



BEA WebLogic Integration™

Tutorials for Trading Partner Integration

Version 8.1
July 2003

Copyright

Copyright © 2003 BEA Systems, Inc. All Rights Reserved.

Restricted Rights Legend

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

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

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

Trademarks or Service Marks

BEA, Jolt, Tuxedo, and WebLogic are registered trademarks of BEA Systems, Inc. BEA Builder, BEA Campaign Manager for WebLogic, BEA eLink, BEA Liquid Data for WebLogic, BEA Manager, BEA WebLogic Commerce Server, BEA WebLogic Enterprise, BEA WebLogic Enterprise Platform, BEA WebLogic Express, BEA WebLogic Integration, BEA WebLogic Personalization Server, BEA WebLogic Platform, BEA WebLogic Portal, BEA WebLogic Server, BEA WebLogic Workshop and How Business Becomes E-Business are trademarks of BEA Systems, Inc.

All other trademarks are the property of their respective companies.

Contents

About This Document

What You Need to Know	viii
e-docs Web Site	ix
How to Print the Document	ix
Related Information	ix
Contact Us!	x
Documentation Conventions	x

1. Introduction

About the Tutorials for Trading Partner Integration	1-1
Before You Begin	1-2
Setting Up the Tutorials	1-2
Step 1: Create a New WebLogic Integration Domain.	1-3
Step 2: View the Default Trading Partner Information.	1-4
Default Trading Partners	1-5
Default Protocol Bindings	1-5
Viewing Trading Partner Information (Optional)	1-6
Step 3: Install the Tutorial Files	1-9
Next Steps	1-10

2. Tutorial: Building ebXML Solutions

Before You Begin	2-1
Prerequisites	2-1

Suggested Reading	2-2
Tutorial Overview	2-3
Step 1: Getting Started	2-4
Creating the Business Process Application	2-5
Importing the Tutorial Sample Data	2-8
Importing the Tutorial Schemas	2-8
Creating the Read and Write Directories	2-9
Step 2: Sending an XML Message through an One-Way ebXML Exchange.	2-9
Building the Seller Business Process	2-10
Building the Buyer Business Process	2-17
Step 3: Selecting the Trading Partner Information Dynamically Through Typed XML	2-23
Building the SelectorSeller Business Process	2-24
Building the SelectorBuyer Business Process	2-27
Step 4: Sending Raw Data (Binary File) Through an ebXML Exchange.	2-32
Building the BinarySeller Business Process	2-32
Building the BinaryBuyer Business Process	2-33
Creating a File Event Generator.	2-36
Step 5: Creating a Roundtrip ebXML Conversation.	2-38
Building the RoundtripSeller Business Process	2-38
Building the RoundtripBuyer Business Process	2-42
Step 6: Implementing the Public/Private Pattern	2-46
Building the PublicBuyer Business Process	2-46
Building the PrivateBuyer Business Process	2-47
Building the PrivateSeller Business Process	2-48
Building the PublicSeller Business Process	2-49
Testing the Public/Private Pattern Example	2-51
Step 7: Using the TPM Control and Callbacks	2-51
Building the BuyerAlert Business Process	2-52

Testing the BuyerAlert Business Process	2-56
Step 8: Setting Partner ID Dynamically Based on Directory Name	2-56
Reviewing the Initiator Side of the Example	2-56
Reviewing the Participant Side of the Example	2-59
Step 9: Creating a Distributed Setup	2-61
Step 10: Configuring Non-Default Protocol Settings	2-62
Configuring the Participant Side	2-62
Configuring the Initiator Side.	2-63
Step 11: Exchanging Messages with WebLogic Integration - Business Connect.	2-64
Configuring WebLogic Integration - Business Connect with Trading Partner Information.	2-65
Configuring WebLogic Integration with Trading Partner Information	2-66
Setting Up a Trading Partner Service Profile	2-67
Sending a Message from WebLogic Workshop to WebLogic Integration - Business Connect	2-69

3. Tutorial: Building RosettaNet Solutions

Tutorial Goals	3-2
Before You Begin	3-2
Prerequisites	3-2
Suggested Reading	3-3
Note About Converting RosettaNet DTDs to W3C Schemas	3-4
Tutorial Overview	3-4
PIPs Implemented In These Examples.	3-4
Folders in the RosettaNet Tutorial Application	3-6
RosettaNet Design Patterns	3-7
Tutorial Steps.	3-8
Step 1: Open the RosettaNet Example Application	3-8

Step 2: Open the PIP 3B2: Advance Shipment Notice Example	3-9
About the PIP 3B2 Example	3-9
Components of the PIP 3B2 Example	3-10
Walkthrough of the Initiator Business Process	3-11
Walkthrough of the Participant Business Process	3-14
Walkthrough of the Private Business Processes	3-15
Running the PIP 3B2 Example	3-16
Step 3: Open the PIP 3A4: Purchase Order Request and Confirmation Example . .	3-16
About the PIP 3A4 Example	3-17
Components of the PIP 3A4 Example	3-17
Walkthrough of the Initiator Business Process	3-20
Walkthrough of the Participant Business Process	3-23
Walkthrough of the Private Business Processes	3-25
Running the PIP 3A4 Example	3-25
Implementing New PIPs Based on the Example PIPs	3-26
About Implementing New PIPs	3-26
Copying and Customizing PIP Implementations	3-26
Converting RosettaNet DTD Schemas to XSD Schemas	3-27

Index

About This Document

This document explains how to build trading partner integration solutions in BEA WebLogic Integration using the ebXML or RosettaNet business protocols.

This document covers the following topics:

- [Chapter 1, “Introduction,”](#) provides a brief overview of the ebXML and RosettaNet tutorials and setup instructions required to use the tutorials. It contains the following sections:
 - [About the Tutorials for Trading Partner Integration](#)
 - [Before You Begin](#)
 - [Setting Up the Tutorials](#)
 - [Next Steps](#)
- [Chapter 2, “Tutorial: Building ebXML Solutions,”](#) describes how to build ebXML business process examples by using ebXML controls and ebXML Participant Business Processes templates. It contains the following sections:
 - [Before You Begin](#)
 - [Tutorial Overview](#)
 - [Step 1: Getting Started](#)
 - [Step 2: Sending an XML Message through an One-Way ebXML Exchange](#)
 - [Step 3: Selecting the Trading Partner Information Dynamically Through Typed XML](#)
 - [Step 4: Sending Raw Data \(Binary File\) Through an ebXML Exchange](#)

- Step 5: Creating a Roundtrip ebXML Conversation
- Step 6: Implementing the Public/Private Pattern
- Step 7: Using the TPM Control and Callbacks
- Step 8: Setting Partner ID Dynamically Based on Directory Name
- Step 9: Creating a Distributed Setup
- Step 10: Configuring Non-Default Protocol Settings
- Step 11: Exchanging Messages with WebLogic Integration - Business Connect
- Chapter 3, “Tutorial: Building RosettaNet Solutions,” describes the components of two completed RosettaNet examples (PIP3B2 and PIP3A4) and suggests how to adapt the examples for other PIP implementations. It contains the following sections:
 - Tutorial Goals
 - Before You Begin
 - Tutorial Overview
 - Tutorial Steps
 - Implementing New PIPs Based on the Example PIPs

What You Need to Know

This document is intended for anyone wanting to learn how to integrate with trading partners via ebXML (Electronic Business using eXtensible Markup Language) or RosettaNet solutions using WebLogic Integration. For more information about these business protocols, see <http://www.ebXML.org> and <http://www.rosettanet.org>, respectively.

This document assumes a familiarity with the trading partner integration concepts described in *Introducing Trading Partner Integration*, which is available at <http://edocs.bea.com/wli/docs81/tpintro/index.html>.

In addition, before you use the tutorials in this document, you should have hands-on experience with building business processes in WebLogic Workshop. Therefore, you should complete the following tutorials in the WebLogic Workshop online help system:

- Tutorial: Building Your First Business Process, which is available at <http://edocs.bea.com/workshop/docs81/doc/en/integration/tutorial/tutWLIPProcessIntro.html>

- [Tutorial: Building Your First Data Transformation](http://edocs.bea.com/workshop/docs81/doc/en/integration/tutorial/tutWLIDataTransformIntro.html), which is available at <http://edocs.bea.com/workshop/docs81/doc/en/integration/tutorial/tutWLIDataTransformIntro.html>

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 WebLogic Integration documents contain information that is relevant to building trading partner integration solutions:

- *Introducing Trading Partner Integration*, which is available at <http://edocs.bea.com/wli/docs81/tpintro/index.html>.
- *Trading Partner Management* in *Managing WebLogic Integration Solutions*, which is available at <http://edocs.bea.com/wli/docs81/manage/tpm.html>.

For more information in general about ebXML, RosettaNet, and Java, refer to the following sources:

- ebXML Web Site at <http://www.ebXML.org>
- RosettaNet Web Site at <http://www.rosettanet.org>
- Sun Microsystems, Inc. Java site at <http://www.java.sun.com>

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.
<i>italics</i>	Indicates emphasis or book titles.

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

Convention	Item
. . .	<p>Indicates one of the following in a command line:</p> <ul style="list-style-type: none">• That an argument can be repeated several times in a command line• That the statement omits additional optional arguments• That you can enter additional parameters, values, or other information <p>The ellipsis itself should never be typed.</p> <p><i>Example:</i></p> <pre>buildobjclient [-v] [-o name] [-f file-list]... [-l file-list]...</pre>
.	<p>Indicates the omission of items from a code example or from a syntax line. The vertical ellipsis itself should never be typed.</p>

Introduction

This topic introduces the trading partner integration tutorials and describes the setup steps required before starting the tutorials. It contains the following sections:

- [About the Tutorials for Trading Partner Integration](#)
- [Before You Begin](#)
- [Setting Up the Tutorials](#)
- [Next Steps](#)

About the Tutorials for Trading Partner Integration

This document provides the following tutorials:

Table 1-1 Tutorials in This Document

Tutorial	Description	Business Protocol
Chapter 2, “Tutorial: Building ebXML Solutions”	Describes how to build the ebXML (Electronic Business using eXtensible Markup Language) business process examples.	http://www.ebXML.org
Chapter 3, “Tutorial: Building RosettaNet Solutions”	Describes the components of two completed RosettaNet examples (PIP3B2 and PIP3A4) and suggests how to adapt the examples for other PIP implementations.	http://www.rosettanet.org

These are standalone tutorials. You can complete both tutorials, if you want, but you can also complete just the ebXML tutorial without the RosettaNet tutorial, and vice versa.

Before You Begin

Before you begin using the trading partner integration tutorials:

- You must have WebLogic Platform 8.1 installed on your system. For more information, see *Installing WebLogic Platform*, which is available at <http://edocs.bea.com/platform/docs81/install/index.html>.
- It is recommended that you complete the following tutorials so that you know how to build business processes and create data transformations in WebLogic Workshop:
 - *Tutorial: Building Your First Business Process*, which is available at <http://edocs.bea.com/workshop/docs81/doc/en/integration/tutorial/tutWLIPProcessIntro.html>
 - *Tutorial: Building Your First Data Transformation*, which is available at <http://edocs.bea.com/workshop/docs81/doc/en/integration/dttutorial/tutWLIDataTransformationIntro.html>
- You should be familiar with basic trading partner integration concepts, as described in *Introducing Trading Partner Integration*, which is available at <http://edocs.bea.com/wli/docs81/tpintro/index.html>.
- You should first create the `tp tutorial` domain and install the tutorial files, as described in “Setting Up the Tutorials” on page 1-2.

Setting Up the Tutorials

Before you start using the trading partner integration tutorials, you need to complete the following steps:

- [Step 1: Create a New WebLogic Integration Domain](#)
- [Step 2: View the Default Trading Partner Information](#)
- [Step 3: Install the Tutorial Files](#)

Step 1: Create a New WebLogic Integration Domain

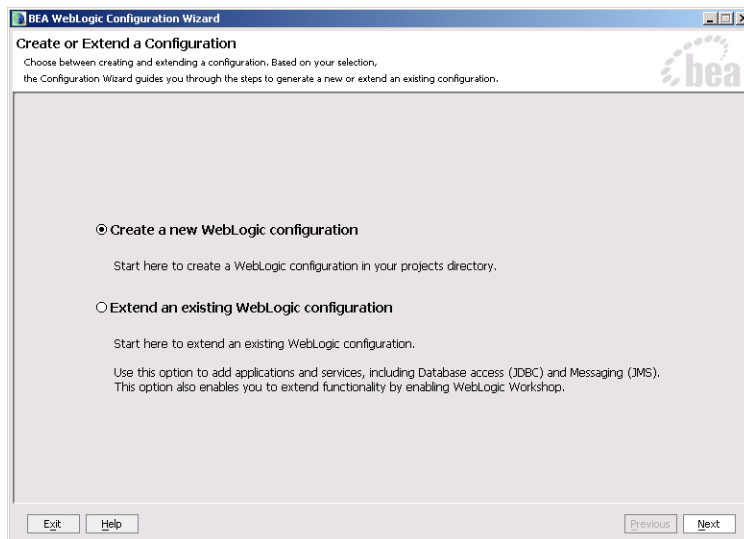
The trading partner integration tutorials require a WebLogic Integration domain that you must create using the BEA WebLogic Platform Configuration Wizard. The domain name used in this document is `tp tutorial`, but you can use any valid domain name you want.

Note: You can also use an existing WebLogic Integration for these tutorials. However, creating a new, separate domain ensures that the required default trading partner configuration is available for use with the tutorials.

To create a new WebLogic Integration domain:

1. Start the Configuration Wizard:
 - **Windows:** From the Start menu, choose **Programs**→**BEA WebLogic Platform**→**Configuration Wizard**.
 - **Unix:** Open a command shell, change to the `/common/bin` subdirectory of the product installation directory (such as `/bea/weblogic81/common/bin`), and enter the following command: `sh config.sh`.

The Configuration Wizard starts and displays the initial screen.



2. In the Configuration Wizard, make the following selections in each screen, and then click the Next button to continue:

Note: These instructions assume mostly default selections. For more information about advanced configuration options, see the Configuration Wizard online help.

Table 1-2 Selections in the Configuration Wizard

Screen Name	Recommended Selection
Create or Extend a Configuration	Create a new WebLogic configuration (default)
Select a Configuration Template	Basic WebLogic Integration Domain (from the list of WebLogic configuration templates).
Choose Express or Custom Configuration	Express (default)
Configure Administrative Username and Password	Specify the administrator password (required) and change defaults as needed.
Configure Server Start Mode and Java SDK	Startup Mode: Development Mode (default) Java SDK Selection: Sun SDK (default)
Create WebLogic Configuration	For Configuration Name, specify <code>tptutorial</code> or another name that you want to use for this domain.

3. Click the Create button.

By default, the Configuration Wizard creates the new domain in `BEA_HOME\user_projects\domains\domainName`, where:

- `BEA_HOME` is the BEA home directory (such as `c:\bea`).
- `domainName` is the name of the domain you created (such as `tptutorial`).

4. On the Creating Configuration screen, select (click) Start Admin Server.

5. Click the Done button.

Step 2: View the Default Trading Partner Information

When you create a new WebLogic Integration domain using the Configuration Wizard, the Configuration Wizard automatically populates the Trading Partner Management (TPM) repository with default trading partners and protocol bindings. The trading partner integration tutorials use this default configuration. To learn more about the TPM repository, see [Trading](#)

Partner Management (available at <http://edocs.bea.com/wli/docs81/manage/tpm.html>) in *Managing WebLogic Integration Solutions*.

By default, WebLogic Integration runs in Test (development) mode, which allows you to use the default protocol bindings and to run business processes from separate trading partners on the same machine (collocated). In a production environment, each trading partner would run its respective business process on its own separate WebLogic Integration server, service profiles would need to be explicitly configured, both trading partners would need to be enabled, and the service profile would need to be enabled. For more information about the Test and Production modes, see “Configuring the Mode and Message Tracking” in **Trading Partner Management** (available at <http://edocs.bea.com/wli/docs81/manage/tpm.html>) in *Managing WebLogic Integration Solutions*.

Default Trading Partners

The WebLogic Integration domain provides two preconfigured trading partners for development and testing:

Table 1-3 Default Trading Partner Configuration in WebLogic Integration Domain

Trading Partner Name	Trading Partner ID	Description
Test_TradingPartner_1	TP1-id	Default local trading partner. In the tutorials, this trading partner is usually the <i>initiator</i> of conversations. In the absence of specific trading partner information, the default trading partner is designated as the trading partner used for sending or receiving messages for the local host system
Test_TradingPartner_2	TP2-id	In the tutorials, this trading partner is usually the <i>participant</i> in conversations.

Default Protocol Bindings

Each default trading partner comes with the following preconfigured protocol bindings:

- ebXML 1.0
- ebXML 2.0
- RosettaNet Implementation FrameWork (RNIF) 1.1

- RNIF 2.0

Each protocol binding (except ebXML 1.0) is marked as default. At run-time, the default binding can be used automatically in the absence of specific protocol information.

