BEA WEbLogic Commerce Server
BEA WebLogic Personalization Server

Migration Guide

BEA WebLogic Comerce Server 3.5
BEA WebLogic Personalization Server 3.5
Document Edition 3.5.1
June 2001

Copyright

Restricted Rights Legend

Trademarks or Service Marks

WebLogic Commerce Server Migration Guide

Document Edition	Date	Software Version
3.5.1	June 2001	BEA WebLogic Commerce Server 3.5 BEA Personalization Server 3.5

Contents

4. Upgrading Database Schemas from Prior Releases

5. Changes to WebLogic Personalization Server JSP Tag Library

Index

1 Migrating WebLogic Commerce Server to Version 3.5

The WebLogic Personalization Server is bundled with the WebLogic Commerce Server. This document uses “WebLogic Commerce Server” to refer to both servers.

Be sure to read the “What’s New” page on the e-docs.bea.com Web site for a preview of the new features in this release:
http://e-docs.bea.com/wlcs/docs35/interm/whatsnew.htm

This chapter addresses the changes to WebLogic Commerce Server and WebLogic Personalization Server since the 3.2 release.

This topic includes the following sections:

- Support for WebLogic Server 6.0
  - Changes to the WebLogic Commerce Server Directory Structure
  - All Pages Must Be in a Web Application
- Introducing the E-Business Control Center
- Changes to the Rules Editor in Release 3.5
- Changes to the JSP Tag Libraries
- Database Schema Migration Information

Support for WebLogic Server 6.0

The Campaign Manager for WebLogic, WebLogic Commerce Server, and WebLogic Personalization Server products now support WebLogic Server 6.0. The support for this new version of WebLogic Server includes a WebLogic Server domain, which contains the WebLogic Server Administration Console and the sample Web applications. A utility to migrate properties from the weblogic.commerce file to the new WebLogic Server Administration Console is provided, and the sample Web applications are reorganized and deployed in a single Enterprise Application.

Changes to the WebLogic Commerce Server Directory Structure

New features and improvements to WebLogic Server 6.0 have caused changes in the WebLogic Commerce Server 3.5 directory structure. This section shows the old and new directory structures and points out key changes. Be sure to examine the installation directory once you have installed WebLogic Commerce Server, to become familiar with the changes.

Figure 1-1 WebLogic Commerce Server 3.2 Sample Directory Structure

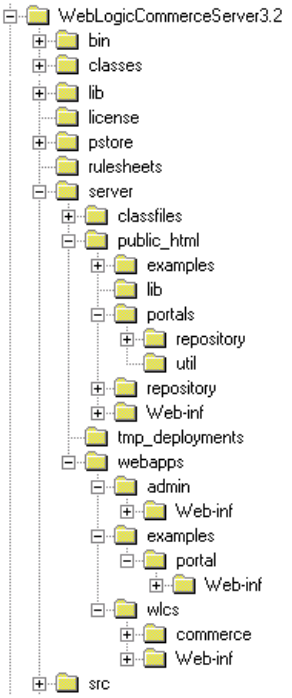
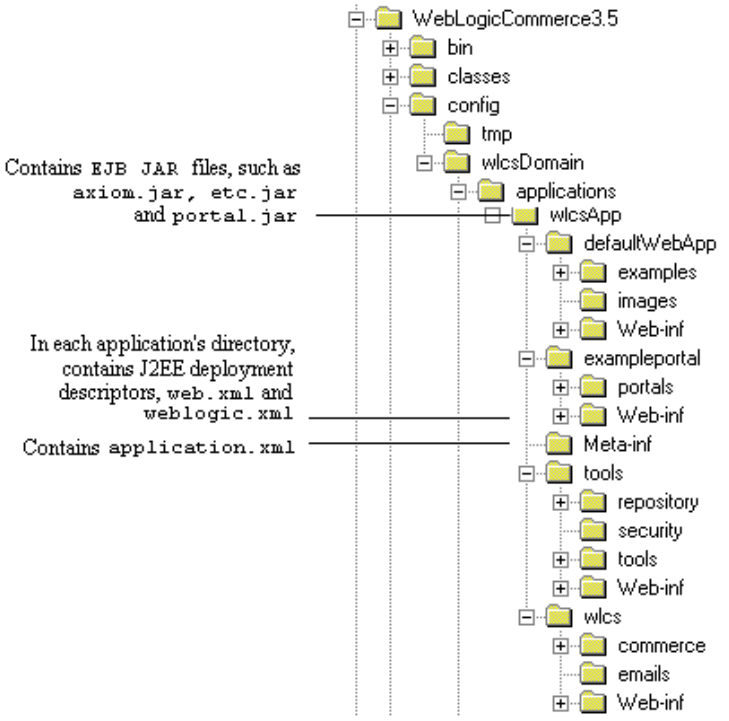


Figure 1-2 WebLogic Commerce Server 3.5 Sample Directory Structure



Key changes in the 3.5 WebLogic Commerce Server include the following:

- The `wlcsDomain` directory is the domain directory; the `wlcsDomain\applications` directory contains all other files and directories listed here.
- The `wlcsApp` directory contains the WebLogic Commerce Server application, including EJB JAR files.
- The `wlcsApp\META-INF` directory contains the `application.xml` file, the J2EE application deployment descriptor.
- The `WEB-INF` directory in each Web application directory contains deployment descriptors.

- The `wlcsApp\defaultWebApp` directory replaces `public_html` as the default Web application directory. You can specify one application as the default.
- Additional Web applications are stored at the same level as `wlcsApp\defaultWebApp`. The `tools` and `wlcs` folders are shown as examples.

For information about migrating the WebLogic Server, refer to the documentation for *WebLogic Server 6.0*.

All Pages Must Be in a Web Application

All Web-based pages are now required to be deployed as a Web application.

The direction of the product is to deploy one portal per Web application and use the application deployment features enabled through the WLS console, instead of the hot deployment portal model of the previous release. When migrating existing portals built on previous releases of the product, convert your non-Web applications to Web applications. This will result in one portal per Web application. You can use the out-of-the-box `exampleportal` as a model.

WebLogic Server supplies a default Web application for simple pages that do not require deployment descriptor properties.

For more information, see the section “Deploying New Portals as Web Applications” in the *Guide to Creating Portals and Portlets*.

Introducing the E-Business Control Center

This release introduces the E-Business Control Center, a GUI tool designed to simplify the way business professionals manage their online customer relationships. This 100% Java client is a powerful desktop application that is easy to use and makes sense to business users by interacting with them in their own terms.

The E-Business Control Center replaces the Rules Manager as the mechanism for creating and editing rules. Using the tool, business users can now create their own customer segments (classification rules) and content selectors (content selector rules).

Specialized versions of the E-Business Control Center are packaged with the WebLogic Commerce Server and the WebLogic Personalization Server products. Or, you may choose to license the new Campaign Manager for WebLogic product, which includes the most comprehensive version of the E-Business Control Center currently available.

Changes to the Rules Editor in Release 3.5

The new Personalization Rules Manager allows business users to fine-tune user-system interactions using plain-English commands within easy-to-use rule editing templates. The Personalization Rules Manager drives BEA's embedded rules engine and eliminates the need to master complex Boolean logic to create and edit rules.

Note: This section assumes you are migrating from Release 3.2. If you are migrating from Release 3.1 or earlier, begin with the section "Changes to the Rules Editor in Release 3.1" in Chapter 3, "Migrating WebLogic Personalization Server to Version 3.1."

The XML format (expressed in XMLSchema) used to describe rule sets for the 3.2 release of WebLogic Commerce Server has been replaced by a more manageable XML format designed to grow with future needs. A new tool, the E-Business Control Center, has also been introduced to allow you to more easily enter rules. This change enables you to specify a greater range of rules.

To use your existing Web Logic Commerce Server 3.2 rules with the new format, follow the steps in this section.

Note: This procedure provides you with files listing all your rules, which you can print and use as a reference to reenter them. As an alternative to steps 1-6, you can view your rules from within the Rules Management JSP pages in WebLogic Commerce Server 3.2, and use that display as the template for the conversion. Then go to step 7 and follow the directions there.

1. Ensure that the correct systems are installed.
   - WebLogic Server 5.1 and the required service pack. See http://edocs.bea.com/wlcs/docs32/relnotes/relnotes.htm#platforms.

- WebLogic Commerce Server 3.2.
- WebLogic Server 6.0 and required service pack and rolling patch. See http://edocs.bea.com/wlcs/docs35/install/platforms.htm.
- Campaign Manager for WebLogic, WebLogic Commerce Server, or WebLogic Personalization Server 3.5.
- E-Business Control Center, which contains the correct rules parser.

2. Install WebLogic Commerce Server 3.5. WLCS must be both *installed* and *running*. Installing WebLogic Commerce Server 3.5 provides the files used during the migration; verify that the files are there before you continue. The paths to the files on your computer might be slightly different, but the filenames must be the same.

   `<wlcs3.5 install dir>/bin/win/dumprules32.bat` (Windows only)

   `<wlcs3.5 install dir>/bin/unix/dumprules32.sh` (UNIX only)

   `<wlcs3.5 install dir>/classes/rules-tools-common.properties`

   `<wlcs3.5 install dir>/classes/rules-tools-query-32.properties`

   `<wlcs3.5 install dir>/classes/rules-tools-dump.properties`

   `<wlcs3.5 install dir>/classes/com/bea/commerce/platform/rules/tools/RuleSetProcessorHarness.class`

   `<wlcs3.5 install dir>/classes/<supporting program classes>`

3. Edit the files shown in Table 1-1 to let the migration script know where your WLS installation, existing rules, and other relevant files are located.

Table 1-1 Files and Corresponding Variables for Rules Migration

File	Variable	Variable Value
`dumprules32.bat` or `dumprules32.bat`	`WLS_51_HOME`	The root directory of an existing WLS 5.1 installation, such as `/opt/bea/wlserver5.1`
	`WLCS_35_TOOLS_LIB_EXT`	The `<root>/lib/ext` directory of the Campaign Manager tool installation, such as `/opt/bea/WebLogicCommerceServer3.5/tools/lib/ext`

Table 1-1 Files and Corresponding Variables for Rules Migration (Continued)

File	Variable	Variable Value

`rules-tools-query-32.properties`	`t3-host`	The name of a running WLCS 3.2 installation host, in the following format:
		Syntax:
		`http://<t3-host>:<t3-port>`
		Example:
		`http://localhost:7501`

	`t3-port`	The name of a running WLCS 3.2 installation port, in the following format:
		Syntax:
		`http://<t3-host>:<t3-port>`
		Example:
		`http://localhost:7501`

4. Use the `dumprules32` script to create files containing lists of your rules. This script creates two files for each of your rule sets in your existing database.

   You will use one of them as a reference to reenter the rules in the new format.

   a. Determine a location (directory) on the local filesystem to write out 3.2 rule sets, such as `C:\rules32`.

   b. Manually create that directory. For example, in Windows on the command line you would type `C:\> mkdir rules32`.

   c. Run the `dumprule32` script from within the home directory of the script, and specify the directory you created in the previous step. For example, in Windows you would run the following command from the command line:

```
C:\opt\bea\WebLogicCommerceServer3.5\bin\win32> dumprules32.bat C:\rules32
```

   Two files are written for each rule set:

   <*rule set name*>`.ruleset`—contains the rule sets in XML.

   <*rule set name*>`-txt.ruleset`—contains the rule sets in a more readable text format.

   You will use the `-txt.ruleset` file for each rule set to reenter your rules.

