BEA WebLogic Integration™

Upgrade Guide

Version 8.1
July 2003

Copyright

Restricted Rights Legend

Trademarks or Service Marks

Contents

About This Document

1. Introduction to Upgrading

2. Installing the Upgrade Utilities and Wizard

About This Document

This document provides information on upgrading WebLogic Integration 2.1 SP2 (Service Pack 2) and WebLogic Integration 7.0 SP2 to WebLogic Integration 8.1.

This document covers the following topics:

- Chapter 1, "Introduction to Upgrading" provides information helpful to read before upgrading to WebLogic Integration 8.1.
- Chapter 2, "Installing the Upgrade Utilities and Wizard"
- Chapter 3, "Step 1: Create Application For Upgrade" contains information about creating a WebLogic Integration 8.1 application for your upgrade
- Chapter 4, "Step 2: Re-deploy Business Operations EJB" provides information about re-deploying WebLogic Integration 2.1 SP2 or 7.0 SP2 EJBs in WebLogic Integration 8.1.
- Chapter 5, "Step 3: Export Package File from WebLogic Integration BPM Studio" provides information about exporting workflows from WebLogic Integration 2.1 SP2 or 7.0 SP2.
- Chapter 6, "Step 4: Export Trading Partner Management Configuration Data" provides information for B2B integration users on how to export Trading Partner configuration data from WebLogic Integration 2.1 SP2 or 7.0 SP2.
- Chapter 7, "Step 5: Upgrade TPM Configuration Data" provide information for B2B integration users on how to import Trading Partner configuration data into WebLogic Integration 8.1.

- Chapter 8, "Step 6: Export Application Views and other AI Repository Artifacts" provides information for application integration users on exporting Application Views from WebLogic Integration 2.1 SP2 or 7.0 SP2.
- Chapter 9, "Step 7: Repackage Application Integration Adapters and Deploy" provides application integration users with the steps needed to repackage custom or third-party adapters for use in WebLogic Integration 8.1.
- Chapter 10, "Step 8: Import Application Views and Other AI Repository Artifacts" provides application integration users with information about importing, testing, and publishing Application Views in WebLogic Integration 8.1
- Chapter 11, "Step 9: Upgrade Workflows" describes how to use the Upgrade Wizard to transform workflows from WebLogic Integration 2.1 SP2 or 7.1 SP2 to business processes in WebLogic Integration 8.1.
- Chapter 12, "Step 10: View the Upgrade Log" provide information about using the Upgrade log to identify the workflows and parts of workflows that could not be upgraded by the Upgrade Wizard.
- Chapter 13, "Step 11: Run and Test Upgraded Business Processes" provides information about running, testing, and optimizing the upgraded business processes.
- Chapter 14, "Upgrading Security Features" describes how to upgrade security features to WebLogic Integration 8.1.
- Chapter 15, "Upgrading Application View Controls Created in WebLogic Workshop" provides information about upgrading Application View controls created in WebLogic Workshop 7.0 SP2.
- Chapter 17, "Upgrading an Adapter Development Project" provides information about upgrading an Adapter Development Project development tree created in WebLogic Integration 2.1 SP2 or 7.0SP2.
- Chapter 16, "Upgrading Utility Adapters" contains information about the correspondence between utility adapter in WebLogic Integration 2.1 SP2 or 7.0 SP2 and features in WebLogic Integration 8.1.

What You Need to Know

This document is intended mainly for application developers who have a knowledge of Java; business process management, especially workflow design; B2B integration; data integration; and WebLogic Server security as it pertains to the application being upgraded. Additionally, you should have a complete understanding of WebLogic Integration 8.1 and WebLogic Workshop.

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 Integration documentation Home page on the e-docs Web site (and also on the documentation CD). You can open the PDF in Adobe Acrobat Reader and print the entire document (or a portion of it) in book format. To access the PDFs, open the WebLogic Integration documentation Home page, click the PDF files button and select the document you want to print.

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

Related Information

The following BEA documents contain information that is relevant to upgrading from WebLogic Integration 2.1 SP2 or 7.0 SP2 to WebLogic Integration 8:

- BEA WebLogic Integration 2.1 SP2 documentation at http://edocs.bea.com/wlintegration/v2_1sp/index.html
- BEA WebLogic Integration 7.0 SP2 documentation at http://edocs.bea.com/wli/docs70/index.html
- BEA WebLogic Workshop 7.0 SP2 documentation at http://edocs.bea.com/workshop/docs70/index.html
- BEA WebLogic Integration 8.1 documentation at http://edocs.bea.com/wli/docs81/index.html
- BEA WebLogic Workshop 8.1 documentation at http://edocs.bea.com/workshop/docs81/index.html
- BEA WebLogic Server 8.1 documentation at http://edocs.bea.com/wls/docs81/index.html

Contact Us!

Your feedback on the BEA WebLogic Integration documentation is important to us. Send us e-mail at **docsupport@bea.com** if you have questions or comments. Your comments will be reviewed directly by the BEA professionals who create and update the WebLogic Integration documentation.

In your e-mail message, please indicate that you are using the documentation for the BEA WebLogic Integration 8.1 release.

If you have any questions about this version of BEA WebLogic Integration, or if you have problems installing and running BEA WebLogic Integration, contact BEA Customer Support through BEA WebSupport at **www.bea.com**. You can also contact Customer Support by using the contact information provided on the Customer Support Card, which is included in the product package.

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

- Your name, e-mail address, phone number, and fax number
- Your company name and company address
- Your machine type and authorization codes
- The name and version of the product you are using
- A description of the problem and the content of pertinent error messages

Documentation Conventions

The following documentation conventions are used throughout this document.

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

Convention	Item

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

`monospace boldface text`	Identifies significant words in code. *Example*: `void` **`commit`** `( )`
`monospace italic text`	Identifies variables in code. *Example*: `String` *`expr`*
UPPERCASE TEXT	Indicates device names, environment variables, and logical operators. *Example*s: LPT1 SIGNON OR
`{ }`	Indicates a set of choices in a syntax line. The braces themselves should never be typed.
`[ ]`	Indicates optional items in a syntax line. The brackets themselves should never be typed. *Example*: `buildobjclient [-v] [-o name ] [-f` *`file-list`*`]... [-l` *`file-list`*`]...`
|	Separates mutually exclusive choices in a syntax line. The symbol itself should never be typed.

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

CHAPTER 1

Introduction to Upgrading

This document provides information on upgrading WebLogic Integration 2.1 SP2 (Service Pack 2) and WebLogic Integration 7.0 SP2 to WebLogic Integration 8.1.

This section provides information about the following topics:

About Upgrading

The following list provides important guidelines for upgrading from WebLogic Integration 2.1 SP2 and 7.0 SP2 to WebLogic Integration 8.1:

- **Upgrading is intended for the development environment**—You cannot upgrade a production system; no support exists for upgrading a running application.
- **Upgrading is a developer task**—Depending on the complexity of your existing WebLogic Integration 2.1 SP2 or 7.0 SP2 application, upgrading may require a knowledge of Java; business process management, especially workflow design, B2B integration, and data integration; and WebLogic Server security. Some of these terms have changed in WebLogic Integration 8.1; see Terminology Changes in this list.

- **Familiarity with the application being upgraded**—You should have a thorough understanding the application you are upgrading.
- **Familiarity with WebLogic Integration 8.1**—Before upgrading your application, you should have a complete understanding of WebLogic Integration 8.1 and WebLogic Workshop 8.1. The following sections in the WebLogic Workshop Help are especially helpful:
  - Developing Applications with WebLogic Workshop at the following URL:

    `http://edocs.bea.com/workshop/docs81/doc/en/workshop/guide/navDevGuide.html`
  - Getting Started with WebLogic Workshop at the following URL:

    `http://edocs.bea.com/workshop/docs81/doc/en/workshop/guide/getstarted/navGettingStartedWorkshop.html`
  - Tutorial: Building Your First Business Process at the following URL:

    `http://edocs.bea.com/workshop/docs81/doc/en/integration/tutorial/tutWLIProcessIntro.html`
  - Tutorial: Your First Data Transformation at the following URL:

    `http://edocs.bea.com/workshop/docs81/doc/en/integration/dttutorial/tutWLIDataTransIntro.html`
- **Terminology Changes**—The following terminology changes have been made in the WebLogic Integration 8.1 documentation:
  - The term *business process* replaces the term *workflow*. In this guide, business process is used when it applies to WebLogic Integration 8.1 and workflow is used when it applies to WebLogic Integration 2.1 or 7.1 SP2 applications.
  - *Business process design* or *process design* replaces the term *workflow design*.
  - *Data Integration* is now called *Data Transformation*.
  - The term *trading partner integration* is used instead of *B2B*.
  - The term *business message* is used instead of *B2B message*.

  Note: More information on terminology changes is available in WebLogic Platform Terminology at the following URL:

  `http://edocs.bea.com/platform/docs81/upgrade/intro.html`

What Is Upgraded by the Upgrade Utilities and Wizard

WebLogic Integration 8.1 includes several utilities and an Upgrade Wizard for helping you upgrade WebLogic Integration 2.1 SP2 and 7.0 SP2 applications. The following list briefly describes these utilities:

- Export21 utility (`aiExport21.cmd`)—Use to export application integration (AI) repository artifacts, such as Application Views, from WebLogic Integration 2.1 SP2. It is located in the `BEA_HOME\weblogic81\integration\upgrade` folder.
- Import-Export utility (`aiimportexport.cmd` or `aiimportexport.sh`)—Use to transform and import application integration (AI) repository artifacts from WebLogic Integration 2.1 SP2 and 7.0 SP2 into WebLogic Integration 8.1. It is located in the `BEA_HOME\weblogic81\integration\bin` folder.
- Repackage Adapter utility for WebLogic Integration 2.1 SP2 and 7.0 SP2 adapters (`aiRepackageAdapter.cmd` or `aiRepackageAdapter.sh`)—Use to upgrade any custom or third-party adapters. It is located in the `BEA_HOME\weblogic81\integration\upgrade` folder.
- TPM upgrade utility (`upgradeTPM.cmd`)—Use for upgrading Trading Partner Management (TPM) configuration data (not run-time state data) and its associated services and profiles. It is located in the `BEA_HOME\weblogic81\integration\upgrade` folder.
- Upgrade Wizard—Use to upgrade workflow definitions and the data integration MFL, XSL, and XML files used by worklflows. The Upgrade Wizard provides a best effort upgrade; some workflows or parts of workflows may not be upgraded. This tool is available in WebLogic Workshop under **Tools→Integration→Upgrade Wizard**.

  Note: For a detailed list, see "What the Wizard Upgrades" on page 11-2.

Features That Require Manual Upgrading

The following features are not upgraded by the Upgrade Wizard or other utilities and need to be manually upgraded:

- **BPM plug-in framework**—All custom BPM plug-ins, including non-data integration, non-AI, B2B integration, and File plug-ins must be manually upgraded. To learn about developing this functionality in WebLogic Integration 8.1, see the following documents:
  - Designing WebLogic Integration Solutions at `http://edocs.bea.com/wli/docs81/design/index.html`

  - Managing WebLogic Integration Solutions at `http://edocs.bea.com/wli/docs81/manage/index.html`
  - WebLogic Workshop Help at `http://e-docs.bea.com/workshop/docs81/doc/en/core/index.html`

- **Utility adapters**—The Email and File adapters in WebLogic Integration 2.1 SP2 and 7.0 SP2 have been replaced with system features in WebLogic Integration 8.1. The Adapter for RDBMS is replaced by a new RDBMS Adapter for 8.1, which has new capabilities. Consequently, these three adapters cannot be upgraded and you will need to replace their implementation with the new features in WebLogic Integration 8.1. For more information, see "Upgrading Utility Adapters" on page 16-1.
- **WebLogic Workshop 7.0 SP2 Application View Controls**—In WebLogic Workshop 7.0 SP2, you could create Java Web Services (JWS) that used Application View controls for accessing enterprise systems through a J2EE Connector Architecture adapter. In WebLogic Integration 8.1, these controls have been completely restructured, and the API has changed. Therefore, you will need to re-develop your Application View controls in WebLogic Integration 8.1. To learn how to manually upgrade Application View Controls, see "Upgrading Application View Controls Created in WebLogic Workshop" on page 15-1.
- **Workflows that use RosettaNet protocols**—You will need to re-develop these business processes in WebLogic Integration 8.1. To learn more, see the following documents:
  - "Building RosettaNet Participant Business Processes" in theWebLogic Workshop Help at `http://edocs.bea.com/workshop/docs81/doc/en/integration/wfguide/wfguideRosettaNet.html`
  - *Designing WebLogic Integration Solutions* at `http://edocs.bea.com/wli/docs81/design/index.html`
- **BPM APIs**—You can achieve the same functionality using various features in WebLogic Integration 8.1. You will need to rewrite your WebLogic Integration 2.1 SP2 or 7.0 SP2 applications using the APIs in WebLogic Integration 8.1.