Viewing Trading Partner Information (Optional)

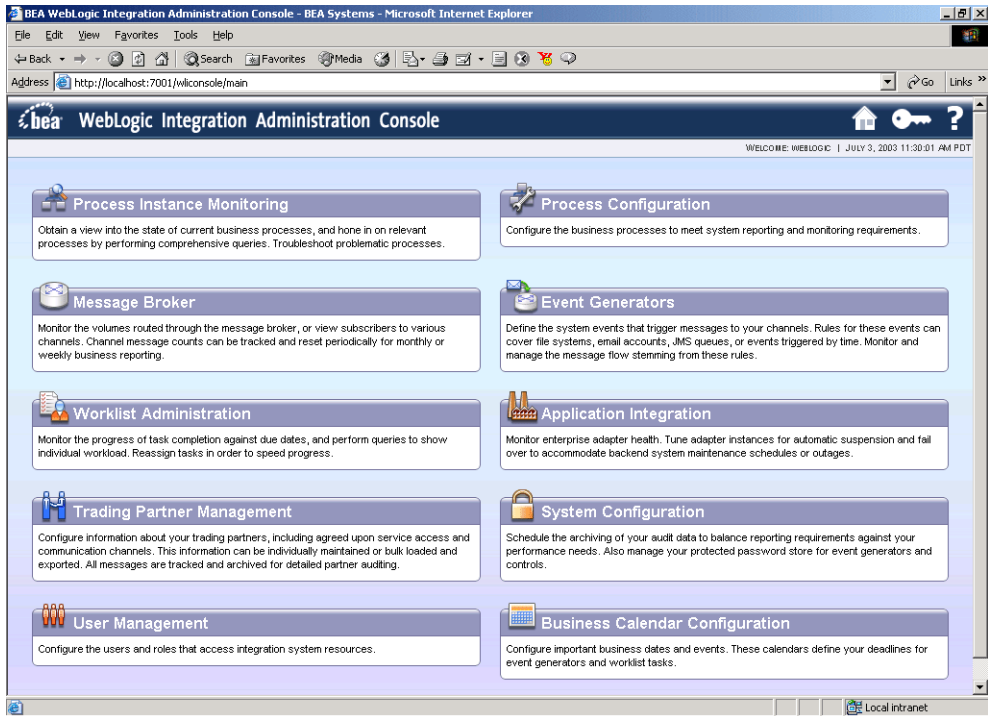
You view and update the contents of the TPM repository using the WebLogic Integration Administration Console. If you want, you can use the WebLogic Integration Administration Console to browse the preconfigured settings in the TPM repository. The trading partner integration tutorials use the preconfigured settings, so no changes to the TPM repository are required unless otherwise stated in the tutorials.

Note: If you make any changes to the preconfigured settings in the TPM repository, you should adjust the instructions in the tutorials accordingly. For configuration instructions, see [Trading Partner Management](http://edocs.bea.com/wli/docs81/manage/tpm.html) (available at <http://edocs.bea.com/wli/docs81/manage/tpm.html>) in *Managing WebLogic Integration Solutions*.

To view the default trading partner information:

1. Start the WebLogic Integration Administration Console by doing either of the following:
 - From the Start menu, choose **Programs→BEA WebLogic Platform→Examples→WebLogic Integration→Integration Admin Console**.
 - In a browser, enter the following URL: `http://localhost:7001/wliconsole`.
2. When prompted, specify the username and password.

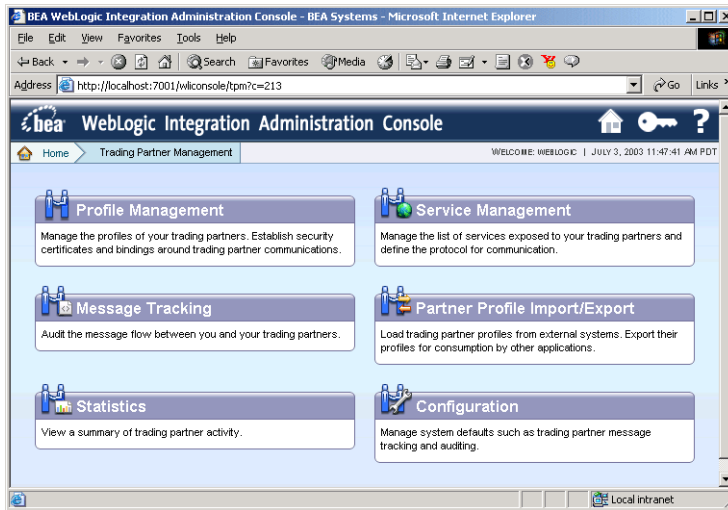
The WebLogic Integration Administration Console displays the home page.



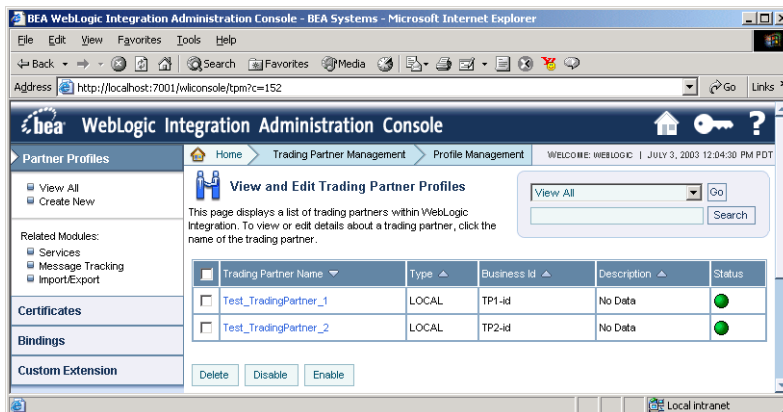
Note: If you want more information about a particular screen in the WebLogic Integration Administration Console, click the Help icon.

3. Select the Trading Partner Management module to see the Trading Partner Management home page.

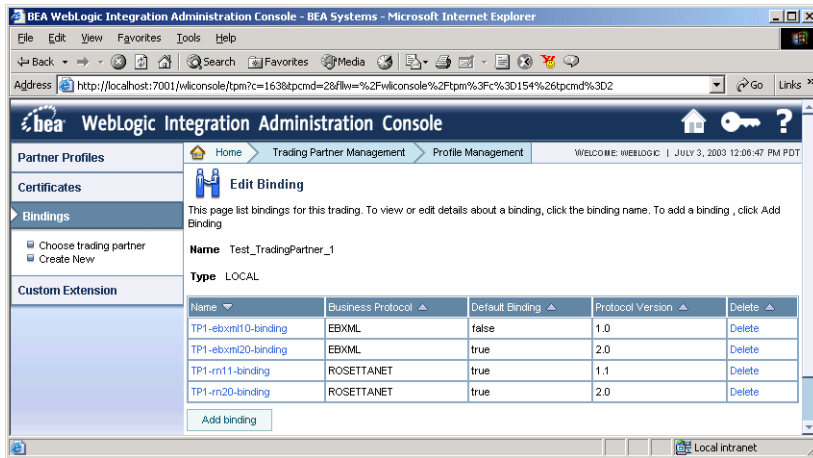
Introduction



4. On the Trading Partner Management home page, select the Profile Management module to see the profiles for the preconfigured trading partners.



5. From the left panel, select Bindings, select Test_TradingPartner_1, and click the Go button to see the preconfigured bindings for the default trading partner.



Note: By default, trading partner endpoints are configured to listen on `localhost:7001`. If you configured WebLogic Server to listen on a different port, adjust the trading partner binding information accordingly.

Step 3: Install the Tutorial Files

The documentation and example files for the trading partner integration tutorials are distributed in an archive file (`tptutorial.zip`) that you obtain from BEA's dev2dev Online site. This archive file contains the completed ebXML and RosettaNet applications used in the tutorials, as well as schema files and sample XML files.

To install the files for the trading partner integration tutorials:

1. Download the `tptutorial.zip` file from the WebLogic Integration 8.1 page at the following URL:
<http://dev2dev.bea.com/code/wli.jsp>
2. Extract the contents of the `tptutorial.zip` file (using folder names) to a local directory. The extracted files have the following directory structure:

Table 1-4 Contents of tptutorial.zip

Folder / File	Description
tptutorialapps\tptutorial.pdf	Documentation for the trading partner integration tutorials.
tptutorialapps\ebxml	Completed application that you can refer to if you encounter difficulties while building the business processes during the ebXML tutorial. Also contains schema files and sample XML data files.
tptutorialapps\rosettanet	Application containing example implementations of PIPs 3B2 and 3A4.

Next Steps

After you have set up the tutorial domain and extracted the tutorial files, you can begin using the following tutorials:

- [Chapter 2, “Tutorial: Building ebXML Solutions”](#)
- [Chapter 3, “Tutorial: Building RosettaNet Solutions”](#)

Both tutorials use the same WebLogic Integration domain that you created in [“Setting Up the Tutorials” on page 1-2](#). These are standalone tutorials. You can complete both tutorials, if you want, but you can also complete just the ebXML tutorial without the RosettaNet tutorial, and vice versa.

Tutorial: Building ebXML Solutions

The ebXML language (Electronic Business using eXtensible Markup Language) is a business protocol that enables enterprises to conduct business over the Internet. WebLogic Workshop uses ebXML controls for initiator processes and ebXML Participant Business Processes templates for participant processes to build business processes systems which exchange ebXML business messages between trading partners. The ebXML control provides the initiator business process with predefined customizable methods for sending and receiving ebXML messages in a conversation. The ebXML Participant Business Process template provides a head start for building public participant business processes for ebXML conversations. Although the template is not required to build ebXML participant business processes, it includes the nodes and business process annotations needed to integrate easily with ebXML initiator business processes. The purpose of this tutorial is to demonstrate the different options available for ebXML trading partner management in WebLogic Workshop and WebLogic Integration.

Before You Begin

This topic describes tasks that you should perform before you begin using this tutorial. It contains the following sections:

- Prerequisites
- Suggested Reading

Prerequisites

To use this tutorial, you must have:

- Installed WebLogic Platform 8.1 on your system according to the instructions in *Installing WebLogic Platform*, which is available at <http://edocs.bea.com/platform/docs81/install/index.html>.
- Completed the tutorial setup procedure according to the instructions in “Setting Up the Tutorials” on page 1-2.

Suggested Reading

To gain a detailed understanding of how ebXML solutions are implemented in WebLogic Integration, consider reading the following material:

- For an overview of integrating trading partners using WebLogic Integration, see *Introducing Trading Partner Integration*, which is available at <http://edocs.bea.com/wli/docs81/tpintro/index.html>.
- For an introduction to ebXML solutions using WebLogic Integration, see *Introducing ebXML Solutions* in *Introducing Trading Partner Integration*, which is available at <http://edocs.bea.com/wli/docs81/tpintro/ebxml.html>.
- For more information about using the ebXML control in initiator business processes, see *ebXML Control* in the WebLogic Workshop Online Help.
- For information about using participant business processes, see the following topics in the WebLogic Workshop Help:
 - *Building ebXML Participant Business Processes*, which is available at <http://edocs.bea.com/workshop/docs81/doc/en/integration/wfguide/wfguideEbXML.html>
 - *@jpd:ebXML Annotation*, which is available at http://edocs.bea.com/workshop/docs81/doc/en/integration/reference/refJpdAnnotations_ebxml.html

To learn how to build business processes and create data transformations in WebLogic Workshop, consider completing the following tutorials:

- *Tutorial: Building Your First Business Process*, which is available at <http://edocs.bea.com/workshop/docs81/doc/en/integration/tutorial/tutWLIPProcessIntro.html>
- *Tutorial: Building Your First Data Transformation*, which is available at <http://edocs.bea.com/workshop/docs81/doc/en/integration/dttutorial/tutWLIDataTransIntro.html>

Tutorial Overview

In this tutorial, the first couple of examples contains detailed instructions of how to use the WebLogic Workshop user interface and the WebLogic Integration Administration Console user interface to perform the steps outlined in the example. The preceding examples build on the skills you learn in the first couple of sections, but do not outline the steps in quite as much detail.

The tutorial is organized into these parts:

“Step 1: Getting Started” on page 2-4

This section describes how to create the business process application in which you will create the business processes and other components required for the examples of the tutorial. It also describes how to import the sample data and schemas which are included in the zipped archive of the tutorial files.

“Step 2: Sending an XML Message through an One-Way ebXML Exchange” on page 2-9

This example demonstrates how to send an XML message from one trading partner to another through a simple one-way ebXML exchange. In this example, you will learn how to create participant and initiator business processes as well as, how to create and configure ebXML and file controls. The section also contains a detailed description of how to deploy and test business processes using the WebLogic Workshop Test Browser.

“Step 3: Selecting the Trading Partner Information Dynamically Through Typed XML” on page 2-23

This example is similar to the preceding one except for that this example demonstrates how to specify the trading partner information dynamically by using an XQuery selector, rather than specifying it statically in the ebXML control. It also describes how to configure the ebXML control to use typed XML data and specific method names.

“Step 4: Sending Raw Data (Binary File) Through an ebXML Exchange” on page 2-32

In this example, you learn how to use ebXML to send binary data between two trading partners through a Message Broker channel.

“Step 5: Creating a Roundtrip ebXML Conversation” on page 2-38

This section describes how to implement an ebXML conversation in which when a request message (order) is received, a response (invoice) message is sent back by adding a Client Response node to the participant business process and using the ebXML control callback feature to send the response message.

“Step 6: Implementing the Public/Private Pattern” on page 2-46

This example illustrates how to use subprocesses to implement the public/private pattern. The public/private pattern can be used to keep the details of backend integration contained

to a private process definition, while the public process definitions are dedicated to trading partner interaction.

“Step 7: Using the TPM Control and Callbacks” on page 2-51

In this section, you learn how to at run time obtain trading partner information from the TPM repository. You also learn how to use the onAck callback of the ebXML control.

“Step 8: Setting Partner ID Dynamically Based on Directory Name” on page 2-56

In this example, you investigate the already built application included with the tutorial files to learn how to read the name of a sub-directory and use that as the partner ID. The business process sets the partner ID dynamically using the setProperties method included in the ebXML control.

“Step 9: Creating a Distributed Setup” on page 2-61

This example briefly explains how to move to a distributed setup where one trading partner operates in one WebLogic Integration instance while the other trading partner operates in another instance. This is the setup that you would use in a production scenario where the two trading partners are running on two physically separated systems.

“Step 10: Configuring Non-Default Protocol Settings” on page 2-62

In this section you will learn how to add a service and a service profile in the WebLogic Integration Administration Console, which will give you control over trading partner communications.

“Step 11: Exchanging Messages with WebLogic Integration - Business Connect” on page 2-64

This example describes how to send messages between WebLogic Integration and WebLogic Integration - Business Connect. WebLogic Integration - Business Connect is a separate application which enables you to securely exchange large volumes of documents with your trading partners. You must have a current license of WebLogic Integration - Business Connect to be able to complete this example.

Step 1: Getting Started

Before you start the ebXML tutorial, you have to complete the procedures described in [“Setting Up the Tutorials” on page 1-2](#). If you have not yet completed these procedures, please do so before proceeding with this section.

In this step, you use WebLogic Workshop to create application, in which you build the ebXML tutorial business processes. You then import the sample data and schemas provided in the zipped archive that came with the tutorial files (see, [“Step 3: Install the Tutorial Files” on page 1-9](#)). The sample data and schemas are used to illustrate the functionality of the ebXML tutorial examples.

Lastly, you create two directories on your hard drive which are used throughout the tutorial to read files from and to write files to.

This section contains the following procedures:

- [“Creating the Business Process Application” on page 2-5](#)
- [“Importing the Tutorial Sample Data” on page 2-8](#)
- [“Importing the Tutorial Schemas” on page 2-8](#)

Creating the Business Process Application

WebLogic Integration extends the WebLogic Workshop to allow the building of integrated enterprise applications. An application in turn contains projects and files. A project can contain several components including, business processes, Web services, and XML files. In this section, you will create the Business Process Application in which you will later build the ebXML tutorial processes and process components.

To Create a Business Process Application

1. Start **WebLogic Workshop**:

To Start WebLogic Workshop on **Windows**:

- a. Invoke WebLogic Workshop from the Start menu by choosing:

Start→Programs→BEA WebLogic Platform 8.1→WebLogic Workshop 8.1.

If this is the first time WebLogic Workshop is started since it was installed, the *samples* project, which contains sample services installed with WebLogic Workshop, is displayed. Otherwise, the project which was opened last is displayed.

To Start WebLogic Workshop on **Linux**:

- a. Using a file system browser or shell, locate the Workshop start script at:

```
$HOME\bea\weblogic81\workshop\Workshop.sh
```

Where `HOME` is your Workshop install directory.

