxlength="30"
      htmlAttributes="onMouseOver=\\\"self.status='Secret Password';return true\\\""
      />

    <br>Number:
    <webflow:text name="number" size="15" maxlength="30"/>

    <br>Phone:
    <webflow:text name="phone" size="15" maxlength="15"/>

    <br>E-mail Phone:
    <webflow:text name="email" size="15" maxlength="30"/>
```



```
<br>Gender:
<webflow:radio name="gender" checked="true" value="M" />Male
<webflow:radio name="gender" value="F" />Female

<br>Animals(s) You Like:
<webflow:checkbox name="cat" value="cat" />Cat<br>
<webflow:checkbox name="dog" checked="true" value="dog" />Dog<br>
<webflow:checkbox name="bird" value="bird" />Bird

<p>Comment:
<webflow:textarea name="comment" cols="20" rows="3" value="hello" />

<br><br>Hobbies:
<webflow:select name="hobbies" size="3" multiple="true">
  <webflow:option value="Running" />Running
  <webflow:option value="Skiing" />Skiing
  <webflow:option value="Motocross" />MotoX
  <webflow:option value="Rugby" />Rugby
</webflow:select>

<br>Pet You Own:
<webflow:select name="pets" size="1" >
  <webflow:option value="dog" selected="true" />Dog
  <webflow:option value="cat" selected="false" />Cat
  <webflow:option value="bird" selected="false" />Bird
</webflow:select>

<br>

<input type="submit" name="Submit" />

</webflow:validatedForm>

<br>
</html>
```

Note: For more information about the `<webflow:validatedForm>` JSP tags, see Chapter 5, “Webflow JSP Tag Library Reference.”

The Input Processor shown in Listing 4-2 can be used to validate the form shown in Listing 4-1.

Listing 4-2 Example of an Input Processor process() Method

```
package com.bea.test.DemoIP;

import com.bea.pl3n.appflow.common.PipelineSession;
import com.bea.pl3n.appflow.exception.ProcessingException;
import com.bea.pl3n.appflow.webflow.forms.ValidatedValues;
import com.bea.pl3n.appflow.webflow.forms.ValidatedValuesFactory;
import com.bea.pl3n.appflow.webflow.forms.ValidatedForm;
import com.bea.pl3n.appflow.webflow.forms.ValidatedFormFactory;
import com.bea.pl3n.appflow.webflow.forms.MissingFormFieldException;
import com.bea.pl3n.appflow.webflow.forms.InvalidFormDataException;
import com.bea.pl3n.appflow.webflow.forms.MinMaxExpression;
import examples.petflow.common.PetflowConstants;
import java.util.ArrayList;
import javax.servlet.http.HttpServletRequest;

public class DemoIP extends InputProcessorSupport
{
    public Object process(HttpServletRequest req, Object requestContext) throws
        ProcessingException, InvalidFormDataException
    {
        PipelineSession pSession = null;
        String namespace = null;
        pSession = getPipelineSession(req);
        namespace = getCurrentNamespace(pSession);
        ValidatedValues vValues = ValidatedValuesFactory.getValidatedValues(request);
        ValidatedForm vForm = ValidatedFormFactory.getValidatedForm();

        // Validate the html form, let any missing fields bubble as a
        // MissingFormFieldException

        MinMaxExpression minMax = new MinMaxExpression();

        String username = vForm.validate(vValues, STRING_VALIDATOR, "username",
            minMax.set(4, 20), "User IDs must be greater then 4");

        String password = vForm.validate(vValues, STRING_VALIDATOR, "password",
            minMax.set(4, 20));

        String number = vForm.validate(vValues, NUMBER_VALIDATOR, "number",
            minMax.set(1, 20));

        String phone = vForm.validate(vValues, PHONE_VALIDATOR, "phone", null);

        String email = vForm.validate(vValues, EMAIL_VALIDATOR, "email", null);
    }
}
```

```
String gender = vForm.validate(vValues, "gender");

String cat = vForm.validate(vValues, "cat");
String dog = vForm.validate(vValues, "dog");
String bird = vForm.validate(vValues, "bird");

String comment = vForm.validate(vValues, STRING_VALIDATOR, "comment",
    minMax(0,255));

Collection hobbies = vForm.validateMultiple(vValues, "hobbies", 2, "Must
select at least 2");

String pets = vForm.validate(vValues, "pets");

// Did all fields pass the validation test
if (vValues.getInvalidFieldCount() > 0)
{
    // No, throw the InvalidFormFieldException and let Webflow
    // redirect the user back to the origin JSP, preserving all our input
    throw new InvalidFormataException("Please fix the fields in red and
    resubmit");
}

// If you got here all form fields were entered correctly you probably want to
// populate the Pipeline Session. Use the base class setRequestAttribute method
// to help you out

return success;
}
}
```

Special Validation Exceptions

There are three prewritten validation exceptions you may want to use with your Input Processors:

- `MissingFormFieldException`

The `MissingFormFieldException` exception can be thrown by the `ValidatedForm` class when validating a field, if the field is not in the `HttpServletRequest`. This exception is usually caused by a typo, and therefore a solution could be to have the Webflow redirect to the `missingformfield.jsp` to display the missing field. This response could be configured in a wildcard processor node.

Note: For an example, see `PORTAL_HOME\applications\petflowApp\petflow\error\missingformfield.jsp`, which you will need to modify for a production site.

- `InvalidFormDataException`

The `InvalidFormDataException` exception can be thrown when a form has invalid fields (determined by the `getInvalidFieldCount()` method of the `ValidatedValues` interface). In this case, you may want to have the Webflow redirect to the origin JSP, to have the Web site visitor resubmit their form.

Note: For more information about the methods in the `ValidatedValues` interface, see “The `ValidatedValues` Interface” on page 4-18.

- `InvalidValidatorException`

The `InvalidValidatorException` exception can be thrown by the `ValidatedForm` interface when validating a field, if the validator specified is not a valid validator. Invalid validators are classes that do not implement the `Validator` interface, classes that do not exist, classes that are abstract, or classes that are interfaces.

Creating a Custom Validator

In addition to the validators previously described, you may choose to create validators of your own.

To create a custom validator, you should create a class that implements the `Validator` interface (see Figure 4-3). Other guidelines for creating a custom validator include:

- You can support an `expression` object. An `expression` object is an arbitrary object that the validator uses to obtain additional information, which is then used to validate the parameter. For example, if a `StringValidator` needs to know the minimum and maximum length of a string, the `expression` object may contain two attributes: `min` and `max`.
- You are responsible for updating the value, status, and message in the `ValidatedValues` object. For more information, see the *Javadoc*.
- Because it is multithreaded, your validator cannot have state.
- There is no need to register your validator; it is automatically registered the first time you use it.

Example of a Custom Validator

Listing 4-3 shows an example of a custom validator that validates credit cards.

Listing 4-3 `CreditCardValidator.java`

```
package examples.petflow.ip.validator;

import com.bea.p13n.appflow.webflow.forms.Validator;
import com.bea.p13n.appflow.webflow.forms.ValidatedValues;

import java.util.StringTokenizer;

public class CreditCardValidator implements Validator
{
    /**
     * Validate a credit card number from an HTML Form.
     * The card number is valid for 16 digits
     * All separators "()- ." are first removed
     * @param validatedValues object containing value, status,
```

4 Customizing and Extending Webflow

```
    messages
    * @param key html form parameter name
    * @return expression
    */

    public String validate(ValidatedValues validatedValues, String
    key, Object expression, String message)

    {

    // The value entered in the HTML form is supplied in the
    // ValidatedValues

        String value = validatedValues.getValueAsString(key);
        value = value.trim();

    // Can optionally use the expression object to further refine
    // validation criteria
    // if (expression != null && expression instanceof
    // CreditCardExpression) ...

        StringBuffer buffer = new StringBuffer();

    // This stuff could come from the expression object

        StringTokenizer st = new StringTokenizer(value, "()-.");
        char separator = ' ';

    // Strip any parentheses, spaces, dots and dashes

        while (st.hasMoreTokens()) {
            buffer.append(st.nextToken());
        }

    // Correct length?

        if (buffer.length() != 16) {
            validatedValues.setValue(key, value);
            validatedValues.setStatus(key, STATUS_INVALID);
        }

    // If the user did not supply a message use the default.
    // This can be retrieved from the internationalization message
    // catalog.

        String msg = (message == null) ? "Credit cards must
            have 16 digits" : message;

        validatedValues.setMessage(key, msg);

        return value;
    }
}
```

```
// Make sure we have nothing but digits...

    try
    {
        Long.parseLong(buffer.toString());
    }
    catch (NumberFormatException e)
    {
        validatedValues.setValue(key, value);
        validatedValues.setStatus(key, STATUS_INVALID);

// If the user did not supply a message use the default.
// This can be retrieved from the internationalization message
// catalog.

        String msg = (message == null) ? "Credit card cannot
            contain letters, only digits" : message;

        validatedValues.setMessage(key, msg);

        return value;

    }

// It is the validators responsibility to do any reformatting
// if necessary.