Contivo Analyst Does Not Require Upgrade

WebLogic Integration 8.1 does not include Contivo Analyst. Instead BEA provides a more powerful XQuery mapping functionality as part of WebLogic Workshop. However, if you wish to continue using Contivo, WebLogic Integration 8.1 provides run-time support for XSLT transformations. Note that when upgrading using the Upgrade Wizard, existing XSLT transformation are not converted to the new functionality, but configured to run unchanged. You continue to maintain XSLT transformations using Contivo Analyst.

To learn more about the XQuery mapper functionality, see "Transforming Data Using XQuery" in the WebLogic Workshop Help at the following URL:

`http://edocs.bea.com/workshop/docs81/doc/en/integration/dtguide/dtguideMapper.html`

Overview of Upgrade Process

The following is a general upgrade strategy. Depending on your application, the steps you use may differ from this approach. Details of each step are described in the remainder of this guide.

1. In WebLogic Integration 8.1, create a Process Application for your upgrade. See "Step 1: Create Application For Upgrade" on page 3-1.
2. Re-deploy the business operations EJB. See "Step 2: Re-deploy Business Operations EJB" on page 4-1.
3. In WebLogic Integration Studio, generate and export a workflow package (`.jar`) file. The JAR will include workflow, XML (Extensible Markup Language), XSLT (eXtensible Stylesheet Language), and MFL (Message Format Language) files. See "Step 3: Export Package File from WebLogic Integration BPM Studio" on page 5-1.

Use steps 4 and 5 only if you are using B2B integration.

4. In WebLogic Integration 2.1 SP2 or 7.0 SP2, export B2B Trading Partner Management (TPM) configuration data using the B2B Console or Bulk Loader. You can export TPM configuration data from either a development environment and a running production system. See "Step 4: Export Trading Partner Management Configuration Data" on page 6-1.
5. Use the WebLogic Integration 8.1 TPM upgrade utility for upgrading TPM configuration data and its associated services and profiles exported from WebLogic Integration 2.1 SP2 and WebLogic Integration 7.0 SP2. See "Using the Trading Partner Management Upgrade Utility" on page 7-3.

Use steps 6, 7, and 8 only if you are using application integration.

6. Using the Export21 utility (included in WebLogic Integration 8.1) or the WebLogic Integration 7.0 import/export utility, export application integration (AI) repository artifacts. These utilities export Application Views, connection factories schemas, and namespaces to a JAR file. See "Step 6: Export Application Views and other AI Repository Artifacts" on page 8-1

7. Using either the Repackage Adapter utility for WebLogic Integration 2.1. SP2 adapters or WebLogic Integration 7.0 SP2 adapters, upgrade any custom or third-party adapters and deploy on WebLogic Integration 8.1. See "Step 7: Repackage Application Integration Adapters and Deploy" on page 9-1.
8. With the WebLogic Integration 8.1 import-export utility, import the AI repository artifacts that you exported in Step 6 into your WebLogic Integration 8.1 application. See "Step 8: Import Application Views and Other AI Repository Artifacts" on page 10-1.
9. Using theWebLogic Integration 8.1 Upgrade Wizard, upgrade the workflows definitions and the data integration MFL, XSL, and XML files used by worklflows. See "Step 9: Upgrade Workflows" on page 11-1.
10. Examine the upgrade log and use the provided information for fixing business processes, identifying what needs to be upgraded manually, and optimizing the upgraded business processes. See "Step 10: View the Upgrade Log" on page 12-1.
11. Run and test your business processes. See "Step 11: Run and Test Upgraded Business Processes" on page 13-1.

CHAPTER 2

Installing the Upgrade Utilities and Wizard

This section describes how to install the Upgrade utilities and Upgrade Wizard. It contains information on the following topics:

- Install the Upgrade Utilities and Wizard
- Edit aiExport21
- Contents of Upgrade JAR

Install the Upgrade Utilities and Wizard

Before upgrading, you must first install the Upgrade utilities and Upgrade Wizard. These utilities are in the Upgrade JAR, which is available on the BEA dev2dev Web site at the following URL:

`http://dev2dev.bea.com/resourcelibrary/utilitiestools/upgrade.jsp#wliupgrade`

After downloading the JAR file, install the utilities with the following steps:

1. If WebLogic Workshop is running, close it.
2. Unjar the `upgrade.jar` file to a temporary directory.
3. In the temporary directory, open the `workshop-lib` directory, then copy the `migration.jar` to the *`BEA_HOME`*`/weblogic81/workshop/lib` directory.

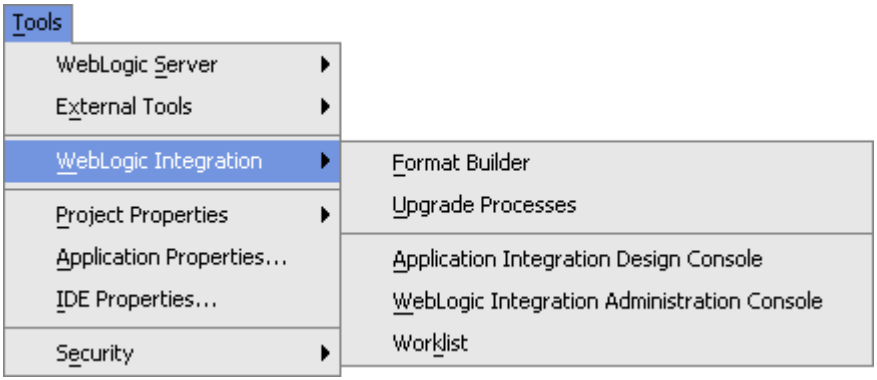
   In the preceding line, *`BEA_HOME`* represents the WebLogic Platform 8.1 home directory.
4. From the temporary directory you created, copy the `upgrade` directory to the *`BEA_HOME`*`/weblogic81/integration` directory.

The upgrade utilities are installed in the upgrade directory

5. Start WebLogic Workshop.

   The Upgrade Wizard and upgrade utilities are now available, as shown in Figure 2-1.

Figure 2-1 Upgrade Wizard is Available



Edit aiExport21

If you plan to upgrade WebLogic Integration 2.1 SP2 Application Views, you need to edit the `aiExport21.cmd` file. Specifically, you need to set the correct path for calling the `setEnv.cmd` in your WebLogic Integration 2.1 SP2 directory. After installing the Upgrade JAR, the `aiExport21.cmd` file is located in the following directory:

```
BEA_HOME\weblogic81\integration\upgrade
```

In the preceding line, *BEA_HOME* represents the WebLogic Platform 8.1 home directory.

For example to set the path for the default installation for WebLogic Integration 2.1 SP2, you would set the path as follows:

```
call C:\bea\wlintegration2.1\setEnv.cmd
```

Contents of Upgrade JAR

The Upgrade JAR file contains the following directories and files:

- `META-INF` directory—Contains the `MANIFEST.MF` file, which is automatically created for the JAR file. This file is not used for upgrading.

- `upgrade` directory—Contains the upgrade utilities and wizard. For a full description of the utilities, see "What Is Upgraded by the Upgrade Utilities and Wizard" on page 1-3.
- `workshop-lib` directory—Contains the `migration.jar` file. This file contains the Java classes that parse the workflows when upgrading WebLogic Integration 2.1 SP2 or 7.0 SP2 to WebLogic Integration 8.1.
- `readme.txt`—Contains instructions on installing the upgrade utilities and the Upgrade Wizard.

CHAPTER 3

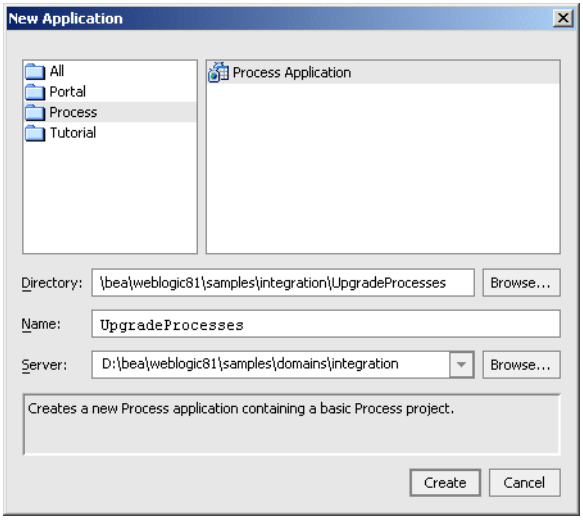
Step 1: Create Application For Upgrade

Before upgrading your WebLogic Integration 2.1 SP2 or 7.0 SP2 application, create a WebLogic Integration 8.1 process application for your upgrade, as described in How Do I: Create a New Application? in the WebLogic Workshop Help at the following URL:

```
http://edocs.bea.com/workshop/docs81/doc/en/integration/howdoI/howStartNewapp.html
```

In the **Name** field, specify a name for the application, then click **Create**, as shown in the following figure.

Figure 3-1 New Application



CHAPTER 4

Step 2: Re-deploy Business Operations EJB

A business operation Enterprise Java Bean (EJB) encapsulates business logic. Business logic is the code that fulfills the purpose of the application, such as checking inventory or ordering a part.

You can use the same business operation EJBs that you created for WebLogic Integration 2.1 SP2 or 7.0 SP2 in WebLogic Integration 8.1. In WebLogic Integration 2.1 SP2 and 7.0 SP2, the Studio was used to enable workflows to use business operation EJBs. In WebLogic Integration 8.1, you use Java and EJB controls to utilize the business logic encapsulated in your business operation EJB. EJB controls make it easy for you to use an existing, deployed EJB from within an application.

To re-deploy your business operation EJB, take the following steps:

1. Add the EJB JAR to the `Modules` folder in your upgrade application, then deploy.
2. For business operations that use Java classes, import the classes into the `Libraries` folder of your upgrade application.

To learn about deploying EJBs, see Deployment in the WebLogic Server documentation at the following URL:

```
http://edocs.bea.com/wls/docs81/deployment.html
```

To learn about using Java Controls, see Working with Java Controls in the WebLogic Workshop Help at the following URL:

```
http://edocs.bea.com/workshop/docs81/doc/en/workshop/guide/controls/navWorkingWithJavaControls.html
```

To learn about using EJB Controls, see Using Integration Controls in the WebLogic Workshop Help at the following URL:

```
http://edocs.bea.com//workshop/docs81/doc/en/integration/controls/contr
olsIntro.html
```

CHAPTER 5

Step 3: Export Package File from WebLogic Integration BPM Studio

Before upgrading your WebLogic Integration 2.1 SP2 or 7.0 SP2 applications, you must first export your workflows and other application data to a JAR file.

Note: In some instances, it may be necessary to modify the original WebLogic Integration 2.1 SP2 or 7.0 SP2 workflows. To see if this applies to any of your workflows, see "Upgrade Wizard Limitations" on page 11-2.

To export your workflows, take the following steps:

1. In WebLogic Integration 2.1 SP2 or 7.0 SP2 Studio, generate and export a workflow package (`.jar`) file. The JAR will include workflow, XML (Extensible Markup Language), XSLT (eXtensible Stylesheet Language), and MFL (Message Format Language) files.

   Note: To learn about exporting a workflow package in WebLogic Integration 2.1 SP2, see Importing and Exporting Workflow Packages in *Using the WebLogic Integration Studio* at the following URL:

   `http://e-docs.bea.com/wlintegration/v2_1/studio/ch11.htm`

   To learn about exporting a workflow package in WebLogic Integration 7.0 SP2, see Importing and Exporting Workflow Packages in *Using the WebLogic Integration Studio* at the following URL:

   `http://edocs.bea.com/wli/docs70/studio/ch11.htm`

2. Transfer the exported TPM XML file to your target environment for the WebLogic Integration 8.1 application.

3. If you are *not* upgrading B2B integration or application integration components go to "Step 9: Upgrade Workflows" on page 11-1.