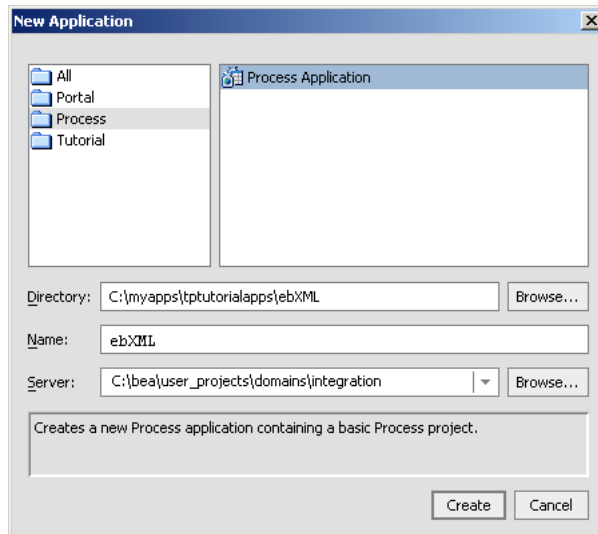
- b. Start WebLogic Workshop using one of the following methods:

-Use the following command from the command line:

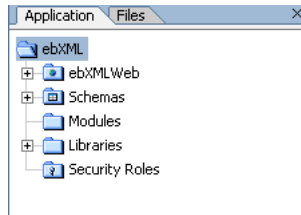
```
sh Workshop.sh
```

-From a file system browser, double-click `Workshop.sh`.

2. From the WebLogic Workshop menu, click **File→New→Application**. The **New Application** dialog box is displayed.



3. In the **New Application** dialog box, select **Process** in the left pane, and select **Process Application** in the right pane.
4. In the **Directory** field, select the directory in which you want to create your application. The examples in this tutorial assumes that the application will be created in: C:\myapps\tputorialapps. If you create the directory in a different location or use another name, adjust the instructions accordingly.
5. In the **Name** field, enter ebXML.
6. Click the arrow beside the **Server** field to display a list of servers. Then choose the server which you created when you set up your domain in [“Step 1: Create a New WebLogic Integration Domain”](#) on page 1-3.
7. Click **Create**.
8. Your Process Application is created and displayed in the **Application** pane. If the **Application** pane is not visible in WebLogic Workshop, choose **View→Application** from the menu bar.



The **Application** pane displays a hierarchical representation of the files and resources available in your application. The components we will work with in this tutorial includes the following:

ebXML—The application folder.

ebXMLWeb—A Web application project folder. Every application contains one or more projects. Projects represent WebLogic Server Web applications. In other words, when you create a project, you are creating a Web application. (The name of your project is included in the URL your clients use to access your application.)

Web Applications are J2EE deployment units that define a collection of Web resources such as business processes, Web services, JSPs, servlets, HTML pages, and can define references to external resources such as EJBs.

Note: The Web application project folder is named by appending **Web** to the name you gave your application.

Schemas—A Schemas project that contains the XML Schemas and the Message Broker channel file used in the application. It also contains ebXML envelope schemas that are used to package ebXML messages.

9. Expand the **ebXMLWeb** folder.

The **processes** folder inside the **ebXMLWeb** folder is created when you created your business process application. Since we do not need it for our tutorial, you can delete it:

10. Right-click on the **processes** folder and select **Delete** from the drop-down menu.

11. Click **Yes** in the confirmation dialogue.

You have now completed the first step in this tutorial and are ready to start creating the tutorial ebXML business processes. For more information about business process applications, see [Guide to Building Business Processes](http://edocs.bea.com/workshop/docs81/doc/en/integration/wfguide/wfguideIntro.html) available at:

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

Importing the Tutorial Sample Data

In this tutorial, we use sample data files to send in the ebXML messages. You can use your own XML data files if you wish, but some sample data files have been provided for you in the tptutorial.zip archive that you downloaded from the <http://dev2dev.bea.com> web site (see, “[Step 3: Install the Tutorial Files](#)” on page 1-9).

To Import the Sample Data Into Your Project Application

1. In the Application pane, right click on ebxmlWeb.
2. Select Import... from the drop-down menu.

The Import File window opens.

3. In the Import File window, navigate to [unzip location]\tptutorialapps\ebxml\ebxmlWeb\sampladata where [unzip location] is the directory to which you unzipped the files from the tptutorial.zip (see, “[Step 3: Install the Tutorial Files](#)” on page 1-9).
4. Click **Import**.

A folder named **sampladata** which contains the files **sampleinvoice.xml** and **sampleorder.xml** is added to your **Application** pane.

Note: Instead of the preceding steps, you could simply drag and drop the sampladata folder from any file browser directly into your Application pane in Weblogic Workshop.

Importing the Tutorial Schemas

In this tutorial, specific schemas are used for the XML data that is sent by the ebXML messages. This section describes how to import the schemas into your schemas folder in your ebXML application.

To Import the Tutorial Schemas

1. In the **Application** tab, right-click on a Schemas folder. (If the **Application** tab is not visible in WebLogic Workshop, choose **View**→**Application** from the menu bar.)
2. From the drop-down menu, select **Import....**

The **Import Files** dialog box is displayed.

3. Navigate to: [unzip location]\tptutorialapps\ebxml\Schemas. Where [unzip location] is the directory to which you unzipped the files from the tptutorial.zip file (see, “[Step 3: Install the Tutorial Files](#)” on page 1-9).
4. Select **b2bborder.xsd** and **b2binvoice.xsd** by clicking on the files while holding down the ctrl key.
5. Click **Import**.

When a XSD or MFL file is imported, a build of the current Schemas project folder is triggered. (The build verifies that the schema file is well formed. For XSD files, it also verifies that the element and attribute names in the XML Schema do not conflict with the XSD files that have already been imported into the current Schemas project folder.) For more information about what gets generated when you import schemas, see [Importing Schemas](http://edocs.bea.com/workshop/docs81/doc/en/integration/dtguide/dtguidemapperimportschema.s.html) available at <http://edocs.bea.com/workshop/docs81/doc/en/integration/dtguide/dtguidemapperimportschema.s.html>.

Creating the Read and Write Directories

To demonstrate how messages can be exchanged by ebXML, several of the examples in this tutorial write out files to and read files from directories on your hard drive. Before you start working with any of the examples in this tutorial, create the following two directories on your hard drive:

- C:\tptutorial\binary-in
- C:\tptutorial\binary-out

Note: This is assuming that you created your application in “[Creating the Business Process Application](#)” on page 2-5 on the C drive. If you created the application on another drive, please place the above directories at the root level of that drive.

Step 2: Sending an XML Message through an One-Way ebXML Exchange

In this example, you will learn how to send an XML message from one trading partner to another using ebXML. Imagine that one of the trading partners is accepting and processing orders. We refer to this partner as the seller and it is the participator of the ebXML conversation. The other trading partner, which we call the buyer, sends an XML order to the seller. The buyer is the initiator of the conversation. The seller accepts the message and simply writes it out to a file. This section contains the following procedures:

- [“Building the Seller Business Process” on page 2-10](#)
- [“Building the Buyer Business Process” on page 2-17](#)

Building the Seller Business Process

The Seller business process is the participator of our ebXML conversation. When building ebXML participator processes, you can use the ebXML Participant Process file which comes pre-configured with the nodes, variables, and other components necessary for building participants processes. In this example, you add a File control to the business process which writes the test data out to a file located on your hard drive. This section includes the following tasks:

- [“To Create the Seller Business Process File”](#)
- [“To Create the File Control and the Control Node” on page 2-12](#)
- [“To Configure the Control Node” on page 2-14](#)
- [“To View the ebXML Source View Parameters” on page 2-15](#)
- [“To Test the Seller Process” on page 2-16](#)

To Create the Seller Business Process File

1. In the Application pane, right-click **ebxmlWeb** and select **New→Folder...**

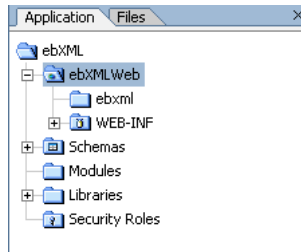
The **New Folder** dialogue opens.

Note: If the **Application** pane is not visible in WebLogic Workshop, choose **View→Application** from menu bar.

2. In the **New Folder** dialogue, enter **ebxml** as the name of your new folder.
3. Click **OK**.

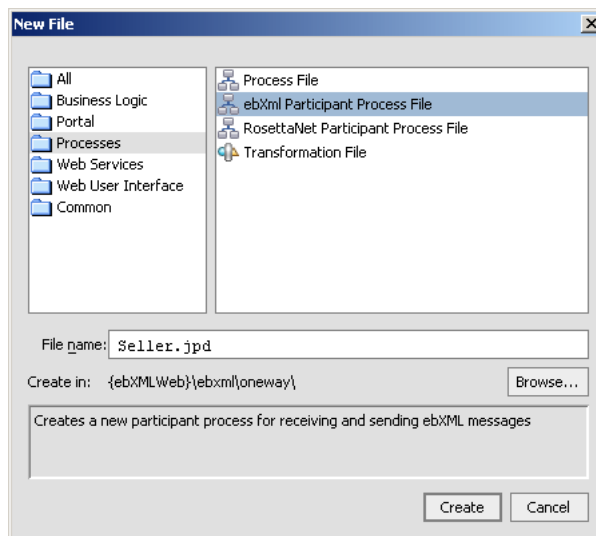
A new folder named **ebxml** is added as a sub folder under **ebxmlWeb** in the **Application** pane.

Step 2: Sending an XML Message through an One-Way ebXML Exchange



4. In the same way as described in the preceding steps, create another new folder as a sub folder under **ebxml** and name it **oneway**.
5. Right-click on **oneway** and select **New→Process File** in the drop-down menu.

The **New File** window opens.



6. In the **New File** window, select **Processes** from the left pane and **ebXML Participant Process File** from the right pane.
7. In the **File name** field, enter **Seller.jpdl**.
8. Click **Create**.

A new ebXML participant process file is created in your **oneway** folder in the **Application** pane and is displayed in Design View.



The ebXML participant process file is created pre-configured with the nodes and business process annotations needed to integrate easily with ebXML initiator business processes. For more information about ebXML participant process files, see [Building ebXML Participant Business Processes](http://edocs.bea.com/workshop/docs81/doc/en/integration/wfguide/wfguideEbXML.html) available at <http://edocs.bea.com/workshop/docs81/doc/en/integration/wfguide/wfguideEbXML.html>.

9. In this example, we will not respond to the buyer process, so we can delete the **Respond to request** node: right-click on the node and select **Delete** from the drop-down menu.

In this example, instead of responding to the buyer process, we add a **File** control which will write out the incoming ebXML messages to a file. This File control is then added to the business process as a Control Send with Return node.

To Create the File Control and the Control Node

1. In the **Controls** section of the **Data** palette, select **Add**→**Integration Controls**→**File**. The **Insert Control - File** window opens.

Step 2: Sending an XML Message through an One-Way ebXML Exchange

Insert Control - File

STEP 1 Variable name for this control:

STEP 2 I would like to :

☐ Use a File control already defined by a JCX file

JCX file:

☒ Create a new File control to use.

New JCX name:

☐ Make this a control factory that can create multiple instances at runtime

STEP 3 (OPTIONAL)

Name of the directory where the file control will look for files

directory-name:

A file name filter

file-mask:

The type of data contained in the file

file-type:

☐ The directory contains large files to be processed

(OPTIONAL) Size of each record (in bytes) in the large files

record-size:

(OPTIONAL) Character set encoding (for String type only)


encoding:

2. In the **Insert Control - File** window, enter the following:
 - a. In the **STEP 1**, enter **file** as the name of the variable to use for the File control. This name must be a valid Java identifier.
 - b. In **STEP 2**, select the **Create a new File control to use** option and enter **File** as the name of the new **JCX** file that will be created.
 - c. In **STEP 3**, enter **/tptutorial** as the directory-name. This is the directory to which the message received by the Seller.jpdl will be written. In this tutorial, we always use the

tutorial directory that you created in the “[Creating the Read and Write Directories](#)” on [page 2-9](#) section as the directory to read from and write to.

- d. In **STEP 3**, enter **order.xml** as the file mask This is the name of the file to which the message received by the Seller.jpdl will be written.
 - e. In **STEP 3**, select **XmlObject** as the type of data contained in the received message from the **file-type** drop-down menu.
3. Click **Create**.

The **File** control instance is added to your list of controls in the **Data** palette, and a **JCX** file corresponding to the **File** control is added to the **Application** palette. For more information about File controls, see [File Control](http://edocs.bea.com/workshop/docs81/doc/en/integration/controls/controlsFile.html) available at <http://edocs.bea.com/workshop/docs81/doc/en/integration/controls/controlsFile.html>. For more information about JCX files, see [JCX Files: Implementing Controls](http://edocs.bea.com/workshop/docs81/doc/en/workshop/guide/devenv/conJwiFiles.html) available at <http://edocs.bea.com/workshop/docs81/doc/en/workshop/guide/devenv/conJwiFiles.html>.

4. In the **Data** palette, expand the **File** control by clicking on the + next to it.
The methods associated with the **File** control are displayed.
5. Select the `FileControlPropertiesDocument write(XmlObject someData)` method.
6. Drag the selected method into **Design View** and drop it on the  which appears immediately following the **Receive request** node.

A new **Control Send with Return** node named **write** is added to your business process. For more information about Control Send with Return nodes, see [Interacting With Resources Using Controls](http://edocs.bea.com/workshop/docs81/doc/en/integration/wfguide/wfguideControls.html) available at <http://edocs.bea.com/workshop/docs81/doc/en/integration/wfguide/wfguideControls.html>.

The next step in our procedure is to configure the Control Node we just created with the correct variables and method assignments.

To Configure the Control Node

1. Double-click the new **write** node.
The **write** node builder is invoked.
2. Click **Send Data**.
3. From the **Select variables to assign** drop-down menu, select **payload** as the variable to assign to the write method.

4. Click **Apply** to apply your selection.
5. Click **Close** to close the node builder.
6. Select **File**→**Save** or enter **Ctrl+S** to save your work.

Your seller side ebXML process is now complete. The process is invoked when an XML message is received from the Buyer initiator process by the **Receive request** node, the **Receive request** node assigns the XML to a variable, and the **write** node writes the XML message to a file named **order.xml** via the File control.

Note: The grey check box icon is there because the receive data tab of the node has not been configured which marks the node as incomplete. However, since the example is not receiving any data, no more configuration is necessary for the business process to run properly.

To View the ebXML Source View Parameters

The ebXML binding information and service name of the business process is displayed in the Property Editor in Source View. To view these properties:

1. Click the **Source View** tab.
2. Click on `public class Seller implements com.bea.jpd.ProcessDefinition` to display the **Property Editor** for the `Seller` class.
3. In the **Property Editor**, note the following:
 - In the **process** section, the **binding** is listed as **ebxml**.
 - In the **ebxml** section, the **ebxml-service-name** property is set to **Seller**. This is the name you gave your process and it corresponds to the `eb:Service` entry in the ebXML message envelope. It is also the name used by the initiator business process as the ebXML service name. This is the name that trading partners will use to identify this service, it also matches the initiator process with the participant process.

For more information about Source View ebXML process annotations, see [@jpd:ebxml Annotation](http://edocs.bea.com/workshop/docs81/doc/en/integration/reference/refJpdAnnotations_ebxml.html) available at http://edocs.bea.com/workshop/docs81/doc/en/integration/reference/refJpdAnnotations_ebxml.html.

We are now ready to test the Seller business process. WebLogic Workshop provides a browser-based interface through which you can test the functionality of your business process. Using this Test View interface, you play the role of the client, invoking the business process's methods and viewing the responses.

To Test the Seller Process

1. If it is not already running, start WebLogic Server. to do so, choose **Tools→WebLogic Server→Start WebLogic Server** from the WebLogic Workshop menu.

If WebLogic Server is running, the following indicator is visible in the status bar at the bottom of the WebLogic Workshop visual development environment:



2. Click  on the WebLogic Workshop menu.

WebLogic Workshop builds your project. If there are no errors in the project, a Web browser is launched to display the Workshop Test Browser, through which you can test your business process using sample input values.

Note: It may take a while before the Test Browser window is launched after the application has completed the built process.

3. If not already displayed in the Test Browser window, click **Test Form**.
4. To enter test data, do one of the following:
 - Click **Browse**, and navigate to
`C:\myapps\tp tutorialapps\ebXML\ebXMLWeb\sampladata\sampleorder.xml`.
 - Cut and paste the content of
`C:\myapps\tp tutorialapps\ebXML\ebXMLWeb\sampladata\sampleorder.xml`,
except for the first line of the file, into the xml variable (payload) field.

5. Click **request**.

The Test Form page refreshes to display a summary of your request parameters.

6. Click **Refresh** to refresh the summary.
7. Scroll down to the **Operation request** section to see the content of the sampleorder.xml file.

8. Using a file browser or command line tool, navigate to your `C:\tp tutorial` directory.

It now contains a file named order.xml with identical content to that of sampleorder.xml. This file was written by the File control that you created and configured in the section [“To Create the Seller Business Process File” on page 2-10](#).