    StringBuffer creditCard = new StringBuffer(19);

    creditCard.append(buffer.substring(0,4));
    creditCard.append(separator);
    creditCard.append(buffer.substring(4,8));
    creditCard.append(separator);
    creditCard.append(buffer.substring(8,12));
    creditCard.append(separator);
    creditCard.append(buffer.substring(12,16));

// Must set the value and the status in the validatedValues
// object.

    value = creditCard.toString();

    validatedValues.setValue(key, value);
    validatedValues.setStatus(key, STATUS_VALID);

    return value;

}
}
```

Creating a New Pipeline Component

As you may recall from “Input Processors and Pipelines” on page 2-5, a Pipeline is one type of processor node that is typically used in a Webflow to execute back-end business logic. Each Pipeline may contain a number Pipeline Components that perform specific tasks. BEA has developed a number of Pipeline Components that are packaged with the WebLogic Portal product suite that you may want to reuse in your own Pipelines. However, you may also want to create your own Pipeline Components to execute your organization’s specific business processes.

How to Create a New Pipeline Component

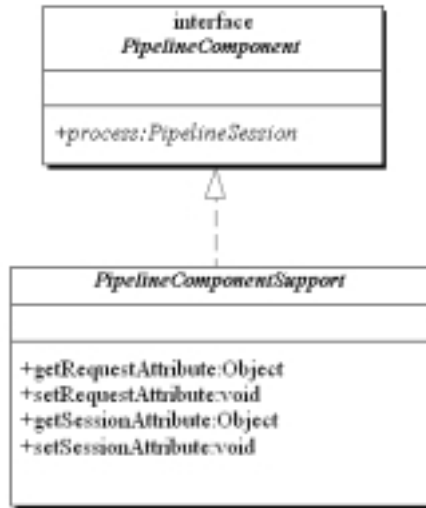
New Pipeline Components must implement the `com.bea.p13n.appflow.pipeline.PipelineComponent` interface and must supply an implementation for the `process()` method. The `process()` method accepts a `PipelineSession` object as a parameter, and returns an updated `PipelineSession` object if the execution is successful, as shown in the following method signature:

```
public PipelineSession process(PipelineSession pipelineSession,  
Object requestContext) throws PipelineException, RemoteException;
```

Note: All of the classes located in the `PORTAL_HOME\applications\petflowApp\petFlow\WEB-INF\src\examples\petflow\ip` directory implement the `InputProcessor` interface, and can be viewed as examples.

Alternately, your new Pipeline Component can extend the `com.bea.p13n.appflow.pipeline.PipelineComponentSupport` class, shown in Figure 4-4. As its name implies, this abstract class allows you to use methods that provide additional support for a Pipeline Component. If your new Pipeline Component class must extend some other class, however, you will not be able to take advantage of the `PipelineComponentSupport` class.

Figure 4-4 Relationship of the PipelineComponentSupport Class



Notes: For more information about the `PipelineComponent` interface or the `PipelineComponentSupport` class, see the Javadoc.

When you are using the Pipeline Editor to specify the properties for a Pipeline Component node you placed on the canvas, simply include the class name of your newly created Pipeline Component in the appropriate field. There are no additional activities you need to perform to make your Pipeline Component work with the existing Webflow mechanism.

Note: For more information about using the Pipeline Editor, see Chapter 3, “Using the Webflow and Pipeline Editors.”

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`. This naming convention should help you keep track of Pipeline Components more easily.

Implementation of Pipeline Components as Stateless Session EJBs or Java Objects

Pipeline Components can be implemented as either stateless session Enterprise JavaBeans (EJBs) or as Java objects. Table 4-2 describes the differences between the two implementations.

Table 4-2 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.
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 Nontransactional Pipelines

If all Pipeline Components within the Pipeline will be invoked under one transaction, you should select the Make This Pipeline Transactional button in the Pipeline Editor. 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 transactional property should be set to `Required`. If the Pipeline's `is-transactional` property is `true` and the participating Pipeline Components (EJBs) have their transaction flag set to `never`, the Pipeline will fail to execute. Similarly, if the Pipeline's `is-transactional` 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 Server Administration Console. A transactional Pipeline containing Pipeline Components implemented as simple Java objects commits the transaction upon success, and rolls back the transaction upon failure. Avoid demarcating transactions within the Pipeline Components themselves.

Including Pipeline Sessions in Transactions

If you include the Pipeline Session in a transaction, any failure in the transaction will cause the Pipeline Session's Pipeline Session-scoped properties to revert back to the state they were in prior to the transaction.

Note: Only Pipeline Session-scoped properties may be included as part of the transaction.

However, it is an expensive operation to serialize (even to memory) the Pipeline Session for each invocation. You will have much better performance if your Pipeline Session is not included in the transaction, so be sure you have a legitimate reason for including it.

Other Development Guidelines for Pipeline Components

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

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

Extending Webflow by Creating Extension Presentation and Processor Nodes

If creating new Input Processors and Pipeline Components to add to those BEA provides out-of-the-box does not meet your needs, you may also choose to extend the Webflow mechanism by creating classes that can be used as Extension (Custom) Presentation or Processor Nodes. Once you create the classes associated with these nodes, you will need to register the new nodes in the `webflow-extensions.wfx` file. This section includes information on how to perform these tasks.

How to Create an Extension Presentation Node

To create an Extension (Custom) Presentation Node, you must first create a class that implements the `com.bea.p13n.appflow.webflow.PresentationNodeHandler` interface. Be sure your class returns a URL that the `WebflowServlet` servlet can forward to. Then, you must register your extension node in the `webflow-extensions.wfx` file so it can be used in the Webflow and Pipeline Editors

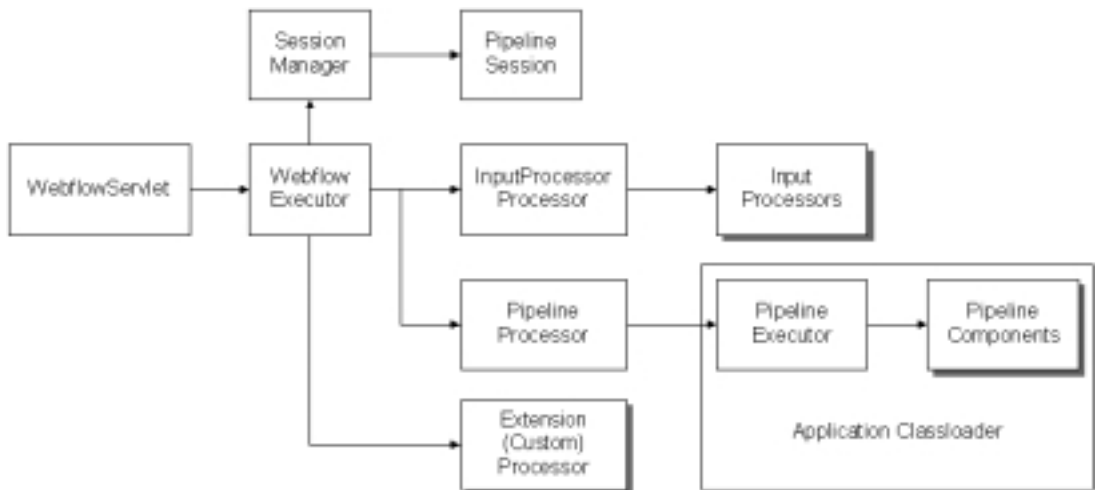
Notes: For more information about the `WebflowServlet` servlet, see “High-Level Architecture” on page 1-2. For instructions on how to register your extension node in the `webflow-extensions.wfx` file, see “Making Your Extension Presentation and Processor Nodes Available in the Webflow and Pipeline Editors” on page 4-35.

The portal application makes use of an Extension (Custom) Presentation Node named `portal`, which you can view as an example. The portal application uses this extension node to indicate to the portal Webflow that the contents of the portlet are to remain unchanged (that is, it indicates that the last URL should be displayed). The `portal` node’s implementation class is `LastContentUrlNodeHandler.java`.

How to Create an Extension Processor Node

Figure 4-5 illustrates the basic architecture of the Webflow mechanism. As the figure shows, you may want to create an Extension (Custom) Processor that functions at the same level as an Input Processor processor or Pipeline Processor. In fact, the Input Processor processor and Pipeline Processor can be thought of as examples of Extension Processors.

Figure 4-5 Webflow High-Level Architecture



Extension (Custom) Processors are processors that your organization (as opposed to BEA) develops for use in your applications' Webflows. Extension Processors may be used to perform activities not currently supported by the Webflow. However, the flow in and out of an Extension Processor is still governed by the Webflow mechanism. Extension Processors are represented as nodes in the Webflow Editor, much like the Input Processor and Pipeline Nodes are, but with a slightly different representation for easy identification.

Note: For more information about using the Webflow Editor, see Chapter 3, "Using the Webflow and Pipeline Editors."

For example, you may want to create an Extension (Custom) Processor that works with the BEA Rules Engine to support different Webflows based on some condition, such as membership in a customer segment. Another, more simple example might be a

layout manager processor that automatically includes a header and footer in your JavaServer Pages (JSPs) when given the page's body content. In fact, we have already created such a processor.

To create an Extension (Custom) Processor Node, you must first create a class that implements the `com.bea.pl3n.appflow.webflow.Processor` interface to define the Extension Processor. Then, you must register your processor in the `webflow-extensions.wfx` file so it can be used in the Webflow and Pipeline Editors.

Note: For instructions on how to register your extension node in the `webflow-extensions.wfx` file, see “Making Your Extension Presentation and Processor Nodes Available in the Webflow and Pipeline Editors” on page 4-35.

Making Your Extension Presentation and Processor Nodes Available in the Webflow and Pipeline Editors

After you have created an Extension (Custom) Presentation or Processor Node, you must make that node available to other developers on your team by registering the node in the `webflow-extensions.wfx` file.

Notes: The `webflow-extensions.wfx` file resides within your Web application's home directory and is therefore scoped to a Web application.

Registering an Extension (Custom) Processor Node will cause its corresponding tool on the Webflow Editor palette to become enabled once you restart the E-Business Control Center (EBCC).

How To Register an Extension Presentation Node

To register an Extension Presentation Node in the `webflow-extensions.wfx` file, follow these steps:

1. Add an `<end-node>` element to the `<end-node-registration>` list.
2. Assign your presentation node a name with the `Name` attribute, and specify the class of the underlying node implementation with the `Node-handler` attribute.

3. Define the input parameters that the class expects upon invocation, using `<node-processor-input>` elements. Give each parameter a Name, and if the parameter is optional, assign the Required attribute a value of `false`.

Note: This information will be used in the Webflow and Pipeline Editors' Property Editors.

4. Save the `webflow-extensions.wfx` file, and restart the E-Business Control Center.

Listing 4-4 provides an example of registering an Extension Presentation Node in the `webflow-extensions.wfx` file.

How To Register an Extension Processor Node

To register an Extension Processor Node in the `webflow-extensions.wfx` file, follow these steps:

1. Add a `<process>` element to the `<process-registration>` list.
2. Assign your processor a name with the Name attribute, and specify the class of the underlying processor implementation with the Executor attribute.
3. Define the input parameters that the class expects upon invocation, using `<node-processor-input>` elements. Give each parameter a Name, and if the parameter is optional, assign the Required attribute a value of `false`.

Note: This information will be used in the Webflow and Pipeline Editors' Property Editors.

4. Save the `webflow-extensions.wfx` file, and restart the E-Business Control Center.

Listing 4-4 provides an example of registering an Extension Processor Node in the `webflow-extensions.wfx` file.

Listing 4-4 Sample `webflow-extensions.wfx` File