If you are upgrading B2B integration components, go to "Step 4: Export Trading Partner Management Configuration Data" on page 6-1.

If you are upgrading application integration components, go to "Step 6: Export Application Views and other AI Repository Artifacts" on page 8-1.

Step 4: Export Trading Partner Management Configuration Data

This step applies only if you are upgrading B2B integration components.

Before you can upgrade your Trading Partner Management (TPM) configuration data and its associated services and profiles from WebLogic Integration 2.1 SP2 or 7.0 SP2, you must first export this information to an XML file.

Note: If a value does not exist for the WLPI Template Name attribute in your B2B conversation-definition roles, you may enter one (if applicable) before upgrading. If this value is missing, the new TPM data will not include the business process or control service derived from the conversation-definition role.

To learn about conversation definitions in WebLogic Integration 2.1 SP2, see Configuring Conversation Definitions in *Administering B2B Integration* at the following URL:

`http://edocs.bea.com/wlintegration/v2_1/b2badmin/cfgtasks.htm#998395`

To learn about conversation definitions in WebLogic Integration 7.0 SP2, see Configuring Conversation Definitions in *Administering B2B Integration* at the following URL:

`http://edocs.bea.com/wli/docs70/b2badmin/cfgtasks.htm#1003805`

To export your TPM configuration data, take the following steps:

1. If you are not using a keystore in your WebLogic Integration 2.1 SP2 or 7.0 SP2 B2B application, you must create one.

The WebLogic Integration 8.1 Trading Partner Management uses the keystore to manage trading partner's certificates. (Trading Partner Management holds only the entry of the certificate in the keystore). To learn about keystores, see "WebLogic B2B Security Upgrade" on page 14-4.

2. Export your WebLogic Integration 2.1 or 7.0 SP2 B2B repository information to an XML file using the Bulk Loader or the WebLogic Integration B2B Console export functions.

   Note: If you are exporting repository information from a running system using the command-line Bulk Loader utility, you must use the `-force` option.

   WebLogic Integration 2.1 SP2: To learn about exporting repository data from this version of WebLogic Integration, see the following documents:

   – Working with the Bulk Loader in *Administering B2B Integration* at the following URL:

     `http://e-docs.bea.com/wlintegration/v2_1/b2badmin/bulkload.htm`.

   – Exporting Repository Data in the *Online Help for the* WebLogic Integration *B2B Console* at the following URL:

     `http://e-docs.bea.com/wlintegration/v2_1/b2bhelp/admncfg.htm#1000060`.

   WebLogic Integration 7.0 SP2: To learn about exporting repository information in this version of WebLogic Integration, see the following documents:

   – Exporting Data from the Repository in the *WebLogic Integration 7.0 Administering B2B Integration* at the following URL:

     `http://e-docs.bea.com/wli/docs70/b2badmin/bulkload.htm#1082771`

   – Exporting Repository Data in the *Online Help for the* WebLogic Integration *B2B Console* at the following URL:

     `http://e-docs.bea.com/wli/docs70/b2bhelp/admncfg.htm#1000060`

3. Transfer the exported TPM XML file to your target environment for the WebLogic Integration 8.1 application.

4. To import the TPM XML file into your WebLogic Integration 8.1 upgrade application, go to "Step 5: Upgrade TPM Configuration Data" on page 7-1.

CHAPTER 7

Step 5: Upgrade TPM Configuration Data

This step applies only if you are upgrading B2B integration components.

WebLogic Integration 8.1 provides a TPM upgrade utility for upgrading Trading Partner Management (TPM) configuration data and its associated services and profiles from WebLogic Integration 2.1 SP2 and WebLogic Integration 7.0 SP2.

Note: The TPM upgrade utility upgrades configuration data, *not* run-time state data.

What the Trading Partner Management Upgrade Utility Does

The TPM upgrade utility upgrades the following:

- Trading Partner information:
  - `trading-partner` and its attributes, including: `name`, `description`, `notes`, `type`, `email`, `phone`, `fax`, and `username`.
  - `address`
  - `extended-property-set`
  - Trading partner's security certificate. See Chapter 6, "Step 4: Export Trading Partner Management Configuration Data" and "WebLogic B2B Security Upgrade" on page 14-4.
  - `ebxml-binding`
  - `transport` (under the `<ebxml-binding>` element)
  - `authentication` (under the `<transport>` element)
  - `rosettaNet-binding`

- Service information:
  - `service`—For each `conversation-definition` where the `business-protocol` is either `ebxml` or `rosettanet`, there are two roles. For each `role`, two `service` elements are generated in the upgraded WebLogic Integration 8.1 repository XML file, one for the process-type `service` and the other for the control-type `service`.
  - `service-profile`

Information Not Upgraded by the Utility

The following information is not mapped by the TPM upgrade utility:

- `wlc/logic-plugin`, `wlc/business-protocol-definition`—Replaced by B2B protocol data in WebLogic Integration 8.1.
- `wlc/trading-partner/xpath-expression`—Applied to XOCP which was deprecated in WebLogic Integration 7.0 SP2.
- `wlc/conversation-definition/role/process-implementation`—Does not apply in WebLogic Integration 8.1.
- `update-count`—Does not apply in WebLogic Integration 8.1.

Limitations

The TPM upgrade utility has the following limitations:

- Does not upgrade run-time state data.
- The deprecated XOCP and cXML business protocols related entries are not upgraded.
- Does not upgrade the B2B protocol stack, which is mostly static and provided with WebLogic Integration 8.1.
- The upgraded trading partner data is not formatted. If formatting is needed, use an editor such as XMLSPY to format this data.

Warning Case

As mentioned in "Step 4: Export Trading Partner Management Configuration Data" on page 6-1, if a value does not exist for the WLPI Template Name attribute in your B2B conversation-definition role, you may enter one (if applicable) before upgrading. If this value does not exist, you will receive the following warning:

```
<date and time> <Error> <Upgrade> <600001> <Missing
wlpi-templateattribute in the conversation-Definition/role element. No
workflow nor control service will be generated for this role.>
```

You can add this information after using the TPM upgrade utility with the WebLogic Integration Administration Console. To learn about importing, see Trading Partner Management in *Managing WebLogic Integration Solutions* at the following URL:

```
http://edocs.bea.com/wli/docs81/manage/tpm.html
```

Using the Trading Partner Management Upgrade Utility

Before you can use the TPM upgrade utility, you must first export this information to an XML file as described in "Step 4: Export Trading Partner Management Configuration Data" on page 6-1.

To upgrade your TPM data using the TPM upgrade utility, take the following steps:

1. On a Windows system, open a command window.
2. In both Windows and UNIX, go to the following directory:

   BEA_HOME/weblogic81/integration/upgrade

   In the preceding line, *BEA_HOME* represents the WebLogic Platform home directory.
3. Execute the upgrade of B2B configuration data by entering:

   Windows: `upgradeTPM.cmd <sourceFileName> <targetFileName> [workflowPath] [ctrlPackage]`

   UNIX: `upgradeTPM.sh <sourceFileName> <targetFileName> [workflowPath] [ctrlPackage]`

   Table 7-1 contains the command-line parameters for the upgradeTPM utility.

   After the file is successfully transformed, the following message is displayed:

   ```
   <date and time> <Info> <Upgrade> <600003> <Transformation Completed
   Successfully>

   Transformation Completed Successfully
   ```

   If the file is not successfully transformed, an error message is displayed.

Table 7-1 Command-Line Parameters for the upgradeTPM Utility

Command	Description
`sourceFileName`	The directory path and name of the TPM XML file to be upgraded.
`targetFileName`	The directory path and name of the upgraded TPM XML file.
`workflowPath` (Optional)	The path name of the workflow that uses the ebXML control, such as `bea.myOrders`. **Note:** If you know the workflow path, enter this parameter; otherwise, leave it blank. After the TPM data is loaded, you can add this parameter in the WebLogic Integration Administration Console→Trading Partner Management→ Service Management→Name field. See Viewing and Changing Services in *Managing WebLogic Integration Solutions* at the following URL: `http://edocs.bea.com/wli/docs81/manage/tpm.html`
`ctrlPackage` (Optional)	The package name of the ebXML Control, such as `bea.ebxmlCtrlPackage`. **Note:** If you know the control package name, enter this parameter; otherwise, leave it blank. After the TPM data is loaded, you can add this parameter using the WebLogic Integration Administration Console→Trading Partner Management→Service Management→Business Service Name field. See Viewing and Changing Services in *Managing WebLogic Integration Solutions* at the following URL: `http://edocs.bea.com/wli/docs81/manage/tpm.html`

4. To verify the upgrade and add optional configuration data, import the upgraded file using the WebLogic Integration Administration Console. To learn about importing, see Importing Management Data in *Managing WebLogic Integration Solutions* at the URL:

   `http://edocs.bea.com/wli/docs81/manage/tpm.html#1090102`

5. If you are upgrading application integration components, go to "Step 6: Export Application Views and other AI Repository Artifacts" on page 8-1; otherwise, go to "Step 9: Upgrade Workflows" on page 11-1.

CHAPTER 8

Step 6: Export Application Views and other AI Repository Artifacts

This step applies only if you are upgrading application integration (AI) components.

Different utilities are needed to export Application Views and other AI repository artifacts from WebLogic Integration 2.1 SP2 and 7.0 SP2 to JARs that can be imported into your WebLogic Integration 8.1 upgrade application. These utilities export Application Views, connection factories schemas, and namespaces into a single JAR file. After exporting, transfer the JAR file to your target environment for the WebLogic Integration 8.1 application.

Note: Upgrading Application Views should be done only if you are using corresponding adapters in WebLogic Integration 2.1 SP2 or 7.0 SP2 and WebLogic Integration 8.1 (for example, WebLogic Adapter for RDBMS 7.1 and WebLogic Adapter for RDBMS for WebLogic Integration 8.1).

Exporting AI Repository Artifacts

Note: If you are exporting WebLogic Integration 2.1 SP2 repository artifacts, you must edit the `aiExport21.cmd` file to use the correct path for the `setEnv.cmd`, as described in "Edit aiExport21" on page 2-2. The aiExport21 utility is included in WebLogic Integration 8.1.

To export AI Repository Artifacts, take the following steps:

1. On a Windows system, open a command window.
2. In both Windows and UNIX for artifacts, go to one of the following directories:

   WebLogic Integration 2.1 SP2: *BEA_HOME*`/weblogic81/integration/upgrade`

   WebLogic Integration 7.0 SP2: *BEA_HOME*`/weblogic700/integration/bin`

In the preceding paragraphs, *BEA_HOME* represents the WebLogic Platform home directory.

3. Execute the exporting of the repository artifacts.

   WebLogic Integration 2.1 SP2

   Usage for exporting:

   ```
   aiexport21 <WLS URL> <user name> <password> <file>
   [-codepage=<#>][-dump=< <namespace> | <'Root'> ] [-append]
   < [-export [object name]*] >
   ```

   Table 8-1 contains the command-line parameters for the aiexport21 utility.

   WebLogic Integration 7.0 SP2

   Usage for exporting:

   ```
   aiimportexport <WLS URL> <user name> <password> <file>
   [-codepage=<#>][-dump=< <namespace> | <'Root'> ] [-append]
   < [-export [object name]*] >
   ```

   Table 8-1 contains the command-line parameters for the aiimportexport and aiexport21 utilities.

Table 8-1 Command-Line Parameters for the aiExport21 and aiimportexport Utilities

Parameter	Description
`WLS URL`	URL of WebLogic Server.
`user name`	Your username for the specified WebLogic Server.
`password`	Your password for the specified WebLogic Server.
`file`	The JAR file exported from the WebLogic Integration 2.1 SP2 or 7.0 SP2 application.