9. Return to WebLogic Workshop and click  on the tool bar to stop and close the Test Browser.

For more information about the different options in the Test Browser and how to use it to test your business processes, see [Running and Testing Your Business Process](http://edocs.bea.com/workshop/docs81/doc/en/integration/wfguide/wfguideTest.html) available at <http://edocs.bea.com/workshop/docs81/doc/en/integration/wfguide/wfguideTest.html>.

Your Seller business process is now deployed and ready to accept ebXML messaged. By default the protocols specified in the default ebXML 2.0 binding will be used, so no additional configurations are required. In production scenarios however, you typically create a TPM service profile using the WebLogic Workshop Administration Console. For more information about creating service profiles, see *Adding Service Profiles to a Service* in [Trading Partner Management](http://edocs.bea.com/wli/docs81/manage/tpm.html) (available at <http://edocs.bea.com/wli/docs81/manage/tpm.html>) in *Managing WebLogic Integration Solutions*.

Building the Buyer Business Process

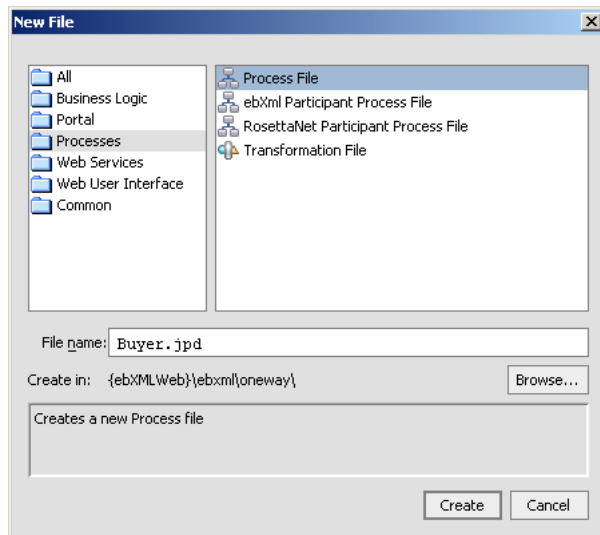
The Buyer business process is the initiator of our ebXML conversation. After creating the Buyer business process, we will use an ebXML control to communicate with the Seller participator business process. This section contains the following procedures:

- “To Create the Buyer Business Process File” on page 2-17
- “To Configure the Client Request Node” on page 2-19
- “To Create the ebXML Control and Control Node” on page 2-20
- “To Configure the Request Node” on page 2-22
- “To Test the Buyer Business Process” on page 2-23

To Create the Buyer Business Process File

1. In the **Application** pane, right-click on **oneway** and select **New→Process File**.

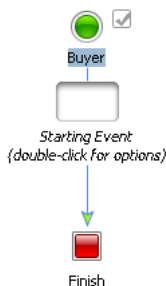
The New File window opens.



Since we are creating an initiator business process file this time and will use an ebXML file control to communicate with the participant process, we will create a default process file rather than an ebXML participant process file.

2. In the **New File** window, select **Processes** from the left pane and **Process File** from the right pane.
3. Enter **Buyer.jpdl** in the **File name** field.
4. Click **Create**.

A new **Buyer.jpdl** file is added to your **Application** pane and displayed in **Design View**.



The Buyer.jpdl file contains a Start node, a Starting Event place holder and a Finish node.

5. Double-click on the *Starting Event* place holder.

The Starting Event selection window is displayed.

6. Select **Invoked via a Client Request**.

7. Click **OK**.

A Client Request node is added as the Starting Event to your business process.

We will now configure the Client Request node with the correct methods and variable types.

To Configure the Client Request Node

1. Double-click the **Client Request** node.

The Client Request node builder is invoked.

2. In the **General Settings** tab:

- a. Change the method name from **clientRequest** to **startBuyer**.

- b. Click **Add**.

- c. If not already selected, select the **XML** option.

- d. Scroll down to the **un-typed XML** types.

- e. Select **XmlObject**.

- f. Click **OK**

The XmlObject parameter type is added to the pane.

- g. Click **Apply** to specify the XmlObject type as the expected type of the startBuyer method.

3. Click **Receive Data**.

4. In the **Receive Data** tab:

- a. Select **Create New Variable...** from the **Select Variables to Assign** drop-down menu.

The **Create Variable** window opens.

- b. Enter **order** as the variable name.

- c. Note that the type **XmlObject** is preselected for you since this is the type you specified on the **General Settings** tab.

- d. Click **OK**.

The new **order** variable is added to the Select variables to assign drop-down list.

- e. Click **Apply**.
- f. Click **Close**.

Your Client Request node is completed. To learn more about Client Request nodes, see [Receiving Messages From Clients](http://edocs.bea.com/workshop/docs81/doc/en/integration/wfguide/wfguideClientsReceiveNode.html) available at <http://edocs.bea.com/workshop/docs81/doc/en/integration/wfguide/wfguideClientsReceiveNode.html>. To learn more about variables and data types, see [Working with Data Types](http://edocs.bea.com/workshop/docs81/doc/en/integration/wfguide/wfguideDataTypesWorking.html) available at <http://edocs.bea.com/workshop/docs81/doc/en/integration/wfguide/wfguideDataTypesWorking.html>.

It is now time to add the ebXML control which will be used to communicate with the Seller process. This control represents what would normally be the remote service we are trying to contact, although in this example, the service runs on the same machine as the Buyer process.

To Create the ebXML Control and Control Node

1. In the **Controls** section of the **Data** palette, select **Add→Integration Controls→ebXML**.
The **Insert Control - ebXML** window opens.

Step 2: Sending an XML Message through an One-Way ebXML Exchange

Insert Control - ebXML

STEP 1 Variable name for this control:

STEP 2 I would like to :

☐ Use an ebXML control already defined by a JCX file

JCX file:

☒ Create a new ebXML control to use.

New JCX name:

STEP 3 Name of the ebXML Service

ebxml-service-name:

(OPTIONAL) Business Identifier of the Initiator of ebXML conversation from:

Business Identifier of the Participant in ebXML conversation to:

The type of attachment

method-arg-type:

(OPTIONAL) Mode of the ebXML Action: default or non-default

ebxml-action-mode:


2. In the **Insert Control - ebXML** window, enter the following:
 - a. In the **STEP 1**, enter **sellerControl** as the name of the variable to use for the control. This name must be a valid Java identifier.
 - b. In **STEP 2**, select the **Create a new File control to use** option and enter **SellerControl** as the name of the new **JCX** file that will be created.
 - c. In **STEP 3**, enter **Seller** as the **ebxml-service-name**. This value must always match the value in the participant (in our case the Seller) business process. This service name corresponds to the `eb:Service` entry in the ebXML message envelope.
 - d. In **STEP 3**, leave the **from** field blank. This will default to the TP1-id at runtime, the default trading partner.

- e. In **STEP 3**, enter **TP2-id** in the **to** field. This sets the participating trading partner id as a static value.
 - f. In **STEP 3**, select **XmlObject** as the type of data contained in the received message from the **file-type** drop-down menu.
3. Click **Create**.

The **ebXML** control instance is added to your list of controls in the **Data** palette, and a **JCX** file corresponding to the **ebXML** control is added to the **Application** palette. You can double-click on the **JCX** file to open it in **Design View** and review the values which you just entered. For more information about ebXML controls, see [ebXML Control](http://edocs.bea.com/workshop/docs81/doc/en/integration/controls/controlsebXML.html).

available at

<http://edocs.bea.com/workshop/docs81/doc/en/integration/controls/controlsebXML.html>.
For more information about JCX files, see [JCX Files: Implementing Controls](http://edocs.bea.com/workshop/docs81/doc/en/workshop/guide/devenv/conJwiFiles.html) available at <http://edocs.bea.com/workshop/docs81/doc/en/workshop/guide/devenv/conJwiFiles.html>.

4. In the **Data** palette, expand the **ebXML** control by clicking on the **+** next to it.
The methods associated with the **ebXML** control are displayed.
5. Select the `void request(XmlObject payload)` method. The name of the method on this control must match the method name of the Receive request node in the participant process.
6. Drag the selected method into **Design View** and drop it on the  which appears immediately following the **Client Request** node.

A new **Control Send** node named **request** is added to your business process. For more information about Control Send nodes, see [Interacting With Resources Using Controls](http://edocs.bea.com/workshop/docs81/doc/en/integration/wfguide/wfguideControls.html) available at

<http://edocs.bea.com/workshop/docs81/doc/en/integration/wfguide/wfguideControls.html>.

We will now configure the request node with the correct variable assignment.


To Configure the Request Node

1. Double-click the new **request** node.
The **request** node builder is invoked.
2. Click **Send Data**.
3. From the **Select variables to assign** drop-down menu, select **order** as the variable to assign to the request method.
4. Click **Apply** to apply your selection.

5. Click **Close** to close the node builder.
6. Select **File→Save** or enter **Ctrl+S** to save your work.

Your buyer side ebXML process is now complete. The process sends a message to the Seller business process which is then written to a file named **order.xml** via the File control in the Seller process.

To Test the Buyer Business Process

1. Click  on the WebLogic Workshop menu. (For more detailed instructions refer back to, [“To Test the Seller Process” on page 2-16.](#))
2. To enter test data, do one of the following:
 - Click **Browse**, and navigate to
`C:\myapps\tp tutorialapps\ebXML\ebXMLWeb\sampl edata\sampleorder.xml.`
 - Cut and paste the content of
`C:\myapps\tp tutorialapps\ebXML\ebXMLWeb\sampl edata\sampleorder.xml`
into the xml variable (payload) field.
3. Confirm that the `order.xml` file was written out correctly to the `C:\tp tutorial` directory by checking that the time stamp of the file is current.

This part of the tutorial is now complete. You have successfully sent an XML message over ebXML from one trading partner to another. The participant (Seller) process wrote the XML out to the file system.

Step 3: Selecting the Trading Partner Information Dynamically Through Typed XML

In this example we build on what you learned in the previous exercise, [“Step 2: Sending an XML Message through an One-Way ebXML Exchange” on page 2-9.](#) You will learn how to specify the trading partner information dynamically using an XQuery selector rather than specifying it statically in the ebXML control. Then you will customize the ebXML control in the initiator (buyer) process and the participant process file (Seller) to use typed XML and business specific method names.

This step contains the following procedures:

- [“Building the SelectorSeller Business Process” on page 2-24](#)
- [“Building the SelectorBuyer Business Process” on page 2-27](#)

Building the SelectorSeller Business Process

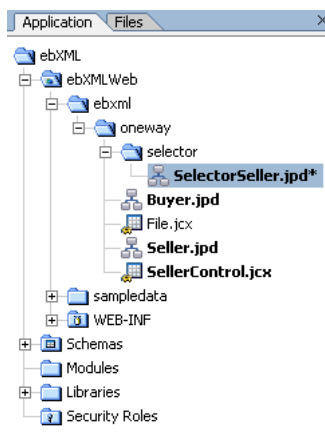
In this procedure, you create an ebXML participant business process which will accept typed XML data instead of the XmlObject variable you used in the previous example's participant process. This section contains the following tasks:

- [“To Create the SelectorSeller Business Process File” on page 2-24](#)
- [“To Configure the Receive Request Node” on page 2-25](#)
- [“To Create the File Control and Control Node” on page 2-26](#)
- [“To Configure the Control Node” on page 2-26](#)
- [“To Test the SelectorSeller Process” on page 2-26](#)

To Create the SelectorSeller Business Process File

1. Create a sub folder named **selector** under **\ebxml\oneway** in your Application tab. (For detailed instructions, refer back to [“To Create the Seller Business Process File” on page 2-10.](#))
2. Create a new **ebXML Participant Process File** named **SelectorSeller.jpdl**.

Your Application tab should look similar to the following.



3. As in the previous example, remove the **Respond to request** node.

The next step involves configuring the Receive request node to accept XML data of a type that is specified in the Schemas which you imported into your application in [“Importing the Tutorial Schemas” on page 2-8.](#)

To Configure the Receive Request Node

1. Double-click the **Receive request** node.

The Receive request node builder is invoked.

2. In the **General Settings** tab:
 - a. Change the method name from **request** to **processOrder**.
 - a. Select **XmlObject** payload.
 - b. Click **Remove**
 - c. Click **Add**.
 - d. If not already selected, select the **XML** option.
 - e. Expand the **b2border.xsd** by clicking on the **+** next to it.
 - f. Select **Order** from the expanded list.
 - g. Click **OK**

The **OrderDocument** parameter type is added to the pane.

- h. Click **Apply** to specify the **OrderDocument** type as the expected type of the **processOrder** method.
3. Click **Receive Data**.
 4. In the **Receive Data** tab:
 - a. Select **Create New Variable...** from the **Select Variables to Assign** drop-down menu.

The **Create Variable** window opens.

- b. Enter **order** as the variable name.

Note: The type of the variable is already specified to be `com.bea.tutorial.order.OrderDocument` which is what you specified in the preceding steps.

- c. Click **OK**.

The new **order** variable is added to the Select variables to assign drop-down list.

- d. Click **Apply**.

- e. Click **Close**.

Your Client Request node is completed. To learn more about Client Request nodes, see [Receiving Messages From Clients](http://edocs.bea.com/workshop/docs81/doc/en/integration/wfguide/wfguideClientsReceivingNode.html) available at <http://edocs.bea.com/workshop/docs81/doc/en/integration/wfguide/wfguideClientsReceivingNode.html>. To learn more about variables and data types, see [Working with Data Types](http://edocs.bea.com/workshop/docs81/doc/en/integration/wfguide/wfguideDataTypesWorking.html) available at <http://edocs.bea.com/workshop/docs81/doc/en/integration/wfguide/wfguideDataTypesWorking.html>.

As in the previous example, you will add a file control to the business process file, which will write the **order** variable out to a file.

To Create the File Control and Control Node


Refer to the instructions in [“To Create the File Control and the Control Node” on page 2-12](#) to create a new **File** control. To save time, you can also drag and drop **File.jcx** from the `\ebxml\oneway` folder in the **Application** tab directly on to the **Design View** of the **SelectorSeller**. The control definitions you created earlier will be reused in a new instance for this business process.

To Configure the Control Node

1. Double-click the new **write** node.
The **write** node builder is invoked.
2. Click **Send Data**.
3. From the **Select variables to assign** drop-down menu, select **order** as the variable to assign to the write method.
4. Click **Apply** to apply your selection.
5. Click **Close** to close the node builder.
6. Select **File**→**Save** or enter **Ctrl+S** to save your work.

This concludes the creation of the SelectorSeller business process.

To Test the SelectorSeller Process

1. Click  on the WebLogic Workshop menu. (For more detailed instructions refer back to, [“To Test the Seller Process” on page 2-16](#).)
2. To enter test data, do one of the following:

- Click **Browse**, and navigate to
C:\myapps\tptutorialapps\ebXML\ebXMLWeb\sampleddata\sampleorder.xml.
 - Cut and paste the content of
C:\myapps\tptutorialapps\ebXML\ebXMLWeb\sampleddata\sampleorder.xml
into the xml variable (payload) field.
3. Confirm that the `order.xml` file was written out correctly to the `C:\tptutorial` directory by checking that the time stamp of the file is current

You are now ready to go on and create a SelectorBuyer initiator process.

Building the SelectorBuyer Business Process

For the initiator process in this example you use an XQuery selector to retrieve the trading partner ID dynamically rather than specifying it statically in the ebXML control as done in the previous example. This section contains the following tasks:

- [“To Create the SelectorBuyer Business Process File” on page 2-27](#)
- [“To Configure the Client Request Node” on page 2-27](#)
- [“To Create the ebXML Control” on page 2-28](#)
- [“To Modify the ebXML Control Definition File” on page 2-29](#)
- [“To Create the processOrder Control Send Node” on page 2-31](#)
- [“To Test the SelectorBuyer Process” on page 2-31](#)

To Create the SelectorBuyer Business Process File

Create a business process file named **SelectorBuyer.jpdl** in the **selector** folder by following the instructions in [“To Create the Buyer Business Process File” on page 2-17](#).

To Configure the Client Request Node

1. Double-click the **Client Request** node.
The Client Request node builder is invoked.
2. In the **General Settings** tab:
 - a. Click **Add**.
 - b. If not already selected, select the **XML** option.

- c. Expand the **b2border.xsd** by clicking on the + next to it.
- d. Select **Order** from the expanded list.
- e. Click **OK**

The **OrderDocument** parameter type is added to the pane.

- f. Click **Apply** to specify the **OrderDocument** type as the expected type of the **processOrder** method.
3. Click **Receive Data**.
 4. In the **Receive Data** tab:
 - a. Select **Create New Variable...** from the **Select Variables to Assign** drop-down menu.
The **Create Variable** window opens.
 - b. Enter **order** as the variable name.
Note: The type of the variable is already specified to be `com.bea.tutorial.order.OrderDocument` which is what you specified in the preceding steps.
 - c. Click **OK**.
The new **order** variable is added to the Select variables to assign drop-down list.
 - d. Click **Apply**.
 - e. Click **Close**.