5. Go to the directory you specified when you ran the script. Print each `-txt.ruleset` file that was created, or open them in a text application, so that they are ready for you to use them.

6. Review the printed file so that you can easily determine what each rule states from the printout. The following is a partial example of a `-txt.ruleset` file.

```
-- Begin File AcmeRules-txt.ruleset --
(RuleSet)
Name: SampleRuleSet
Description: Demonstrating And/or Logic in Rules Conversion
(Rule)
       Name: SampleRule1
       Type: classifier
           (When)
               REQUEST.DefaultRequestPropertySet.Authorization Scheme eq 1
                   (Or)
                       REQUEST.DefaultRequestPropertySet.Character Encoding eq 2
                       REQUEST.DefaultRequestPropertySet.Character Encoding gt 5
           (Then)
               (New)
                   ClassName: Classification
                       (Arguments)
                       (Constant)
                       Type: string
                       Value: RuleDemonstrated
```

You need to pay particular attention to the lines under the (`When`) heading in order to recreate your rules correctly, if there are lines under (`When`) as well as (`Or`). Both the line after the (`When`) and the *first* line after the (`Or`) must be true for the logic under (`Then`) to be run.

Note: The rule set printout makes it seems as though either the (`When`) item or the first (`Or`) item could be true to fulfill the condition; however, that is not the case. As stated previously, both must be true.

For example, in the sample rule set given, the string "`RuleDemonstrated`" will be displayed in either of the following situations:

If `DefaultRequestPropertySet.Authorization Scheme` equals 1 *and* `REQUEST.DefaultRequestPropertySet.Character Encoding` equals 2

or

If `DefaultRequestPropertySet.Authorization Scheme` equals 1 *and* `REQUEST.DefaultRequestPropertySet.Character Encoding` is greater than 5

The following would *not* fulfill the requirements for the condition (`Then`):

If `DefaultRequestPropertySet.Authorization Scheme` equals 1 **or** `REQUEST.DefaultRequestPropertySet.Character Encoding` is greater than 5

7. Enter the new rules in the E-Business Control Center. For information about using the E-Business Control Center, see *Using the BEA E-Business Control Center*.

Note: Rules are created in the E-Business Control Center. This GUI tool is designed to allow Business Analysts to develop their own content selector rules and classifier rules. Because the Business Analysts are not exposed to the concept of rules, you will see content selector rules called simply "content selectors" and classifier rules referred to as "customer segments."

For more information about rules, see the following chapters in the *Guide to Building Personalized Applications*:
Creating Personalized Applications with the Advisor
Foundation Classes and Utilities
Introducing the Rules Manager
Working with Content Selectors

Changes to the JSP Tag Libraries

The following changes were made to JSP tag libraries in Release 3.5.

- Removed Tags

  `<es:preparedStatement>`

- New Ads and Placeholder Tags

  `<ad:adTarget>`

  `<ph:placeholder>`

- New Event Tracking Tags

  `<tr:clickContentEvent> Content Tag`

  `<tr:displayContentEvent> Content Tag`

  `<trp:clickProductEvent> Product Tag`

  `<trp:displayProductEvent> Product Tag`

  `<trc:clickCampaignEvent> Campaign Tag`

- New Webflow Tag

  `<webflow:setValidated Value>`

- New E-Business Tag

  `<eb:smnav>`

- Changes to Personalization Tags

  `<pz:div>`

  `<pz:contentSelector>`

For more information about these tags, see the section "JSP Tag Changes in Version 3.5," in Chapter 5, "Changes to WebLogic Personalization Server JSP Tag Library."

Database Schema Migration Information

Release 3.5 of WebLogic Commerce Server and WebLogic Personalization Server provides enhancements and changes that require you to update the schemas and migrate the data.

Note: If you are using the Oracle 8.0.5 or 8.1.5 database, you must upgrade to Oracle 8.1.6 or greater before migrating to Release 3.5 of WebLogic Commerce Server and WebLogic Personalization Server. Release 3.5 uses CLOBs and BLOBs instead of LONG RAW characters. Oracle 8.0.5 and 8.1.5 do not support CLOBs and BLOBs.

For detailed migration information, refer to Chapter 4, "Upgrading Database Schemas from Prior Releases." The information specific to the 3.5 release can be found in the section "Upgrading Database Schemas from 3.2 to 3.5."

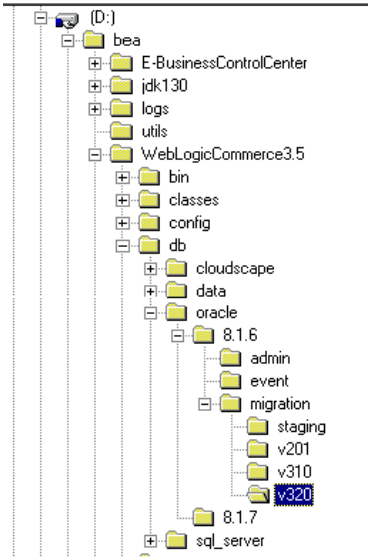
Note: If you are starting at a release earlier than Release 3.2 of WebLogic Commerce Server and WebLogic Personalization Server, begin at the appropriate section in that chapter.

Information about migrating specific databases is also provided in readme files. Navigate to the readme file for your specific database using this directory path:

`../db/<vendor>/<version>/migration/v320/readme.text`

where `<vendor>` is the name of the database (for example, Oracle) and `<version>` is the release number for that database. Figure 1-3 shows the directory structure on a Windows system.

Figure 1-3 Navigate to the Database readme Files



2 Migrating WebLogic Commerce Server to Version 3.2

This chapter describes the changes between WebLogic Commerce Server and WebLogic Personalization Server Release 3.2 and the previous release, 3.1.1.

The WebLogic Personalization Server is bundled with the WebLogic Commerce Server. This document uses “WebLogic Commerce Server” to refer to both servers.

This topic includes the following sections:

- New Java 2 SDK for Enhanced Performance
- New Third-Party Integrations
  - International Tax Support from TAXWARE
  - E-Marketing Analysis Using Broadbase
  - Content Management with Interwoven Content Express
- New Webflow and Pipeline Editor
- Changes to the JSP Tag Libraries
- Database Schema Migration Information

New Java 2 SDK for Enhanced Performance

The BEA WebLogic Commerce Server and WebLogic Personalization Server now require the Java 2 SDK with the HotSpot Server Virtual Machine. Using the Java 2 SDK with HotSpot may enhance performance for your production environment.

New Third-Party Integrations

BEA WebLogic Commerce is now integrated with the following third-party products:

- International Tax Support from TAXWARE
- E-Marketing Analysis Using Broadbase
- Content Management with Interwoven Content Express

International Tax Support from TAXWARE

The WORLDTAX System from TAXWARE International, Inc. is the most comprehensive calculation system for international taxes available in the industry. The WORLDTAX System calculates and reports Value Added Tax (VAT), Goods and Services Tax (GST), sales tax, and consumption tax in many countries. BEA has tested the countries of France, Germany, Italy, South Korea, Spain, and the United Kingdom for accuracy with the WebLogic Commerce Server.

E-Marketing Analysis Using Broadbase

This release of the BEA WebLogic Commerce and Personalization Server provides integration with Broadbase for customer analysis. The user profile, customer, and order information can be extracted from the WebLogic Commerce and Personalization Server database and used to enable E-Marketing analysis within Broadbase. Please contact Broadbase Software, Inc. for more information.

Content Management with Interwoven Content Express

In addition to the existing limited-user version of Documentum 4i, this release includes Interwoven Content Express, an entry-level content management tool. These content management products allow developers to quickly create personalized content applications, using software from either of the leading content management vendors.

New Webflow and Pipeline Editor

The BEA WebLogic Commerce Server now includes the Webflow and Pipeline Editor, a JSP-based administration tool specifically designed to help you modify the default `webflow.properties` and `pipeline.properties` configuration files. It also provides you with validation tools that enable you to check the syntax of your Webflow and verify whether the necessary components exist within the Webflow. By modifying and validating your Webflow with the Webflow and Pipeline Editor, you can eliminate errors that may otherwise be difficult to track.

Changes to the JSP Tag Libraries

The following changes were made to JSP tag libraries in Release 3.2.

- New Content Management tag

`<cm:getProperty>`

- Changes to Content Management tags

  `<cm:printDoc>` has a new attribute, `baseHref`

- New tags to support the Flow Manager

  `<fm:getApplicationURI>`

  `<fm:getCachedAttribute>`

  `<fm:setCachedAttribute>`

  `<fm:removeCachedAttribute>`

  `<fm:getSessionAttribute>`

  `<fm:setSessionAttribute>`

  `<fm:removeSessionAttribute>`

- Changes to Utility tags

  `<es:preparedStatement>` has a new attribute, `transactionIsolationLevel`

For more information about these tags, see the section "JSP Tag Changes in Version 3.2," in Chapter 5, "Changes to WebLogic Personalization Server JSP Tag Library."

Database Schema Migration Information

Release 3.2 of WebLogic Commerce Server and WebLogic Personalization Server introduces scripts that you can use to create Oracle tablespaces and user accounts for the WebLogic Commerce Server and WebLogic Personalization Server database schema. It also introduces scripts that you can use to export the Oracle database objects from a source environment and import them into a destination environment. With these export/import scripts, you can move data from a staging environment into your production environment without having to recreate all your content.

Release 3.2 introduces schema changes and restrictions for the length of data allowed in various columns. To upgrade databases from Release 3.1.1 to Release 3.2, complete the following tasks:

- Upgrade the WebLogic Personalization Server Schema

- Upgrade the WebLogic Commerce Server Schema (only if you use WebLogic Commerce Server)
- Verify the Upgrade
- Remove Temporary Tables

For detailed migration information, refer to Chapter 4, "Upgrading Database Schemas from Prior Releases." The information specific to the 3.2 release can be found in the section "Upgrading Database Schemas from 3.1.1 to 3.2."

Note: If you are starting at a release earlier than Release 3.1.1 of WebLogic Commerce Server and WebLogic Personalization Server, begin at the appropriate section in that chapter. Database migration must be done sequentially. You cannot skip a release.

Information about migrating specific databases is also provided in readme files. Navigate to the readme file for your specific database using this directory path:

```
../db/<vendor>/<version>/migration/v320/readme.text
```

where `<vendor>` is the name of the database (for example, Oracle) and `<version>` is the release number for that database.

3 Migrating WebLogic Personalization Server to Version 3.1

This chapter describes the changes between WebLogic Personalization Server 2.0.1 and WebLogic Personalization Server 3.1. It includes specific information for migrating existing code to WebLogic Personalization Server 3.1.

Note: Both the WebLogic Commerce Server and WebLogic Personalization Server functionality now reside in a unified Java package hierarchy located at com.beasys.commerce.

This section includes the following topics:

- Navigating with Flow Manager
- Changes to the Personalization Advisor
- Changes to the Rules Editor in Release 3.1
- Changes to Content Management
- Changes to the JSP Tag Library
- Database Schema Migration Information

Navigating with Flow Manager

The Flow Manager is a servlet implementation that allows the hot deployment of applications within the WebLogic Application Server. Flow Manager also adds flexibility to navigation through the system—it moves navigation information off the JSP page and into a single point of control. Using a destination determiner and a destination handler, the Flow Manager dynamically determines a destination for a given page request and dynamically handles it.

This topic includes the following sections:

- Deprecated Service Managers
- Hot Deployment
- Dynamic Flow Determination and Handling
- Property Set Usage
- Go With the Flow: Migrating to the Flow Manager
- Accessing Your Application via the Flow Manager

For more information, see "Flow Manager" in the Foundation chapter in the *WebLogic Personalization Server Developer's Guide* (in the release 3.1 documentation set).