```
<?xml version="1.0" encoding="UTF-8" ?>

<webflow-extensions
  xmlns="http://www.bea.com/servers/pl3n/xsd/webflow-extension/1.0"
  xmlns:xsi="http://www.w3.org/2000/10/XMLSchema-instance"
```



```
xsi:schemaLocation="http://www.bea.com/servers/pl3n/xsd/webflow-extension/1.0
webflow-extensions.xsd">

<process-registration>

  <process name="inputprocessor"
    executor="com.bea.pl3n.appflow.webflow.internal.IPProcessor">
    <node-processor-input name="class-name" required="true" />
  </process>

  <process name="pipeline"
    executor="com.bea.pl3n.appflow.webflow.internal.PipelineProcessor">
    <node-processor-input name="pipeline-namespace" required="false" />
    <node-processor-input name="pipeline-name" required="true" />
  </process>

  <process name="layoutmanager"
    executor="examples.petflow.layout.LayoutProcessor">
    <node-processor-input name="header" required="true" />
    <node-processor-input name="content" required="true" />
    <node-processor-input name="footer" required="true" />
  </process>

</process-registration>

<end-node-registration>

  <end-node name="jsp"
    node-handler="com.bea.pl3n.appflow.webflow.internal.GenericNodeHandler">
    <node-processor-input name="page-relative-path" required="false" />
    <node-processor-input name="page-name" required="true" />
  </end-node>

  <end-node name="html"
    node-handler="com.bea.pl3n.appflow.webflow.internal.GenericNodeHandler">
    <node-processor-input name="page-relative-path" required="false" />
    <node-processor-input name="page-name" required="true" />
  </end-node>

  <end-node name="htm"
    node-handler="com.bea.pl3n.appflow.webflow.internal.GenericNodeHandler">
    <node-processor-input name="page-relative-path" required="false" />
    <node-processor-input name="page-name" required="true" />
  </end-node>

  <end-node name="servlet"
    node-handler="com.bea.pl3n.appflow.webflow.internal.ServletNodeHandler">
    <node-processor-input name="request-uri-path" required="true" />
  </end-node>


```

```
<end-node name="portal"
  node-handler="com.bea.portal.appflow.internal.LastContentUrlNodeHandler">
  <node-processor-input name="page-name" required="true"/>
</end-node>

</end-node-registration>

</webflow-extensions>
```

Webflow Internationalization

The WebLogic Portal product suite uses message catalogs that are in XML format for purposes of internationalization. For more information about the message catalog and internationalization, see “Overview of Internationalization for WebLogic Server” in the *BEA WebLogic Server Internationalization Guide*.

5 Webflow JSP Tag Library Reference

The WebLogic Portal product suite includes a set of JSP tags designed to facilitate the development of JSPs using Webflow. Use of these predefined tags will eliminate the need for your JSPs to contain any Java code related to Webflow. This topic explains how to import this set of tags into your Web pages, and describes the purpose of each tag.

This topic includes the following sections:

- Importing the Webflow Tag Library
- URL Creation Tags
 - `<webflow:createWebflowURL>`
 - `<webflow:createResourceURL>`
- Form Tags
 - `<webflow:form>`
- Validated Form Tags
 - `<webflow:validatedForm>`
 - `<webflow:text>`
 - `<webflow:password>`
 - `<webflow:radio>`
 - `<webflow:checkbox>`
 - `<webflow:textarea>`

- <webflow:select>
- <webflow:option>
- Pipeline Session Tags
 - <webflow:setProperty>
 - <webflow:getProperty>
 - <webflow:setValidatedValue>
 - <webflow:getValidatedValue>
 - <webflow:getException>

Note: If you are using Webflow in a portal application, you will need to use specialized versions of the Webflow JSP tags. For more information, see “Portal Management JSP Tag Library Reference” in the *Getting Started with Portals and Portlets* documentation.

Importing the Webflow Tag Library

The Webflow JSP tags are utility tags used to simplify the implementation of JSPs that utilize the Webflow mechanism. To import all of Webflow JSP tags described in this topic, use the following code:

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

URL Creation Tags

The Webflow’s URL tags described in this section are used to create dynamic or static URLs for links and other resources within a JSP.

Note: In the following tables, the Required column specifies if the attribute is required (yes) or optional (no). In the R/C column, C means that the attribute is a Compile time expression, and R means that the attribute can be either a Request time expression or a Compile time expression.

<webflow:createWebflowURL>

The <webflow:createWebflowURL> tag (Table 5-1) is used in a JSP to dynamically create a Webflow URL in a JSP. The Webflow URL may include the protocol, domain name, port, Web application URI, `webflowServlet` URI, and query string.

Table 5-1 <webflow:createWebflowURL>

Tag Attribute	Required	Type	Description	R/C
<code>domainName</code>	No	String	Used to change the domain name or IP address of the URL. This may be used if Webflow is fronted by a proxy server and that server resides on another machine.	R
<code>doRedirect</code>	No	String	Causes the <code>webflowServlet</code> to perform a redirect instead of a forward to a presentation node. Valid values are <code>true</code> and <code>false</code> . The default is <code>false</code> (not to redirect).	R
<code>encode</code>	No	String	Informs Webflow to encode the URL. URLs need to be encoded if you wish to maintain session state and the browser does not accept cookies. Valid values are <code>true</code> and <code>false</code> . The default value is <code>true</code> , as URLs will only need to be encoded if the browser does not accept cookies.	R
<code>event</code>	Yes	String	Webflow will use this in combination with the origin to resolve a destination in the supplied namespace XML file.	R
<code>extraParams</code>	No	String	Used to supply additional request parameters as name/value pairs.	R

Table 5-1 <webflow:createWebflowURL> (Continued)

Tag Attribute	Required	Type	Description	R/C
httpsInd	No	String	<p>Informs Webflow to calculate the protocol or use HTTPS or HTTP. Therefore, valid values are <code>calculate</code>, <code>http</code>, and <code>https</code>. The default value is <code>http</code>.</p> <p><code>Calculate</code> will yield HTTPS if any node in the origin/event branch chain list is matched under the <code>HTTPS_URL_PATTERNS</code> context parameter in the application's <code>WEB-INF/web.xml</code> file. <code>Calculate</code> is more dynamic and expensive, but if the protocol needs to be forced you can specify it here.</p>	R
namespace	No	String	<p>Indicates which Webflow configuration file the origin and event are defined in. If omitted, then the current (last successful) namespace is used.</p>	R
origin	No	String	<p>The node where the event will be coming from. Origins follow the <code>node-name.node-type</code> format. This may or may not be equal to the page name. If omitted, then the JSP page name is used.</p>	R
pathPrefix	No	String	<p>Used to prefix the path information. This string will be placed in front of the Web application URI. This can be used when the proxy server is located on the same machine.</p> <p>Note: The proxy must strip the path prefix before forwarding the request to Webflow.</p>	R
pathSuffix	No	String	<p>Used to suffix the path information. The additional path information will be placed after the <code>WebflowServlet</code> URI. This information can then be retrieved via the <code>request.getPathInfo()</code> method.</p>	R

Example

The following code fragments illustrate how to use the `<webflow:createWebflowURL>` JSP tag and its many attributes:

```
<a href="<webflow:createWebflowURL event="link.yo" pathSuffix="/morepath"
/>">A Path Suffix</a>
<br>
```

```
<a href="<webflow:createWebflowURL event="link.yo" pathPrefix="/pathprefix"
/>">A Path Prefix</a>
<br>
```

```
<a href="<webflow:createWebflowURL event="link.yo" pathPrefix="/pathprefix"
pathSuffix="/suffix" />">A Path Prefix and Path Suffix</a>
<br>
```

```
<a href="<webflow:createWebflowURL event="link.yo" domainName="123.123.123.123"
/>">A Domain Name</a>
<br>
```

```
<a href="<webflow:createWebflowURL event="link.yo" pathSuffix="/morepath"
extraParams="sex=male" />">A Path Suffix and One Extra Parameter</a>
<br>
```

```
<a href="<webflow:createWebflowURL event="link.yo" pathSuffix="/morepath"
extraParams="sex=male&animal=dog" />">A Path Suffix and Two Extra Parameters</a>
<br>
```

```
<a href="<webflow:createWebflowURL event="link.yo" httpsInd="https" />">Always
Use HTTPS</a>
<br>
```

```
<a href="<webflow:createWebflowURL event="link.yo" httpsInd="http" />">Always Use
HTTP</a>
<br>
```

```
<a href="<webflow:createWebflowURL event="link.yo" httpsInd="calculate"
/>">Calculate HTTPS</a>
<br>
```

```
<a href="<webflow:createWebflowURL event="link.yo" encode="false" />">Do Not
Encode the URL</a>
<br>
```

```
<a href="<webflow:createWebflowURL event="link.yo" doRedirect="true"
/>">Redirect, Instead of Forward</a>
<br>
```

<webflow:createResourceURL>

The <webflow:createResourceURL> tag (Table 5-2) is used in a JSP to create a static URL for a resource, using the value of the P13N_STATIC_ROOT context parameter in the application's WEB-INF/web.xml file. This tag may be used to load GIF images from a separate server.

Table 5-2 <webflow:createResourceURL>

Tag Attribute	Required	Type	Description	R/C
encode	No	Boolean	Indicates whether or not the URL should be encoded. Valid values are <code>true</code> and <code>false</code> . The default value is <code>true</code> .	R
resource	No	String	Relative path to the file or image.	R

Example

The following code fragment illustrates how you might use the <webflow:createResourceURL> JSP tag:

```
" border="0" alt="Proceed
To Checkout" border="0"></a>
```

Form Tags

The Webflow JSP form tags described in this section are used to create simple dynamic links for form actions.

Note: In the following tables, the Required column specifies if the attribute is required (yes) or optional (no). In the R/C column, C means that the attribute is a Compile time expression, and R means that the attribute can be either a Request time expression or a Compile time expression.

<webflow:form>

The `<webflow:form>` tag (Table 5-3) is used in a JSP to dynamically generate an HTML form tag. This tag is not as sophisticated as the `<webflow:validatedForm>` tag, but is simpler. For more information about the `<webflow:validatedForm>` tag, refer to the next section.

Note: This tag does not support the embedded Webflow form tags like the `<webflow:validatedForm>` does.

Table 5-3 `<webflow:form>`

Tag Attribute	Required	Type	Description	R/C
<code>domainName</code>	No	String	Used to change the domain name or IP address of the URL. This may be used if Webflow is fronted by a proxy server and that server resides on another machine.	R
<code>doRedirect</code>	No	String	Causes the <code>WebflowServlet</code> to perform a redirect instead of a forward to a presentation node. Valid values are <code>true</code> and <code>false</code> . The default is <code>false</code> (not to redirect).	R
<code>encode</code>	No	String	Informs Webflow to encode the URL. URLs need to be encoded if you wish to maintain session state and the browser does not except cookies. Valid values are <code>true</code> and <code>false</code> . The default value is <code>true</code> , as URLs will only need to be encoded if the browser does not except cookies.	R
<code>event</code>	Yes	String	Webflow will use this in combination with the <code>origin</code> to resolve a destination in the supplied namespace XML file.	R
<code>extraParams</code>	No	String	Used to supply additional request parameters as name/value pairs.	R

Table 5-3 <webflow:form> (Continued)

Tag Attribute	Required	Type	Description	R/C
hide	No	String	If set to <code>false</code> , the namespace, origin, and event will be displayed on the command line instead of as hidden form fields. Valid values are <code>true</code> and <code>false</code> . The default value is <code>true</code> .	R
httpsInd	No	String	<p>Informs Webflow to calculate the protocol or use HTTPS or HTTP. Therefore, valid values are <code>calculate</code>, <code>http</code>, and <code>https</code>. The default value is <code>http</code>.</p> <p><code>Calculate</code> will yield HTTPS if any node in the origin/event branch chain list is matched under the <code>HTTPS_URL_PATTERNS</code> context parameter in the application's <code>WEB-INF/web.xml</code> file. <code>Calculate</code> is more dynamic and expensive, but if the protocol needs to be forced you can specify it here.</p>	R
method	No	String	The method to be used for the form. Valid values are <code>get</code> and <code>post</code> . The default value is <code>post</code> .	R
name	No	String	The name of the form.	R
namespace	No	String	Indicates which Webflow configuration file the origin and event are defined in. If omitted, then the current (last successful) namespace is used.	R
origin	No	String	The node where the event will be coming from. Origins follow the <code>node-name.node-type</code> format. This may or may not be equal to the page name. If omitted, then the JSP page name is used.	R

Table 5-3 <webflow:form> (Continued)

Tag Attribute	Required	Type	Description	R/C
pathPrefix	No	String	Used to prefix the path information. This string will be placed in front of the Web application URI. This can be used when the proxy server is located on the same machine. Note: The proxy must strip the path prefix before forwarding the request to Webflow.	R
pathSuffix	No	String	Used to suffix the path information. The additional path information will be placed after the WebflowServlet URI. This information can then be retrieved via the <code>request.getPathInfo()</code> method.	R

Example

The following code fragment illustrates how you might use the <webflow:form> JSP tag:

```
<webflow:form event="button.go" >
  <input type="text" name="username" >
</webflow:form>
```

Validated Form Tags

The Webflow's validated form JSP tags are used to dynamically generate HTML forms that can be validated. These tags work in conjunction with an Input Processor and the `ValidatedValues` class, described in “Webflow Validators and Input Processors” on page 4-17. When a Web site visitor enters invalid information, the the visitor's input is preserved and redisplayed with an appropriate error message.

<webflow:validatedForm>

The `<webflow:validatedForm>` tag (Table 5-4) is used in a JSP to dynamically create the URL that defines the action in an HTML form. This tag should be used in conjunction with the `com.bea.p13n.appflow.webflow.forms.*` package and the other nested form tags defined in the `webflow.tld` file (which are described later in this section).

Table 5-4 `<webflow:validatedForm>`

Tag Attribute	Required	Type	Description	R/C
<code>applyStyle</code>	No	String	Applies the associated style as indicated by the field status to the message, the field, or to none. Therefore, valid values are <code>message</code> , <code>field</code> , and <code>none</code> . The default value is <code>message</code> .	R
<code>domainName</code>	No	String	Used to change the domain name or IP address of the URL. This may be used if Webflow is fronted by a proxy server and that server resides on another machine.	R
<code>doRedirect</code>	No	String	Causes the <code>WebflowServlet</code> to perform a redirect instead of a forward to a presentation node. Valid values are <code>true</code> and <code>false</code> . The default is <code>false</code> (not to redirect).	R

Table 5-4 <webflow:validatedForm> (Continued)

Tag Attribute	Required	Type	Description	R/C
encode	No	String	Informs Webflow to encode the URL. URLs need to be encoded if you wish to maintain session state and the browser does not accept cookies. Valid values are <code>true</code> and <code>false</code> . The default value is <code>true</code> , as URLs will only need to be encoded if the browser does not accept cookies.	R
event	Yes	String	Webflow will use this in combination with the origin to resolve a destination in the supplied namespace XML file.	R
extraParams	No	String	Used to supply additional request parameters as name/value pairs.	R
hide	No	String	If set to <code>false</code> , the namespace, origin, and event will be displayed on the command line instead of as hidden form fields. Valid values are <code>true</code> and <code>false</code> . The default value is <code>true</code> .	R
httpsInd	No	String	Informs Webflow to calculate the protocol or use HTTPS or HTTP. Therefore, valid values are <code>calculate</code> , <code>http</code> , and <code>https</code> . The default value is <code>http</code> . Calculate will yield HTTPS if any node in the origin/event branch chain list is matched under the <code>HTTPS_URL_PATTERNS</code> context parameter in the application's <code>WEB-INF/web.xml</code> file. Calculate is more dynamic and expensive, but if the protocol needs to be forced you can specify it here.	R
invalidStyle	No	String	The style used to format the HTML field or the message when the field is invalid.	R

Table 5-4 <webflow:validatedForm> (Continued)

Tag Attribute	Required	Type	Description	R/C
messageAlign	No	String	Indicates whether to align the error message above the field, to the right of the field, or below the field. Therefore, value values are top, right, and bottom. The default value is right.	R
method	No	String	The method to be used for the form. Valid values are get and post. The default value is post.	R
name	No	String	The name of the form.	R
namespace	No	String	Indicates which Webflow configuration file the origin and event are defined in. If omitted, then the current (last successful) namespace is used.	R
origin	No	String	The node where the event will be coming from. Origins follow the node-name.node-type format. This may or may not be equal to the page name. If omitted, then the JSP page name is used.	R
pathPrefix	No	String	Used to prefix the path information. This string will be placed in front of the Web application URI. This can be used when the proxy server is located on the same machine. Note: The proxy must strip the path prefix before forwarding the request to Webflow.	R
pathSuffix	No	String	Used to suffix the path information. The additional path information will be placed after the webflowServlet URI. This information can then be retrieved via the request.getPathInfo() method.	R

Table 5-4 <webflow:validatedForm> (Continued)

Tag Attribute	Required	Type	Description	R/C
styleId	No	String	Scripting variable that will be set to one of invalidStyle, unspecifiedStyle, or validStyle, depending on the field's status: valid, invalid, unspecified. Can be used for finer control of formatting the HTML form.	R
unspecifiedStyle	No	String	Used to specify the initial style of the HTML field before validation occurs.	R
validStyle	No	String	The style used to format the HTML field when it is valid.	R

<webflow:text>

The <webflow:text> tag (Table 5-5) is used in a JSP to validate an HTML text field. This tag must be nested in the <webflow:validatedForm> tag.

Table 5-5 <webflow:text>

Tag Attribute	Required	Type	Description	R/C
htmlAttributes	No	String	Additional HTML attributes. Any attribute not supported directly can be supplied here.	R
maxLength	No	String	The maximum length of the text field.	R
name	Yes	String	The name of the text field.	R
retainValue	No	String	Determines whether or not the text field should retain its input upon redisplay. Valid values are true and false.	R
size	No	String	The size of the text field.	R
style	No	String	The HTML style associated with the text field.	R
value	No	String	The initial value of the text field.	R

<webflow:password>

The <webflow:password> tag (Table 5-6) is similar to the <webflow:text> tag except that field input is masked with asterisks (****). This tag must be nested in the <webflow:validatedForm> tag.

Table 5-6 <webflow:password>

Tag Attribute	Required	Type	Description	R/C
htmlAttributes	No	String	Additional HTML attributes. Any attribute not supported directly can be supplied here.	R
maxLength	No	String	The maximum length of the password field.	R
name	Yes	String	The name of the password field.	R
retainValue	No	String	Determines whether or not the password field should retain its input upon redisplay. Valid values are <code>true</code> and <code>false</code> . The default value is <code>false</code> .	R
size	No	String	The size of the password field.	R
style	No	String	The HTML style associated with the password field.	R
value	No	String	The initial value of the password field.	R

<webflow:radio>

The `<webflow:radio>` tag (Table 5-7) is used in a JSP to represent an HTML radio button, but preserves the input. This tag must be nested in the `<webflow:validatedForm>` tag.

Table 5-7 `<webflow:radio>`

Tag Attribute	Required	Type	Description	R/C
<code>checked</code>	No	String	Determines whether or not the radio button is initially selected. Valid values are <code>true</code> and <code>false</code> . The default value is <code>false</code> .	R
<code>htmlAttributes</code>	No	String	Additional HTML attributes. Any attribute not supported directly can be supplied here.	R
<code>name</code>	Yes	String	The name of the radio button field.	R
<code>value</code>	Yes	String	The initial value of the radio button field.	R

<webflow:checkbox>

The `<webflow:checkbox>` tag (Table 5-8) is used in a JSP to generate the required HTML for a check box. This tag will preserve input upon the `InputProcessor` throwing an `InvalidFormFieldException`. This tag must be nested in the `<webflow:validatedForm>` tag.

Table 5-8 `<webflow:checkbox>`

Tag Attribute	Required	Type	Description	R/C
<code>checked</code>	No	String	Determines whether or not the check box field is initially selected. Valid values are <code>true</code> and <code>false</code> . The default value is <code>false</code> .	R
<code>htmlAttributes</code>	No	String	Additional HTML attributes. Any attribute not supported directly can be supplied here.	R
<code>name</code>	Yes	String	The name of the check box field.	R

Table 5-8 <webflow:checkbox> (Continued)

Tag Attribute	Required	Type	Description	R/C
value	Yes	String	The initial value of the check box field.	R

<webflow:textarea>

The <webflow:textarea> tag (Table 5-9) is used in a JSP to represent an HTML text area, but preserves the input. This tag must be nested in the <webflow:validatedForm> tag.

Table 5-9 <webflow:textarea>

Tag Attribute	Required	Type	Description	R/C
cols	No	String	The number of columns for the text area.	R
name	Yes	String	The name of the text area.	R
retainValue	No	String	Determines whether or not the text area should retain its input upon redisplay. Valid values are <code>true</code> and <code>false</code> .	R
rows	No	String	The number of rows for the text area.	R
style	No	String	The HTML style associated with the text area.	R
value	No	String	The initial value of the text area.	R
wrap	No	String	Determines whether or not the text area should wrap input. Valid values are <code>true</code> and <code>false</code> . The default value is <code>true</code> .	R

<webflow:select>

The `<webflow:select>` tag (Table 5-10) is used in a JSP to represent a list box, but preserves its input. This tag must be nested in the `<webflow:validatedForm>` tag.

Table 5-10 `<webflow:select>`

Tag Attribute	Required	Type	Description	R/C
<code>htmlAttributes</code>	No	String	Additional HTML attributes. Any attribute not supported directly can be supplied here.	R
<code>multiple</code>	No	String	Determines whether or not the list box allows multiple selections. Valid values are <code>true</code> and <code>false</code> . The default value is <code>false</code> .	R
<code>name</code>	Yes	String	The name of the list box.	R
<code>size</code>	No	String	The size of the list box.	R
<code>style</code>	No	String	The HTML style attribute.	R

<webflow:option>

The `<webflow:option>` tag (Table 5-11) is used in a JSP to represent an HTML option, but preserves the input. An option is one value in a list box. This tag must be nested in the `<webflow:select>` tag.

Table 5-11 `<webflow:option>`

Tag Attribute	Required	Type	Description	R/C
<code>selected</code>	No	String	Determines whether or not the option is initially selected. Valid values are <code>true</code> and <code>false</code> . The default value is <code>false</code> .	R
<code>style</code>	No	String	The HTML style attribute.	R
<code>value</code>	Yes	String	The value the option represents.	R

Example

The following code uses each of the validated form tags in an HTML form page that gathers some information about a Web site's visitors.

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

<% String validStyle="background: white; color: black; font-family: Arial"; %>
<% String invalidStyle="background: white; color: red; font-style: italic"; %>

<!-- If there was an InvalidFormException thrown, display the message -->

<font size="5" color="green"><webflow:getException/></font>
<br>

<webflow:validatedForm event="button.go" applyStyle="message"
  messageAlign="right" validStyle="<%=validStyle%>"
  invalidStyle="<%=invalidStyle%>" unspecifiedStyle="<%=validStyle%>" >
<p>

Username:
<webflow:text name="username" value="start" size="15" maxlength="30"
  htmlAttributes="onMouseOver=\\\"self.status='User ID';return true\\\" " />
<br>

Password:
<webflow:password name="password" size="15" retainValue="true" maxlength="30"
  htmlAttributes="onMouseOver=\\\"self.status='Secret Password';return true\\\" " />
<br>

Sex:
<webflow:radio name="sex" checked="true" value="M"/>Male
<webflow:radio name="sex" value="F" />Female
<br>

Favorite Pet(s):
<webflow:checkbox name="cat" value="cat" />Cat <br>
<webflow:checkbox name="dog" checked="true" value="dog" />Dog <br>
<webflow:checkbox name="bird" value="bird" />Bird
<p>

Comment:
<webflow:textarea name="comment" cols="20" rows="3" value="hello" />
<br>

Hobbies:
<webflow:select name="hobbies" size="3" multiple="true">
  <webflow:option value="Running"/>Running
  <webflow:option value="Skiing"/>Skiing
```

```

    <webflow:option value="Motocross" />MotoX
    <webflow:option value="Rugby" />Rugby
</webflow:select>
<br>

<input type="submit" name="Submit" />

</webflow:validatedForm>