Your Client Request node is completed. To learn more about Client Request nodes, see [Receiving Messages From Clients](http://edocs.bea.com/workshop/docs81/doc/en/integration/wfguide/wfguideClientsReceiveNode.html) available at <http://edocs.bea.com/workshop/docs81/doc/en/integration/wfguide/wfguideClientsReceiveNode.html>. To learn more about variables and data types, see [Working with Data Types](http://edocs.bea.com/workshop/docs81/doc/en/integration/wfguide/wfguideDataTypesWorking.html) available at <http://edocs.bea.com/workshop/docs81/doc/en/integration/wfguide/wfguideDataTypesWorking.html>.

In the next step you add a ebXML control and configure this control with an XQuery selector.

To Create the ebXML Control

1. In the **Controls** section of the **Data** palette, select **Add→Integration Controls→ebXML**.
The **Insert Control - ebXML** window opens.

2. In the **Insert Control - ebXML** window, enter the following:
 - a. In the **STEP 1**, enter **selectorSellerControl** as the name of the variable to use for the control. This name must be a valid Java identifier.
 - b. In **STEP 2**, select the **Create a new File control to use** option and enter **SelectorSellerControl** as the name of the new **JCX** file that will be created.
 - c. In **STEP 3**, enter **SelectorSeller** as the **ebxml-service-name**. This value must always match the value in the participant (in our case the SelectorSeller) business process. This service name corresponds to the `eb:Service` entry in the ebXML message envelope.
 - d. In **STEP 3**, leave the **from** field blank. This will default to the TP1-id at runtime, the default trading partner.
 - e. In **STEP 3**, leave the **to** field blank. This will be determined at runtime by the XQuery selector which you will define in the next section.
 - f. In **STEP 3**, select **XmlObject** as the type of data contained in the received message from the **file-type** drop-down menu.
3. Click **Create**.

The **ebXML** control instance is added to your list of controls in the **Data** palette, and a **JCX** file corresponding to the **ebXML** control is added to the **Application** palette. You can double-click on the **JCX** file to open it in **Design View** and review the values which you just entered. For more information about ebXML controls, see [ebXML Control](http://edocs.bea.com/workshop/docs81/doc/en/integration/controls/controlsebXML.html) available at <http://edocs.bea.com/workshop/docs81/doc/en/integration/controls/controlsebXML.html>. For more information about JCX files, see [JCX Files: Implementing Controls](http://edocs.bea.com/workshop/docs81/doc/en/workshop/guide/devenv/conJwiFiles.html) available at <http://edocs.bea.com/workshop/docs81/doc/en/workshop/guide/devenv/conJwiFiles.html>.

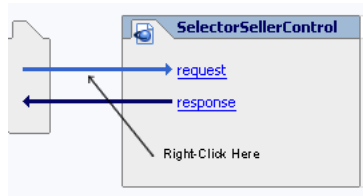
You will now modify the control definition file of the ebXML control. In the next section, you change the method name, change the message type, and add an XQuery selector.

To Modify the ebXML Control Definition File

1. In the **Data Palette**, right-click on the ebXML **selectorSellerControl**.
2. Select **Edit** from the drop-down menu.

The **SelectorSellerControl** is displayed in **Design View**.

3. Right-click on the arrow going to the **request** method, as illustrated in the following graphic.



4. Select **Rename** from the drop-down menu.

5. Enter **processOrder**.

6. Press the **Enter** key.

The method name now matches the name of the method on the participant business process. This method name corresponds to the `eb:Action` element in the ebXML message envelope.

7. Right-click on the arrow going away from the **response** method.

8. Select Delete from the drop-down menu.

You deleted the pre-configured response method since we are not sending a response in this example.

9. Click on processOrder.

The ebXML Source View opens.

10. Change the line `void processOrder(XmlObject payload);` to `void processOrder(OrderDocument payload);`

11. The tooltip editor displays a suggested package to import, press **Alt+Enter** to accept.

This ebXML control is now set up to use XML typed according to the schema you imported in [“To Import the Tutorial Schemas” on page 2-8](#) rather than generic untyped XML.

12. Click the Design View tab.

13. Double-click the arrow which points to processOrder.

The **Property Editor** window opens with a graphical representation of the Start Method Schema. In this example, we assume that the target trading partner ID is contained in the **Supplier ID** element.

14. Click **Supplier_ID** to select it.

A selector XQuery statement is generated.


15. Click **OK**.

16. Select **File**→**Save** or enter **Ctrl+S** to save your work.

17. Close the **SelectorSellerControl.jcx** by clicking the **x** in the top right corner of the window.

The ebXML control is now configured to pick out the trading partner ID from the message according to the specifications of the XQuery statement you generated in the selector. The next step is to add a processOrder Control Send node to the business process.

To Create the processOrder Control Send Node


1. In the **Data Palette**, expand the **selectorSellerControl** by clicking on the **+** next to it.
2. Select the `void processOrder(OrderDocument payload)` method. The name of the method on this control must match the method name of the Receive request node in the participant process.
3. Drag the selected method into **Design View** and drop it on the  which appears immediately following the **Client Request** node.

A new **Control Send** node named **processOrder** is added to your business process. For more information about Control Send nodes, see [Interacting With Resources Using Controls](http://edocs.bea.com/workshop/docs81/doc/en/integration/wfguide/wfguideControls.html) available at <http://edocs.bea.com/workshop/docs81/doc/en/integration/wfguide/wfguideControls.html>.

4. To specify the `order` variable as the input variable:
 - a. In the **Data Palette**, select `order` from the list of XML variables.
 - b. Drag and drop it onto the **processOrder** Control Send node.

This concludes the creation of the SelectorBuyer business process. In this section, you learned how to customize the participant and initiator processes to use typed XML and custom method names. You also learned how to use an XQuery selector to extract the target trading partner ID from the payload message rather than declaring it statically. The next section describes how to test your business processes.

To Test the SelectorBuyer Process

1. Click  on the WebLogic Workshop menu. (For more detailed instructions refer back to, “[To Test the Seller Process](#)” on page 2-16.)

2. To enter test data, do one of the following:
 - Click **Browse**, and navigate to
`C:\myapps\tp tutorialapps\ebXML\ebXMLWeb\sampl edata\sampleorder.xml.`
 - Cut and paste the content of
`C:\myapps\tp tutorialapps\ebXML\ebXMLWeb\sampl edata\sampleorder.xml`
into the `xml` variable (payload) field.
3. Confirm that the `order.xml` file was written out correctly to the `C:\tp tutorial` directory by checking that the time stamp of the file is current.

Step 4: Sending Raw Data (Binary File) Through an ebXML Exchange

In this example, we will use ebXML to send binary data between two trading partners through a Message Broker channel. This example builds on the examples describes in previous sections. If at any time you need more information about how to complete a task that was previously explained, refer back to the preceding sections. This section contains the following procedures:

- [“Building the BinarySeller Business Process” on page 2-32](#)
- [“Building the BinaryBuyer Business Process” on page 2-33](#)

Building the BinarySeller Business Process

This example builds on what you have learned in the previous examples. If at any time you need more detailed instructions, refer back to [“Building the Seller Business Process” on page 2-10](#). This section contains the following tasks:


- [“To Create the BinarySeller Business Process” on page 2-32](#)
- [“To Test the BinarySeller Business Process” on page 2-33](#)

To Create the BinarySeller Business Process

1. Create a new folder in your application under `\ebxml\oneway\binary`.
2. Create a new ebXML participant process in this folder named **BinarySeller.jpdl**.
3. Remove the **Respond to request** node.
4. Modify the **Receive request** node as follows:
 - a. Remove the **XmlObject** parameter data type.

- b. Add a **RawData** parameter data type (found in NonXML option).
 - c. Create a new variable named **data** of type RawData.
 - d. Specify the new variable as the one to assign to the request method.
5. Create a new **File** control with the following values specified:
 - Variable name: **file**
 - New JCX name: **File**
 - directory-name: /tptutorial/binary-in
 - file-mask: **data.bin** (this is an arbitrary file name, it could be anything)
 - file-type: **RawData**
6. Drag and drop the **write** method of the **file** control after the Receive request node in the business process.
7. Double-click the write node and on the Send Data tab, specify **data** as the variable to assign to the write method.

To Test the BinarySeller Business Process

1. Click  on the WebLogic Workshop menu. (For more detailed instructions refer back to, [“To Test the Seller Process” on page 2-16.](#))
2. Use any binary file (for example an image file) as binary payload test data.
3. Confirm that the `data.bin` file was written out correctly to the `C:\tptutorial\binary-in` directory by checking that the time stamp of the file is current.

Building the BinaryBuyer Business Process

For the initiator side of this example, the process will look very similar to those in previous examples, except that it will be invoked through a subscription to a file event which will cause the process to pick up a file from a directory. This section contains the following tasks:

- [“To Create the BinaryBuyer Business Process File” on page 2-34](#)
- [“To Create the Channel File” on page 2-34](#)
- [“To Configure the Subscription Node” on page 2-35](#)

- “To Create the ebXML Control” on page 2-35
- “To Test the BinaryBuyer Business Process” on page 2-36

To Create the BinaryBuyer Business Process File

Use the instructions outlined in detail in “Building the Buyer Business Process” on page 2-17 to complete the following tasks:

1. In the **ebxml\oneway\binary** folder, create a new business process named **BinaryBuyer.jpdl**.
2. Double-click the Starting Event and select the **Subscribe to a Message Broker channel and start via an event (Timer, Email, File, Adapter, etc.)** option.

Before you can start to configure the Message Broker Subscription node, you need to create a subscription channel file to which the subscription node can listen for events.

To Create the Channel File

1. In the **Application** tab, right-click on the **Schemas** folder.
2. Select **New→Channel File**
3. Enter `tptutorial.channel` as the file name.
4. Click **Create**.

The new channel file is created, added under the Schemas folder in the Application tab, and displayed on screen. An application build is also started automatically to build the Schemas project.

5. In the channel file code, locate the following line:
`<channel name="SampleRawDataChannel" messageType="rawData" />`
6. Remove the comment tags `<!--` and `-->` preceding and following the line.
7. Select **File→Save** or enter **Ctrl+S** to save the channel file.

The Schemas project builds again.

8. Close the Channel file by clicking the **x** in the top right corner of the window.

The sample channels defined in the file are now available for selection in the subscription node. For more information about channels, see [Publishing and Subscribing to Channels](http://edocs.bea.com/workshop/docs81/doc/en/integration/howdof/howPubSub.html) available at <http://edocs.bea.com/workshop/docs81/doc/en/integration/howdof/howPubSub.html>.

To Configure the Subscription Node

1. Double-click the Subscription node.
The Subscription node builder is invoked.
2. On the **General Settings** tab, select the **SampleRawDataChannel** from the **Channel name** drop-down list.
3. In the **Receive Data** tab:
 - a. Create a variable named **data** of type **RawData**
 - b. Select it to be the variable to assign to the subscription method.
4. Click **Apply**.
5. Click **Close**.

You have now completed the configuration of your subscription node. To learn more about subscription nodes, see [Subscription Start](http://edocs.bea.com/workshop/docs81/doc/en/integration/wfguide/wfguideStartSub.html) available at <http://edocs.bea.com/workshop/docs81/doc/en/integration/wfguide/wfguideStartSub.html>.

In the next section, you create a ebXML control to be used to communicate with the participator business process.


To Create the ebXML Control

1. Use the instructions in “[To Create the ebXML Control and Control Node](#)” on page 2-20 to create an ebXML with the following values:
 - Variable name: **binarySellerControl**
 - New JCX name: **BinarySellerControl**
 - ebxml-service-name: **BinarySeller** (must match the value in the participant process)
 - from: leave blank
 - to: **TP2-id**
 - method-arg-type: **RawData**
2. Click **Create**.
3. From the **Data Palette**, drag and drop the `void request(RawData payload)` method on the target appearing in the business process below the **Subscription** node.

4. From the Data Palette, drag and drop the `data` variable in the Non-XML list onto the **request** node.

The BinaryBuyer business process is now complete, run and test it to ensure that it is working properly.

To Test the BinaryBuyer Business Process

1. Click  on the WebLogic Workshop menu. (For more detailed instructions refer back to, “[To Test the Seller Process](#)” on page 2-16.)
2. Use an image file as the binary payload test data.
3. Confirm that the `data.bin` file was written out correctly to the `C:\tptutorial\binary-in` directory by checking that the time stamp of the file is current.

Note: Before you test your business process, make sure that you have completed the “[To Create the Channel File](#)” on page 2-34 section. If you do not create a channel file before testing the process, you will encounter an error.

Creating a File Event Generator

In this section you create the file event which will invoke the BinaryBuyer business process when a raw data type file is dropped in a directory.

To Create the File Event

1. Create the following new directory on your hard drive:
 - `C:\tptutorial\errors`
2. If not already running, start your WebLogic Server by selecting **Tools→WebLogic Server→Start WebLogic Server**.
3. After the Server is running, open the WebLogic Administration Console by selecting **Tools→WebLogic Integration→WebLogic Administration Console**.
4. Log in to the console using the server user name and password that you specified in the “[Step 1: Create a New WebLogic Integration Domain](#)” on page 1-3 section.
5. Click **Event Generators**.
The **Event Generators** page opens.
6. Click **Create New**.

7. Enter **TPTutorial** as the **Generator Name**.

8. Click **Submit**.

The **File Event Generator Definition** page opens

9. Click **Define a New Channel Rule**.

The **File Generator Channel Rule Definition** page opens.


10. Enter the following parameters:

- **File Type:** Disk File
- **Channel Name:** /SamplePrefix/Samples/SamplesRawDataChannel (rawData)
- **Message Encoding:** *leave blank*
- **Directory:** /tptutorial/binary-out
- **Pass by file name:** No
- **Scan subdirectories:** No
- **File pattern:** *leave blank*, it defaults to *.*
- **Sort by Arrival?:** No
- **Polling interval:** 3 seconds (small enough to avoid a long wait)
- **Read limit:** 0
- **Post Read Action:** Delete
- **Archive Directory:** *leave blank*
- **Error Directory:** /tptutorial/errors
- **Description:** Raw data file.

11. Click **Submit**.

You have completed creating the File Event. For more information about the WebLogic Administration Console and Event Generators, see [Event Generators](http://edocs.bea.com/wli/docs81/manage/evntgen.html) available at <http://edocs.bea.com/wli/docs81/manage/evntgen.html>.

To Test the Sending Raw Data Example

1. Click  on the WebLogic Workshop menu to deploy and run the BinaryBuyer business process.

2. Put a binary file in the `binary-out` directory you created earlier. In a few seconds it should disappear and a file named `data.bin` should appear in the `binary-in` directory.

Note: About Message Tracking: In this and previous examples, all messages sent via ebXML are tracked in the tracking database. In the WebLogic Administration Console, navigate to **Trading Partner Management**→**Message Tracking** to look at the messages. Basic message information and payload information are both tracked. You can also navigate from the message to the process that created or consumed the message. For more information see, [Trading Partner Management](http://edocs.bea.com/wli/docs81/manage/tpm.html) (available at <http://edocs.bea.com/wli/docs81/manage/tpm.html>) in *Managing WebLogic Integration Solutions*.

Step 5: Creating a Roundtrip ebXML Conversation

This example illustrates how to implement an ebXML conversation in which a request message (order request) is followed by a response message (invoice). The first part of the conversation is identical to the one-way scenario you developed earlier. In this example, you add a Client Response node to the participant business process and use the ebXML control callback for the response message. This section contains the following procedures:

- [“Building the RoundtripSeller Business Process” on page 2-38](#)
- [“Building the RoundtripBuyer Business Process” on page 2-42](#)

Building the RoundtripSeller Business Process

This example builds on the previous examples that you have completed. If at any time you need more detailed instructions, refer back to [“Building the Seller Business Process” on page 2-10](#) to complete the following sections:

- [“To Create the Business Process File and Configure the Receive Request Node” on page 2-38](#)
- [“To Create the File Controls and Configure the Control Nodes” on page 2-39](#)
- [“To Configure the Respond to Request Node” on page 2-41](#)
- [“To Deploy the RoundtripSeller Business Process” on page 2-41](#)

To Create the Business Process File and Configure the Receive Request Node

1. Create a new folder in your application named **roundtrip** as a sub folder of the **ebxml** folder.

2. In the roundtrip folder, create a new ebXML participant process named **RoundtripSeller.jpdl**, but do not delete the **Respond to request node**.
3. Double-click the Receive request node and configure as follows:
 - a. Rename the method to **processOrder**.
 - b. Remove the **XmlObject payload** parameter type.
 - c. Add the parameter type **OrderDocument** (from b2border.xsd)
 - d. Create a new variable named **order** of type **OrderDocument** and select it as the variable to be assigned to the method.
 - e. Click **Apply**, then click **Close**.