Deprecated Service Managers

In WebLogic Personalization Server 3.1, all of the functionality of the JSP Service Manager and the Portal Service Manager has been ported to the new Flow Manager. The JSP Service Manager and the Portal Service Manager have been deprecated.

Hot Deployment

The Flow Manager is a servlet implementation that allows the hot deployment of applications within the WebLogic Application Server.

Registering a new portal or a new application no longer requires restarting the server, as it did in WebLogic Personalization Server 2.0.1. Instead of registering servlets in the `weblogic.properties` file, the Flow Manager relies on a property set to obtain information about a specific application or portal. You simply create a new instance of a property set to hold the equivalent parameters that were in the properties file. Default values are supplied during property set creation. Any changes become visible according to a configurable refresh setting in the property set.

Note: In Release 3.5, the direction of the product will be to deploy one portal per Web application and use the application deployment features enabled through the WLS console, instead of the hot deployment portal model. For more information, see "All Pages Must Be in a Web Application" in Chapter 1, "Migrating WebLogic Commerce Server to Version 3.5."

Dynamic Flow Determination and Handling

Flow Manager also provides the basic infrastructure to support the new Webflow functionality. Webflow dynamically determines a destination for a given page request and dynamically handles it. Using a destination determiner and a destination handler, the Flow Manager moves navigation information off the JSP page and into a single point of control.

The old service managers relied on a hidden form field in the current page to determine where an HTTP request should be routed:

```
<input type="hidden" name="<%=DESTINATION_TAG%>"
value="<%=PortalJspBase.getRequestURI(request)%>">
```

This scheme required destination (or routing) information to be distributed across the JSP/HTML pages. While this works fine, it can be cumbersome to modify if destination values need to change.

The Flow Manager, on the other hand, allows the determination of page routing to be centralized on the server based on an application's needs.

Backward Compatibility

For backward compatibility, default implementations of the destination determiner and the destination handler are provided which support destination information being passed via the DESTINATION_TAG mentioned above. These implementations are:

`com.beasys.commerce.portal.flow.PortalDestinationDeterminer`
and
`com.beasys.commerce.foundation.flow.ServletDestinationHandler`

Also, for non-portal-based personalized applications, the following default implementations may be used:

`com.beasys.commerce.foundation.flow.jsp.DefaultDestinationDeterminer`
and
`com.beasys.commerce.foundation.flow.ServletDestinationHandler`

Property Set Usage

A new class of property sets, "Application Initialization Property Sets" has been added to the Property Set Management Administration Tools. These are the property sets used by the Flow Manager in support of portal (`_DEFAULT_PORTAL_INIT`) and non-portal-based (`_DEFAULT_APP_INIT`) personalized applications.

Three new properties have been added to support the Flow Manager:

- **destinationdeterminer** Property

  The destination determiner is responsible for evaluating an HTTP request and determining which servlet to route it to.

  The value provided for this property should be the name of a class that implements the `com.beasys.commerce.foundation.flow.DestinationDeterminer` interface. If appropriate, use a default implementation provided by WebLogic Personalization Server or WebLogic Commerce Server. Otherwise, develop your own implementation according to the needs of your application.

- **destinationhandler** Property

  Given a destination route, the destination handler is responsible for invoking the requested processing.

  The value provided for this property should be the name of a class that implements the `com.beasys.commerce.foundation.flow.DestinationHandler` interface. If appropriate, use a default implementation provided by WebLogic Personalization Server or WebLogic Commerce Server. Otherwise, develop your own implementation according to the needs of your application.

- **ttl (time-to-live)** Property

  ttl, which stands for time-to-live, represents how often (in milliseconds) the Flow Manager reloads the `_APPLICATION_INIT` property set from the database. This allows changes that you make to the `_APPLICATION_INIT` property set to be visible while the application or portal is running.

Note: To force immediate reloading of the property set, append the "flowReset" argument to your URL, like this:
`http://localhost:7001/application/exampleportal?flowReset=true`

Go With the Flow: Migrating to the Flow Manager

To migrate your portal or non-portal application to use the Flow Manager, do the following:

To create a new property set:

1. Open the Administration Tools Home page. Click the Property Set Management icon to open the Property Set Management screen.
2. From the main Property Set Management screen, click Create.
3. Name the new property set you are creating (100 character maximum). The name of the property set should be the same as the name you used to create the portal, or the name you will use to access the application.
4. Enter a description of the property set (255 character maximum).

5. From the Copy Properties From drop-down list, select
   `APPLICATION_INIT._DEFAULT_PORTAL_INIT` (for a portal)
   or
   `APPLICATION_INIT._DEFAULT_APP_INIT` (for a non-portal application).
6. From the Property Set Type drop-down list, select Application Init.
7. Click the Create button.
8. At the top of the page, in red, you will see the message "Property Set creation was successful." (Or, you will see an error message indicating why the property set was not created.)
9. Click Back to return to the main Property Set Management screen.

To set parameters for your portal or application:

1. From the Property Set Management Home page, under the Application Initialization Property Sets heading, click the name of the property set you just created.
2. A Property Set page comes up, allowing you to set parameters.
3. **(Note:** For non-portal applications, skip this step.) To edit the portal name, click the Edit button to the right of the "portal name" property. Change the default value from `UNKNOWN` to the name of your portal, as you created it in Portal Management.
4. Edit the `destinationdeterminer` property. Either accept the default, or edit to provide your own implementation of these classes.
5. Edit the `destinationhandler` property. Either accept the default, or edit to provide your own implementation of these classes.
6. Customize any other properties you choose. For information about customizing properties in portals, see Creating and Managing Portals in the *WebLogic Personalization Server User's Guide* (in the release 3.1 documentation set) and Building a Custom Portal Step-by-Step in the*WebLogic Personalization Server Developer's Guide* (in the release 3.1 documentation set).
7. When you have finished setting properties, click the Finished button at the bottom of the page.

Note: In WebLogic Personalization Server 2.0.1, you registered servlets in the `weblogic.properties` file. This is not required for WebLogic Personalization Server 3.1. You have the option to remove them, but it is not required. The WebLogic Personalization Server will ignore them.

Accessing Your Application via the Flow Manager

The exact URL you use depends upon whether or not you have deployed your application as a Web application. WebLogic Personalization Server 3.1 includes sample configurations for both a Web application/Web archive deployment and a non-Web application configuration. For more information, see the chapter "Using the Catalog Application in a Portal" in the *WebLogic Personalization Server Developer's Guide* (in the release 3.1 documentation set).

Changes to the Personalization Advisor

For WebLogic Personalization Server 3.1, the Personalization Advisor has been renamed to Advisor and has undergone some API changes. However, its functionality remains the same. The Advisor has been improved to provide better error reporting and to make use of the unified logging facility provided by WebLogic Commerce Server 3.1.

This topic includes the following sections:

- JSP Tags Ported to Use the New Advisor
- Deprecated Personalization Advisor Classes
- Changes in Advisor APIs
- Terminology Change: Agents Changed to Advislets

JSP Tags Ported to Use the New Advisor

The three pz library tags (`pz:div`, `pz:contentQuery`, and `pz:contentSelector`) have been changed to use the new Advisor Session Bean. However, the tag usage remains the same. For more information, see the JSP Tag Library Reference in the *WebLogic Personalization Server Developer's Guide* (in the release 3.1 documentation set).

To use the `<pz:div>` and `<pz:contentSelector>` tags, you are no longer required to insert the following JSP directive into your JSP code:

```
<%@ page extends="com.beasys.commerce.axiom.p13n.jsp.P13NJspBase"
%>
```

However if it is already in your code, you do not need to remove it.

Deprecated Personalization Advisor Classes

All of the Java classes for the Personalization Advisor released in WebLogic Personalization Server 2.0.1 have been deprecated. This includes all of the Java classes in the following Java packages:

```
com.beasys.commerce.axiom.p13n.advisor
com.beasys.commerce.axiom.p13n.agents
```

In WebLogic Personalization Server 3.1, these deprecated classes are replaced by new Advisor Java packages. They include:

```
com.beasys.commerce.axiom.advisor
com.beasys.commerce.axiom.advislets
```

The Personalization Advisor Bean has been replaced by the new Advisor Bean.

This change only affects the case when the Advisor API is used directly and is transparent to JSP tag users.

Changes in Advisor APIs

The changes made while porting the WebLogic Personalization Server 2.0.1 Personalization Advisor interface to the new Advisor interface are as follows:

- The Personalization Advisor `pzTechnique` parameter is not supported in the new Advisor implementation.
- The `createRequestTemplate` method parameters have been simplified to use a single string lookup name for the advice request, instead of a fully qualified class name. The three advice request lookup names supported for WebLogic Personalization Server 3.1 are `ClassificationAdviceRequest`, `ContentSelectorAdviceRequest`, and `ContentQueryAdviceRequest`.

The following example shows the difference in the `createRequestTemplate` method between the Personalization Advisor and the Advisor.

Personalization Advisor Interface

```
public AdviceRequest createRequestTemplate(
    String adviceRequestClassName,
    String pzTechnique)
    throws IllegalArgumentException,
           PersonalizationAdvisorException,
           RemoteException;
```

Advisor Interface

```
public AdviceRequest createRequestTemplate(
    String theKindOfRequest)
    throws IllegalArgumentException,
           AdvisorException,
           RemoteException;
```

Terminology Change: Agents Changed to Advislets

The three WebLogic Personalization Server 2.0.1 Personalization Agents have been renamed and repackaged to advislets. The following table defines the mapping between the WebLogic Personalization Server 2.0.1 Agent Java classes to the WebLogic Personalization Server 3.0 advislet Java classes.

2.0 Agent class	com.beasys.commerce.axiom.p13n.agents.ClassificationAgentImpl
3.1 Advislet class	com.beasys.commerce.axiom.advisor.advislets.ClassificationAdvisletImpl
2.0 Agent class	com.beasys.commerce.axiom.p13n.agents. ContentSelectorAgentImpl
3.1 Advislet class	com.beasys.commerce.axiom.advisor.advislets.ContentSelectorAdvisletImpl
2.0 Agent class	com.beasys.commerce.axiom.p13n.agents. ContentQueryAgentImpl
3.1 Advislet class	com.beasys.commerce.axiom.advisor.advislets.ContentQueryAdvisletImpl

Changes to the Rules Editor in Release 3.1

The WebLogic Personalization Server provides rule sets that include a set of classifier and content selector rules. These rule sets act as containers for rules that match personalized content with users.

This topic includes the following sections:

- Relationship Between Rules and Property Sets
- The Use of AND or OR to Connect Expressions
- Change the Word 'Rule Sheet' to 'Rule Set'

For more information, see "Creating and Managing Rules" in the *WebLogic Personalization Server User's Guide* (in the release 3.1 documentation set).

Relationship Between Rules and Property Sets

In previous releases, the rule sets (also called rule sheets) were associated with property sets that defined the attributes available for user and group profiles. Once defined, this relationship between rules and property sets could not be undone.

In the current WebLogic Personalization Server 3.1 release, there is no longer an association between a rule set and a property set. Rules within a rule set may refer to any properties.

The Use of AND or OR to Connect Expressions

The Rules Editor now allows the use of "and" or "or" to connect expressions that contain comparators.