```

Pipeline Session Tags

A Pipeline Session is used to share information between Input Processors, Pipeline Components, and presentation nodes. The Pipeline Session JSP tags are used to retrieve and set properties in the Pipeline Session. Presentation nodes (such as JSPs) are typically used to retrieve information from the Pipeline Session, while Input Processors and Pipeline Components place properties into the Pipeline Session. There are, however, JSP tags for setting properties in the Pipeline Session.

Note: For more information about the Pipeline Session, see “The Pipeline Session” on page 2-6 and “Pipeline Session Internals” on page 4-3.

<webflow:setProperty>

The <webflow:setProperty> tag (Table 5-12) sets a property in the Pipeline Session.

Table 5-12 <webflow:setProperty>

Tag Attribute	Required	Type	Description	R/C
namespace	No	String	Use the namespace attribute to force webflow to use a particular webflow configuration file defining a specific origin and event. If omitted then the current namespace (last successful namespace) is used.	R

Table 5-12 <webflow:setProperty> (Continued)

Tag Attribute	Required	Type	Description	R/C
property	Yes	String	Name or key with which the given property is to be associated.	R
scope	No	String	The scope of the property. Valid values are <code>session</code> and <code>request</code> . The default value is <code>session</code> .	R
value	Yes	Object	The value to associate with the property, specified as an object name or JavaScript expression.	R

Example

The following code fragment illustrates how you might use the <webflow:setProperty> JSP tag to set some arbitrary object in the Pipeline Session (Request-scoped):

```
<% SomeObject so = new SomeObject("TWO"); %>
<webflow:setProperty property="myobject" value="<%= so %>"
scope="request" />
```

<webflow:getProperty>

The <webflow:getProperty> tag (Table 5-13) retrieves a named property from the Pipeline Session. This tag can be inlined or return a scripting variable.

Table 5-13 <webflow:getProperty>

Tag Attribute	Required	Type	Description	R/C
namespace	No	String	Use the namespace attribute to force webflow to use a particular webflow configuration file defining a specific origin and event. If omitted then the current namespace (last successful namespace) is used.	R

Table 5-13 <webflow:getProperty> (Continued)

Tag Attribute	Required	Type	Description	R/C
id	No	Object	Java scripting variable to receive the instance of the returned object. If omitted, the <code>toString()</code> method will be called on the object and the results will be displayed in the browser.	R
property	Yes	String	The name or key of the property to obtain from the Pipeline Session.	R
scope	No	String	The scope of the property, which can be <code>request</code> or <code>session</code> . Request-scoped properties can improve performance, especially in a cluster because they do not need to be replicated. Valid values are <code>session</code> and <code>request</code> . The default value is <code>session</code> .	R
type	No	String	A Java class name, which can be used to cast your exception.	R

Example 1

The following code fragment is an example of how you might use the <webflow:getProperty> JSP tag inline. The `toString()` method is called on the instance of `SomeObject`:

```
result = <webflow:getProperty property="myobject" scope="request" />
```

Example 2

The following code fragment is an example of how you might use the <webflow:getProperty> JSP tag to return a scripting variable of type `SomeObject`:

```
<webflow:getProperty id="myObj" property="myobject"
type="com.bea.test.SomeObject" scope="request" />
result = <%= myObj.getValue() %>
```

<webflow:setValidatedValue>

The <webflow:setValidatedValue> tag (Table 5-14) is used in a JSP to configure the display of fields in a form that a Web site visitor must correct. Usually this is done within an Input Processor, but it can also be done from a JSP by using this tag. The <webflow:setValidatedValue> tag is used in tandem with the <webflow:getValidatedValue> tag.

Note: You may want to consider using the <webflow:validatedForm> tags instead. This tag supports the `validatedValues` class from previous releases. However, if there is some low-level functionality that needs to be accessed, then these tags are still valid.

Table 5-14 <webflow:setValidatedValue>

Tag Attribute	Required	Type	Description	R/C
fieldName	No	String	The name of the field for which the status is desired. This should match the HTML form field name.	R
fieldStatus	No	String	The processing status of the field. Valid values are: unspecified—Field was left blank; Web site visitor must enter some data. invalid—Data is entered incorrectly. valid—Data is entered correctly.	R
fieldValue	No	String	The new value of the field.	R

Example

When used in a JSP, this sample code will obtain the current value and processing status of the <field_name> form field.

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


<webflow:getValidatedValue>

The <webflow:getValidatedValue> tag (Table 5-15) is used in a JSP to display the fields in a form that a Web site visitor must correct. The

<webflow:getValidatedValue> tag is used in tandem with the <webflow:setValidatedValue> tag.

Note: You may want to consider using the <webflow:validatedForm> tags instead. This tag supports the `ValidatedValues` class from previous releases. However, if there is some low-level functionality that needs to be accessed, then these tags are still valid.

Table 5-15 <webflow:getValidatedValue>

Tag Attribute	Required	Type	Description	R/C
fieldColor	No	String	Scripting variable set to one of <code>invalidColor</code> , <code>validColor</code> , or <code>unspecifiedColor</code> (depending on the status). This can be used to change the color of the field or message.	R
fieldDefaultValue	No	String	The default value to use if the <code>fieldValue</code> is missing.	R
fieldMessage	No	String	A scripting variable used to provide a specific message for the current field.	R
fieldName	Yes	String	The name of the field for which the status is desired. This should match the HTML form field name.	R
fieldStatus	Yes	String	The status of the field. Valid values are: <code>unspecified</code> —Field was left blank; Web site visitor must enter some data. <code>invalid</code> —Data is entered incorrectly. <code>valid</code> —Data is entered correctly.	R
fieldValue	Yes	String	Scripting variable representing the value of the form field.	R

Table 5-15 <webflow:getValidatedValue> (Continued)

Tag Attribute	Required	Type	Description	R/C
invalidColor	No	String	The color with which the label for an invalid field is to be marked. Defaults to red.	R
unspecifiedColor	No	String	If the Web site visitor leaves a required field blank, this will be the color of the label for that field. Defaults to red.	R
validColor	No	String	The color with which the label for a valid field is to be marked. Defaults to black.	R

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. For more information, see “Webflow Validators and Input Processors” on page 4-17.

Example 1

When used in a JSP, this sample code will obtain the current value and processing status of the <field_name> form field.