Parameter	Description
`-codepage`	Optional. The codepage used when writing to console. It ensures that characters are displayed correctly. The default is Cp437 (United States).Other valid values include: • Cp850 Multilingual (Latin I) • Cp852 Slavic (Latin II) • Cp855 Cyrillic (Russian) • Cp857 Turkish • Cp860 Portuguese • Cp861 Icelandic • Cp863 Canadian-French • Cp865 Nordic • Cp866 Russian • Cp869 Modern Greek • MS932 Japanese The value specified must match your console's codepage. **Note:** On Window systems the *chcp* command displays the console's codepage.
`-dump`	Prints a list of all objects within both the specified namespace and other namespaces nested within it. To print a list of objects for the entire directory structure, specify `Root`.
`-append`	Appends exported items to the file specified by `file` instead of overwriting it.
`-export`	Specifies an export operation and the name of the objects to be exported (namespaces and application views) and any objects they contain. These objects are stored into file. When specifying an object within a namespace use "." as the delimiter (for example, `mynamespc.myappview`). **Note:** To export the entire directory structure, include `Root` in the list of object names.

4. To upgrade WebLogic Integration 2.1 SP2 and 7.0 SP2 custom or third-party adapters, go to "Step 7: Repackage Application Integration Adapters and Deploy" on page 9-1.

To import Application Views and other AI repository artifacts, go to "Step 8: Import Application Views and Other AI Repository Artifacts" on page 10-1.

Step 7: Repackage Application Integration Adapters and Deploy

This step applies only if your are upgrading application integration (AI) components.

WebLogic Integration 8.1 provides a aiRepackageAdapter utility for upgrading WebLogic Integration 2.1 SP2 and 7.0 SP2 custom and third-party adapters. Specifically, this utility converts adapter EARs (Enterprise Archive file) to WebLogic Integration 8.1 EARs.

Note: Note, do *not* use the aiRepackageAdapter utility for upgrading BEA Adapters. You should use BEA WebLogic 8.1 Adapters. To learn about BEA adapters, see BEA WebLogic Adapters 8.1 at the following URL:

`http://edocs.bea.com/wladapters/docs81/index.html`

What the aiRepackageAdapter Utility Does

Differences exist between the structure of WebLogic Integration 2.1 SP2 or 7.0 SP2 EAR files and WebLogic Integration 8.1 EAR files. Previous WebLogic Integration EARs contained a design-time WAR (Web Application file), a run-time RAR (Resource Adapter Archive), and an event router WAR, while the WebLogic Integration 8.1 EAR contains only a design-time WAR and a run-time RAR.

The aiRepackageAdapter utility performs the following functions:

- Removes the event router WAR and incorporates the event generator classes into the run-time RAR.
- Generates a `ResourceAdapter` implementation class using the `init-param` information from the event router WAR's `web.xml` and the other overview information in the run-time RAR's `ra.xml` file. This `ResourceAdapter` implementation takes the place of the event

router for accepting event subscriptions and delivering events, and also acts as a container for connection factories.

- Updates the ADK JSP files that have changed and removes obsolete ADK JSP files. Updates design-time WAR's `web.xml` and `weblogic.xml` to refer to the new JSPs.
- Update the adapter properties file with new property information from the ADK.
- Provides new versions of all ADK and WLAI JARs.

Using the aiRepackageAdapter Utility

To repackage your adapters, take the following steps:

1. On a Windows system, open a command window.
2. In both Windows and UNIX, go to the following directory:

   BEA_Home`\weblogic81\integration\upgrade`

   In the preceding line, *BEA_HOME* represents the WebLogic Platform home directory.
3. Execute the repackaging of the adapter by entering (without the `.EAR` extension):

   Windows: `aiRepackageAdapter.cmd <parent_dir> <Adapter EAR File Name>`

   UNIX: `aiRepackageAdapter.sh <parent_dir> <Adapter EAR File Name>`

   In the preceding paragraphs, `parent_dir` represents the parent directory of the WebLogic Integration 7.0 SP2 EAR file.

   For example:

   ```
   aiRackageAdapter.cmd d:\weblogic700\integration\adapters\dbms\lib
   BEA_WLS_DBMS_ADK
   ```

   The repackaged adapter is named `<adapter name>_8.1.ear` and located in the parent directory.
4. Deploy the repackaged adapter in your WebLogic Integration 8.1 application. See *Deploying WebLogic Integration Solutions* at the following URL:

   `http://edocs.bea.com/wli/docs81/deploy/index.html.`

   Note: When you deploy, the application name should be the name of the adapter without the `_81.ear` suffix. If the suffix is used, the adapter cannot be used by WebLogic Integration 8.1. For example, the application name of a `supplier_8.1.ear` is supplier. The module (file) name can remain the same.

Step 8: Import Application Views and Other AI Repository Artifacts

This step applies only if you upgrading application integration (AI) components.

WebLogic Integration 8.1 provides an import-export utility for importing Application Views and other AI repository artifacts from a WebLogic Integration 2.1 SP2 or 7.0 SP2 AI repository. This utility can be executed from the command line, or incorporated into your code with the import-export API. The output of the utility is a JAR file.

What the Import-Export Utility Does

Upgrading your WebLogic Integration 7.0 SP2 AI repository is required because the WebLogic Integration 8.1 AI repository is contained in the directory structure of the application's local file system, while the repository for WebLogic Integration 7.0 SP2 is contained in a Relational DataBase Management System (RDBMS) database. Moreover, objects stored in the WebLogic Integration 8.1 repository have changed, as shown in the following list:

- Application View descriptor
  - No longer has an Access Control List (ACL). All access control is defined in the WebLogic Integration Administration Console. See "WebLogic Server Security Upgrade" on page 14-1.
  - Contains an imports section for importing adapter instances and connection factories for use in event delivery and service invocation.
- The Connection factory descriptor is no longer a top-level AI object. This object is now a child element of the new adapter descriptor.

- Adapter descriptor—This is a new object type in WebLogic Integration 8.1. This descriptor contains information about the adapter type and inbound messaging (event delivery), as well as zero or more connection factory descriptors.
- Schema descriptor—This descriptor is unchanged for 8.1.

Limitations

Shared connection factories—When an Application View that uses a shared connection factory is imported into WebLogic Integration 8.1, it will have an invalid adapter instance reference. When you are ready to test and publish the upgraded Application View, you need to correct the invalid reference. To make the correction, take the following steps:

1. In the Application Integration Design Console, select the Application View with the invalid adapter instance reference.
2. After the console page displays the **Define Application View** page, select the correct adapter from the **Associated Adapter** drop list.
3. Click the **Reuse Existing Connection** button.
4. Select an existing Adapter Instance for the Application View to use.

Note: To learn more about testing and publishing Application Views, see Defining an Application View in *Using the Application Integration Design Console* at the following URL:

`http://edocs.bea.com/wli/docs81/aiuser/1usrntr.html`

Namespace mismatch—Application Views upgraded to WebLogic Integration 8.1 using the import-export utility will have a namespace added to the request and response schema. If an `XmlObject` is used in the upgraded business process, such as in an XPath function, the upgraded files will not contain the namespace information. Subsequently, the XPath function will not work properly because of the namespace mismatch. The DTF (Data Transfer Format) file generated by the Upgrade Wizard must be edited to contain the namespace information. The DTF file is located in the *HOME_BEA*`/weblogic81/<Upgrade Project>/<Upgrade Project>Web/process` folder.

In the preceding paragraph, *BEA_HOME* represents the WebLogic Platform home directory.

For example, if a WebLogic Integration 7.0 SP2 document contains the following:

```
<RowsAffected>1</RowsAffected>
```

in WebLogic Integration 8.1, it is upgraded to look like this:

```
<ns0:RowsAffected xmlns:ns0="wlai/DemoDBMS_CreateCustomer_response"> 1
</ns0:RowsAffected>
```

The XPath function that is generated by the import-export utility is:

```
/**
     * Original xpath = "/RowsAffected/text()"
     * @dtf:transform xquery="string(
$xmlInput/self::RowsAffected/text())"
     *
     */
    abstract  public String xpath_2(XmlObject xmlInput);
```

You must manually add the namespace information so the XPath function can work properly, as indicated by the bold text:

```
/**
     * Original xpath = "/RowsAffected/text()"
     * @dtf:transform xquery="declare namespace
ns0='wlai/DemoDBMS_CreateCustomer_response" string(
$xmlInput/self::ns0:RowsAffected/text())"
     *
     */
    abstract public String xpath_2(XmlObject xmlInput);
```

Using the Import-Export Utility

To upgrade AI repository artifacts using the WebLogic Integration 8.1 import-export utility, complete the following steps:

Note: Before upgrading your repository, you must have created an application in WebLogic Workshop, as described in "Step 1: Create Application For Upgrade" on page 3-1. By default a new application contains a project named "Schemas". If your application does not have a Schemas project, you must create one using WLI System Schemas as the project type. The project determines the location of the repository. To learn about creating a Schemas project, see How Do I: Create a Schemas Project Folder in the WebLogic Workshop Help at the following URL:

`http://edocs.bea.com/workshop/docs81/doc/en/integration/howdoI/howSchemasCreate.html`

Note: Before you can use the import-export utility, you must first export this information to a JAR file as described in "Step 6: Export Application Views and other AI Repository Artifacts" on page 8-1.

1. Using the WebLogic Integration 8.1 import-export utility, import the JAR into your applications project's Schemas folder. Detailed steps are provided in "Invoking the Import-Export Utility from the Command Line" on page 10-4.
2. (Optional) Create an event properties file. This file provides the properties needed for event generation within the adapter instance that is created for the named connection factory. BEA recommends that this file is named `<adapter name>-eventProps.properties`.
3. (Optional) Create an event properties file. The recommended name for this file is `<adapter name>-eventProps.properties`.

   The event properties file represents the properties for an event connection that will be paired with a named connection factory in a newly generated adapter instance.

   The event properties file is used to configure event delivery on the newly created adapter instance that wraps the existing WebLogic Integration 2.1 SP2 or 7.0 SP2 connection factory descriptors. This is only necessary when the generated adapter instance is used to support event delivery for Application Views that use events. If you do not specify the event properties file, and the Application Views that uses that connection factory has defined events, you will need use the Application Integration Design Console to edit the event connection being used by the Application View to allow for event delivery on that Application View.
4. If you do not specify an event properties file when importing connection factories, you will have to edit your adapter instances using the Application Integration Design Console.

   To learn more about specifying event properties, see Defining an Application View in *Using the Application Integration Design Console* at the following URL:

   `http://edocs.bea.com/wli/docs81/aiuser/1usrntr.html`
5. Test and publish your imported Application Views and other AI repository artifacts.

   Note: You must complete this step before upgrading your workflows, as described in "Step 9: Upgrade Workflows" on page 11-1.

Note: To learn more about testing and publishing Application Views, see Defining an Application View in *Using the Application Integration Design Console* at the following URL:

`http://edocs.bea.com/wli/docs81/aiuser/1usrntr.html`

Invoking the Import-Export Utility from the Command Line

The usage for importing with the import-export utility is:

```
Usage: aiimportexport <app name> <root dir> <file> <codepage>
```

```
-import [-publish]
[-eventProps=<connection factory name>,<properties file name>]*
```

For example:

```
aiimportexport UpgradeProcesses
c:bea\weblogic81\samples\integration\UpgradeProcesses
d:bea\weblogic81\integration\70exports\appvies.jar cp437 -import
-publish -eventProps=myFactory,myAdapter-eventProps.properties
```

In the proceeding command, `myFactory` is the connection factory name within the JAR exported from the WebLogic Integration 2.1 SP2 or 7.0 SP2 repository.

Note: This usage is specific to importing JAR files that have been exported from WebLogic Integration 2.1 and 7.0 SP2 applications. For full usage, see Importing and Exporting Application Views in the *Using the Application Integration Design Console* at the following URL:

`http://edocs.bea.com/wli/docs81/aisuer/imp_appx.html`

The following table contains the command-line parameters for the import-export utility.

Parameter	Description
`app name`	The name of the J2EE application that receives the artifacts from the WebLogic Integration 2.1 or 7.0SP2 JAR.
`root dir`	The root directory of the AI repository within the WebLogic Integration 8.1 application. **Note:** You only need to specify the root directory of the application, not the location of the Schemas folder.
`file`	The JAR file exported from the WebLogic Integration 2.1 SP2 or 7.0 SP2 application.