Change the Word 'Rule Sheet' to 'Rule Set'

For consistency, an effort has been made to change the word "rule sheet" to "rule set" or `ruleSet` in all cases. However, the following legacy code continues to use Rulesheet:
`jdbc://com.beasys.commerce.axiom.reasoning.rules.RulesheetDefinitionHome`

Changes to Content Management

This topic includes the following sections:

- New Features in <cm:select> and <cm:selectById> Tags
- Changes to EJB Deployment Descriptors
- Changes to Object Interfaces
- Changes to the BulkLoader

New Features in <cm:select> and <cm:selectById> Tags

To retrieve Content or Documents, use a ContentManager or DocumentManager with `<cm:select>` or `<cm:selectById>`. The default DocumentManager is deployed at `com.beasys.commerce.axiom.document.DocumentManager`. For more information, see "Configuring WebLogic Commerce Properties" in the chapter Creating and Managing Content in the *WebLogic Personalization Server User's Guide* (in the release 3.1 documentation set).

The `<cm:select>` and `<cm:selectById>` tags now support a session-based, per-user Content cache for content searches. For more information, see "Content Cache" in the chapter Creating and Managing Content in the *WebLogic Personalization Server User's Guide* (in the release 3.1 documentation set).

The Content Manager now supports non-EJB context objects. The `<cm:select>` and `<cm:selectById>` tags support an optional `readOnly` parameter. For more information, see "readOnly Content Tag" in the chapter Creating and Managing Content in the *WebLogic Personalization Server User's Guide* (in the release 3.1 documentation set).

Changes to EJB Deployment Descriptors

Deployment descriptors handle the configuration for the Content Manager. This section describes the changes to the deployment descriptors:

- Document Schema EJB Deployment Descriptor
- DocumentManager EJB Deployment Descriptor
- Document EJB Deployment Descriptor (Deprecated)

Document Schema EJB Deployment Descriptor

Two EJB variables have been removed:
`SmartConnectionPoolClass`
`SmartBMP_URL`

Five EJB variables have been added:
`UseDataSource`
`DocPoolURL`
`DocPoolDriver`
`jdbc/docPool`
`jdbc/commercePool`

One EJB variable remains the same:
`SmartBMPUpdate`

For more information, see "Configuring the Document Schema EJB Deployment Descriptor" in the chapter Creating and Managing Content in the *WebLogic Personalization Server User's Guide* (in the release 3.1 documentation set).

DocumentManager EJB Deployment Descriptor

All the EJB variables have been removed:
`UseDefaultHomeNames`
`ContentHome`
`SchemaHome`

Six EJB variables have been added:
`PropertyCase`
`jdbc/docPool`
`ejb/ContentHome`
`ejb/SchemaHome`
`UseDataSource`
`DocPoolURL`
`DocPoolDriver`

For more information, see "Configuring the DocumentManager EJB Deployment Descriptor" in the chapter Creating and Managing Content in the *WebLogic Personalization Server User's Guide* (in the release 3.1 documentation set).

Document EJB Deployment Descriptor (Deprecated)

Note: The Document EJB has been deprecated and should not be used. Use the DocumentManager EJB instead.

To support legacy code, the Document EJB has been upgraded as follows:

Two EJB variables have been removed:
`SmartConnectionPoolClass`
`SmartBMP_URL`

Four EJB variables have been added:
`UseDataSource`
`DocPoolURL`
`DocPoolDriver`
`jdbc/docPool`

Two EJB variables remain the same:
`SmartBMPUpdate`
`Propertycase`

- *`SmartBMPUpdate`*: Set to `false`.
- *`UseDataSource`*: Controls whether `jdbc/docPool` (`true`) or `DocPoolURL` (`false`) is used to get connections. Defaults to `true`.

- *DocPoolURL*: Specifies the JDBC URL to the document JDBC connection to use (if `UseDataSource` is `false`). Should point to a connection pool. For example: `jdbc:weblogic:pool:docPool`
- *DocPoolDriver*: Specifies the JDBC driver class to use to connect to the `DocPoolURL`. This is optional. If not specified, the EJB will try to determine the appropriate JDBC driver class from the `DocPoolURL`.
- *jdbc/docPool*: A Data Source reference to the document JDBC connection pool. This should correspond to the Data Source attached to the WebLogic connection pool that uses the document reference implementation JDBC driver.
- *PropertyCase*: This sets how the `DocumentImpl` modifies incoming property names. If this is *lower*, all property names are converted to lowercase. If this is *upper*, all property names are converted to uppercase. If this is anything else or not specified, property names are not modified. Use *lower* or *upper* if the `SmartBMP` class expects everything in a certain case (for example, the Documentum `SmartBMP` expects everything in lowercase). For the document reference implementation, do not specify the *PropertyCase*.

Other `SmartBMP` classes for other document management systems will possibly require more and/or different EJB environment variables.

Changes to Object Interfaces

The `ConfigurableEntity`, `Content`, `Document`, `User` and `Group` interfaces no longer extend EJBObject. Instead, those interfaces are code-identical to the original 2.0.1 versions (same method signatures).

The interfaces `ConfigurableEntityRemote`, `ContentRemote`, `DocumentRemote`, `UserRemote` and `GroupRemote` extend both EJBObject and their respective non-EJBObject interfaces.

For more information, see “Object Interfaces” in the chapter “Creating and Managing Content” in the *WebLogic Personalization Server User’s Guide* (in the release 3.1 documentation set).

Changes to the BulkLoader

The BulkLoader now accepts a `-encoding <enc>` and `-schemaName` option. For more information, see "Command Line Usage" in the chapter Creating and Managing Content in the *WebLogic Personalization Server User's Guide* (in the release 3.1 documentation set).

Changes to the JSP Tag Library

Five new tags were introduced in WebLogic Personalization Server Release 3.1:

<ps:getPropertyNames>

<ps:getPropertySetNames>

<i18n:localize>

<i18n:getMessage>

<wl:repeat>

For information about these tags, see Chapter 5, "Changes to WebLogic Personalization Server JSP Tag Library," in this guide. Also refer to the following sections in that chapter for information about changes to the tags:

- New JSP 1.1 Naming Conventions
- Changes to Tag Attributes
- Global Changes
- Tag Migration Roadmap
- Additional Notes About JSP Tags

Database Schema Migration Information

The WebLogic Personalization Server Schema is now documented in the *WebLogic Personalization Server Developer's Guide* (in the release 3.1 documentation set).

Updated User Management Schema Table

A new column called PROFILE_TYPE was added to the WLCS_USER table since WebLogic Personalization Server Release 2.0.1. This column holds the name of the Unified Profile Type of which the User is an instance.

For more information about migrating your database, refer to the section "Upgrading WebLogic Personalization Server Database Schemas from 2.0.1 to 3.1.1" in Chapter 4, "Upgrading Database Schemas from Prior Releases," in this guide.

4 Upgrading Database Schemas from Prior Releases

If you are upgrading your WebLogic Commerce Server and WebLogic Personalization Server installation from a prior release, you must also upgrade your database to the corresponding schema. This chapter describes the following tasks:

- Upgrading WebLogic Personalization Server Database Schemas from 2.0.1 to 3.1.1
- Upgrading WebLogic Personalization Server Database Schemas from 2.0.1 to 3.2
- Upgrading Database Schemas from 3.1.1 to 3.2
- Upgrading Database Schemas from 3.2 to 3.5

You must upgrade databases in the sequence of the preceding list; you cannot skip a Release in the migration path.

Note: For information on upgrading a WebLogic Commerce Server 2.0.1 database to the WebLogic Commerce Server 3.1.1 or 3.2 database schemas, contact BEA Customer Support.

Upgrading WebLogic Personalization Server Database Schemas from 2.0.1 to 3.1.1

In WebLogic Personalization Server Release 3.1, a new column called `PROFILE_TYPE` was added to the WLCS_USER table. This column contains the name of the Unified Profile Type of which the User is an instance. (For more information about Unified Profile Types, see the chapter "Creating and Managing Users"in the *Guide to Building Personalized Applications* (in the release 3.5 documentation set.))

The `PROFILE_TYPE` column can be added to existing WLCS_USER tables with the following statement:

ALTER TABLE WLCS_USER ADD PROFILE_TYPE VARCHAR2(100);

For User objects that are of the standard type com.beasys.commerce.axiom.contact.User, this should be left as null. If the User is an extended User type, such as the 'Unified Profile Example', the column should be set to that type name. The example user for the Unified Profile Example should be updated with the following statement:

UPDATE WLCS_USER SET PROFILE_TYPE = 'Unified Profile Example' WHERE IDENTIFIER = 'unifieduser_bob';

Note: In this document, `$WL_COMMERCE_HOME` refers to the directory into which you installed WebLogic Commerce Server and/or WebLogic Personalization Server and *`database-type`* refers to the type and version of RDBMS that you installed.

To upgrade a WebLogic Personalization Server database from release 2.0.1 to release 3.1, follow these steps:

1. Make a backup copy of the following file:
   `$WL_COMMERCE_HOME/db/`*`database-type`*`/staging /upgrade-to-310.sql`
2. Open `upgrade-to-310.sql` in a text editor.
3. Move the cursor immediately below the following statement:

```
ALTER TABLE WLCS_USER ADD (
     PROFILE_TYPE      VARCHAR2(100)
);
```

4. For each user that is of an extended User type, add the following statement on a single line:

   ```
   UPDATE WLCS_USER SET PROFILE_TYPE = '<profile-type>' WHERE
   IDENTIFIER = '<user-name>';
   ```

   For example:

   ```
   UPDATE WLCS_USER SET PROFILE_TYPE = 'Unified Profile Example'
   WHERE IDENTIFIER = 'unifieduser_bob';
   ```

5. Save your modifications and close `update-to-310.sql`.

6. Run the following SQL command:
   `@ $WL_COMMERCE_HOME/db/`*`database-type`*`/update-to-310.sql`

   For example, if you installed WebLogic Commerce Server in `~/WebLogicCommerceServer3.2`, enter the following from SQL*Plus:
   `@`
   `~/WebLogicCommerceServer3.2/db/`*`database-type`*`/update-to-310.sql`

When the command successfully completes upgrading the database, it prints the following message:

```
************  SCRIPT HAS FINISHED EXECUTING   *************

******  PLEASE REVIEW UPDATE-TO-310.LOG  FILE  *****
```

Upgrading WebLogic Personalization Server Database Schemas from 2.0.1 to 3.2

This section describes scripts that do the following:

- Update tables that were created for the WebLogic Personalization Server 2.0.1 schema to the WebLogic Personalization Server 3.2 schema.
- Copy existing WebLogic Personalization Server data to the new 3.2 tables.
- Create empty WebLogic Personalization Server tables that are new for 3.1.1 and 3.2.
- Create empty WebLogic Commerce Server tables that the release 3.2 schema defines. The scripts do not copy existing WebLogic Commerce Server data into the tables.

The following instructions also assume that you are working with an Oracle database and will be using the scripts in either of the following:

- `WL_COMMERCE_HOME/db/oracle` (for Oracle versions 8.1.5 and below)
- `WL_COMMERCE_HOME/db/oracle816` (for Oracle version 8.1.6)

To upgrade your Personalization Server database from 2.0.1, do the following:

1. In `WL_COMMERCE_HOME/db/<oracle directory>/migration/v201`, log into SQL*Plus and execute the following:

   ```
   SQL> @upgrade-to-310.sql
   ```

2. In `WL_COMMERCE_HOME/db/<oracle directory>`, log into SQL*Plus and execute the following:

   ```
   SQL> @create-wlcs-oracle.sql
   ```

   At this point the database format should be WLCS 3.1.1 compliant.

3. Follow the steps in "Upgrading Database Schemas from 3.1.1 to 3.2" on page 4-5. Then return to these instructions and continue with step 4.