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

Example 2

The <webflow:getValidatedValue> tag refers to the `webflow.tld` tag library to retrieve available elements/attributes. In this example, a request is being made to obtain the following values from the HTTP session:

```
fieldName  
fieldValue  
fieldStatus  
validColor  
invalidColor  
unspecifiedColor
```

```
fieldColor
```

These attributes are used for display purposes. (In this case, indicate that this field is required and mark it in red.) The overall goal is to display values back to the Web site visitor, indicating which pieces are valid/invalid as returned from the Input Processor.

```
<webflow:getValidatedValue
fieldName="<%=HttpRequestConstants.CUSTOMER_FIRST_NAME%>"
fieldValue="customerFirstName" fieldStatus="status"
validColor="black"
invalidColor="red" unspecifiedColor="black" fieldColor="fontColor"
/>
```

<webflow:getException>

The `<webflow:getException>` tag (Table 5-16) is used to retrieve the exception or message thrown by a Webflow processor. This can be the message associated with an `InvalidFormFieldException` exception or a `ProcessingException` exception. This tag can be inlined (in which it calls the `getMessage()` method on the exception) or return a scripting variable representing the exception.

Table 5-16 `<webflow:getException>`

Tag Attribute	Required	Type	Description	R/C
id	Yes	Exception	Java scripting variable, which can be used to retrieve an instance of the exception.	R
type	No	String	Java class name, which can be used to cast the exception.	R

Example

The following code fragment illustrates how you might use the

`<webflow:getException>` JSP tag:

```
<%-- If there was an InvalidFormDataException thrown, display the
message --%>
```

```
<font size="5" color="red"><webflow:getException/></font>
```

6 An Example of Webflow: The Pet Flow Application

Pet Flow is a sample Web application that comes packaged with the WebLogic Portal product suite. The Pet Flow application has been designed to illustrate some of the features of the Webflow mechanism, which may clarify the concepts described in other topics of this *Guide to Managing Presentation and Business Logic: Using Webflow and Pipeline* documentation.

This topic includes the following sections:

- About the Pet Flow Sample Application
- What Webflow Features Does the Pet Flow Sample Application Illustrate?
- Accessing the Pet Flow Sample Application
 - Location of Pet Flow Files
 - Running Pet Flow in a Web Browser
 - Opening a Pet Flow Namespace in the Webflow Editor

About the Pet Flow Sample Application

As a developer, you are probably familiar with the Sun Microsystems Pet Store Demo, a comprehensive e-commerce application based on the Java 2 Enterprise Edition (J2EE) specification. The Pet Store Demo showcases the main features of J2EE and can be used as a reference implementation. Because BEA recognizes this sample application as a great way to help you understand how to build Web applications using J2EE, we have modified it to use some features of our Webflow mechanism. The resulting sample application is called Pet Flow.

Note: For more information on the original Pet Store Demo, see the “Architecture Overview” at <http://java.sun.com/j2ee/blueprints/jps11/archoverview.html>.

What Webflow Features Does the Pet Flow Sample Application Illustrate?

Because it is only a sample application, the Pet Flow Web application does not show every feature available in the Webflow mechanism. However, it does illustrate the following:

- Link and button event transitions within the same namespace, when the following information is provided in the URL:
 - namespace, no origin, no event
 - namespace, origin, event

Additionally, the Pet Flow application shows link and button event transitions across different namespaces when the URL provides the namespace, origin, and event to the Webflow mechanism.

- Exception transitions within the same namespace and across different namespaces.
- Wildcard presentation nodes (JavaServer Pages only) that work within the same namespace and across different namespaces.

- Handling of wildcard processor node exceptions (Input Processors and Pipelines).
- Input Processors used for conditional branching, and general handling of Input Processor exceptions.
- Form validation using Input Processors, as well as use of a custom validator.
- Pipelines that may produce fatal exceptions, and general handling of Pipeline exceptions.
- Pipelines that are transactional and Pipelines that are not transactional. Additionally, Pipeline Sessions that transactional, and Pipeline Sessions that are not transactional.
- Getting Pipeline Session properties via JavaServer Pages (JSPs), Input Processors, and Pipeline Components. Additionally, setting Pipeline Session properties via Input Processors and Pipeline Components.
- Use of an extension (custom) processor node.
- Frame-based and all-in-one configurations.

Use the Pet Flow examples to increase your understanding of Webflow's capabilities, but be sure to use your organization's established processes and your own best judgement in your implementations.

Accessing the Pet Flow Sample Application

If you have successfully installed the WebLogic Portal product suite, you can access the Pet Flow sample application. If you have not yet installed the product and would like instructions for performing the installation, see the *Installation Guide*.

Location of Pet Flow Files

The files comprising the Pet Flow sample Web application are located in the `PORTAL_HOME\applications\petflowApp\petflow` directory, where `PORTAL_HOME` is the directory in which you installed WebLogic Portal.

Running Pet Flow in a Web Browser

To run the Pet Flow application in a Web browser, follow the instructions for the `petflowDomain` in the “Review the Reference Domains” section in the *Deployment Guide*.

Opening a Pet Flow Namespace in the Webflow Editor

To open a Pet Flow namespace file in the Webflow Editor, follow these steps:

1. Start the E-Business Control Center (EBCC). For detailed instructions on starting the EBCC, see “Starting the E-Business Control Center” in the *Guide to Using the E-Business Control Center* documentation.
2. Open the Pet Flow application so you can view its Webflow. For instructions on opening an application, see “Opening an Application” in the *Guide to Using the E-Business Control Center* documentation.
3. Select the Site Infrastructure tab in the EBCC’s Explorer window, then click the Webflows/Pipelines icon.
4. Click the `webapps` and `petflow` objects in the Webflows/Pipelines list box to view the list of namespace files that comprise the Pet Flow application’s Webflow.
5. Double click any of the Pet Flow application’s `<namespace>.wf` files that are shown in the EBCC’s Explorer list to view them in the Webflow Editor canvas.
6. From the Webflow Editor canvas, click on any Pipeline Node to view its Pipeline Components the Pipeline Editor. Or, you can expand the Pipelines shown under the Pipeline Namespaces object in the Webflows/Pipelines list box, then double click on the Pipeline name to open the Pipeline Editor.

Index

Symbols

- <namespace>.pln files 1-9, 3-6, 3-12
- <namespace>.pln.<pipeline_name>.ui files 3-12
- <namespace>.wf files 1-9, 3-6, 3-12
- <namespace>.wf.ui files 3-12

A

- abort exception node 3-16, 4-14
 - default 4-14
- accessing
 - exceptions 4-13
 - the Pet Flow application 6-3
- adding Webflow components in Editors 3-24
- application.xml file 1-14
- applications
 - enterprise
 - relationship to Webflow 1-7
 - portal
 - events 2-3, 2-8
 - specifying namespaces 2-9
 - Web
 - configuring to use Webflow 1-19
 - creating in the EBCC 3-3
 - default behavior 2-10
 - Pet Flow 6-1, 6-2
 - relationship to Webflow 1-7
 - URI 1-13
 - welcome files 1-18

- architecture, Webflow 1-1, 1-2, 1-3
 - portal 1-4
- attributes, Pipeline Session 2-7
- automatic URL generation in JavaServer Pages (JSPs) 1-17, 2-12

B

- begin node
 - definition 2-9
 - designating in Webflow Editor 3-25
- begin origin 1-14, 1-19
- benefits
 - namespaces 3-9
 - Webflow 1-1, 1-8
- branch exceptions 4-14
- branching Input Processors and Pipelines 2-12
- business logic in Pipelines 2-6
- button events 2-7

C

- chaining Input Processors and Pipelines 2-12
- class
 - InputProcessorSupport 4-3, 4-15
 - helper methods 4-5
 - PipelineComponentSupport 4-3, 4-28
 - helper methods 4-6
 - SessionManagerFactory 4-4
 - ValidatedValues 2-5, 4-17, 5-10, 5-

- WebflowJSPHelper 2-12
- ClassCastException 4-11, 4-13
- ClassNotFoundException 4-11, 4-12
- clusters
 - data synchronization in 3-7
 - Pipeline Session 2-6
- communication among Pipeline Components 2-6
- Components, Pipeline
 - association with Pipeline Session 4-3
 - communication 2-6
 - creating 4-28
 - development guidelines 4-32
 - exception handling 4-11
 - exceptions
 - creating 4-13
 - implementation 2-6
 - as Java objects 4-30
 - as stateless session EJBs 4-30
 - naming conventions 4-29
 - relationship to Pipelines 2-6
 - state 4-31
- components, Webflow 1-7, 2-1
 - adding in Editors 3-24
 - creating 1-20
 - editing names in Editors 3-24
 - graphical representations 3-13
 - in Editors 3-9
 - organizing in Editor canvas 3-23
 - relationships 2-3
 - selecting in Editors 3-23
 - using in portals 2-13
 - using in Web pages 2-12
- configuration error page 2-16, 4-11
 - definition 2-11
- configuration files
 - <namespace>.pln 1-9, 3-6, 3-12
 - <namespace>.wf 1-9, 3-6, 3-12
 - application.xml 1-14
 - creating and modifying with Webflow
 - and Pipeline Editors 1-12
- web.xml
 - registering the PortalServlet 1-18
 - registering the WebflowServlet 1-17, 1-18
- Webflow 1-1, 1-9
 - Content Management System 3-6, 3-11, 3-12
 - contents 1-9
 - creating 1-12, 1-19
 - location 1-10
 - modifying 1-12
 - pipeline.properties 1-10
 - webflow.properties 1-10
- ConfigurationException 4-12
- configuring
 - Pipeline Session 4-3
 - Web applications to use Webflow 1-19
- container
 - Enterprise JavaBean (EJB) 2-5
 - Web application 2-5
- Content Management System
 - Webflow and Pipeline configuration files in 3-6, 3-11, 3-12
- context parameters
 - HTTP_PORT and HTTPS_PORT 1-13
 - P13N_APPLICATION_URL 1-15
 - relationship to URL pattern 1-17, 1-18
 - P13N_DEFAULT_NAMESPACE 1-14, 1-15, 2-9, 2-11
 - P13N_STATIC_ROOT 1-16, 5-6
 - P13N_URL_DOMAIN 1-16
 - P13N_URL_PREFIX 1-16
 - specifying 1-15
- conversational state 1-4
- createStaticResourceURL() method 1-16
- createWebflowURL() method 1-16, 2-12
- creating
 - custom validators 4-25
 - extension (custom) nodes 2-4, 2-5, 4-