Parameter	Description
`-codepage`	The codepage used when writing to console. It ensures that characters are displayed correctly. The default is Cp437 (United States).Other valid values include: • Cp850 Multilingual (Latin I) • Cp852 Slavic (Latin II) • Cp855 Cyrillic (Russian) • Cp857 Turkish • Cp860 Portuguese • Cp861 Icelandic • Cp863 Canadian-French • Cp865 Nordic • Cp866 Russian • Cp869 Modern Greek • MS932 Japanese The value specified must match your console's codepage. **Note:** On Window systems the *chcp* command displays the console's codepage.
`-import`	Specifies that objects contained in the JAR should be imported into the repository.
`-publish` (Optional)	Used only with `-import`. Requests the import/export utility to publish the imported Application Views to the target application. This generates the EJB JAR for the Application View and prepares the Application View for use in the target application.

Parameter	Description
`-eventProps` (Optional)	Used only with `-import` and when importing a WebLogic Integration 2.1 or 7.0 SP2 JAR. Defines a mapping between the qualified connection factory name and a properties file. This properties file provides the properties needed for event generation within the adapter instance that is created for the named connection factory. If needed, you may use multiple `-eventProps` arguments.These properties are specified in WebLogic Integration 2.1 or 7.0 SP2 within the EventRouter WAR file's `web.xml`. You can extract the name and value from the `init-param` elements of the `web.xml` and place them in a Java properties file where name is the key and the value is the value portion on each line in the file. When creating the properties file for the `-eventProps` arguments, you should take the property name/value pairs from the `init-param` elements in the `web.xml` of the WebLogic Integration 2.1 or 7.0 SP2 adapter's EventRouter WAR file. You do not need to put the following properties in `web.xml properties` in the `-eventProps` properties file: • `eventGeneratorClassName` • `RootLogContext` • `AdditionalLogContext` • `LogConfigFile` • `LogLevel` • `MessageBundleBase` • `LanguageCode` • `CountryCode` If the `-eventProps` argument does not exist or if it is not defined so that it maps a properties file to the qualified name for a connection factory, the adapter instance generated will have inbound messaging disabled. After importing, you can specify event properties for the adapter instance your using the Application Integration Design Console. To learn more about specifying event properties, see Defining an Application View in the *Using the Application Integration Design Console* at the following URL: `http://edocs.bea.com/wli/docs81/aiuser/2usrdef.html`

Parameter	Description
`connection factory name`	The qualified name of the connection factory that is mapped to the properties file. Defines a mapping between the qualified connection factory name and a properties file.
`properties file name`	The name of the properties file that the properties from the `web.xml` file are mapped to.

The following properties are typically found in a WebLogic Integration 7.0 SP2 EventRouter `web.xml` file. You can safely *not* include them in the event properties file.

- `eventGeneratorClassName`
- `RootLogContext`
- `AdditionalLogContext`
- `LogConfigFile`
- `LogLevel`
- `MessageBundleBase`
- `LanguageCode`
- `CountryCode`

Using the Import-Export API

You can invoke the import-export utility directly from the API. The class name for the import-export API is `com.bea.wlai.client.ImportExport`. The following code provides an example for importing.

Listing 10-1 Sample Code for Importing

```
String exportFilename = "myExportFile.jar";
String appName = "MyApp";
String appRootDir = "d:\\apps\\MyApp";
boolean overwrite = true;
boolean publish = true;
List errors = new LinkedList(); // List of Exception objects
```

```
Map eventPropsMap = new HashMap();
// Add properties file names (String) as the values keyed by the
// qualified names of the connection factories the properties are
// to be used with. This list can be null if you don't want any
// mappings applied.
eventPropsMap.put("Folder1.AppView1_connectionFactory",
                "d:/myFiles/propsFor_Folder1.AppView1_connectionFactory");

ImportExport ie = new ImportExport(new File(exportFilename));

try
{
  ie.connect(appName, appRootDir);

  ie.importNamespaceObjects(overwrite, publish, eventPropsMap, errors);
}
catch (Exception e)
{
  System.err.println("Fatal errors encountered:\n");
  e.printStackTrace();
  return;
}

if (errors.size() > 0)
{
  System.err.println("Non-fatal errors encountered:\n");
  int pos = 1;
  Iterator i = errors.iterator();
  while (i.hasNext())
  {
    Exception e = (Exception) i.next();
    System.err.println("Error " + pos + ": " + e);
    e.printStackTrace(m_out);
    pos++;
```

```
    }
}
```

CHAPTER 11

Step 9: Upgrade Workflows

WebLogic Integration 8.1 provides an Upgrade Wizard for upgrading workflows and other application information developed in WebLogic Integration 2.1 SP2 and 7.0 SP2 to WebLogic Integration 8.1. This section contains the following topics:

About the Upgrade Wizard

The Upgrade Wizard automatically upgrades most workflows, including business process templates that use Data Integration, (DI), Application Integration (AI) and B2B plug-ins. Basically, the wizard takes an exported JAR file from WebLogic Integration 2.1 or 7.0 SP2 that contains workflows, business operations definitions, event keys, and data integration MFL, XSL, and XML files and transforms them into a business process file (`.jpd`), control files (`.jcx`), and extracted DI files. The Upgrade Wizard provides success, warning, error, and failure messages about the upgrade process and records this information to a log file.

The Upgrade Wizard optimizes the upgraded business processes as much as possible. However, upgraded processes may not be as optimal as if they were created in WebLogic Integration 8.1. Additionally, some processes cannot be fully upgraded, especially if they are extremely complex or improperly designed. These processes may require manual upgrading or redesign.

To learn about developing business processes, see *Designing WebLogic Integration Solutions* at the following URL:

```
http://edocs.bea.com/wli/docs81/design/index.html
```

and the Guide to Building Business Processes in the WebLogic Workshop Help at the following URL:

```
http://edocs.bea.com/workshop/docs81/doc/en/integration/wfguide/wfguideIntro.html
```

What the Wizard Upgrades

The following list provides an overview of what the Upgrade Wizard upgrades:

- Workflow (business process) definitions, including the following:
  - Flow chart, start, task, branch, condition, join/or, and done.
  - Process actions, such as assign task, post XML event, set variable, call sub workflow, and call business-operations.
  - Process event subscription, such as event start and event resume.
- Data integration MFL, XSL, and XML files used by workflows.

Upgrade Wizard Limitations

As previously mentioned, the Upgrade Wizard cannot upgrade some components of particular workflows. You will need to manually upgrade these parts. The following list contains information on the Upgrade Wizard limitation and references to documents that can help you re-develop these components.

- **Running applications**—You cannot upgrade a production system.
- **Unstructured parallel workflows**—WebLogic Integration 8.1 supports only structured parallel business processes (All parallel paths coming out of a node should merge at the same Join node). If your WebLogic Integration 2.1 or 7.0 SP2 application, contains unstructured parallel workflows (cross paths between potential threads and Join nodes as Task nodes), the Upgrade Wizard cannot upgrade these workflows and you will need to re-develop them. To learn about structured business process design, see *Designing WebLogic Integration Solutions* at the following URL:

  ```
  http://edocs.bea.com/wli/docs81/design/index.html
  ```

- **Workflows That Send Email**—After upgrading a workflow that sends email, you must manually edit the SMTP server settings in the annotation for email control the upgraded process. This is required because the SMTP server information is not contained in the original workflow, it is part of the domain configuration. To learn about editing the SMTP server settings, see Configuring an Email Control in the WebLogic Workshop at the following URL:

  `http://edocs.bea.com/workshop/docs81/doc/en/integration/controls/controlsEmailConfig.html`

- **Run-time exception if output parameter is not initialized**—In WebLogic Integration Studio, in the Variable Properties dialog box, if the Output parameter is checked, the callback to the client will include this variable as one of its arguments. However, if you do not initialize this variable, a run-time exception is thrown. Primitive type and String are exceptions to this rule, as they have a default value when declared.

- **WebLogic Integration 7.0 SP2 DI plug-in binary event messages**—The DI plug-in supports data translations between binary formats and XML using a message format language (MFL) document. The Upgrade Wizard upgrades workflows that can handle only XML based events even though the actual incoming event may consist of binary data. In this case, you need to do one of the following:

  – If the receiving WebLogic Integration 8.1 business process can determine the MFL for transforming the binary data after it receives the incoming message, it will translate the data to XML. See Transforming Non-XML Data in the WebLogic Workshop Help at the following URL:

    `http://edocs.bea.com/workshop/docs81/doc/en/integration/dtguide/dtguideNonXML.html`

  – If the receiving WebLogic Integration 8.1 business process cannot determine the MFL for transforming the incoming binary data, you can use Java or another business process to utilize MFL to convert the message to XML before sending the binary data. See the Guide to Building Business Processes in the WebLogic Workshop Help at the following URL:

    `http://edocs.bea.com/workshop/docs81/doc/en/integration/wfguide/wfguideIntro.html`

- **Custom business operation classes**—For upgrading workflows with business operations, you need to import any custom business operation classes into your upgrade project's *HOME_BEA*`/weblogic81/<upgrade_project>/WEB-INF/classes` folder or, for JAR files, the `WEB-INF/lib` folder, where *BEA_HOME* represents the WebLogic Platform home directory.

- **Workflows using the same Application View service synchronously and asynchronously**—You need to manually change the WebLogic Integration 2.1 SP2 or 7.0 SP2 workflow to make the service be used either synchronously or asynchronously before running the Upgrade Wizard.
- **Multiple Start Nodes**—The Upgrade Wizard cannot separate the different possible threads of workflow execution when multiple start nodes exist. You need to manually upgrade these workflows.
- **Workflows node without inbound connection**—Normally, the only workflow node that does not have an inbound connection is the Start node. If any other node in a workflow lacks an inbound connection, the workflow cannot be upgraded by the Upgrade Wizard, as shown in the following figure:

- **Timeouts**—During upgrade, timeouts are set to have a count of at least one second; any timeouts that are set to less than a second are reset to one second. If you require less than one second, use the `sleep()` function after upgrading.
- **Session EJB and Entity EJB variables**—Although WebLogic Integration 7.0 SP2 workflow variables of type Session EJB and Entity EJB are upgraded to WebLogic Integration 8.1 EJB controls, these variables cannot be passed between business processes in WebLogic 8.1. In WebLogic 7.0 SP2, these variables could be passed between workflows.
- **Timed starts**—When upgrading a workflow that uses a timed start, the upgraded business process contains a Subscription start node. Subscription start nodes start processes on receipt of a message from a Message Broker channel. Subsequently, you need to configure the channel to have a Timer event generator that publishes to the Subscription start node using the WebLogic Integration Administration Console. See Event Generators in *Managing WebLogic Integration Solutions* at the following URL:

  `http://e-docs.bea.com/wli/docs81/manage/evntgen.html`

- **Transaction boundaries**—To allow the transaction of the caller to propagate into subprocesses, the Upgrade Wizard introduces the **Client Request with Return** construct into any upgraded business process that has a called start. Other than this contsruct, the Upgrade Wizard does not deliberately set transaction boundaries in upgraded processes. However, transactional boundaries can sometimes be different in an upgraded business process for the following reasons:

  - Whenever a parallel construct is found, boundaries change because each branch is its own transaction block.
  - Called business processes have a **Client Request with Return** that gets as close as possible to the quiescent point in the original workflow and still maintains structured logic.
  - Upgraded nodes are inherently different because WebLogic Integration 8.1 uses controls and WebLogic Integration 7.0 SP2 BPM uses actions in. This incongruence may introduce slight transactional boundary changes.

- **Addressed messaging**—To enable upgrading of addressed messaging, you must add a comment in the original subscribing workflow's Notes field. To do this, include the word *addressedTarget* in the Notes field for either the workflow Notes (Right-click the template definition in the workflow tree in WebLogic Integration 7.0 SP2 Studio.) or the Event node (under the properties for the Event node). Enabling addressed messaging at the workflow level is the same as enabling it individually for every event node in the workflow.

Using the Upgrade Wizard

Before using the Upgrade Wizard, you must first export the workflows you created in WebLogic Integration 2.1 SP2 or 7.0 SP2 as described in "Step 3: Export Package File from WebLogic Integration BPM Studio" on page 5-1.