4. Start WebLogic Personalization Server and WebLogic Commerce Server if it is not already running.

5. In `WL_COMMERCE_HOME\bin\win32\` (Windows) or in `WL_COMMERCE_HOME/bin/unix/` (UNIX), run the following on a command line:

   ```
   loadrules
   ```

6. Finally, in `WL_COMMERCE_HOME\bin\win32\` (Windows) or in `WL_COMMERCE_HOME/bin/unix/` (UNIX), run the following on a command line:

   ```
   loaddocs -delete -cleanup
   ```

Upgrading Database Schemas from 3.1.1 to 3.2

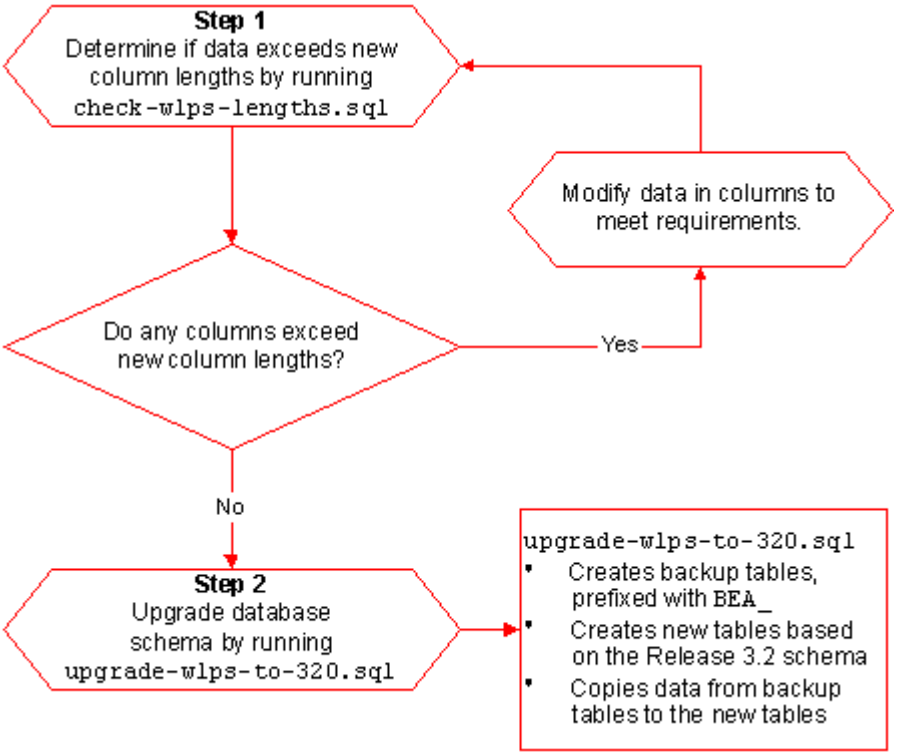
Release 3.2 of WebLogic Commerce Server and WebLogic Personalization Server introduces schema changes and restrictions for the length of data allowed in various columns. To upgrade databases from Release 3.1.1 to Release 3.2, complete the following tasks:

- Upgrade the WebLogic Personalization Server Schema
- Upgrade the WebLogic Commerce Server Schema (only if you use WebLogic Commerce Server)
- Verify the Upgrade
- Remove Temporary Tables

Upgrade the WebLogic Personalization Server Schema

If you are upgrading WebLogic Personalization Server from Release 3.1.1 to Release 3.2, complete the tasks described in this section. The following diagram illustrates the process for upgrading the WebLogic Personalization Server schema, and subsequent topics provide more information about the process.

Figure 4-1 Upgrading the Personalization Server Schema from 3.1.1 to 3.2



Step 1: Determine if Data Exceeds New Column Lengths and Modify When Necessary

Start the migration process by finding and correcting any columns in your existing databases that contain data exceeding the new column length in Release 3.2.

To start the migration, do the following:

1. Make a backup copy of your database.
2. Run the following SQL command:
   `@ $WL_COMMERCE_HOME/db/`*`database-type`*`/check-wlps-lengths.sql`

   For example, if you installed WebLogic Commerce Server in `~/WebLogicCommerceServer3.2`, enter the following command in SQL*Plus:
   `@ ~/WebLogicCommerceServer3.2/db/`*`database-type`*`/check-wlps-lengths.sql`
3. To see the results of the script, open the following log file in a text editor:
   `$WL_COMMERCE_HOME/db/`*`database-type`*`/check-wlps-lengths.log`

   The log file lists each table for which the maximum number of characters has changed. As Listing 4-1 illustrates, the log file states `no rows selected` for tables that meet the new maximum-length requirements. For tables that exceed requirements, the log file lists each row and describes the error condition.

Listing 4-1 Output of check-wlps-lengths.sql

```
*****  WLCS_DOCUMENT.ID  *****
no rows selected

*****  WLCS_DOCUMENT_METADATA.ID  *****
no rows selected
```

4. For any table containing data that exceeds a row's maximum length requirement:

   a. For information on length requirements for the table, see the schema chapters in the WebLogic Commerce Server and WebLogic Personalization Server documentation.

   b. Modify the data in the row to meet the new requirements.

5. Repeat steps 2-4 until the log file reports "no rows selected" for all tables.

Step 2: Upgrade the Database Schema

After correcting any rows that do not conform to new column length requirements, you must upgrade the Release 3.1.1 schema to the Release 3.2 schema by doing the following:

1. Make backup copies of your database and the following file:
   `$WL_COMMERCE_HOME/db/`*`database-type`*`/upgrade-wlps-to-320.sql`

2. Open `upgrade-wlps-to-320.sql` in a text editor.

3. Make sure that the following lines assign values that match your tablespace:

   ```
   define DATA_TABLESPACE=WLCS_DATA
   define INDEX_TABLESPACE=WLCS_INDEX
   ```

   By default, WebLogic Commerce Server and WebLogic Personalization Server place data in `WLCS_DATA` and indexes in `WLCS_INDEX`. If you are using other tablespaces, you must modify `upgrade-wlps-to-320.sql` to specify your tablespaces instead.

4. Save your modifications to `upgrade-wlps-to-320.sql`.

5. Run the following SQL command:
   `@ $WL_COMMERCE_HOME/db/`*`database-type`*`/upgrade-wlps-to-320.sql`

Note: Enter this command only once and only after you have modified all rows that contain data exceeding new length requirements.

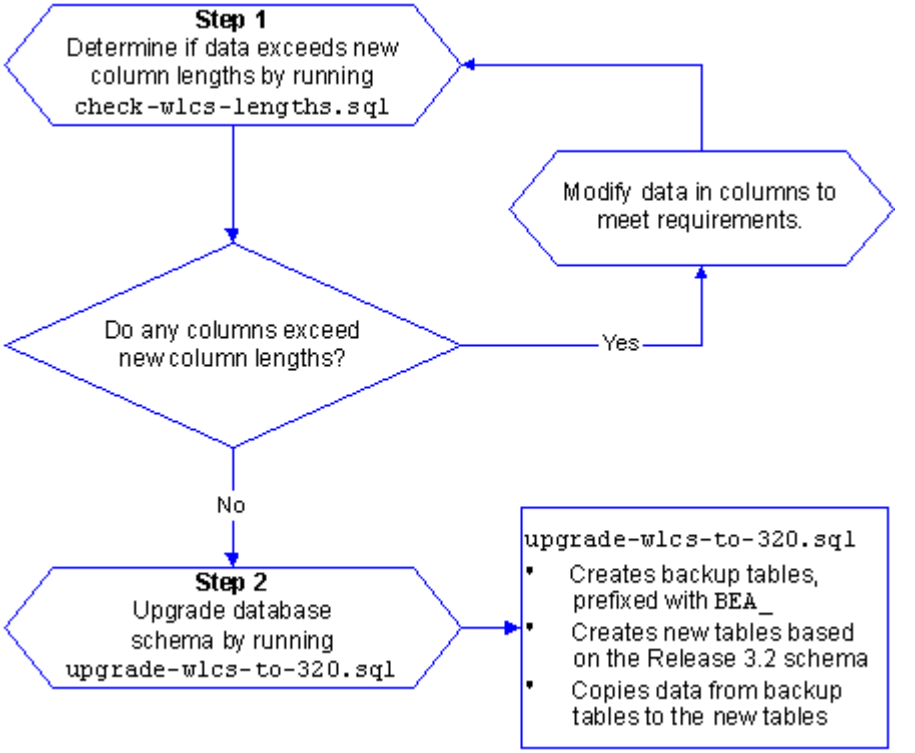
When the command successfully completes updating tables, it prints the following message:

```
************  SCRIPT HAS FINISHED EXECUTING   *************
******  PLEASE REVIEW UPDATE-TO-320.LOG  FILE  *****
```

Upgrade the WebLogic Commerce Server Schema

To upgrade WebLogic Commerce Server from Release 3.1.1 to Release 3.2, complete the tasks described in this section. The following diagram illustrates the process for upgrading the WebLogic Commerce Server schema, and subsequent topics provide more information about the process.

Figure 4-2 Upgrading the Commerce Server Schema from 3.1.1 to 3.2



Step 1: Determine if Data Exceeds New Column Lengths and Modify When Necessary

Start the migration process by finding and correcting any columns in your existing databases that contain data exceeding the new column length in Release 3.2.

To start the migration, do the following:

1. Make a backup copy of your database.
2. Run the following SQL command:
   `@ $WL_COMMERCE_HOME/db/`*`database-type`*`/check-wlcs-lengths.sql`
3. To see the results of the script, open the following log file in a text editor:
   `$WL_COMMERCE_HOME/db/`*`database-type`*`/check-wlcs-lengths.log`

   The log file lists each table for which the maximum number of characters has changed. As Listing 4-2 illustrates, the log file states `no rows selected` for tables that meet the new maximum-length requirements. For tables that exceed requirements, the log file lists each row and describes the error condition.

Listing 4-2 Output of check-wlcs-lengths.sql

```
*****  WLCS_CATEGORY  *****
no rows selected

*****  WLCS_PRODUCT  *****
no rows selected
```

4. For any table containing data that exceeds a row's maximum length requirement:
   a. For information on length requirements for the table, see the schema chapters in the WebLogic Commerce Server and WebLogic Personalization Server documentation.
   b. Modify the data in the row to meet the new requirements.
5. Repeat steps 2-4 until the log file reports `no rows selected` for all tables.

Step 2: Upgrade the Database Schema

After correcting any rows that do not conform to new column length requirements, you must upgrade the Release 3.1.1 schema to the Release 3.2 schema by doing the following:

1. Make backup copies of your database and the following file:
   `$WL_COMMERCE_HOME/db/`*`database-type`*`/upgrade-wlcs-to-320.sql`
2. Open `upgrade-wlcs-to-320.sql` in a text editor.
3. Make sure that the following lines assign values that match your tablespace:

   ```
   define DATA_TABLESPACE=WLCS_DATA
   define INDEX_TABLESPACE=WLCS_INDEX
   ```

   By default, WebLogic Commerce Server and WebLogic Personalization Server place data in `WLCS_DATA` and indexes in `WLCS_INDEX`. If you are using other tablespaces, you must modify `upgrade-wlps-to-320.sql` to specify your tablespaces instead.
4. Save your modifications to `upgrade-wlcs-to-320.sql`.
5. Run the following SQL command:
   `@ $WL_COMMERCE_HOME/db/`*`database-type`*`/upgrade-wlcs-to-320.sql.`

Note: Enter this command only once and only after you have modified all rows that contain data exceeding new length requirements.