- Input Processor exceptions 4-13
 - Input Processors 4-15
 - Pipeline Component exceptions 4-13
 - Pipeline Components 4-28
 - Pipelines in the Pipeline Editor 3-5
 - Webflow components 1-20
 - Webflow configuration files 1-12, 1-19
 - methods for 1-12
 - Webflow namespaces in the Webflow Editor 3-5
 - customizing Webflow 4-1
- D**
- data, synchronizing 3-1, 3-7
 - clusters 3-7
 - errors 3-7
 - default abort exceptions 4-14
 - deployment descriptors, XML 1-3, 1-7, 1-13, 1-16, 1-18, 1-19, 2-9, 2-11
 - setting up 1-15
 - destination and origin nodes 2-4
 - developer roles 1-12
 - development guidelines
 - Input Processors 4-17
 - Pipeline Components 4-32
 - domain name, server 1-13, 1-16
 - dynamic and static URLs 5-2
- E**
- Editors, Webflow and Pipeline
 - creating Webflow namespaces and Pipelines 3-5
 - definition 3-1
 - important information about 3-6
 - opening multiple namespace files 3-9
 - opening multiple Pipelines 3-11
 - organizing Webflow components 3-23
 - palettes 3-17
 - read-only mode 3-6, 3-11, 3-12, 3-17
 - role-based security 3-7
 - title bar information 3-11
 - toolbars 3-20
 - using to create and modify Webflow
 - configuration files 1-12
 - validation features 1-12, 2-15, 2-16, 3-5
 - Webflow components 3-9
 - ensuring validity of Webflows 1-12
 - enterprise applications
 - and Webflow 1-7
 - Enterprise JavaBean (EJB) container 2-5
 - error handling, Webflow 4-10
 - error page
 - 500 internal server 2-16, 4-11
 - configuration 2-16, 4-11
 - definition 2-11
 - events
 - definition 2-7
 - portal applications 2-3, 2-8
 - presentation node 2-7
 - processor node 2-7
 - role in transitions 2-3
 - Webflow use of 1-6
 - exceptions 2-7
 - abort 3-16, 4-14
 - default 4-14
 - accessing 4-13
 - branch 4-14
 - ClassCastException 4-11, 4-13
 - ClassNotFoundException 4-11, 4-12
 - ConfigurationException 4-12
 - Exception 4-11
 - handling
 - Input Processor 4-11
 - JavaServer Page (JSPs) 4-13
 - JavaServer Pages (JSPs) 4-13
 - Pipeline Component 4-11
 - IllegalAccessException 4-12, 4-13

- IllegalArgumentExcep`tion` 4-12
- Input Processor
 - creating 4-13
- InstantiationException 4-11, 4-12
- InvalidArgumentException 4-4
 - messages 4-6, 4-13
 - non-runtime 4-9
- Pipeline
 - fatal 4-14
 - nonfatal 4-14
- Pipeline Component
 - creating 4-13
- PipelineException 4-11
- PipelineSystemException 4-12
- RemoteException 4-12
- runtime 4-9
- RuntimeException 4-11, 4-12
- validation 4-24
 - InvalidFormDataException 4-24
 - InvalidValidatorException 4-24
 - MissingFormFieldException 4-24
- execution order, Webflow 1-3, 2-13
 - Presentation Nodes 2-14
 - Processor Nodes 2-15
- Executor, Pipeline 1-4
- Executor, Webflow 1-3
- extending Webflow 1-4, 4-1, 4-33
- extension (custom) nodes
 - creating 4-33
 - presentation 2-4
 - processor 2-5
 - registering 1-10, 1-12, 3-12

F

- fatal Pipeline exceptions 4-14
- file
 - `<namespace>.pln.<pipeline_name>.ui` 3-12
 - `<namespace>.wf.ui` 3-12
- namespace

- opening multiple 3-9
- Pet Flow
 - location 6-3
- webflow.tld 5-10
- webflow-extensions.wfx 3-12, 4-33, 4-35
 - location 4-35
 - modifying 4-35
- welcome 1-18
- files, configuration
 - `<namespace>.pln` 1-9, 3-6, 3-12
 - `<namespace>.wf` 1-9, 3-6, 3-12
 - web.xml 1-3, 1-7, 1-13, 1-18, 1-19
 - setting up 1-15, 1-16, 2-9, 2-11
- Webflow 1-1, 1-9
 - Content Management System 3-6, 3-11, 3-12
 - contents 1-9
 - creating 1-12, 1-19
 - location 1-10
 - modifying 1-12
 - pipeline.properties 1-10
 - webflow.properties 1-10
- flow of control 1-4, 1-5

G

- `getInvalidFieldCount()` method 4-24
- `getMessage()` method 5-25
- getting and setting Pipeline Session properties 4-4
- GIF images, loading 5-6
- graphical representations of Webflow components 3-13
- guidelines, development
 - Input Processors 4-17
 - Pipeline Components 4-32

H

- HTTP_PORT and HTTPS_PORT context

- parameters 1-13
- HTTPRequest
 - Pipeline Session use of 2-6, 4-7
- HTTPSession
 - Pipeline Session use of 2-6

I

- IllegalAccessException 4-12, 4-13
- IllegalArgumentException 4-12
- images, GIF
 - loading 5-6
- implementation of JSPs 5-2
- initial state
 - Pipelines 2-10
 - Webflow 2-9
- Input Processor
 - as Model 1-7
 - branching 2-12
 - chaining 2-12
 - creating 4-15
 - definition 1-4, 2-5
 - development guidelines 4-17
 - exception handling 4-11
 - exceptions
 - creating 4-13
 - InvalidFormFieldException 5-15, 5-25
 - naming conventions 4-16
 - ProcessingException 5-25
 - statelessness 4-17
 - typical use 2-5
 - ValidatedValues class 4-17, 5-10, 5-24
- InputProcessor interface 4-15
- InputProcessorSupport class 4-3, 4-15
 - helper methods 4-5
- InstantiationException 4-11, 4-12
- interface
 - InputProcessor 4-15
 - Pipeline Session 4-3
 - PipelineComponent 4-28

- PresentationNodeHandler 4-33
- Processor 4-35
- ValidatedValues 4-17
 - purpose 4-18
- Validator 4-25

- internal server error page 2-16, 4-11
- internals of Webflow 1-4
- internationalization, Webflow 4-38
- InvalidArgumentException 4-4
- InvalidFormDataException 4-24
- InvalidFormFieldException 5-15, 5-25
- InvalidValidatorException 4-24
- invocation of Webflow from a URL 1-13

J

- Java object Pipeline Components 2-6, 4-30
- JavaServer Pages (JSPs)
 - automatic URL generation 1-17, 2-12
 - exception handling 4-13
 - implementation of 5-2
- JSP tags
 - Pipeline Session 2-7, 5-19
 - Webflow 1-16, 2-5, 2-12, 5-1
 - importing 5-1

L

- library, tag
 - Webflow 5-2, 5-10
- link events 2-7
- listen port 1-13
- loading GIF images 5-6
- location
 - Webflow configuration files 1-10
 - webflow-extensions.wfx file 1-11

M

- Manager, Session 1-3
- messages, exception 4-6, 4-13

methods

- createStaticResourceURL() 1-16
- createWebflowURL() 1-16, 2-12
- getInvalidFieldCount() 4-24
- getMessage() 5-25
- helper
 - InputProcessorSupport 4-5
 - PipelineComponentSupport 4-6
- support 2-7
- Webflow configuration files
 - creating 1-12
 - modifying 1-12

migration

- Webflow 1-1, 1-10, 1-13

MissingFormFieldException 4-24

modifying

- namespaces in the Webflow Editor 3-4
- Pipelines in the Pipeline Editor 3-5
- Webflow configuration files 1-12
 - methods for 1-12
- webflow-extensions.wfx file 1-12

MVC and Webflow 1-2, 1-7

N

namespaces

- benefits 3-9
- default 1-14
 - begin node 2-9
 - configuration error page 2-11
- definition 1-7, 2-8, 3-9
- division of Pipeline Session 2-8
- files 1-9
- Pipeline 3-4
- specifying for portal applications 2-9
- Webflow
 - in EBCC 3-4
- wildcard nodes 2-10

naming conventions

- Input Processors 4-16
- Pipeline Components 4-29

nodes

- abort exception 3-16, 4-14
- begin
 - definition 2-9
 - designating in Webflow Editor 3-25
- definition 2-3
- extension (custom)
 - creating 4-33
 - presentation 2-4
 - processor 2-5
 - registering 1-10, 1-12, 3-12
- presentation
 - as View 1-7
 - definition 2-4
- processor
 - as Model 1-7
 - definition 2-4
 - Input Processor 2-5
 - Pipelines 2-5
- proxy 3-15
- root component
 - definition 2-10
 - designating in Pipeline Editor 3-25
- wildcard
 - definition 2-10

nonfatal Pipeline exceptions 4-14

non-runtime exception 4-9

O

objects

- Java
 - Pipeline Components implemented as 2-6, 4-30
- return 2-7
- serializable 2-7

opening

- multiple namespace files 3-9, 3-11
- Pet Flow application in the Webflow Editor 6-4

- Pipeline Editor 3-5, 3-10
- origin or destination nodes 2-4
- origin, begin 1-14, 1-19