For illustration purposes, this example simplifies the order/invoice process. The seller will again write the incoming order out to file and for the response, the seller will read an invoice document from the file system. The invoice will be the same for each order. In production scenarios, the seller process could instead read the information from a database, obtain it from a message queue, or get it from a backend application as an event from an Application View control. The process could also delegate processing of the order to a subprocess, which is illustrated in the next example “[Step 6: Implementing the Public/Private Pattern](#)” on page 2-46. For more information about Application View controls, see [Application View Control](http://edocs.bea.com/workshop/docs81/doc/en/integration/controls/controlsAppView.html) available at <http://edocs.bea.com/workshop/docs81/doc/en/integration/controls/controlsAppView.html>.

To Create the File Controls and Configure the Control Nodes

1. Add a new **File** control named **file** and configure it to write out order.xml to C:\tptutorial\
Note: To save time, you can drag and drop the **File.jcx** in the **oneway** folder onto the **RoundtripSeller.jpdl** Design View. This will create a new instance of the **File.jcx**.
2. Drag and drop the **write** method to the target that appears below the **Receive request** node.
3. Double-click on the **write** node.
4. Configure the node to send the **order** variable to the **write** method.
5. Create a new directory on your hard drive named C:\tptutorial\invoice.
6. Copy the file
 C:\myapps\tptutorialapps\ebxml\ebxmlWeb\sampladata\sampleinvoice.xml to
 the C:\tptutorial\invoice directory
7. Rename **sampleinvoice.xml** to **invoice.xml**.

8. Create a second new **File** control named **invoiceFile** and configure it to read an **XmlObject** (invoice.xml) from the `C:\tptutorial\invoice` directory by using the following configurations:

- **file-mask:** invoice.xml
- **directory-name:** /tptutorial/invoice

9. Drag and drop the **read** method to the target that appears below the **write** node.

10. Double-click the **read** node.

11. Click **Receive Data**.

The method expects untyped XML (XmlObject) but our invoice.xml file contains typed XML so we will need to do a simple transformation between the two.

12. Select the **Transformation** Option.

13. Click **Select Variable→Create New Variable...**

The **Create Variable** window opens.

14. Enter **invoice** as variable name.

15. Expand **b2binvoice.xml** and select **Invoice**.

16. Click **OK**.

The **Create Variable** window closes and your new variable is listed in the variable pane of the node builder.

17. Click **Create Transformation**.

The **Transformation Tool** window opens.

18. Drag and drop the **XmlObjectDoc** in the **Source Schema** pane to the **Invoice** root in the **Target Schema** pane.

An XQuery transformation is created a corresponding transformation control is created and stored in the **RoundtripSellerTransformation.dtf** file, which was created in your **Application** pane when you opened the Transformation tool.

19. Close the Transformation Tool window by clicking on the x in the top right corner of the window.

20. Click **Yes** in the **Save Before Closing** dialogue window.

21. Click **Close** in the node builder.

The transformation you created in the preceding steps, only transforms from one data type to another. In production scenarios, you would use the Transformation Tool to convert from the backend system format to the format expected by the trading partner. You can also use the Transformation Tool to manipulate data. For more information about the Transformation Tool, see [Guide to Data Transformation](http://edocs.bea.com/workshop/docs81/doc/en/integration/dtguide/dtguideIntro.html) available at <http://edocs.bea.com/workshop/docs81/doc/en/integration/dtguide/dtguideIntro.html>.


The next step in this example is to configure the Respond to request node.

To Configure the Respond to Request Node

1. Double-click the **Respond to request** node.
2. Rename the method name to **onInvoice**.
3. Remove the **XmlObject payload** parameter type.
4. Add a parameter of type **Invoice** from the **b2binvoice.xsd** schema.
5. Click **Apply**.
6. Click **Send Data**.
7. Select **invoice** from the **Select variables to assign** drop-down list.
8. Click **Apply**.
9. Click **Close**.

You have completed building the seller (participant) side of this example. The seller will receive an order document from the buyer, write it out to file, read in an invoice file and send it to the buyer. The name of the ebXML service created for this transaction is RoundtripSeller. The message exchange will be carried out as a single ebXML conversation. The Trading Partner Integration (TPI) system automatically manages the ebXML conversation and ensures that the same conversation ID is used within the same participant process instance.

To Deploy the RoundtripSeller Business Process

1. Click  on the WebLogic Workshop menu to deploy the business process.
2. Close the Test Browser window.

Testing the process at this point would create an exception on the callback method since you have not yet created the initiator process. That is why you only deploy the process so that it will run properly when you test the initiator process later on.

Building the RoundtripBuyer Business Process

This section builds on the tasks that you have learned in previous examples. If at any time you need more detailed descriptions, refer back to [“Building the Buyer Business Process” on page 2-17](#) while completing the following tasks:

- [“To Create the Business Process and Configure the Client Request Node” on page 2-42](#)
- [“To Create and Configure the ebXML Control” on page 2-42](#)
- [“To Create and Configure the ebXML Control Nodes” on page 2-43](#)
- [“To Create and Configure the Client Response Node” on page 2-44](#)
- [“To Specify the Trading Partner ID in an XML Variable” on page 2-44](#)
- [“To Test the Business Processes” on page 2-45](#)

To Create the Business Process and Configure the Client Request Node

1. In the **roundtrip** folder, create a new business process named **RoundtripBuyer.jpd**.
2. Double-click the *Starting Event*.
3. Select **Invoked via a Client Request**.
4. Click **OK**.
5. Double-click the **Client Request** node and configure as follows:
 - a. Change the method name to **startBuyer**.
 - b. Add a parameter type of type **Order** from the **b2border.xsd** schema
 - c. On the **Receive Data** tab, create a new variable named **order** of type **Order** and specify it to be assigned to the method.
 - d. Click **Apply**, then click **Close**.

To Create and Configure the ebXML Control

1. Add a new ebXML control and configure it as follows:

- **Variable name:** roundtripSellerControl
 - **New JCX name:** RoundtripSellerControl
 - **ebxml-service-name:** RoundtripSeller (matches the participant side)
 - Accept defaults for all other fields.
2. In the **Data Palette**, right-click on the **ebXML** control and select **Edit**.
 3. Right-click on the arrow pointing to **request** and select **rename**.
 4. Enter **processOrder** and press the enter key.
 5. In a similar fashion, rename **response** to **onInvoice**.
 6. Click **processOrder**.
 7. In `processOrder`, change `XmlObject` to `OrderDocument`.
 8. Press **Alt+Enter** to import the suggested package.
 9. In `onInvoice`, replace `XmlObject` with `InvoiceDocument`.
 10. Press **Alt+Enter** to import the suggested package.
 11. Save your work by selecting **File→Save** or enter **Ctrl+S**
 12. Close the ebXML Source View by clicking the x in the top right corner of the window.
- You have now completed the ebXML control configuration. In the next step, you create control nodes associated with the methods of your ebXML control.

To Create and Configure the ebXML Control Nodes

1. In the Data Palette, expand the ebXML control.
2. Drag and drop the **processOrder** method to the target that appears below the **Client Request** node.
3. Drag and drop the **order** variable onto the **processOrder** node.
4. Drag and drop the **onInvoice** callback to the target that appears below the **processOrder** node.
5. Double-click the **onInvoice** node.
6. Click **Receive Data**.

7. Select **Create New Variable...** from the **Select variables to assign** drop-down list.
8. Create a new variable named **invoice** of type **Invoice**.
9. Click **OK**.
10. In the node builder, click **Apply**.
11. Click **Close**.

You have completed adding the ebXML controls. The next step is to add a Client Response node and configure it to return the contents of the invoice variable.

To Create and Configure the Client Response Node

1. Drag and drop a **Client Response** node from the **Palette** to the target that appears below the **onInvoice** node.
2. Double-click the Client Response node.
3. Change the method name to **onBuyerComplete**.
4. Add a parameter type of type **Invoice** (under b2binvoice.xsd).
5. Click **Send Data**.
6. Select the **invoice** variable from the **Select variables to assign** drop-down list.
7. Click **Apply**.
8. Click **Close**.

The buyer side of this example is almost complete, but you have not yet specified the ID of the trading partner. In the previous examples, you provided it as a static value in the ebXML control and also configured the control to pick the ID out at runtime from the payload. In this example, you specify the ID in a process XML variable using an XQuery selector.

To Specify the Trading Partner ID in an XML Variable

1. In the **Data Palette**, click on **roundtripSellerControl**.
2. In the **Property Editor**, locate the **ebxml** attribute section.

In the previous examples, we specified the ebXML attributes in the control definition (the JCX file). The values of those attributes apply wherever the control is used. Here you specify the control instance attributes, which only apply to the instance of the control declared in this business process. For more information about control properties for control

types and control instances, see [Setting Control Properties](http://edocs.bea.com/workshop/docs81/doc/en/integration/wfguide/wfguideControlsProperties.html) available at <http://edocs.bea.com/workshop/docs81/doc/en/integration/wfguide/wfguideControlsProperties.html>

3. In the **ebxml-service-name** field, enter **RoundtripSeller**.

Note: This value has to be specified again although it exists in the controls definition JCX file. This is because any required fields in the configuration will not be inherited from the controls definition file but instead needs to be specified in each instance of the control.

4. In the **to-selector** field, click on....

The **Property Editor** window opens with a graphical representation of the schemas of the typed variables in the business process.


5. Click on **Supplier_ID** in the order schema.

An selector XQuery is generated in the Selected Element pane. Based on this XQuery, the trading partner ID will be determined at runtime.

6. Click **OK**.

The buyer side is now complete. You can run and test your processes using the WebLogic Test Browser.

To Test the Business Processes

1. Click  on the WebLogic Workshop menu. (For more detailed instructions refer back to, “[To Test the Seller Process](#)” on page 2-16.)
2. To enter test data, cut and paste the content of `C:\myapps\tptutorialapps\ebXML\ebXMLWeb\sampladata\sampleorder.xml` into the xml variable (payload) field.
3. Confirm that the **Supplier_ID** field in the test data has the correct trading partner ID (TP2-id).
4. Click the Refresh link to see the callback method.
5. Confirm that the `order.xml` file was written out correctly to the `C:\tptutorial` directory by checking that the time stamp of the file is current.

Step 6: Implementing the Public/Private Pattern

This example illustrates how to use subprocesses to implement the public/private pattern. You can use this pattern to keep the details of backend integration confined to a private process definition and keep the public process definitions dedicated to trading partner interaction. This example contains the following procedures:

- [“Building the PublicBuyer Business Process” on page 2-46](#)
- [“Building the PrivateBuyer Business Process” on page 2-47](#)
- [“Building the PrivateSeller Business Process” on page 2-48](#)
- [“Testing the Public/Private Pattern Example” on page 2-51](#)

Building the PublicBuyer Business Process

In this example we reuse some of the files that you have created in the previous examples.

1. Create a new folder under the **ebxml** folder in the **Application** pane named **publicprivate**.
2. Click on the **RoundtripBuyer.jpd** in the **roundtrip** folder and drag it to the **publicprivate** folder while holding down your **Ctrl** key.
3. Release your mouse button.

A copy of the **RoundtripBuyer.jpd** file is created in the **publicprivate** folder.

Note: Whenever you duplicate a business process file, the new copy still reference the same (old) controls, even if you duplicate folders which include control files.

4. Right-click on the new business process and select rename.

Note: You may have to stop the WebLogic Server before you can rename the business process. To do so, select **Tools→WebLogic ServerStop→WebLogic Server**.

5. Enter **PublicBuyer.jpd**.
6. Right-click on **PublicBuyer.jpd** and select **Generate Process Control**.

A new JCX control definition file is created in your publicprivate folder. This control can be used in other processes to invoke the PublicBuyer.jpd business process. You use this control in the PrivateBuyer business process to invoke the PublicBuyer process.

Building the PrivateBuyer Business Process

This section builds on the tasks you learned in previous examples. If at any time you need more detailed information, refer back to [“Building the Buyer Business Process” on page 2-17](#) while completing the following tasks:

1. Create a new business process in the **publicprivate** folder named **PrivateBuyer.jpdl**.
2. Configure the process to be invoked by a **Client Request** node.
3. Configure the **Client Request** node according to the following:
 - **Method name:** startPrivateBuyer
 - **Expected parameter type:** Order from the b2border.xsd schema.
 - **Receive data tab:** create a new XML variable named **order** of type **Order** and specify it to be assigned to the method.
4. Add a **Client Response** node.
5. Configure the **Client Response** node according to the following:
 - **Method name:** onPrivateBuyerComplete
 - **Expected parameter type:** Invoice from the b2binvoice.xsd schema.
 - **Send Data tab:** create a new XML variable named **invoice** of type **Invoice** and specify it to be assigned to the method.
6. Drag and drop the **PublicBuyerControl.jcx** from the **Application** palette onto the **Design View**.
 An instance of the **publicBuyerControl** is added to your **Controls** list in the **Data Palette**.
7. Expand the new process control.
8. Drag and drop the **startBuyer** method onto the target that appears after the **Client Request** node.
9. Drag and drop the **order** variable onto the **startBuyer** node.
10. Drag and drop the **onBuyerComplete** method onto the target that appears after the **startBuyer** node.
11. Drag and drop the **invoice** variable onto the **onBuyerComplete** node.
12. Save your work.

The buyer side business processes are now complete.

Building the PrivateSeller Business Process

If at any time you need detailed instructions while completing the tasks in the section, refer back to [“Building the Seller Business Process” on page 2-10](#).

1. In the publicprivate folder, create a new business process named **PrivateSeller.jpdl**.
2. Configure it to be invoked by a **Client Request** node.
3. Configure the **Client Request** node as follows:
 - **Method name:** privateProcessOrder
 - **Expected variable type:** Order from the b2border.xsd schema.
 - **Receive Data tab:** Create a new variable named order of type Order and specify it to be assigned to the method.
4. Add a **Client Response** node to the business process, following the **Client Request** node.
5. Configure the **Client Response** node as follows:
 - **Method name:** onPrivateInvoice
 - **Expected variable type:** Invoice from the b2binvoice.xsd schema.
 - **Send Data tab:** create a new variable named invoice of type Invoice and specify it to be assigned to the method.

There are several options for processing a private process. For example, you may choose to send the order to an ERP system (most likely having transformed it first) via an Application View control, and then subscribe to an event from the backend system with invoice information. Since this example is for illustration purposes only, you use a simple file control to read and write the data from and to the file directory system.

6. Drag and drop **File.jcx** from the oneway folder in the **Application** pane onto the **Design View**.
7. Drag and drop the **write** method onto the target that appears below the **Client Request** node.
8. Configure the node to send the `order` variable to the `write` method.
9. Drag and drop **InvoiceFile.jcx** from the **roundtrip** folder in the **Application** pane to the **Design View**.

10. Drag and drop the **read** method to the target that appears below the **write** node.

11. Double-click the **read** node.

12. Click **Receive Data**.

The method expects untyped XML (XmlObject) but our invoice.xml file contains typed XML so we will need to do a simple transformation between the two.

13. Select the **Transformation** Option.

14. Click **Select Variable** and select **invoice** from the drop-down menu.

15. Click **Create Transformation**.

The **Transformation Tool** window opens.

16. Drag and drop the **XmlObjectDoc** in the **Source Schema** pane to the **Invoice** root in the **Target Schema** pane.

An XQuery transformation is created a corresponding transformation control is created and stored in the **PrivateSellerTransformation.dtf** file, which was created in your **Application** pane when you opened the Transformation tool.

17. Close the Transformation Tool window by clicking on the x in the top right corner of the window.

18. Click **Yes** in the **Save Before Closing** dialogue window.

19. Click **Close** in the node builder.

20. Save your work.

21. Right-click on **PrivateSeller.jpdl** and select **Generate Process Control**.

You have completed building the PrivateSeller business process. In the next section, you will create the public process for the seller side of the conversation.

Building the PublicSeller Business Process

To create the PublicSeller business process, you re-use some of the files that you created in the previous example. You can refer back to the detailed instruction in [“Building the Seller Business Process” on page 2-10](#) to complete the tasks in this section.

1. Drop and drag the **RoundtripSeller.jpdl** file from the **roundtrip** folder to the **publicprivate** folder while holding down your ctrl key.



2. Right-click on the new process file and select rename.
3. Enter **PublicSeller.jpdl**.
4. Double-click on **PublicSeller.jpdl**.
5. Click the **Source View** tab.
6. In the **Property Editor**, change the **ebxml-service-name** to **PublicSeller**.
7. Click **Design View**.
8. Delete the **write** node.
9. Delete the **read** node
10. In the **Data Palette** Controls section, delete all three control instances (file, invoiceFile, and transformations).
11. Drag and drop the **PrivateSellerControl.jcx** file from the **Application** pane onto **Design View**.
12. Expand the **privateSellerControl** in the **Data Palette**.
13. Drag and drop the **privateProcessOrder** method onto the target that appears just below the **Receive request** node.
14. Drag and drop the **order** variable onto the **privateProcessOrder** node.
15. Drag and drop the **onPrivateInvoice** method onto the target that appears just below the **privateProcessOrder** node.
16. Drag and drop the **invoice** variable onto the **onPrivateInvoice** node.

You have completed the seller side of the conversation. The public process receives a message via ebXML and passes it on to the private process. When the private process responds, it calls back to the public process, which returns the invoice information via ebXML to the initiating trading partner.