When the command successfully completes updating tables, it prints the following message:

```
************  SCRIPT HAS FINISHED EXECUTING   *************
******  PLEASE REVIEW UPDATE-TO-320.LOG  FILE  *****
```

Verify the Upgrade

After you upgrade the schema for each server that you are using, verify the upgrade by starting the server and Administration Tool and testing the application. For example, if you use both WebLogic Commerce Server and WebLogic Personalization Server, open the Administration Tool to verify that the users and groups you upgraded are available under User Administration, and all items and categories that you upgraded are available under Catalog Administration. Then access the server through a Web browser to verify that data transferred successfully.

To Start the Server

To start WebLogic Commerce Server and/or WebLogic Personalization Server on UNIX, enter the following command from a WebLogic Commerce Server and WebLogic Personalization Server host:

```
$WL_COMMERCE_HOME/StartCommerce.sh
```

To start WebLogic Commerce Server and/or WebLogic Personalization Server on Windows, on a WebLogic Commerce Server and WebLogic Personalization Server host, do one of the following:

- Click Start → Programs → WebLogic Commerce Server 3.2 → Start Commerce Server.
- From a command prompt, enter the following command:
  `%WL_COMMERCE_HOME%\StartCommerce.bat`

Remove Temporary Tables

Note: Do not complete this step until you have successfully upgraded the schema for **both** servers (if you use both servers) to Release 3.2 **and** started the application and verified the data migration.

After you have verified that WebLogic Commerce Server and WebLogic Personalization Server function properly with the imported data, remove the temporary `BEA_`*`table-name`* tables by running the following SQL command:
`@ $WL_COMMERCE_HOME/db/`*`database-type`*`/drop_bea_tables.sql`

When the command successfully completes removing `BEA_` tables, it prints the following message:

```
************  SCRIPT HAS FINISHED EXECUTING   *************

******  PLEASE REVIEW  DROP_BEA_TABLES.LOG  FILE  *****
```

Upgrading Database Schemas from 3.2 to 3.5

Release 3.5 of WebLogic Commerce Server and WebLogic Personalization Server provides enhancements and changes that require you to update the schemas and migrate the data.

Note: If you are using the Oracle 8.0.5 or 8.1.5 database, you must upgrade to Oracle 8.1.6 or greater before migrating to Release 3.5 of WebLogic Commerce Server and WebLogic Personalization Server. Release 3.5 uses CLOBs and BLOBs instead of LONG RAW characters. Oracle 8.0.5 and 8.1.5 do not support CLOBs and BLOBs.

To upgrade databases from Release 3.2 to Release 3.5, complete the following tasks:

- Make a Backup
- Validate Data
- Upgrade Current Tables to New Schema
- Drop Any Backup Tables (Optional)
- Add New Tables to Bring the Schema Current
- Verify the Upgrade

Make a Backup

We strongly recommend that you make a complete backup of the WebLogic Personalization Server/WebLogic Commerce Server database before beginning the process to migrate the database to a more current schema.

Validate Data

To validate the data before beginning the upgrade, run the following scripts. These scripts are used to check which WebLogic Commerce Server tables have column values exceeding 254 characters, or whatever the new length for the specific column in question is.

1. From a command prompt, type the following:

```
@ $WL_COMMERCE_HOME/db/db-vendor/db-version/migration/v320/check_common_lengths.sql
@ $WL_COMMERCE_HOME/db/db-vendor/db-version/migration/v320/check_wlps_lengths.sql
@ $WL_COMMERCE_HOME/db/db-vendor/db-version/migration/v320/check_wlcs_lengths.sql
```

For example, if you are using Oracle 8.1.6 and installed WebLogic Commerce Server in `~/WebLogicCommerceServer3.5`, enter the following command in SQL*Plus:

```
@ $WL_COMMERCE_HOME/db/oracle/8.1.6/migration/v320/check_common_lengths.sql
```

2. To see the results of the script, open the following log file in a text editor:

```
@ $WL_COMMERCE_HOME/db/db-vendor/db-version/migration/v320/check_common_lengths.log
```

There will also be `check_wlps_lengths.log` and `check_wlcs_lengths.log`.

The log file lists each table for which the maximum number of characters has changed. As Listing 4-2 illustrates, the log file states `no rows selected` for tables that meet the new maximum-length requirements. For tables that exceed requirements, the log file lists each row and describes the error condition.

Listing 4-3 Output of check-wlcs-lengths.sql

```
*****  WLCS_CATEGORY  *****
no rows selected

*****  WLCS_PRODUCT  *****
no rows selected
```

3. For any table containing data that exceeds a row's maximum length requirement:

a. For information on length requirements for the table, see the schema chapters in the WebLogic Commerce Server and WebLogic Personalization Server documentation.

b. Modify the data in the row to meet the new requirements.

4. Repeat the previous steps until the log file for each script reports `no rows selected` for all tables.

Upgrade Current Tables to New Schema

Once you have successfully verified the current tables, you can run the upgrade scripts. These scripts make backup copies of current tables, drop the existing table, recreate each table with current column definitions, and populate the new table with data from the backup table.

From a command prompt, type the following:

```
@ $WL_COMMERCE_HOME/db/db-vendor/db-version/migration/v320/upgrade_common_to_350.sql
@ $WL_COMMERCE_HOME/db/db-vendor/db-version/migration/v320/upgrade_wlps_to_350.sql
@ $WL_COMMERCE_HOME/db/db-vendor/db-version/migration/v320/upgrade_wlcs_to_350.sql
```

Drop Any Backup Tables

Note: This step is optional and should not be executed until you are certain that all data has been retained. Only when you are comfortable with the state of your data should you consider running this script.

The script in this step is used to drop the temporary tables that held your original data. During the upgrade process, the data contained in the original table (named with the WLCS_ prefix) was copied to a backup table using the BEA_ prefix and renamed with BEA_ prefix.

1. Restart the WebLogic Commerce Server before executing this script, to test data accuracy and to ensure the application(s) are working as intended.

2. From a command prompt, type the following:

```
@ $WL_COMMERCE_HOME/db/db-vendor/db-version/migration/v320/drop_bea_tables.sql
```

Add New Tables to Bring the Schema Current

Run the following scripts to create the necessary new tables.

- `insert_event_properties.sql` is used to populate the property management tables with records of various Behavior Tracking events.
- `upgrade_wlcs_add_tables_350.sql` is used to create new tables used by the WebLogic Commerce Server. It is involved in making changes to the existing database objects: adding columns, dropping columns, resizing columns, and so on.
- `create_campaign.sql` and `create_mail_ad.sql` are used in the new WebLogic Campaign Manager component. Each script is involved in creating new database objects such as tables and indexes.

From a command prompt, type the following:

```
@ $WL_COMMERCE_HOME/db/db-vendor/db-version/migration/v320/
upgrade_wlcs_add_tables_350.sql

@ $WL_COMMERCE_HOME/db/db-vendor/db-version/migration/v320/
insert_event_properties.sql
```

Note: The path from which to run the next two scripts is different from where you ran previous scripts.

```
@ $WL_COMMERCE_HOME/db/db-vendor/db-version/create_campaign.sql

@ $WL_COMMERCE_HOME/db/db-vendor/db-version/create_mail_ad.sql
```

Verify the Upgrade

After you upgrade the schema for each server that you are using, verify the upgrade by starting the server and Administration Tool and testing the application. For example, if you use both WebLogic Commerce Server and WebLogic Personalization Server, open the Administration Tool to verify that the users and groups you upgraded are

available under User Administration, and all items and categories that you upgraded are available under Catalog Administration. Then access the server through a Web browser to verify that data transferred successfully.

To Start the Server

To start WebLogic Commerce Server and/or WebLogic Personalization Server on UNIX, enter the following command from a WebLogic Commerce Server and WebLogic Personalization Server host:

```
$WL_COMMERCE_HOME/StartCommerce.sh
```

To start WebLogic Commerce Server and/or WebLogic Personalization Server on Windows, on a WebLogic Commerce Server and WebLogic Personalization Server host, do one of the following:

- Click Start → Programs → WebLogic Commerce Server 3.5 → Start Commerce Server.
- From a command prompt, enter the following command:
  `%WL_COMMERCE_HOME%\StartCommerce.bat`

5 Changes to WebLogic Personalization Server JSP Tag Library

This topic includes the following sections:

- JSP Tag Changes in Version 3.5
  - New Ads and Placeholder Tag
  - New Event Tracking Tags
  - New Webflow Tag
  - New E-Business Tag
  - Changes to Personalization Tags
- JSP Tag Changes in Version 3.2
  - Changes to Content Management Tags in Release 3.2
  - Changes to Utility Tags in Release 3.2
  - New Flow Manager Tags in Release 3.2
- New JSP Tags Introduced in Release 3.1
  - New Property Set Management Tags in Release 3.1
  - New Internationalization Tags in Release 3.1
  - New WebLogic Utility Tag in Release 3.1
- Changes to the JSP Tag Library in Release 3.1

- New JSP 1.1 Naming Conventions
- Changes to Tag Attributes
- Global Changes
- Tag Migration Roadmap
- Additional Notes About JSP Tags

JSP Tag Changes in Version 3.5

This section covers the tags that have been removed from and added to the WebLogic Commerce Server and WebLogic Personalization Server tag libraries.

- Removed Tags
  - <es:preparedStatement>
- New Ads and Placeholder Tag
  - <ad:adTarget>
  - <ph:placeholder>
- New Event Tracking Tags
  - <tr:clickContentEvent> Content Tag
  - <tr:displayContentEvent> Content Tag
  - <trp:clickProductEvent> Product Tag
  - <trp:displayProductEvent> Product Tag
  - <trc:clickCampaignEvent> Campaign Tag
- New Webflow Tag
  - <webflow:setValidated Value>
- New E-Business Tag
  - <eb:smnav>
- Changes to Personalization Tags

Removed Tags

One tag has been removed in this release.

All tags that were deprecated in previous releases have now been removed. For more information, see the Note at the top of the section "JSP Tag Changes in Version 3.2" on page 5-6.

<es:preparedStatement>

The `<es:preparedStatement>` tag has been removed in this release.

New Ads and Placeholder Tag

Two new tags have been added to create placeholders and query for ad content.

These tags are documented in the "Personalization Server JSP Tag Library Reference" chapter in the *Guide to Building Personalized Applications*.

<ad:adTarget>

The `<ad:adTarget>` tag uses the Ad Service to send an ad query to the content management system. Unlike the tag, the query in the `<ad:adTarget>` tag does not compete with other queries in an ad placeholder.

<ph:placeholder>

The `<ph:placeholder>` tag implements a placeholder, which describes the behavior for a location on a JSP page.

New Event Tracking Tags

Five new JSP tags have been added to track events.

These `<tr*:>` tags are documented in the chapter "Events and Behavior Tracking JSP Tag Library Reference" in the *Guide to Events and Behavior Tracking*.

<tr:clickContentEvent> Content Tag

The `<tr:clickContentEvent>` tag is used to generate a behavior event when a user has clicked (through) on an ad impression.

<tr:displayContentEvent> Content Tag

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

<trp:clickProductEvent> Product Tag

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

<trp:displayProductEvent> Product Tag

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

<trc:clickCampaignEvent> Campaign Tag

The `<trc:clickCampaignEvent>` tag is used to explicitly generate a clickthrough event relevant to a campaign.

New Webflow Tag

One new tag has been added to the Webflow feature set.

The Pipeline and Webflow tags are documented in the chapter "Webflow and Pipeline JSP Tag Library Reference" in the *Guide to Managing Presentation and Business Logic: Using Webflow and Pipeline.*

<webflow:setValidated Value>

The tag `<webflow:setValidatedValue>` is used in a JSP to configure the display of fields in a form that a customer must correct.