To upgrade your workflows, take the following steps:

1. Start WebLogic Workshop and WebLogic Server, as described in How Do I: Start WebLogic Workshop? in the WebLogic Workshop Help at the following URL:

   ```
   http://edocs.bea.com/workshop/docs81/doc/en/integration/howdoI/howStartWorkshop.html
   ```

2. Open the Process Application for your upgrade that you created in "Step 1: Create Application For Upgrade" on page 3-1.

3. From the WebLogic Workshop menu, choose **Tools→WebLogic Integration→Upgrade Processes**.

   The Upgrade Wizard is displayed.

4. In Upgrade Wizard - Step1: Select Process:

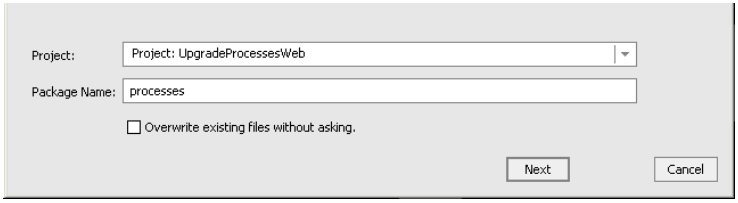
   a. In the **Select a JAR File** field, enter the JAR that contains a set of exported workflows from a previous version of WebLogic Integration. Alternatively, click **Browse** to navigate to the JAR.

b. In the **Select Workflows to Upgrade** field, select one or more workflows from the list of workflows.

   Note: If multiple workflow template definitions exist, you must choose which template definition you want to upgrade. The Upgrade Wizard allows only a single definition to be used for creating the JPD.

c. In the Project drop list, select the project for your upgraded workflows. It should be the name of your application with *Web* appended. For example, *UpgradeProcessesWeb*. See the following figure.

Figure 11-1 Upgrade Wizard - Step 1: Select Process

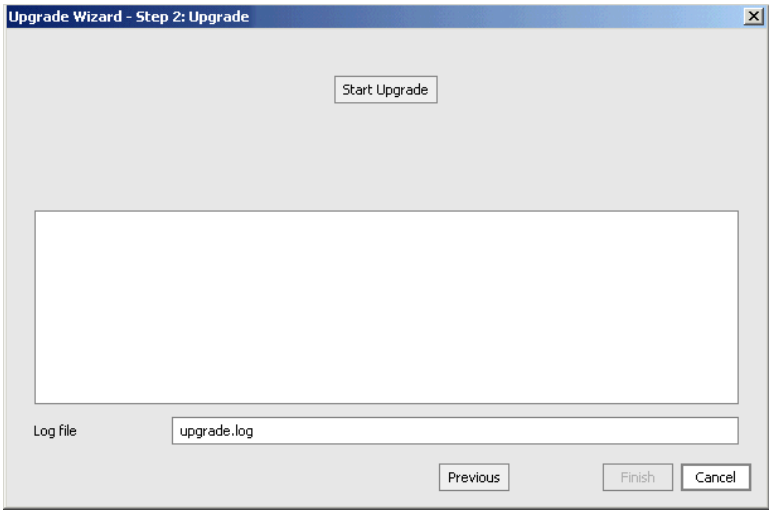


d. In the **Package Name** field, enter the name "processes" (which is the location of the `default.jpd`).

e. Click **Next**.

   The Upgrade Wizard - Step2: Upgrade is displayed, as shown in the following figure.

Figure 11-2 Upgrade Wizard - Step 2: Upgrade

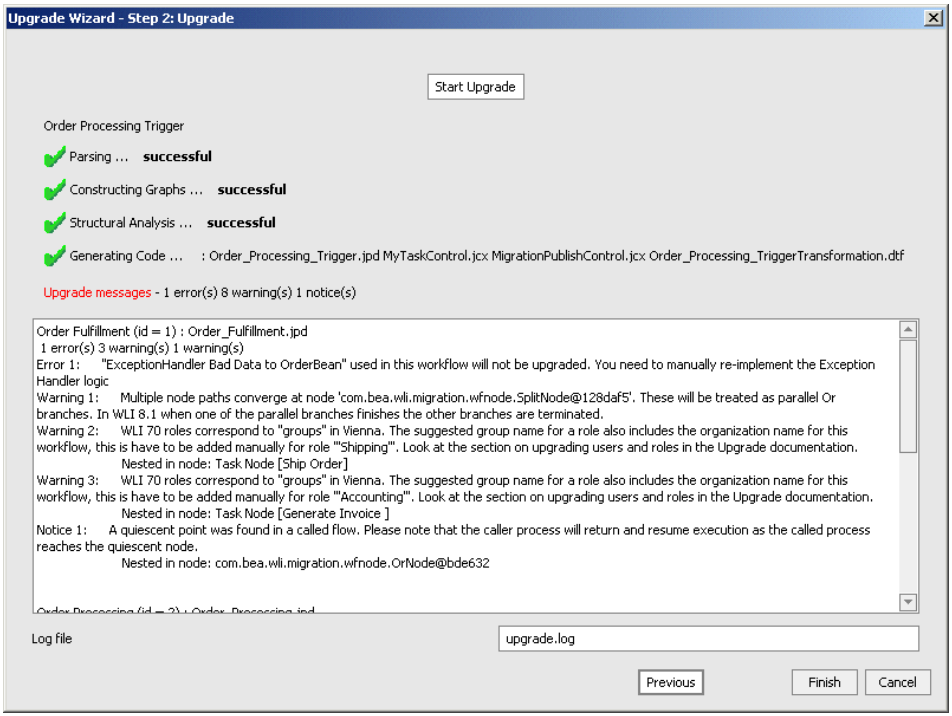


5. If you do not want to use the default name for the upgrade log, enter a name in the **Log file** field.

6. Click **Start Upgrade**.

   Messages indicating the status of the business process upgrade are displayed in the Upgrade Wizard. These messages indicate which steps in the upgrade process succeed and which steps fail; whether or not the JPD file is generated; the number or errors, warnings, and notices; and the type of errors, warnings, and notices, as shown in the following figure:

Figure 11-3 Upgrade Messages

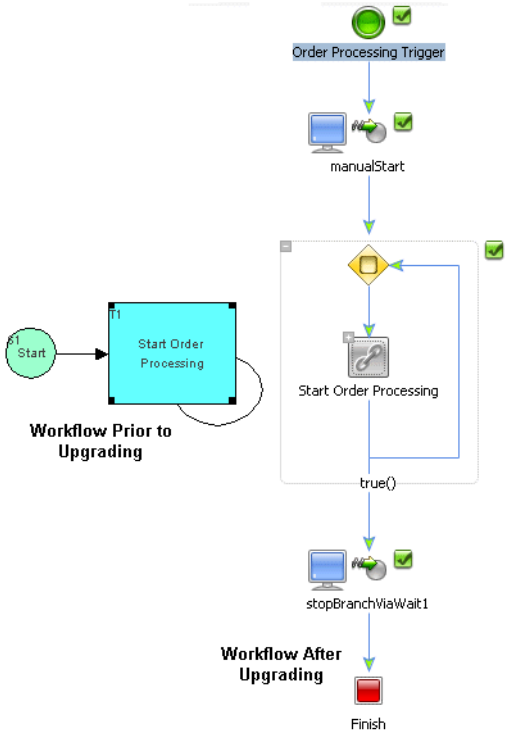


7. Click **Finish**.

   The Upgrade message field displays the results of the upgrade. The errors, warnings, failures, and notices are also recorded in the upgrade log. To learn more about using these messages, see "Step 10: View the Upgrade Log" on page 12-1.

   A **Design View** of the generated business process is displayed and the upgraded workflows (now called business processes) are saved in the directory specified the **Package Path** field, as shown in the following figure.

Figure 11-4 Original Workflow and Upgraded Workflow



Note: Some nodes may not display properly.

8. To view the source code of the generated business process, select **Source View**.

Note: Some nodes in business processes generated by the Upgrade Wizard may not display properly in the **Design View**. However, these parameters are visible in the source view.

Step 10: View the Upgrade Log

The upgrade log helps you with upgrading workflows. You can use the upgrade log to identify the workflows and parts of workflows that could not be upgraded by the Upgrade Wizard. The messages in the log provide guidance for fixing, redeveloping, and optimizing the upgraded business processes. In general, they provide the following:

- Notices indicate successful upgrade and identifies quiescent nodes.
- Warnings indicate success, but you may need to modify a part of a workflow or otherwise modify it to improve its functionality.
- Errors require minor to moderate fixes to the business process.
- Errors with a failure to produce a JPD file means you must fix the original workflow or redevelop it in WebLogic Integration 8.1.

The following steps will help you use the upgrade log most effectively:

1. To view the log, select `upgrade.log` in the **Application** pane. The log is in the `Project/ProjectWeb/processes` folder.
2. Examine the upgrade log and use it to identify the workflows and parts of workflows that could not be upgraded. The log also provides information about the upgraded business process that you can you use to improve their performance.
3. Consult the "Upgrade Wizard Limitations" on page 11-2 section of the Upgrade Wizard for information about specific limitations and solutions.
4. If an entire workflow cannot be upgraded, you need to redevelop that workflow (business process) in WebLogic Integration 8.1.

5. If a part of a workflow cannot be upgraded, you can add the missing functionality to that workflow (business process) using WebLogic Integration 8.1.

To learn about creating and modifying business processes, see Building Integration Applications in the WebLogic Workshop Help at the following URL:

```
http://edocs.bea.com/workshop/docs81/doc/en/integration/navIntegration.
html
```

To learn about optimizing business processes and your application, see *Designing WebLogic Integration Solutions* at the following URL:

```
http://edocs.bea.com/wli/docs81/design/index.html
```

Step 11: Run and Test Upgraded Business Processes

To learn about running and testing business processes, see Running and Testing Your Business Process at the following URL:

```
http://edocs.bea.com/workshop/docs81/doc/en/integration/wfguide/wfguideTest.html
```

You may be able to further optimize the upgraded business processes. To learn about improving the efficiency of business processes, see Designing WebLogic Integration Solutions at the following URL:

```
http://edocs.bea.com/wli/docs81/design/index.html
```

CHAPTER 14

Upgrading Security Features

When upgrading from WebLogic Integration 2.1 SP2 or 7.0 SP2 to 8.1, security features must be upgraded manually. This section contains information about these procedures:

- WebLogic Server Security Upgrade
- WebLogic BPM Security Upgrade
- WebLogic B2B Security Upgrade
- WebLogic Application Integration Security Upgrade

WebLogic Server Security Upgrade

WebLogic Server ACLs, users, groups, certificates, and so on must be upgraded by following the Security section of the *WebLogic Server 8.1 Upgrade Guide* at the following URL:

```
http://edocs.bea.com/wls/docs81/security.html
```

WebLogic Integration 8.1 uses the Default Security Configuration in WebLogic Server 8.1. For more information refer to Managing WebLogic Security.

WebLogic BPM Security Upgrade

Upgrading WebLogic business process management (BPM) security affects users, roles, organizations, calendars, email, and permissions.

WebLogic BPM Users, Roles, and Organizations

WebLogic business process management (BPM) security upgrades to users are handled separately from roles and organizations.

WebLogic BPM Users

All WebLogic BPM users must become WebLogic Server users.

User `wlisystem` is no longer a special user. The following table shows its replacement in WebLogic Integration 8.1.

Table 14-1 WebLogic Integration 2.1 SP2 and 7.0 SP2 wlisystem User vs 8.1 Functionality

WebLogic Integration 2.1 and 7.0 SP2 wlisystem User	Replacement in WebLogic Integration 8.1
The wlisystem user was used when an event or trigger invoking a workflow had no associated user.	A business process started by a message will be run as user `anonymous` by default, unless the business process defines the `<run-as>` attribute. Business processes that have been migrated using the upgrade wizard will use `<run-as> wlisystem` as the running user.

WebLogic BPM Roles and Organizations

The following table shows the replacement in WebLogic Integration 8.1 for BPM roles and organizations.

Table 14-2 WebLogic Integration 2.1 SP2 and 7.0 SP2 BPM Roles and Organizations vs 8.1 Functionality

WebLogic Integration 2.1 SP2 and 7.0 SP2 Roles and Organizations	Replacement in WebLogic Integration 8.1
Users had roles and were assigned to organizations.	A BPM role and organization are combined and mapped to a WebLogic Server group. **Note:** The concept of an organization no longer exists.