P

- palettes in the Webflow and Pipeline Editors 3-17

- parameters, context

 - HTTP_PORT and HTTPS_PORT 1-13

 - P13N_APPLICATION_URL 1-15

 - relationship to URL pattern 1-17, 1-18

 - P13N_DEFAULT_NAMESPACE 1-14, 1-15, 2-9, 2-11

 - P13N_STATIC_ROOT 1-16, 5-6

 - P13N_URL_DOMAIN 1-16

 - P13N_URL_PREFIX 1-16

 - specifying 1-15

- pattern, URL 1-17

- performance, Pipeline Session 2-6

- Pet Flow sample Web application 6-1

 - about 6-2

 - accessing 6-3

 - features 6-2

 - opening in Webflow Editor 6-4

 - running 6-4

- Pipeline Components

 - association with Pipeline Session 4-3

 - communication 2-6

 - creating 4-28

 - development guidelines 4-32

 - exception handling 4-11

 - exceptions

 - creating 4-13

 - implementation

 - as Java objects 4-30

 - as stateless session EJBs 4-30

 - implementations 2-6

 - naming conventions 4-29

 - relationship to Pipelines 2-6

 - state 4-31

- Pipeline Editor

 - opening 3-5, 3-10

- Pipeline Executor 1-4

- Pipeline Processor

 - definition 1-4

- Pipeline Session

 - as storage 1-6

 - association with Pipeline Components 4-3

 - clustering 2-6

 - configuring 4-3

 - conversational state 1-4

 - definition 2-6

 - division into namespaces 2-8

 - HTTPSession 2-6

 - including in transactions 4-32

 - interface 4-3

 - JSP tags 2-7, 5-19

 - lifecycle 1-3

 - performance 2-6

 - properties 2-7

 - getting and setting 4-4

 - scope 4-7

 - serializing 4-8

 - serializable objects 2-7

 - storing information in 2-6

 - transactions 2-6

 - use of HTTPRequest 2-6

- Pipeline Session scope for Pipeline Session

 - properties 4-8

- PipelineComponent interface 4-28

- PipelineComponentSupport class 4-3, 4-28

 - helper methods 4-6

- PipelineException 4-11

- Pipelines

 - as Model 1-7

 - branching 2-12

 - chaining 2-12

 - creating in Pipeline Editor 3-5

 - definition 2-5

- exceptions
 - fatal 4-14
 - nonfatal 4-14
- initial state 2-10
- modifying in Pipeline Editor 3-5
- nontransactional 4-31
- opening multiple 3-11
- relationship to Pipeline Components 2-6
- scope 1-7, 1-9
- synchronizing data 3-1, 3-7
 - clusters 3-7
 - errors 3-7
- transactional 1-4, 2-5, 4-31
- within namespaces 3-4
- PipelineSystemException 4-12
- port, listen 1-13
- portal applications
 - events 2-3, 2-8
 - specifying namespaces 2-9
 - using Webflow 2-13
- PortalServlet
 - 1-4
 - registering 1-18
- presentation nodes
 - as View 1-7
 - definition 2-4
 - events 2-7
- PresentationNodeHandler interface 4-33
- ProcessingException 5-25
- processor
 - Input Processor 1-4
 - branching 2-12
 - chaining 2-12
 - creating 4-15
 - definition 2-5
 - development guidelines 4-17
 - InvalidFormFieldException 5-15, 5-25
 - naming conventions 4-16
 - ProcessingException 5-25
 - statelessness 4-17
 - typical use 2-5
 - ValidatedValues class 5-10, 5-24
 - nodes
 - as Model 1-7
 - definition 2-4
 - events 2-7
 - Input Processors 2-5
 - Pipelines 2-5
 - Pipeline 1-4
 - Processor interface 4-35
 - properties
 - Pipeline Session 2-7
 - getting and setting 4-4
 - Pipeline Session scope 4-8
 - Request scope 4-7
 - serializing 4-8
 - proxy node 3-15

R

- read-only mode, Webflow and Pipeline Editors 3-6, 3-11, 3-12, 3-17
- registering extension (custom) nodes 1-10, 1-12, 3-12, 4-35
- relationships among Webflow components 2-3
- RemoteException 4-12
- Request scope for Pipeline Session properties 4-7
- request, HTTP 4-7
- return objects, processor node 2-7
- role-based security in Webflow and Pipeline Editors 3-7
- roles, developer 1-12
- root component node
 - definition 2-10
 - designating in Pipeline Editor 3-25
- running the Pet Flow application 6-4
- runtime exception 4-9
- RuntimeException 4-11, 4-12

S

scope

- Pipeline Session properties 4-7
 - Pipeline Session 4-8
 - Request 4-7
- Pipelines 1-7, 1-9
- Webflow 1-7, 1-9

selecting Webflow components in Editors 3-23

serializable objects in the Pipeline Session 2-7

serializing Pipeline Session properties 4-8

server domain name 1-13, 1-16

servlets

- PortalServlet 1-4
 - registering 1-18
- WebflowServlet 1-3, 1-14
 - as Controller 1-7
 - registering 1-17, 1-18

Session Manager 1-3

Session, Pipeline

- clustering 2-6
- configuring 4-3
- definition 2-6
- HTTPSession 2-6
- including in transactions 4-32
- interface 4-3
- JSP tags 2-7
- lifecycle 1-3
- performance 2-6
- properties 2-7
 - getting and setting 4-4
 - scope 4-7
 - serializing 4-8
- serializable objects 2-7
- storing information in 1-6, 2-6
- transactions 2-6
- use of HTTPRequest 2-6

SessionManagerFactory class 4-4

setting up the web.xml file 1-15, 1-16, 2-9,

2-11

- specifying context parameters 1-15
- state machine, Webflow as 1-2, 1-6
- stateless session EJB Pipeline Components 2-6, 4-30
- statelessness and Input Processors 4-17
- static and dynamic URLs 5-2
- support methods 2-7
- synchronizing Webflow and Pipeline data 3-1, 3-7
 - clusters 3-7
 - errors 3-7

T

tags, JSP

- Pipeline Session 2-7, 5-19
- Webflow 1-16, 2-5, 2-12, 5-1
 - importing 5-1

title bar information in Webflow and Pipeline Editors 3-11

toolbars in the Webflow and Pipeline Editors 3-20

transactions

- including Pipeline Session in 2-6, 4-32
- Pipelines 1-4, 2-5, 4-31

transitions, Webflow 1-6

- definition 2-3
- event 2-3, 2-7
- exception 2-8

U

URI for the Web application 1-13

URL

- automatic generation of 1-17, 2-12
- pattern 1-17
- static and dynamic 5-2
- use of begin node 2-9
- Webflow invocation from 1-13

using Webflow 1-1, 1-13

with a proxy front-end 1-13, 1-16
with portals 2-13

V

ValidatedValues class 2-5, 4-17, 5-10, 5-23
 purpose 4-18
validation exceptions 4-24
 InvalidFormDataException 4-24
 InvalidValidatorException 4-24
 MissingFormFieldException 4-24
validation features of Webflow and Pipeline
 Editors 1-12, 2-15, 2-16, 3-5
Validator interface 4-25
validators
 creating custom 4-25

W

Web application container 2-5
Web applications
 and Webflow 1-7
 configuring to use Webflow 1-19
 creating in the EBCC 3-3
 default behavior 2-10
 Pet Flow 6-1
 about 6-2
 accessing 6-3
 features 6-2
 opening in Webflow Editor 6-4
 running 6-4
 URI 1-13
 welcome files 1-18
Web pages
 using Webflow 2-12
web.xml file 1-3, 1-7, 1-13, 1-18, 1-19
Webflow
 architecture 1-1, 1-2, 1-3
 portal 1-4
 as state machine 1-2, 1-6

benefits 1-1, 1-8
components 1-7, 2-1
 adding in Editors 3-24
 begin node 2-9
 configuration error page 2-11
 creating 1-20
 editing names in Editors 3-24
 graphical representations 3-13
 organizing in Editor canvas 3-23
 relationships 2-3
 root component node 2-10
 selecting in Editors 3-23
 using in portals 2-13
 using in Web pages 2-12
 wildcard node 2-10
configuration files 1-1, 1-9
 Content Management System 3-6,
 3-11, 3-12
 contents 1-9
 creating 1-12, 1-19
 location 1-10
 modifying 1-12
 pipeline.properties 1-10
 webflow.properties 1-10
customizing 4-1
definition 1-1
error handling 4-10
execution order 1-3, 2-13
 Presentation Nodes 2-14
 Processor Nodes 2-15
extending 1-4, 4-1, 4-33
flow of control 1-4, 1-5
initial state 2-9
internals 1-4
internationalization 4-38
invocation from a URL 1-13
JSP tags 1-16, 2-5, 2-12, 5-1
 importing 5-1
migration 1-1, 1-10, 1-13
modifying namespaces in Webflow
 Editor 3-4

-
- MVC 1-2, 1-7
 - namespaces
 - default 1-14
 - definition 1-7
 - files 1-9
 - in EBCC 3-4
 - overview 1-1
 - relationship
 - to enterprise applications 1-7
 - to Web applications 1-7
 - scope 1-7, 1-9
 - synchronizing data 3-1, 3-7
 - clusters 3-7
 - errors 3-7
 - tag library 5-2, 5-10
 - transitions 1-6
 - definition 2-3
 - event 2-3, 2-7
 - exception 2-8
 - use of events 1-6
 - using 1-1, 1-13
 - example 6-1
 - in Web pages 2-12
 - with a proxy front-end 1-13, 1-16
 - with portals 2-13
 - validity
 - ensuring 1-12
 - Webflow and Pipeline Editors
 - creating Webflow namespaces and Pipelines 3-5
 - definition 3-1
 - important information about 3-6
 - opening multiple namespace files 3-9
 - opening multiple Pipelines 3-11
 - organizing Webflow components 3-23
 - palettes 3-17
 - read-only mode 3-6, 3-11, 3-12, 3-17
 - role-based security 3-7
 - title bar information 3-11
 - toolbars 3-20
 - using to create and modify Webflow
 - configuration files 1-12
 - validation features 1-12, 2-15, 2-16, 3-5
 - Webflow components 3-9
 - Webflow Executor 1-3
 - Webflow namespaces
 - creating in the Webflow Editor 3-5
 - webflow.tld file 5-10
 - webflow-extensions.wfx file
 - definition 1-10, 3-12
 - location 1-11, 4-35
 - modifying 1-12, 4-33, 4-35
 - WebflowJSPHelper class 2-12
 - WebflowServlet 1-3, 1-14, 4-33
 - as Controller 1-7
 - registering 1-17, 1-18
 - welcome files for Web applications 1-18
 - wildcard node
 - definition 2-10
- X**
- XML configuration files
 - Webflow 1-1, 1-9
 - Content Management System 3-6, 3-11, 3-12
 - contents 1-9
 - creating 1-12, 1-19
 - location 1-10
 - modifying 1-12
 - pipeline.properties 1-10
 - webflow.properties 1-10
 - XML deployment descriptors 1-3, 1-7, 1-13, 1-16, 1-18, 1-19, 2-9, 2-11
 - setting up 1-15
 - XML Schema Definition (XSD) 1-9, 1-10