Several previously undocumented attributes have been added to the documentation for the `<webflow:getValidatedValue>` tag.

New E-Business Tag

One tag has been created for the new E-Business tag library.

See the chapter "Product Catalog JSP Tag Library Reference" in the *Guide to Building a Product Catalog*.

<eb:smnav>

The `<eb:>` preface stands for E-Business. The Scrollable Model can be use throughout the E-Business package to iterate through a list of objects. It can be used in conjunction with transaction, shopping cart, order history, or shipping services.

Changes to Personalization Tags

<pz:div> and <pz:contentSelector>

The `ruleSet` attribute has been removed from the `<pz:div>` tag and the `<pz:contentSelector>` tag. This attribute was used to define the URI for the rulesets. In code that already used the `ruleSet` attribute, the attribute is no longer required and will be ignored.

It is no longer necessary for programmers to define rule sets (or rulesheets) because rule set names are no longer controlled through the tags. Rules are created using the new GUI tool, E-Business Control Center. The tool saves rules into predefined rule sets in the advislet registry. User classifier rules are saved into the `GlobalClassification.xml` file. Content selectors are saved into the `GlobalContentSelectors.xml` file.

JSP Tag Changes in Version 3.2

Note: **Backward Compatibility Will Stop After Version 3.2.** The tag libraries were updated in WebLogic Personalization Server (WLPS) version 3.1 to comply with the JSP 1.1 Specification. If you are upgrading from WebLogic Personalization Server 2.0.1, you can continue to use your existing code with WebLogic Personalization Server 3.2. However, *future releases will no longer be backward compatible*, so you will need to migrate to the new tags if you intend to continue to use your legacy code with the latest WebLogic Personalization Server releases.

The WebLogic Personalization Server documentation has been revised to reflect the changes to the tag libraries. Until you migrate to the new tags, you can continue to use the WebLogic Personalization Server 2.0 *JSP Tag Reference* found at http://e-docs.bea.com/wlcs/docs20/p13ndev/jsptags.htm.

WebLogic Personalization Server 3.2 introduces eight new tags:

```
<cm:getProperty>
<fm:getApplicationURI>
<fm:getCachedAttribute>
<fm:setCachedAttribute>
<fm:removeCachedAttribute>
<fm:getSessionAttribute>
<fm:setSessionAttribute>
<fm:removeSessionAttribute>
```

Changes to Content Management Tags in Release 3.2

A new Content Management tag has been added in WebLogic Personalization Server 3.2. In addition, a new attribute has been added to the `<cm:printDoc>` tag.

<cm:getProperty>

Retrieves the value of the specified content metadata property into a variable specified by `resultId`. This tag is similar to the `<cm:printProperty>` tag, with the addition of two new parameters, `resultId` and `resultType`.

<cm:printDoc>

A new attribute, `baseHref`, has been added to the `<cm:printDoc>` tag. This attribute provides the URL of the document's BASE HREF.

Changes to Utility Tags in Release 3.2

<es:preparedStatement>

The Personalization Utility tag `<es:preparedStatement>` has a new attribute, `transactionIsolationLevel`.

Note: This tag is removed from the tag library in WLPS Release 3.5.

New Flow Manager Tags in Release 3.2

Seven new tags have been added to support the Flow Manager:

<fm:getApplicationURI>

Gets the Flow Manager.

<fm:getCachedAttribute>

Gets an attribute out of the session/global cache.

<fm:setCachedAttribute>

Sets an attribute in the session/global cache.

<fm:removeCachedAttribute>

Removes an attribute from the session/global cache.

<fm:getSessionAttribute>

Gets an attribute out of the HttpSession.

<fm:setSessionAttribute>

Sets an attribute in the HttpSession.

<fm:removeSessionAttribute>

Removes an attribute from the HttpSession.

New JSP Tags Introduced in Release 3.1

Five new tags were introduced in WebLogic Personalization Server Release 3.1:

```
<ps:getPropertyNames>
<ps:getPropertySetNames>
<i18n:localize>
<i18n:getMessage>
<wl:repeat>
```

New Property Set Management Tags in Release 3.1

Two new Property Set Management JSP extension tags provide the following services:

- Lists all properties associated with a property set.
- Lists all property set names for a property set group name (for example, `USER` or `CONTENT`).

The two new Property Set tags are:

<ps:getPropertyNames>

Returns a list of property names for a given property set in a String array.

<ps:getPropertySetNames>

Returns a list of property set names for a given schema group name in a String array.

New Internationalization Tags in Release 3.1

In earlier releases of WebLogic Personalization Server, Internationalization (I18N) was applied from JSP beans that supported sample portal pages, and administration tools pages. The JSP beans employed a simple MessageBundle Java class that allowed access to localized text labels and messages.

For this release, this basic MessageBundle has been extended using a simple framework that is accessible from JSPs via a small I18N extension tag library. The JSP extension tag library provides the following services:

- Retrieves a static text label or a message from a resource bundle (implemented as a property file).
- Initializes a page context with a particular language, country, and variant for label and message retrieval throughout a page.
- Properly sets the content type (text/html) and character encoding for a page.

The following new tags are included in the I18N framework:

<i18n:localize>

Allows you to define the language, content type, and character encoding to be used in a page. It also allows you to specify a country, variant, and resource bundle name to use throughout a page when accessing resource bundles via the `<i18n:getMessage>` tag described below.

<i18n:getMessage>

Retrieves a localized label, or message (based on the absence/presence of an "args" attribute). This tag optionally takes a bundle name, language, country, and variant to aid in locating the appropriate properties file for resource bundle loading.

New WebLogic Utility Tag in Release 3.1

<wl:repeat>

This WebLogic Server tag is used to iterate over a variety of Java objects that includes:

- Enumerations
- Iterators
- Collections
- Arrays
- Vectors
- Result Sets
- Result Set Metadata
- Hashtable keys

Changes to the JSP Tag Library in Release 3.1

The tag libraries have been updated in WebLogic Personalization Server version 3.1 to comply with the JSP 1.1 Specification. If you are upgrading from WebLogic Personalization Server 2.0.1, you can continue to use your existing code with WebLogic Personalization Server 3.1. However, *future releases will no longer be backward compatible*, so you will need to migrate to the new tags if you intend to continue to use your legacy code with the latest WebLogic Personalization Server releases.

The WebLogic Personalization Server 3.1 documentation has been revised to reflect the changes to the tag libraries. Until you migrate to the new tags, you can continue to use the WebLogic Personalization Server 2.0 *JSP Tag Reference* located at http://e-docs.bea.com/wlcs/docs20/p13ndev/jsptags.htm.

New JSP 1.1 Naming Conventions

Beginning with WebLogic Personalization Server version 3.1, all tags use the JSP 1.1 naming conventions. Old style tags that were used in previous WebLogic Personalization Server releases have been changed to reflect the new camel case naming conventions.

For example, the old-style tag `<um:getgroupnamesforusers>` is now `<um:getGroupNamesForUsers>`.

Old tag names can still be used in the WebLogic Personalization Server 3.1 release. However, *old style tag names will not be supported in future releases of* WebLogic Personalization Server.

Note: Each time you use a deprecated tag, a message is logged to WebLogic Server. To turn off the deprecation messages, add the following property to weblogiccommerce.properties:
`commerce.log.display.deprecated=false`

For consistency, the Portal Management tags <pt:*> have a new esp: prefix. For example, the old-style tag <pt:eval> is now called <esp:eval>, and the old <pt:portalmanager> is now <esp:portalManager>. When you change to the new prefix, you will need to update each Portal Management tag invocation in the page to use the new prefix.

Note: The es: prefix stands for e-commerce services.
The esp: prefix stands for e-commerce services portal.
The pz: prefix stands for personalization.

Changes to Tag Attributes

The Content Management Tags Have Been Changed as Follows:

- For the Content Management <cm:printDoc> tag, a new attribute, baseHref, has been added. This attribute provides the URL of the document's BASE HREF.

The User Management Tags Have Been Changed as Follows:

- For the User Management <um:*> tags, the resultId attribute has been changed to result, and is now an Integer instead of an int. Usage and functionality remain the same.
- For the User Management tags <um:getProperty> and <um:setProperty>, the usecache attribute has been dropped.

The WebLogic Personalization Server Utility tags Have Been Changed as Follows:

- For the WebLogic Personalization Server Utility tags <es:isNull> and <es:notNull>, the id attribute has been changed to item.
- For the WebLogic Personalization Server Utility tag <es:preparedStatement>, the pool attribute has been dropped (see "Note 4: <es:preparedStatement>" on page 5-23) and a new attribute, transactionIsolationLevel, has been added.

Tag Attributes Require Camel Casing

All of the tag attributes used in previous WebLogic Personalization Server releases already use the camel-case convention, with a few exceptions. The tags that do not already use camel-cased attributes are the three Advisor tags (formerly called Personalization Advisor) <pz:*>, and the single WebLogic utility <wl:process>.

Table 5-1 lists the attributes that you will need to camel case. Note that all of these attributes are optional, so it is possible that you did not use them in your existing code.

Table 5-1 Camel-Cased Attributes

Tag	Attribute
<pz:div>	ruleSet
<pz:contentQuery>	sortBy contentHome
<pz:contentSelector>	ruleSet sortBy contentHome
<wl:process>	notName notValue

New Library Descriptors

Any JSP migrating from old-style tags to new-style tags will need to point to new library descriptors.

- For Portal Management <pt:*> tags, change "lib/esportal.jar" to "esp.tld". (Also, change prefix="pt" to prefix= "esp". Update each invocation of a Portal Management tag on the page to use the "esp" prefix.)
- For User Management <um:*> tags, change "lib/um_tags.jar" to "um.tld".
- For Personalization Utilities <es:*> tags, change "lib/esjsp.jar" to "es.tld".
- For the WebLogic Utility <wl:process> tag, change "lib/wljsp.jar" to "weblogic.tld".

For example:

In the JSP page, `<%@ taglib uri="lib/um_tags.jar" prefix="um" %>` would change to `<%@ taglib uri="um.tld" prefix="um" %>`.

Note: The Personalization Advisor is now simply called the Advisor. The Advisor `<pz:*>` tags already use `taglib uri="pz.tld"`, so these do not need to be changed.

The Content Management `<cm:*>` tags already use `taglib uri="cm.tld"`, so these do not need to be changed.

Global Changes

Tags no longer return primitive types, they only return objects. For example, `<es:counter>` used to return an int, and now it returns an Integer object.

Any tags (`es`, `um`, `wl`, etc.) with a `<jsp:include page=.../>` in their body must be replaced with their scriptlet equivalent. (See Section 5.4.5 of the JSP 1.1 Specification.)

Old Usage:

```
<es:notNull item="renderer">
  <jsp:include page="<%=reconcileFile(request, renderer)%>"/>
</es:notNull>
```

New Usage:

```
<% if (renderer != null) { %>
  <jsp:include page="<%=reconcileFile(request, renderer)%>"/>
<% } %>
```

Tag Migration Roadmap

Table 5-2 maps the old tag names to the new JSP 1.1 camel-cased tag names. In addition, changes made to the tags in the WebLogic Personalization Server 3.1 release are noted in the Change column.