WebLogic BPM Calendars and Email

WebLogic Integration BPM Calendars and Email have been replaced with new functionality. The following table shows the change in WebLogic Integration 8.1 for BPM calendars and email.

Table 14-3 WebLogic Integration 2.1 SP2 and 7.0 SP2 BPM Calendars and Email vs 8.1 Functionality

WebLogic Integration 2.1 SP2 and 7.0 SP2 BPM Calendars and Email	Replacement in WebLogic Integration 8.1
Calendars existed at the organization, role, and user levels.	Calendars now exists only at the user level, and are configured via the WebLogic Integration Administration Console. Email addresses for users must be reentered via the WebLogic Integration Administration Console.

WebLogic BPM Permissions

Permissions in WebLogic Integration 8.1 are set via the WebLogic Integration Administration Console. The following table shows the change in WebLogic Integration 8.1 for BPM permissions.

Table 14-4 WebLogic Integration 2.1 SP2 and 7.0 SP2 BPM permissions vs 8.1 Functionality

WebLogic Integration 2.1 SP2 and 7.0 SP2 BPM permissions	Replacement in WebLogic Integration 8.1
Permissions were set for users and roles.	Permissions are defined using specially named roles that must be configured via the WebLogic Integration Administration Console. The following permissions should be associated with the `admin` role: • ConfigureSystems • ConfigureComponents • MonitorInstances • CreateTemplate • DeleteTemplate • AdministerUser **Note:** The ExecuteTemplate permission has been replaced by the security policy on business process methods.

WebLogic B2B Security Upgrade

Upgrading WebLogic B2B security affects certificates, trading partner configuration, and the packaging of some Java classes you may be using.

For WebLogic Integration 8.1, the B2B system user is no longer used. Instead you will use the users and roles provided when you create a new WebLogic Integration domain.

Certificates must be placed in keystores before they can be upgraded. In WebLogic Integration 2.1 SP2 keystores were not available. In WebLogic Integration 7.0 SP2 the use of keystores was optional.

Upgrading Certificates in WebLogic Integration 2.1 SP2

Certificates used by WebLogic Integration 2.1 SP2 B2B must be imported into the WebLogic Integration 8.1 keystore one at a time by using a JavaSoft JDK `keytool` utility, or the WebLogic

ImportPrivateKey utility as described in "ImportPrivateKey" in the Using the WebLogic Java Utilities section of the *WebLogic Server Administration Guide* at the following URL:

```
http://edocs.bea.com/wls/docs70/adminguide/utils.html
```

and in the Configuring the Keystore section of the *WebLogic Integration 7.0 B2B Security Guide* at the following URL:

```
http://edocs.bea.com/wli/docs70/b2bsecur/keystore.htm
```

Upgrading Certificates in WebLogic Integration 7.0 SP2

To upgrade certificates used by WebLogic Integration 7.0 SP2 B2B to 8.1, your certificates must be in a private keystore and your trusted certificate authorities must be in the CA keystore. If you have not yet set up these keystores, follow the instructions in the Configuring the Keystore section of the *WebLogic Integration 7.0 B2B Security Guide*.at the following URL:

```
http://edocs.bea.com/wli/docs70/b2bsecur/keystore.htm
```

The main steps are:

1. Generate and configure the private and CA keystores.
2. Specify the password for the keystores.
3. In WebLogic Integration 7.0 SP2, enable the auto-migrate mode to allow bulk loading of certificates into the keystore.

After your certificates are in keystores, you can upgrade to WebLogic Integration 8.1 following these steps:

1. Copy the keystore files to another location.
2. Configure the WebLogic Server keystore to use the new keystore files. Note that in WebLogic Integration 8.1, clustered keystore configuration is node-specific. If you have clustered nodes, they will need to access a shared directory containing the keystores, or the keystores must be replicated on each node.
3. Enter the primary key password using the Trading Partner Management (TPM) console. Note that the TPM must be running.

Upgrading Trading Partner Security Configuration

A script is provided to upgrade your trading partner security and message encryption configuration.

On Windows, run:

```
BEA_HOME/weblogic81/integration/upgrade/upgradeTPM.cmd
```

On UNIX, run:

```
BEA_HOME/weblogic81/integration/upgrade/upgradeTPM.sh
```

In these commands, *BEA_HOME* represents the WebLogic Platform home directory.

Upgrading Use of com.bea.b2b.security Classes

You will need to change and recompile your applications that use the com.bea.b2b.security package. The following table shows the changes in class names:

Table 14-5 WebLogic Integration 2.1 SP2 and 7.0 SP2 Classes vs 8.1 Classes

WebLogic Integration 2.1 SP2 and 7.0 SP2 Classes	Replacement in WebLogic Integration 8.1
com.bea.b2b.security.AuditLogger	com.bea.wli.security.audit.AuditLogger
com.bea.b2b.security.CertificateVerification	com.bea.wli.security.verification.CertificateVerification
com.bea.b2b.security.Timestamp	com.bea.wli.security.time.Timestamp

WebLogic Application Integration Security Upgrade

Upgrading WebLogic Application Integration security affects EIS authentication and authorization, and application view access control.

Repackaging Adapter Code

Java classes for adapters must conform to a new package scheme. A script is provided to repackage your adapter code.

On Windows, run:

```
BEA_HOME/weblogic81/integration/upgrade/aiRepackageAdapter.cmd
```

On UNIX, run:

```
BEA_HOME/weblogic81/integration/upgrade/aiRepackageAdapter.sh
```

In these commands, *BEA_HOME* represents the WebLogic Platform home directory.

Upgrading Application View Access Control

The security information for WebLogic Application Integration is no longer held in ACL format. Instead, a role-based authorization scheme uses the underlying WebLogic Server 8.1 security infrastructure. Go to the Application Integration section of the WebLogic Integration 8.1 console to reconfigure the security information to access the application view.

CHAPTER 15

Upgrading Application View Controls Created in WebLogic Workshop

In WebLogic Workshop 7.0 SP2, you could create Java Web Services (JWS) that used Application View controls for accessing enterprise systems through a J2EE Connector Architecture adapter. In WebLogic Integration 8.1, these controls have been completely restructured, and the API has changed. Therefore, you will need to manually upgrade your Application View controls, as described in the following steps:

1. Make sure your Application Views and other artifacts (Application View descriptors, connection factory descriptors, and schemas) have been imported into a WebLogic Integration 8.1 application using the import-export utility. To learn about this utility, see "Step 8: Import Application Views and Other AI Repository Artifacts" on page 10-1.

2. Copy the JWS file from your WebLogic Workshop 7.0 SP2 project and perform any conversion needed for the JWS files. To learn about upgrading JWS files, see "Run the jwsUpgrade Command-line Tool" in Upgrading Workshop Applications in the WebLogic Workshop Help at the following URL:

   ```
   http://edocs.bea.com/workshop/docs81/doc/en/workshop/guide/migration
   /conMigratingWorkshopApplications.html
   ```

3. Remove all member variable declarations for Application View controls from the JWS file and record the variable names and their associated Application Views. You will need this information later.

   For example, in a file called `DMBS1Service.jws`:

   ```
   public class DBMS1Service
   {
       /**
        * @jws:control
   ```

```
*/
private DBMS1Control m_dbms;
/** @jws:context */
JwsContext context;
...
```

a. Remove the member variable declaration.

```
    /**
     * @jws:control
*/
private DBMS1Control m_dbms;
```

b. Write down the variable name.

```
m_dbms
```

For example, in the `DBMS1Control.ctrl` file:

```
import weblogic.jws.control.ApplicationViewControl;
/**
* @jws:av-identity name="DBMS.DBMS1" user-id="system"
password="password"
*/
```

c. Write down the name of the application:

```
DBMS.DBMS1
```

4. Create an Application View control in WebLogic Workshop 8.1 for all Application Views you recorded. When providing the variable name for the control, use the variable name you recorded previously. This creates the control instances and links them to your JWS as variable declarations.

   Note: To learn about creating an Application View control, see Creating a New Application View Control in the WebLogic Workshop Help at the following URL:

   `http://edocs.bea.com/workshop/docs81/doc/en/integration/controls/controlsAppViewCreate.html`

5. Examine your JWS code and change the places where the Application View control variables are used, using the following guidelines:

   Note: A control created in WebLogic Workshop 7.0 SP2 has a `.ctrl` extension, while a control created in WebLogic Workshop 8.1 has a `.jcx` extension.

   – In WebLogic Integration 7.0, the request/response Java types for the control were generated as static inner classes within the control's CTRL file. For example, for a control named `DBMS1Control`, a service called `GetAllCustomers`, and a response schema describing the following document:

```
<Rows>
  <Row>
    <FIRSTNAME>Joe</FIRSTNAME>
    <LASTNAME>User</LASTNAME>
  </Row>
</Rows>
```

The inner class within the `DBMS1Control.ctrl` file is as follows:

```
public static class GetAllCustomersResponse
  implements java.io.Serializable
{
    public static class Rows implements java.io.Serializable
    {
        public Row[] Row;
    }

    public static class Row implements java.io.Serializable
    {
        public java.lang.String FIRSTNAME;
        public java.lang.String LASTNAME;
    }
}
```

Additionally, code would exist in the JWS to get the first and last names from the response rows as follows:

```
DBMS1Control.GetAllCustomersResponse.Rows rows =
  m_dbms.GetAllCustomers();
DBMS1Control.GetAllCustomersResponse.Rows.Row[] custs =
  rows.Row;

for (int i=0; i < custs.length; i++)
{
  DBMS1Control.GetAllCustomersResponse.Row cust = custs[i];
  String firstName = cust.FIRSTNAME;
  String lastName = cust.LASTNAME;
  System.out.println("First name=" + firstName + "last name=" +
                     lastName);
}
```

Note: The variable name `m_dbms` refers to the Application View control instance `DBMS1Control`.

– In WebLogic Integration 8.1, the same functionality is achieved using an XBean type. No static inner classes are generated inside the CTRL file. Instead, the XBean type is compiled directly from the schema for the request/response. This makes it available to all code within the application, not only the JWS. In WebLogic Integration 8.1, you

would have a control instance generated from the DBMS1 Application View called `DBMS1.jcx` as follows:

```
public interface DBMS1 extends ApplicationViewControl
{
    public wlai.dbms1GetAllCustomersResponse.RowsDocument
 GetAllCustomers()
        throws Exception;

}
```

and the code in the JWS (or a business process JWF file) that looks like this:

```
wlai.dbms1GetAllCustomersResponse.RowsDocument response =
  m_dbms.GetAllCustomers();
wlai.dbms1GetAllCustomersResponse.Rows.Row[] custs =
  response.getRows().getRowArray();

for (int i=0; i < custs.length; i++)
{
  wlai.dbms1GetAllCustomersResponse.Rows.Row cust = custs[i];
  String firstName = cust.getFIRSTNAME();
  String lastName = cust.getLASTNAME();
  System.out.println("First name=" + firstName + " last name=" +
                     lastName);
}
```

As you may have noted, the code is similar in both the WebLogic Integration 7.0 and WebLogic Integration 8.1 JWS files. However, the class/interface names have changed. In general, the name for the XBean type for the request is structured as follows:

```
wlai.<mangled namespace URI name>.<decapitalized AppView
name><Service name>.<Root element name>Document
```

Mangling replaces slashes with dots, and changes the first character of each qualifier to lower case.

The `mangled namespace URI name` equals the qualified name of the namespace that contains the schema in the application integration (AI) repository. For instance, `Folder1.Folder2.Schema1` uses `Folder1/Folder2` as the URI. The mangled name is `folder1.folder2`. The mangled lower-cased AppView name equals the lower case first letter of the AppView name.

Note: WebLogic Workshop provides context-sensitive code help that will guide you in using the XBean types correctly.

CHAPTER 16

Upgrading Utility Adapters

The Utility adapters, Email and File, in WebLogic Integration 2.1 SP2 and 7.0 SP2, have been replaced in WebLogic Integration 8.1 with system features. Consequently, these two adapters cannot be upgraded and you will need to replace their implementation with the new features.

Note: The Adapter for RDBMS had been replaced by a new RDBMS Adapter for 8.1 and has new capabilities.