You need to do one final adjustment before running and testing this example:

1. Double-click on **PublicBuyer.jpdl**.
2. In the **Data Palette**, select the **roundtripSellerControl**.
3. In the **Property Editor**, change the **ebxml-service-name** to **PublicSeller** to match the participant side.

Testing the Public/Private Pattern Example

1. Open **PublicSeller.jpdl**.
2. Click  on the WebLogic Workshop menu. (For more detailed instructions refer back to, [“To Test the Seller Process” on page 2-16.](#))
3. Switch back to WebLogic Workshop and open **PrivateBuyer.jpdl**.
4. Click  on the WebLogic Workshop menu.
5. To enter test data, do one of the following:
 - Click **Browse**, and navigate to
`C:\myapps\tptutorialapps\ebXML\ebXMLWeb\sampladata\sampleorder.xml`.
 - Cut and paste the content of
`C:\myapps\tptutorialapps\ebXML\ebXMLWeb\sampladata\sampleorder.xml`
 into the xml variable (payload) field.
6. Refresh the browser after a few seconds. The log entry for
`callback.onPrivateBuyerComplete` will have the content of the response with the Invoice document.
7. Confirm that the `order.xml` file was written out correctly to the `C:\tptutorial` directory by checking that the time stamp of the file is current.

Step 7: Using the TPM Control and Callbacks

This example illustrates how to obtain trading partner information from the Trading Partner Management (TPM) repository at runtime by using the TPM control. It will also show you how to use the `onAck` callback of the ebXML control. In the example you send an ebXML message and wait for the ebXML acknowledgement. If the acknowledgement times out, for example, because the remote partner is offline, and the TPM protocol binding for both local and remote trading partners have “once and only once” or “at least once” delivery schematics, an email alert is sent to that remote trading partner. You obtain the email address by querying the TPM database using a TPM control.

This example contains the following sections:

- [“Building the BuyerAlert Business Process” on page 2-52](#)
- [“Testing the BuyerAlert Business Process” on page 2-56](#)

Building the BuyerAlert Business Process

Use the skills you have learned in previous sections to complete the following tasks:

- [“To Create the BuyerAlert Business Process” on page 2-52](#)
- [“To Create and Add the TPM Control to the Process” on page 2-53](#)
- [“To Create and Add an Email Control to the Business Process” on page 2-53](#)
- [“To Specify the Target Trading Partner ID” on page 2-55](#)
- [“To Specify the Email Address for the Trading Partner Profile” on page 2-55](#)

To Create the BuyerAlert Business Process

1. Create a new folder named **tpm** off the root (ebxmlWeb) of your project.
2. Add a new process file named **BuyerAlert.jpdl**.
3. Configure the business process to be invoked by a **Client Request** node.
4. Configure the **Client Request** node as follows:
 - **Expected Parameter Types:** XmlObject and String
 - **Receive Data tab:** Create a new XmlObject variable named payload and a new String variable named partnerId. Select them to be assigned to the corresponding methods.
5. Add an ebXML control and configure it as follows:
 - **Variable name:** sellerControl
 - **New JCX file:** SellerControl.jcx
 - **ebxml-service-name:** Nonesuch (use any value that does not correspond to an actual deployed participant service)
 - Accept defaults for all other fields
6. From the **Data Palette**, drag and drop the **request** method of the sellerControl onto the target that appear below the **Client Request** node.
7. Drag and drop the **payload** variable onto the **request** node.
8. Drag and drop the **onAck** callback onto the target that appears below the **request** node.

This callback will occur when the ebXML message sent in the previous step is acknowledged. In case the acknowledgement does not arrive, you add logic in the next steps that alerts the trading partner.

9. Right-click on the **onAck** node.
10. Select **Add Timeout Path** from the drop-down menu.
11. In the **Property Editor** for the timer, provide a **duration** value of 10 seconds by typing **10s**.

To Create and Add the TPM Control to the Process

1. Create a new **TPM** control by selecting **Add→Integration Controls→TPM** in the Controls section of the **Data Palette**.
2. Enter **tpm** as the variable name.
3. Click **Create**.

For more information about the TPM control, see [TPM Control](http://edocs.bea.com/workshop/docs81/doc/en/integration/controls/controlsTPM.html) available at <http://edocs.bea.com/workshop/docs81/doc/en/integration/controls/controlsTPM.html>

4. Expand the new **TPM** control.
5. Drag and drop the `TradingPartnerDocument getBasicProperties(String partnerId)` onto the target that appears in the timeout path below the timer.
6. Double-click the **getBasicProperties** node.
7. Click **Send Data**.
8. Select **partnerId** from the **Select variables to assign** drop-down list.
9. Click **Receive Data**.
10. Create a new variable named **basicPartnerInfo** of type **TradingPartnerDocument** and specify it to be assigned to the method.
11. Click **Apply**.
12. Click **Close**.

To Create and Add an Email Control to the Business Process

1. Create a new **Email** control by selecting **Add→Integration Controls→Email** in the Controls section of the **Data Palette**.

2. Configure the **Email** control as follows:

- **Variable name:** alert
- **New JCX file name:** Alert
- **SMTP host:** <your SMTP host name, e.g. mail.mycompany.com>
- **from-address:** <a reply address, e.g.admin@mycompany.com>
- **from-name:** <Name, e.g. Administrator>
- **body-type:** String

For more information about the Email control, see [Email Control](http://edocs.bea.com/workshop/docs81/doc/en/integration/controls/controlsEmail.html) available at <http://edocs.bea.com/workshop/docs81/doc/en/integration/controls/controlsEmail.html>.

3. Expand the new **Email** control.
4. Drag and drop the `sendEmail` method onto the target that appears after the **getBasicProperties** node in the timeout path.
5. Double-click the **sendEmail** node.
6. Click on **Send Data**.
7. Select the **Transformation** option.
8. Click **Select Variable** and select **basicPartnerInfo**.
9. Click **Create Transformation**.
10. Drop and drag the **email** leaf from the **Source Schema** onto the **arg1** (corresponds to the email 'To' field) leaf in the **Target Schema**.
11. Right-Click on **arg4** (corresponds to the email 'Subject' field).
12. Select **Create Constant**.
13. Enter **Problem Sending Order**.
14. Click **OK**.
15. Right-Click on **arg5** (corresponds to the email body).
16. Select **Create Constant**.
17. Enter **We encountered a problem while sending you an order**.
18. Click **OK**.

19. Close the **Transformation Tool** by clicking the **x** in the top right corner.
20. Click **Yes** in the **Save Before Closing?** dialogue.
21. Click **Close** in the node builder.

The last step in this example is to specify the ID of the target trading partner in our ebXML control. In this example, you use the **setProperties** method of the control to accomplish this.

To Specify the Target Trading Partner ID

1. Select the **setProperties** method under the ebXML control in the Data Palette.
2. Drop and drag the method onto the target that appear between the Client Request node and the request node.
3. Double-click on the **setProperties** node.
4. Click **Send Data**.
5. Select the **Transformation** option.
6. Select **partnerId** as the variable.
7. Click on **Create Transformation**.
8. Drag the **string variable** from the Source Schema pane to the **'to'** element in the Target Schema pane.
9. Close and Save your transformation.
10. Close the node builder


Before you can test the example, you need to specify the email address in the trading partner profile.

To Specify the Email Address for the Trading Partner Profile

1. If not already running, start your WebLogic Server by selecting **Tools→WebLogic Server→Start WebLogic Server**.
2. After the Server is running, open the WebLogic Administration Console by selecting **Tools→WebLogic Integration→WebLogic Administration Console**.
3. Log into the console using your server user name and password you specified in the [“Step 1: Create a New WebLogic Integration Domain”](#) on page 1-3 section.

4. Click **Trading Partner Managment**.
5. Click **Profile Management**.
6. Click **Test_TradingPartner_2**.
7. Click **Edit profile**.
8. In the **Email** field, enter you email. This is where the alert messages will be deliver.
9. Click **Submit**.
10. Close the console.

Testing the BuyerAlert Business Process

1. Click  on the WebLogic Workshop menu to test the **BuyerAlert** business process.
2. Specify any XML data as the payload.
3. Enter **TP2-id** for the string parameter.
4. After the process completes, check your email for the alert message.

These emails where sent since an error occurred and triggered the email control to send an email to the address you specified when you set up the control.

Step 8: Setting Partner ID Dynamically Based on Directory Name

In this example you look at the already built application which was distributed with the zipped archive that contained this tutorial (see, [“Step 3: Install the Tutorial Files” on page 1-9](#)). This example demonstrates how to use the name of a sub-directory on your hard drive as the partner ID and set it dynamically using the setProperties method in the ebXML control. This example contains the following sections:

- [“Reviewing the Initiator Side of the Example” on page 2-56](#)
- [“Reviewing the Participant Side of the Example” on page 2-59](#)

Reviewing the Initiator Side of the Example

In this example you begin with looking at the initiator side of the application. Use the skills you learned in previous sections to complete the following tasks:

- [“To Set Up the Initiator Side of the Example” on page 2-57](#)
- [“To Review the Configuration of the DynamicBinaryBuyer Process” on page 2-57](#)
- [“To Add a New Rule in the File Event Generator” on page 2-58](#)
- [“To Test the DynamicBinaryBuyer Process” on page 2-59](#)

To Set Up the Initiator Side of the Example

1. Create the following two directories on your hard drive:
 - C:\tptutorial\binary-out\dynamic\TP2-id
 - C:\tptutorial\binary-in\dynamic\TP1-id
2. Open up your file browser and navigate to
3. Select **File→Open→Application**.
4. Locate the **advanced** folder in the [unzip location]\tptutorialapps\ebxml\ebxmlWeb\ebxml\oneway\binary directory where [unzip location] is the location where you unzipped the tutorial archive in [“Step 3: Install the Tutorial Files” on page 1-9](#).
5. Drag and drop the **advanced** folder from the file browser onto the **binary** folder in your **Application** tab.

WebLogic Workshop will update your class and package names.
6. In the **Application** pane, navigate to and expand the ebxml\ebxmlWeb\ebxml\oneway\binary\advanced folder.
7. Double-click on **DynamicBinaryBuyer.jpdl**

DynamicBinaryBuyer.jpdl opens in Design View.

To Review the Configuration of the DynamicBinaryBuyer Process

1. This application uses the name of a sub directory as the partner ID and sets it dynamically using the setProperties method in the ebXML control. You can double-click on any of the nodes to note the following:
 - The business process is invoked by a RawData message coming through on the channel which the starting event subscribes to through the **Subscription** node.
 - The **Extract partner id** node contains custom Java code which extracts the partner id from the incoming message. To learn about writing custom Java code in business

processes, see [Writing Custom Java in Perform Nodes](http://edocs.bea.com/workshop/docs81/doc/en/integration/wfguide/wfguidePerform.html) available at <http://edocs.bea.com/workshop/docs81/doc/en/integration/wfguide/wfguidePerform.html>.

- The **setProperties** node takes the trading partner ID and assigns it to a string variable which corresponds to the ‘to’ parameter of the ebXML envelope.
 - The **request** sends the message to the trading partner.
2. In the **Application** pane, double-click on the `ebxml\Schemas\tptutorial.channel` file. The channel file opens in Source View.
 3. Add the following line to your channel file:

```
<channel name="TutorialChannel" messageType="rawData"
qualifiedMetadataType="eg:FileEventGenerator"/>
```

This line specifies a new channel which listens messages that are sent using event a File event generator. All event generators and some applications send metadata with messages.


The next step in the process is to add a new rule in the File Even Generator in the WebLogic Integration Administration Console to listen to the correct folder on your hard drive.

To Add a New Rule in the File Event Generator

1. If not already running, start your WebLogic Server by selecting **Tools→WebLogic Server→Start WebLogic Server**.
2. After the Server is running, open the WebLogic Administration Console by selecting **Tools→WebLogic Integration→WebLogic Administration Console**.
3. Log into the console using your server user name and password you specified in the “[Step 1: Create a New WebLogic Integration Domain](#)” on page 1-3 section.
4. Click **Event Generators**.
5. Click **TPTutorial** in the **Name** column.
6. Click **Define a New Channel Rule**.
7. Enter the following parameters:
 - **File Type**: Disk File
 - **Channel Name**: /SamplePrefix/Samples/TutorialChannel (rawData)

- **Message Encoding:** *leave blank*
 - **Directory:** /tptutorial/binary-out/dynamic
 - **Pass by file name:** No
 - **Scan subdirectories:** Yes
 - **File pattern:** *leave blank*, it defaults to *.*
 - **Sort by Arrival?:** No
 - **Polling interval:** 3 seconds (small enough to avoid a long wait)
 - **Read limit:** 0
 - **Post Read Action:** Delete
 - **Archive Directory:** *leave blank*
 - **Error Directory:** /tptutorial/errors
 - **Description:** Raw data file.
8. Click **Submit**.
 9. Close the WebLogic Integration Administration Console.
- You are now ready to build and test the application.

To Test the DynamicBinaryBuyer Process

1. Click  on the WebLogic Workshop menu to build and deploy the DynamicBinaryBuyer.jpd business process.
2. Place a binary file in the C:\tptutorial\binary-out\dynammic\TP2-id directory.

The file disappears is delivered to the trading partner.

You have completed the steps for the initiator side of this example.

Reviewing the Participant Side of the Example


The next step involves reviewing the participant side. Use what you have learned in previous examples to complete the following tasks:

- [“To Review the Participant Side of the Example” on page 2-60](#)
- [“To Test the Example” on page 2-60](#)

To Review the Participant Side of the Example

1. Double-click on **DynamicBinarySeller.jpdl**
The **DynamicBinarySeller.jpdl** opens in **Design View**.
2. Double-click on the **Receive request** node and note that there are two parameter types; one for the payload and the other for the ebXML envelope.
3. Close the node builder.
4. Right-Click on the **Receive request** node and select **View Code**.
5. Select the **request** method and look at the **Property Editor** parameter entered for the **ebxml-method envelope**. The **{env}** parameter indicates that the ebXML envelope will be placed in the second parameter, **env**.
6. Click the **Design View** tab.
7. In the **Data Palette**, select the **baseDirectory** variable. In the **Property Editor**, note that the value of the variable is `"/tptutorial/binary-in/dynamic/"`. This is the base directory from which the application will read the sub-directory name and use it as the trading partner ID.
8. The remaining two nodes:
 - **setProperty**—invokes the `setProperty` on the file control with a transformation that concatenates the “From” ID from the ebXML envelope and the directory base string. To review the transformation, double-click on the **DynamicBinarySellerfileSetProperties.xq** in the **Application** tab.
 - **write**—this node reads and writes the binary files from and out to the directories you created on your hard drive.

To Test the Example

1. Click  on the WebLogic Workshop menu to run and deploy the process.
2. Drop a file into the `C:\tptutorial\binary-out\dynamic\TP2-id` directory and watch it appear in a few moments in the `C:\tptutorial\binary-in\dynamic\TP1-id` directory.

Step 9: Creating a Distributed Setup

In all of the examples so far, the initiator (TP1-id) and the participant (TP2-id) are both located in the same WLI instance and are using the same database repository. In practice, the two sides will operate on two physically separated systems. This examples briefly describes how to move to a distributed setup where TP1-id operates in one WebLogic Integration instance and the TP2-id in another.

1. Install and configure a second WebLogic instance on another machine. (For detailed instructions, refer back to [“Step 1: Create a New WebLogic Integration Domain”](#) on page 1-3.
2. In the **WebLogic Integration Administration Console** navigate to the **Trading Partner Management**→**Profile Management**, for the first WebLogic Integration Server, modify the Test_Trading Partner_2 (TP2-id) trading partner information as follows:
 - a. Change Test_TradingPartner_2(TP2-id) to **remote** instead of **local**.
 - b. Modify the ebXML 2.0 binding by changing the **EndPoint URL** to include the correct server/port of the second WebLogic Integration server. The examples use ebXML 2.0 binding by default, modify this to other bindings if you use them.
3. For the second WebLogic Integration server modify the Test_TradingPartner_1 (TP1-id) as follows:
 - a. Deselect the **default** option.
 - b. Change **local** to **remote**.
 - c. Modify the ebXML 2.0 binding by changing the **EndPoint URL** to include the correct server/port of the first WebLogic Integration server.
4. For the second WebLogic Integration server modify the Test_TradingPartner_2 (TP2-id) as follows:
 - a. Select the default option.
 - b. Modify the ebXML 2.0 binding by changing the **EndPoint URL** to include the correct server/port of the second WebLogic Integration server.
5. On each of the servers, start WebLogic Workshop and open the tutorial application.
6. Navigate to **Tools**→**Application Properties**→**WebLogic Server** and ensure that the **Server Home Directory** field is set to the correct path.

7. Before testing any of the examples: use the WebLogic Test Browser to run the participant side processes to force them to build and deploy.