Table 5-2 Tag Changes for WebLogic Personalization Server 3.1

Library	Old Style Tag Name	Change	New JSP 1.1 Tag
Advisor	`<pz:contentquery>`	Camel case Attribute `sortby = sortBy` Attribute `contenthome = contentHome` It is no longer necessary to extend the JSP. See below - **Note 1: `<pz:>` tags**.	`<pz:contentQuery>`
	`<pz:contentselector>`	Camel case Attribute `ruleset = ruleSet` Attribute `sortby = sortBy` Attribute `contenthome = contentHome`	`<pz:contentSelector>`
	`<pz:div>`	`ruleset = ruleSet` It is no longer necessary to extend the JSP. See below - **Note 1: `<pz:>` tags**.	`<pz:div>`
Content Mngmt	---	**New**	`<cm:getProperty>`
	`<cm:printproperty>`	Camel case	`<cm:printProperty>`
	`<cm:printdoc>`	Camel case New attribute: `baseHref`	`<cm:printDoc>`
	`<cm:select>`	No change	`<cm:select>`

Table 5-2 Tag Changes for WebLogic Personalization Server 3.1 (Continued)

Library	Old Style Tag Name	Change	New JSP 1.1 Tag
	`<cm:selectbyid>`	Camel case	`<cm:selectById>`
Flow Manager	---	**New**	`<fm:getApplicationURI>`
	---	**New**	`<fm:getCachedAttribute>`
	---	**New**	`<fm:setCachedAttribute>`
	---	**New**	`<fm:removeCachedAttribute>`
	---	**New**	`<fm:getSessionAttribute>`
	---	**New**	`<fm:setSessionAttribute>`
	---	**New**	`<fm:removeSessionAttribute>`
I18N	---	**New**	`<i18n:initialize>`
	---	**New**	`<i18n:getMessage>`
Property Set	---	**New**	`<ps:getPropertyName>`
	---	**New**	`<ps:setPropertyName>`
Portal	`<pt:eval>`	`taglib uri="esp.tld"` Change preface `pt:` to `esp:`	`<esp:eval>`
	`<pt:get>`	`taglib uri="esp.tld"` Change preface `pt:` to `esp:`	`<esp:get>`
	`<pt:getgroupsforportal>`	Camel case Change preface `pt:` to `esp:` `taglib uri="esp.tld"`	`<esp:getGroupsForPortal>`

Table 5-2 Tag Changes for WebLogic Personalization Server 3.1 (Continued)

Library	Old Style Tag Name	Change	New JSP 1.1 Tag
	`<pt:monitorsession>`	Camel case `taglib uri="esp.tld"` Change preface `pt:` to `esp:`	`<esp:monitorSession>`
	`<pt:portalmanager>`	Camel case `taglib uri="esp.tld"` Change preface `pt:` to `esp:`	`<esp:portalManager>`
	`<pt:portletmanager>`	Camel case `taglib uri="esp.tld"` Change preface `pt:` to `esp:`	`<esp:portletManager>`
	`<pt:props>`	`taglib uri="esp.tld"` Change preface `pt:` to `esp:`	`<esp:props>`
User/ Profile	`<um:getprofile>`	Camel case `taglib uri="um.tld"`	`<um:getProfile>`
	`<um:getproperty>`	Camel case `taglib uri="um.tldv"`	`<um:getProperty>`
	`<um:getpropertyasstring>`	Camel case `taglib uri="um.tld"`	`<um:getPropertyAsString>`
	`<um:removeproperty>`	Camel case `taglib uri="um.tld"`	`<um:removeProperty>`

Table 5-2 Tag Changes for WebLogic Personalization Server 3.1 (Continued)

Library	Old Style Tag Name	Change	New JSP 1.1 Tag
	`<um:setproperty>`	Camel case `taglib uri="um.tld"`	`<um:setProperty>`
User/ Group	`<um:addgrouptogroup>`	Camel case `taglib uri="um.tld"` Attribute `resultId = result`	`<um:addGroupToGroup>`
	`<um:addusertogroup>`	Camel case `taglib uri="um.tld"` Attribute `resultId = result`	`<um:addUserToGroup>`
	`<um:changegroupname>`	Camel case `taglib uri="um.tld"` Attribute `resultId = result`	`<um:changeGroupName>`
	`<um:creategroup>`	Camel case `taglib uri="um.tld"` Attribute `resultId = result`	`<um:createGroup>`
	`<um:createuser>`	Camel case `taglib uri="um.tld"` Attribute `resultId = result`	`<um:createUser>`
	`<um:getchildgroupnames>` *(previously undocumented)*	Camel case `taglib uri="um.tld"`	`<um:getChildGroupNames>`
	`<um:getchildgroups>`	Camel case `taglib uri="um.tld"`	`<um:getChildGroups>`
	`<um:getgroupnamesforuser>`	Camel case `taglib uri="um.tld"`	`<um:getGroupNamesForUser>`
	`<um:getparentgroupname>`	Camel case `taglib uri="um.tld"`	`<um:getParentGroupName>`

Table 5-2 Tag Changes for WebLogic Personalization Server 3.1 (Continued)

Library	Old Style Tag Name	Change	New JSP 1.1 Tag
	<um:gettoplevelgroups>	Camel case taglib uri="um.tld"	<um:getTopLevelGroups>
	<um:getusernames>	Camel case taglib uri="um.tld" Attribute resultId = result	<um:getUsernames>
	<um:getusernamesforgroup>	Camel case taglib uri="um.tld"	<um:getUsernamesForGroup>
	<um:removegroup>	Camel case taglib uri="um.tld" Attribute resultId = result	<um:removeGroup>
	<um:removegroupfromgroup> *(previously undocumented)*	Camel case taglib uri="um.tld" Attribute resultId = result	<um:removeGroupFromGroup>
	<um:removeuser>	Camel case taglib uri="um.tld" Attribute resultId = result	<um:removeUser>
	<um:removeuserfromgroup> *(previously undocumented)*	Camel case taglib uri="um.tld" attribute resultId = result	<um:removeUserFromGroup>
User / Security	<um:login>	Camel case taglib uri="um.tld" Attribute resultId = result	<um:login>
	---	**New**	<um:logout>

Table 5-2 Tag Changes for WebLogic Personalization Server 3.1 (Continued)

Library	Old Style Tag Name	Change	New JSP 1.1 Tag
	`<um:setpassword>`	Camel case `taglib uri="um.tld"` Attribute `resultId = result`	`<um:setPassword>`
WLPS Utilities	`<es:condition>`	Tag no longer supported. Requires manual replacement. See below - **Note 2: <es:condition>**.	
	`<es:counter>`	`taglib uri="es.tld"` Attribute `id` returns an Integer or Long object. You can no longer change the value of the `counter` variable `"id"`. See below - **Note 3: <es:counter>**. Optional attribute `type` can be `long` or `Long` or `Integer` or if not specified is assumed to be `Integer`.	`<es:counter>`
	`<es:date>`	`taglib uri="es.tld"`	`<es:date>`
	`<es:foreachinarray>`	Camel case `taglib uri="es.tld"` Attribute `array` must be a run-time expression (`<%=expression%>`). Attribute `counterId` returns an Integer object (use `id.intValue()`).	`<es:forEachInArray>`

Table 5-2 Tag Changes for WebLogic Personalization Server 3.1 (Continued)

Library	Old Style Tag Name	Change	New JSP 1.1 Tag
	`<es:isnull>`	Camel case `taglib uri="es.tld"` Attribute `id = item` Attribute `item` must be a run-time expression. An empty string is now treated as a value. (An empty string is not null.)	`<es:isNull>`
	`<es:monitorsession>`	Camel case `taglib uri="es.tld"`	`<es:monitorSession>`
	`<es:notnull>`	Camel case `taglib uri="es.tld"` Attribute `id = item` Attribute `item` must be a run-time expression. An empty string is now treated as a value. (An empty string is not null.)	`<es:notNull>`
	`<es:preparedstatement>`	Camel case `taglib uri="es.tld"` Add two new scriptlets. See below - **Note 4: <es:preparedStatement>**. Attribute `pool` no longer supported. See below - **Note 4: <es:preparedStatement>**.	`<es:preparedStatement>`
	`<es:simplereport>`	Camel case `taglib uri="es.tld"` Attribute `resultSet` must be a run-time expression.	`<es:simpleReport>`

Table 5-2 Tag Changes for WebLogic Personalization Server 3.1 (Continued)

Library	Old Style Tag Name	Change	New JSP 1.1 Tag
	`<es:transposearray>`	Camel case `taglib uri="es.tld"` Attribute `array` must be a run-time expression.	`<es:transposeArray>`
	`<es:usertransaction>`	Tag no longer supported. Replace with new scriptlets. See below - **Note 5: <es:usertransaction>**.	
	`<es:uricontent>`	Camel case `taglib uri="es.tld"`	`<es:uriContent>`

WLS Utilities	`<wl:process>`	`taglib uri="weblogic.tld"`
		`notname = notName`
		`notvalue = notValue`
			`<wl:process>`

	---	**New**	`<wl:repeat>`

Additional Notes About JSP Tags

Note 1: <pz:> Tags

To use the `<pz:div>` and `<pz:contentSelector>` tags, you no longer need to have the JSP extended. You are no longer required to insert the following directive into your code:

```
<%@ page extends="com.beasys.commerce.axiom.p13n.jsp.P13NJspBase"
%>.
```

(If you have already added this code, it does no harm to leave it.)

Note 2: <es:condition>

The `<es:condition>` tag is no longer supported. Replace it manually with a scriptlet, creating your own `if` statement.

Old Usage

```
<es:condition test="schemaPortletNames.length>0"> </es:condition>
```

New Usage

```
<% if (schemaPortletNames.length>0) { %>
<% } %>
```

Note 3: <es:counter>

If you were manipulating the `counter` variable within the `<es:counter>` tag, you will now need to use a scriptlet instead.

Old Usage

```
<es:counter id="colIter" minCount="0"
maxCount="<%=numOfCols%>">
colIter++;
</es:counter>
```

New Usage

```
<%
for (int colIter = 0; colIter<numOfCols; colIter++) {
colIter++;
}
%>
```

Note 4: <es:preparedStatement>

The new `<es: preparedStatement>` tag includes two new scriptlets. In addition, this tag no longer supports the `"pool"` attribute. (The pool defined in `commerce.propeties` as `"commerce.jdbc.pool.name"` is used for connections.)

Old Usage

```
<es:preparedstatement id="ps" sql"<%=bookmarkBean.QUERY%>"
pool="commercePool">
<%
```

```
    bookmarkBean.createQuery(ps, owner);
    java.sql.ResultSet resultSet = ps.executeQuery();
    bookmarkBean.load(resultSet);
%>
</es:preparedstatement>
```

New Usage

```
<es:preparedStatement id="ps" sql="<%=bookmarkBean.QUERY%>">
<%@ include file="startPreparedStatement.inc" %>
<%
    bookmarkBean.createQuery(ps, owner);
    java.sql.ResultSet resultSet = ps.executeQuery();
    bookmarkBean.load(resultSet);
%>
<%@ include file="endPreparedStatement.inc" %>
</es:preparedStatement>
```

Note 5: <es:usertransaction>

The old `<es:usertransaction>` tag is no longer supported. The following code illustrates how to create equivalent functionality.

Old Usage

```
<es:usertransaction>
---- body of page --------
</es:usertransaction>
```

New Usage

```
<%
setSessionValue(com.beasys.commerce.axiom.jsp.JspConstants.
USER_TRANS_TIMEOUT, "500",request);
// tx timeout defaults to 600 sec. without above line
%>
<%@ include file="startUserTransaction.inc" %>
---- body of page --------
<%@ include file="endUserTransaction.inc" %>
```

Index

Symbols

A

B

C

D

E

F

G

H

I

J

L

M

N

O

P

R

S