Table 16-1 shows the WebLogic Integration 2.1 and 7.0 SP2 adapters and their replacement in WebLogic Integration 8.1.

Table 16-1 WebLogic Integration 2.1 and 7.0 SP2 Utility Adapter vs 8.1 Functionality

WebLogic Integration 2.1 and 7.0 SP2 Adapters	Replacement in WebLogic Integration 8.1
Adapter for Email	Email Control (send e-mail) Email Event Generator (receive e-mail)
Adapter for File	File Control (read, write, or append files) File Event Generator (receive file from polled directory)
Adapter for RDBMS	Adapter for RDBMS 8.1

Use Email Controls and Event Generators Instead of Adapter for Email

Email controls enable WebLogic Integration business processes to send e-mails to specific destinations. To learn more about Email controls, see Email Control in the WebLogic Workshop Help at the following URL:

```
http://edocs.bea.com/workshop/docs81/doc/en/integration/controls/controlsEmail.html
```

Business processes use Email event generators to read e-mail messages from a Post Office Protocol (POP3) or Internet Message Access Protocol (IMAP) account on a mail server and publish the contents to Message Broker channels. To learn more about Email event generators, see Using Event Generators to Publish to Message Broker Channels in the WebLogic Workshop Help at the following URL:

```
http://edocs.bea.com/workshop/docs81/doc/en/integration/controls/controlsBrokerEventGenerators.html
```

Use File Controls and Event Generators Instead of the Adapter for File

File controls read, write, or append to a file in a file system. The files can be one of the following types: `XmlObject`, Binary (raw data), or `String`. In addition, the File control supports file operations such as copy, rename, and delete. Typically, you use these operations to manipulate large files, without having to process them in any way. To learn more about File controls, see File Control in the WebLogic Workshop Help at the following URL:

```
http://edocs.bea.com/workshop/docs81/doc/en/integration/controls/controlsFile.html
```

File event generators poll for files in file systems and publish the files to Message Broker channels. Business processes use File event generators to receive events—for example, a business process receives an event when a file appears in a file system that is polled by a File event generator. To learn more about File event generators, see Using Event Generators to Publish to Message Broker Channels in the WebLogic Workshop Help at the following URL:

```
http://edocs.bea.com/workshop/docs81/doc/en/integration/controls/controlsBrokerEventGenerators.html
```

New Adapter for RDBMS 8.1

The Adapter for RDBMS 8.1 provides a similar user experience to the previous WebLogic Integration RDBMS adapters. However, several important features have been added, including the following:

- Support for stored procedure calls.
- Multiple mechanisms for event handling, including triggering, shadow tables, and *destructive gets* (where the table that is polled for events is a staging table in which rows are removed after they are processed).
- Metadata browsing is now available from the WebLogic Integration – Application Integration Design Console.
- Support for all JDBC (Java Database Connectivity) native types, including BLOB (Binary Large Object) and CLOB (Character Large Object Block).

To learn about the Adapter for RDBMS, see BEA WebLogic Adapters 8.1 at the following URL:

```
http://edocs.bea.com/wladapters/docs81/index.html
```

CHAPTER 17

Upgrading an Adapter Development Project

This section describes the steps for converting a WebLogic Integration 7.0 SP2 Adapter Development Kit (ADK) adapter development tree to an 8.1 ADK adapter development tree.

1. Implement WebLogic Integration Resource Adapter by extending `AbstractWLIResourceAdapter` using *`BEA_HOME`*`/weblogic81/integration/adapters/sample/src/sample/spi/ResourceAdapterImpl.java` as a guide, where *`BEA_HOME`* represents the WebLogic Platform 8.1 home directory. Keep the following in mind as you implement the adapter.
   - The `AbstractResourceAdapter` class defines the properties needed for the generic adapter object that will hold both the inbound and outbound adapter sections. For use in this situation, this property set is the set of properties you have in your `Event Router web.xml` without the ADK *standard* properties.
   - Generally, if the sample adapter's `ResourceAdapterImpl` does not have a particular property from your `event router web.xml` *and* the property is not specific to your adapter, do not include it on the `ResourceAdapterImpl` class.
   - You must have a setter/getter pair of methods for each property on the `ResourceAdapterImpl` class.
   - Using *`BEA_HOME`*`/weblogic81/integration/adapters/sample/src/wli-ra.xml` as a guide, add the `wli-ra.xml` file to your `<adapter root>/src` directory (for example, `MyAdapter/src/wli-ra.xml`). This file has the format of the Connector 1.5 draft specification and describes the `ResourceAdapterImpl` class you are creating.

     To learn about this specification, see JSR 112: J2EE Connector Architecture 1.5 in the Java Community Process (JCP) program at the following URL:

```
http://jcp.org/en/jsr/detail?id=112
```

2. The following steps prepare your WebLogic Integration 7.0 SP2 adapter properties file to work in the build scheme for WebLogic Integration 8.1. The new scheme separates the adapter-specific properties from the ADK-defined properties into separate files. These files are merged at build time. See step 5 below for the build changes.

   a. Rename the adapter properties file named `<Adapter Logical Name>.properties` to `<Adapter Logical Name>-base.properties` (for example, `MyAdapter_1_0.properties` is renamed to `MyAdapter_1_0-base.properties`).

   b. In the renamed file, remove all properties not related to the JSP pages in your adapter's source tree. For example remove any properties starting with `depappvw_` because the `depappvw.jsp` exists in the ADK, not your adapter.

   c. Remove all properties not related to the run-time messages that your adapter explicitly defines. The rule of thumb is that if the property or message exists in `ADK.properties` do not include it in your adapter properties file.

3. The following steps prepare your WebLogic Integration 7.0 SP2 design-time Web application's web descriptors (`web.xml`, `weblogic.xml`) to work in WebLogic Integration 8.1 build scheme. The new scheme separates the adapter-specific information in these descriptors into separate files from the generic (ADK or WebLogic Server defined) information. The new scheme generates the `web.xml` and `weblogic.xml` based on a simple properties file that you provide in your adapter source. See step 5 below for the build changes.

   a. Create a new `.properties` file called `web-gen.properties` in your WebLogic Integration 8.1 `<adapter root>/src/war/WEB-INF` directory. The properties that go into this file are described in the steps that follow. Basically, these properties will be derived from the contents of the `web.xml` file in the same directory.

   b. Add a property called `display-name`, and give it the value you used for the `display-name` of you `web.xml` descriptor. For example, the following display-name element in the `web.xml`:

      ```
      <web-app>
      <display-name>BEA_WLS_SAMPLE_ADK</display-name>
      …
      ```

      becomes

      ```
      display-name=BEA_WLS_SAMPLE_ADK
      ```

   c. Add a property called `version`, and give it the value of the `context-param` element in `web.xml` with param-name of `version`.

d. Add a property called `request-handler-class` and give its value as the class name for your design-time request handler. This value can be found in the existing `web.xml` in the controller servlet definition's `RequestHandlerClass init-param`. It looks like this in `web.xml`:

```
<!-- Controller servlet -->

<servlet>
  <servlet-name>controller</servlet-name>
  <servlet-class>com.bea.web.ControllerServlet</servlet-class>

  <init-param>
    <param-name>MessageBundleBase</param-name>
    <param-value>BEA_WLS_SAMPLE_ADK</param-value>
    <description>…</description>
  </init-param>

  …

  <init-param>
    <param-name>RequestHandlerClass</param-name>
    <param-value>sample.web.DesignTimeRequestHandler</param-value>
   <description>Class that handles design time requests</description>
  </init-param>

  …
  <load-on-startup>1</load-on-startup>

</servlet>
```

e. Add a property called `adapter-logical-name` and give its value as the logical name for your adapter (for example, `MyAdapter_1_0`).

f. Add a property called `debug-setting` and give it the value `on` or `off`. This should be the same value as the value in your controller servlet's `init-params` in the old `web.xml`.

g. Add a property called `extra-jsp-list` that lists the JSP pages you have added for your adapter, above and beyond those defined in the ADK This list is a comma-separated list of *extra* JSPs. The standard JSPs are `addevent`, `addservc`, `confconn`, `edtevent`, `edtservc`, `event`, `service`, and `testform`; do *not* include them in the `extra-jsp-list`. For example if you add a JSP called `mybrowser.jsp`, your `extra-jsp-list` would look like the following:

```
extra-jsp-list=mybrowser
```

h. Delete your existing `web.xml` and `weblogic.xml` descriptors. You can make a back-up copy of these files if you have customized them. Most adapters do not have custom `web.xml` and `weblogic.xml` descriptors.

4. Edit the `<adapter root>/src/ear/META-INF/application.xml` and remove the web module definition for the event router. For example, the text removed for the sample adapter is as follows:

```
<module>
  <web>
    <web-uri>BEA_WLS_SAMPLE_ADK_EventRouter.war</web-uri>
    <context-root>BEA_WLS_SAMPLE_ADK_EventRouter</context-root>
  </web>
</module>
```

5. Modify your `<adapter root>/build.xml`, as follows:

   Note: The following instructions assume that your `build.xml` is based on the `build.xml` provided with the ADK. If your `build.xml` in not based on the ADK's `build.xml` or you have heavily modified it, you will need to adjust these instructions to account for these differences.

   a. Eliminate the event router JAR and WAR.

   b. In `build.xml`, delete the `eventrouter_jar` and `eventrouter_war` targets and any references to them.

   c. Add `<your adapter package>/event/*.class` to the class `includes` definition of the JAR target. It looks like this:

```
...

<!--
From the adapter's source directory, include the "includes" list
For this adapter, all the classes in the sample/cci, sample/event
and sample/spi packages are included as well as the log configuration
file and message bundles
-->
<fileset dir='${SRC_DIR}'
         includes='sample/cci/*.class,sample/event/*.class,
                   sample/spi/*.class, sample/eis/*.class,
                   *.xml,*.properties'/>

...
```

   d. Add the following to the JAR command under your EAR target after the `zipfileset` element for the `adk.jar` file:

```
<zipfileset src='${WLI_LIB_DIR}/adk-eventgenerator.jar'>
  <exclude name='META-INF/MANIFEST.MF'/>
</zipfileset>
```

e. Add the following text to the top of your *packages* build target. This text calls ant tasks in the ADK that generate the web descriptors (`web.xml` and `weblogic.xml`) for your adapter. It also merges your adapter-specific properties file with the `ADK.properties` properties file to yield the final merged properties file required for proper operation of the adapter.

```
<!-- Generate web descriptors. NOTE: You can turn this off if you want
          to tightly control your web.xml/weblogic.xml. In this case,
        simply maintain these files in your src/war/WEB-INF instead of
          web-gen.properties -->
    <ant dir='${WLI_HOME}/adapters/utils/ant'
         target='generate_web_descriptors'
         inheritAll='false'>
      <property name='web_gen_props_file'
                value='${SRC_DIR}/war/WEB-INF/web-gen.properties'/>
    </ant>

    <!-- Merge the ADK.properties file and your adapter-specific
properties
        into the final properties file that will be used by the adapter
     -->
    <ant dir='${WLI_HOME}/adapters/utils/ant'
         target='merge_properties'
         inheritAll='false'>
      <property name='props_dir' value='${SRC_DIR}'/>
      <property name='adapter_props_file'
                value='BEA_WLS_DBMS_ADK-base.properties'/>
      <property name='target_props_file'
                value='BEA_WLS_DBMS_ADK.properties'/>
    </ant>
```

6. Remove the entire `<adapter root>/src/eventrouter` directory.

7. Change the `SYSTEM` identifier in `<adapter root>/src/rar/META-INF/weblogic-ra.xml` from

```
PUBLIC '-//BEA Systems, Inc.//DTD WebLogic 6.0.0 Connector//EN'
'http://www.bea.com/servers/wls600/dtd/weblogic600-ra.dtd'
```

to

```
PUBLIC '-//BEA Systems, Inc.//DTD WebLogic 8.1.0 Connector//EN'
'http://www.bea.com/servers/wls810/dtd/weblogic810-ra.dtd'
```

8. Change the `<shrink-period-minutes>` element in `<adapter root>/src/weblogic-ra.xml` to `<shrink-frequency-seconds>` and multiply the current element's value by 60.

   This completes upgrading your ADK.

Index

W