Step 10: Configuring Non-Default Protocol Settings

In the previous examples, default values were used for the initiator trading partner and the protocol information. The default values were used as follows:

- **from field** (initiator ID)—trading partner which is marked as default in the WebLogic Integration Administration console, TP1-id in the previous examples.
- **Protocol information**—taken from the binding which is marked as default for each business protocol (ebXML MS 2.0; RNIF 1.1, 2.0).

In production scenarios, you will typically add a service and service profile entry which gives you control over the trading partner communication. This example illustrates how to add a service and a service profile by using the Seller and Buyer business processes which you created in [“Step 2: Sending an XML Message through an One-Way ebXML Exchange” on page 2-9](#). It contains the following sections:

- [“Configuring the Participant Side” on page 2-62](#)
- [“Configuring the Initiator Side” on page 2-63](#)

Configuring the Participant Side

The Seller business process is the participant of the ebXML communication.

1. If not already running, start your WebLogic Server by selecting **Tools→WebLogic Server→Start WebLogic Server**.
2. After the Server is running, open the WebLogic Administration Console by selecting **Tools→WebLogic Integration→WebLogic Administration Console**.
3. Log in to the console using the server user name and password that you specified in the [“Step 1: Create a New WebLogic Integration Domain” on page 1-3](#) section.
4. Click **Trading Partner Management**.
5. Click **Service Management**.
6. In the **Services** panel, click **Create New**.
7. Next to the **Name** field, click **Browse**.

A list of all participant processes which are currently deployed on your server is displayed.

8. Click on `ebxmlWeb\ebxml\oneway\Seller.jpdl`

The **Add Service** window opens.

9. Click **Add Service**.

The **View And Edit Service Details** window opens.

10. Click **Add Service Profile**.

The **Add Service Profile** window opens.

11. In the **Name** row, select **Test_TradingPartner_2** as the **LOCAL** trading partner. This is the partner actually hosting the Seller service.

12. In the **Name** row, select **Test_TradingPartner_1** as the **REMOTE** trading partner. This is the partner who will use the service.

13. Select **ebXML 2.0** bindings for both the **LOCAL** and **REMOTE** binding.

14. Click **Submit**.

Configuring the Initiator Side

For the initiator side, service configuration information is added to the ebXML control that represents the remote service.

1. Go back to the **Service Management** page by clicking on **Service Management** in the navigation tool bar.

2. In the **Services** pane, click **Create New**.

3. Next to the name field, click **Browse**.

A list of currently deployed participant business processes on your server and controls which are referenced from currently deployed processes is displayed.

4. Click on `ebxml.oneway.SellerControl`.

5. Click **Add Service**.

6. Click **Add Service Profile**.

7. In the **Name** row, select **Test_TradingPartner_1** as the **LOCAL** trading partner. This is the initiator partner on the local system.

8. In the **Name** row, select **Test_TradingPartner_2** as the **REMOTE** trading partner. This is the remote partner who hosts the actual service.
9. Select **ebXML 2.0** bindings for both the **LOCAL** and **REMOTE** binding.
10. Click **Submit**.

You have now completed setting up the protocol settings. For each of the examples in this tutorial, you can create a corresponding service entry for each participant and initiator side. There is no limit to how many service profiles you enter, you can create as many as you need for each service entry to reference different protocol bindings.

Step 11: Exchanging Messages with WebLogic Integration - Business Connect

WebLogic Integration - Business Connect is a separate application which enables you to securely exchange large volumes of documents with your trading partners. The application packages documents in secure envelopes that are transmitted among trading partners according to schedules.

Note: You must have WebLogic Integration - Business Connect installed to complete this example.

To learn more about WebLogic Integration - Business Connect, see the [Using WebLogic Integration - Business Connect](http://edocs.bea.com/wlibc/docs81/admin/index.html), available at <http://edocs.bea.com/wlibc/docs81/admin/index.html>.

This example contains the following sections:

- “[Configuring WebLogic Integration - Business Connect with Trading Partner Information.](#)” on page 2-65
- “[Configuring WebLogic Integration with Trading Partner Information](#)” on page 2-66
- “[Setting Up a Trading Partner Service Profile](#)” on page 2-67
- “[Sending a Message from WebLogic Workshop to WebLogic Integration - Business Connect](#)” on page 2-69

Configuring WebLogic Integration - Business Connect with Trading Partner Information.

To avoid typing the trading partner information by hand, you can export it directly from WebLogic Integration and then import it into WebLogic Integration - Business Connect.

To Export the Trading Partner Information

1. If not already running, start your WebLogic Server by selecting **Tools→WebLogic Server→Start WebLogic Server**.
2. After the Server is running, open the WebLogic Administration Console by selecting **Tools→WebLogic Integration→WebLogic Administration Console**.
3. Log in to the console using the server user name and password that you specified in the [“Step 1: Create a New WebLogic Integration Domain” on page 1-3](#) section.
4. Navigate to **Trading Partner Management→Partner Profile Import/Export**.
The **Import Trading Partner Management Data** window opens.
5. In the **Import/Export** pane, select **Export**.
6. Select the **Trading Partner** option.
7. Click **Browse** next to **Trading Partner**.
8. Deselect **Test_TradingPartner_2**.
9. Click **Done**.
10. For the **Format** option, select **Business Connect**.
11. Click **Export**.
12. If a **File Download** dialogue opens, click **Save**.
13. In the **Save As** window navigate to the location which you want to save the exported file to.
14. Enter a filename and click **Save**.

Note: Remember the navigation path to the file or copy it to the clipboard. You will need this in the next procedures.

To Import the Trading Partner Information

1. Start the Weblogic Integration - Business Connect Administration tool by selecting **Start→Programs→BEA WebLogic Integration - Business Connect 8.1→Administrator**.
2. Use Administrator as the user name and leave the password field blank.
3. Click **Partner Profiles**.
4. Right-click in the white area and select **Import** from the menu.
5. Enter the path to the exported file location or paste it from the clipboard.
6. When prompted to select active protocol, click **OK**.

You have now imported the information about the trading partner hosted on the WebLogic Integration server into WebLogic Integration - Business Connect. The next step is to configure WebLogic Integration with the trading partner information of the partner which is using WebLogic Integration - Business Connect.

Configuring WebLogic Integration with Trading Partner Information

In this section you create a trading partner in WebLogic Integration - Business Connect and export the trading partner profile to a file which will be imported into WebLogic Integration.

To Create and Export the Trading Partner Profile

1. In the **Weblogic Integration - Business Connect Administration** tool, click **Company Profiles**.
2. Click **New** and enter **BCTradingPartner** as the **name** and **BCTP-id** as the **ID**.
3. Fill out the basic profile contact information.
4. In the **Inbound Protocols** tab, add **ebXML** to the list.
5. Click **OK**.
6. In the **Do you want to setup a certificate for this company?** window, click **No**. Since we are not testing the security in this example, there is no need to set up a certificate.
7. Right-click on the profile entry and select **Export** from the menu.
8. Accept the defaults.

9. Remember the path to the export file location or copy it to the clipboard.
10. Click **OK**.

To Import the Trading Partner Profile

1. On the **WebLogic Integration Administration Console**→**Trading Partner Management**→**Partner Profile Import/Export** screen, in the **Import/Export** pane, click **Import**.
2. In the **File Name** field, enter the location to the file you exported from WebLogic Integration - Business Connect or paste the path from the clipboard.
3. Select **Business Connect** as the **Import Format**.
4. Click **Import**.

You have now completed the configuration of WebLogic Integration and WebLogic Integration - Business Connect. If for any reason you have to create the WebLogic Integration - Business Connect trading partner by hand in the WebLogic Integration Administration Console, use the following values:

- **TP Name:** BCTradingPartner.
- **ID:** BCTP-id
- **Default:** false
- **LOCAL/REMOTE:** REMOTE
- **Binding Information:** add ebXML 2.0 with the following settings:
 - **Name:** ebxml-20
 - **Default binding:** true
 - **Transport:** HTTP, version: 1.1
 - **Endpoint:** http//*host*:4080/exchange/BCTP-id where *host* is the IP address of your machine.
 - **Delivery semantics:** Once and only once, retry count: 3

Setting Up a Trading Partner Service Profile

In this example you set up a trading partner service profile between the TP1-id trading partner in WebLogic Integration and the BCTP-id trading partner in WebLogic Integration - Business

Connect. Service profiles specify the protocol binding and URL endpoints for the local and remote trading partners that offer and call the service.

To Set Up the Trading Partner Service Profile

1. If not already running, start your WebLogic Server by selecting **Tools→WebLogic Server→Start WebLogic Server**.
2. After the Server is running, open the WebLogic Administration Console by selecting **Tools→WebLogic Integration→WebLogic Administration Console**.
3. Log in to the console using the server user name and password that you specified in the [“Step 1: Create a New WebLogic Integration Domain” on page 1-3](#) section.

4. Navigate to **Trading Partner Management→Service Management**.

The **View and Edit Services** window opens.

5. In the **Services** panel, select **Create New**.

The **Add Service** window opens.

6. Click **Browse** (by the Name field).
7. The **Choose Service** window opens.
8. Select **ebxml.oneway.binary.BinarySellerControl**.

The **Add Service** window is displayed with the BinarySellerControl added to the Name field.

9. Click **Add Service**.

The **View and Edit Services** window is displayed with the new Service Details.

10. Click **Add Service Profile**.


The **Add Service Profile** window opens.

11. From the **Name LOCAL** drop-down menu, select **Test_TradingPartner_1**.
12. From the **Name REMOTE** drop-down menu, select **BCPartner**.
13. Click **Submit**.
14. Skip the authentication steps by selecting **No**.

You have completed setting up the trading partner service profile and are ready to begin the tutorial examples of this section.

Sending a Message from WebLogic Workshop to WebLogic Integration - Business Connect

In this section, you use one of the initiator processes that you created in an earlier example to send a message. Recall that the SelectorBuyer business process in “[Step 3: Selecting the Trading Partner Information Dynamically Through Typed XML](#)” on page 2-23 selects the trading partner ID from the incoming data. In this example you use the files from that example, but modify the incoming data with the correct trading partner ID to send the message to WebLogic Integration - Business Connect.

1. In the **Applications** tab, double-click on `ebxml\oneway\selector\SelectorBuyer.jpd`.
1. Click  on the WebLogic Workshop menu. (For more detailed instructions refer back to, “[To Test the Seller Process](#)” on page 2-16.)
2. To enter test data, cut and paste the content of
`C:\myapps\tptutorialapps\ebXML\ebXMLWeb\sampladata\sampleorder.xml` into the xml variable (payload) field.
3. In the test data pane, modify the **Supplier_ID** element to have the value **BCTP-id**.
4. Start the WebLogic Integration - Business Connect selecting **Start→Programs→BEA Weblogic Integration-Business Connect 8.1→Start Server**.

Note: For information about how to start the server on platforms other than Windows, see the [Getting Started](#) of *Using WebLogic Integration - Business Connect* available at http://edocs.bea.com/wlibc/docs81/admin/get_started.html.
5. Run the test.
6. Confirm that a file appeared in the inbound XML directory of your WebLogic Integration - Business Connect installation:
`[BC_HOME]\data\BCTP%2did\xml` in where `[BC_HOME]` is the directory in which you installed WebLogic Integration - Business Connect (such as `C:\WLI-BC`).

Tutorial: Building ebXML Solutions

Tutorial: Building RosettaNet Solutions

This topic describes how to implement RosettaNet solutions for trading partner integration in WebLogic Integration. It contains the following sections:

- [Before You Begin](#)
- [Tutorial Goals](#)
- [Tutorial Overview](#)
- [Tutorial Steps](#)
- [Implementing New PIPs Based on the Example PIPs](#)

RosettaNet is a business protocol that enables enterprises to conduct business over the Internet. To learn about RosettaNet, see <http://www.rosettanet.org>.

Tutorial Goals

This tutorial has the following goals:

- To demonstrate two common RosettaNet design patterns (asynchronous single-action and asynchronous two-action activity) that form the basis of most PIP implementations.
- To describe example WebLogic Workshop public business processes that implement two RosettaNet Partner Interface Processes (PIPs)—PIP 3B2 (an asynchronous single-action activity) and 3A4 (an asynchronous two-action activity).
- To describe the example private business processes, which are stubs that represent integration with back-end systems.
- To explain how to run the example PIPs, using example XML data, so that you can view them at runtime.
- To explain how you can easily implement business processes for other PIPs using the provided PIP implementations as starting points.

Before You Begin

This topic describes tasks that you should perform before you begin using this tutorial. It contains the following sections:

- [Prerequisites](#)
- [Suggested Reading](#)
- [Note About Converting RosettaNet DTDs to W3C Schemas](#)

Prerequisites

To use this tutorial, you must have:

- Installed WebLogic Platform 8.1 on your system according to the instructions in [Installing WebLogic Platform](#), which is available at <http://edocs.bea.com/platform/docs81/install/index.html>.
- Completed the tutorial setup procedure according to the instructions in “[Setting Up the Tutorials](#)” on page 1-2.

Suggested Reading

To gain a detailed understanding of how RosettaNet solutions are implemented in WebLogic Integration, consider reading the following material:

- For an overview of integrating trading partners using WebLogic Integration, see *Introducing Trading Partner Integration*, which is available at <http://edocs.bea.com/wli/docs81/tpintro/index.html>.
- For an introduction to RosettaNet solutions using WebLogic Integration, see *Introducing RosettaNet Solutions* (available at <http://edocs.bea.com/wli/docs81/tpintro/rosettanet.html>) in *Introducing Trading Partner Integration*.
- For more information about using the RosettaNet control in initiator business processes, see *RosettaNet Control* in the WebLogic Workshop Online Help, which is available at: <http://edocs.bea.com/workshop/docs81/doc/en/integration/controls/controlsRosettaNet.html>
- For information about using participant business processes, see the following topics in the WebLogic Workshop Help:
 - *Building RosettaNet Participant Business Processes*, which is available at: <http://edocs.bea.com/workshop/docs81/doc/en/integration/wfguide/wfguideRosettaNet.html>
 - *@jpd:rosettanet Annotation*, which is available at: http://edocs.bea.com/workshop/docs81/doc/en/integration/reference/refJpdAnnotations_rosettanet.html

To learn how to build business processes and create data transformations in WebLogic Workshop, consider completing the following tutorials:

- *Tutorial: Building Your First Business Process*, which is available at <http://edocs.bea.com/workshop/docs81/doc/en/integration/tutorial/tutWLIPProcessIntro.html>
- *Tutorial: Building Your First Data Transformation*, which is available at <http://edocs.bea.com/workshop/docs81/doc/en/integration/dttutorial/tutWLIDataTransIntro.html>

Note About Converting RosettaNet DTDs to W3C Schemas

The RosettaNet PIP message definitions are specified in DTD files. The graphical XQuery mapping tools in WebLogic Workshop require the W3C Schema format (XSD files). WebLogic Integration 8.1 supports the use of DTD files, and you can implement RosettaNet PIPs using DTD files.

However, if you want to use the graphical XQuery mapping tools to define data transformations, you need to convert the RosettaNet DTD files to W3C XSD files (using a tool like XML Spy, which ships with WebLogic Platform) and then import the XSD files into your project. For instructions on how to convert the files using XML Spy, see [“Converting RosettaNet DTD Schemas to XSD Schemas” on page 3-27](#).

Tutorial Overview

The RosettaNet example includes business process definitions (JPDs), schemas, and other files that illustrate how to implement RosettaNet PIPs using WebLogic Integration 8.1. The example business process files implement common design patterns and provide a head start for building initiator and participant business processes for RosettaNet conversations. Rather than build the business processes from scratch, you can easily adapt the example files to implement any PIP.

This topic contains the following sections:

- [PIPs Implemented In These Examples](#)
- [Folders in the RosettaNet Tutorial Application](#)
- [RosettaNet Design Patterns](#)

The PIPs implemented in this tutorial follow *RosettaNet Implementation Framework* (RNIF) 2.0.

PIPs Implemented In These Examples

The following table describes the RosettaNet examples that are provided in this tutorial:

Table 3-1 RosettaNet Examples

PIP	Specification	Description
0A1	http://www.rosettanet.org/PIP0A1	<p>Implements PIP 0A1: Notification of Failure. The example files are provided with the distribution but the tutorial does not walk through this implementation. Similar to the PIP 3B2 example, below, this is an example of an asynchronous single-action activity design pattern described in “RosettaNet Design Patterns” on page 3-7.</p> <p>You can initiate this business process from any other business process in which a RosettaNet notification of failure is required. For example, you can use this business process in conjunction with the RosettaNet Participant Business Process template provided in WebLogic Workshop. For more information, see Building RosettaNet Participant Business Processes in the WebLogic Workshop Help.</p>
3B2	http://www.rosettanet.org/PIP3B2	<p>Implements PIP 3B2: Advance Shipment Notice. Described in “Step 2: Open the PIP 3B2: Advance Shipment Notice Example” on page 3-9.</p>
3A4	http://www.rosettanet.org/PIP3A4	<p>Implements PIP 3A4: Purchase Order Request and Confirmation. Described in “Step 3: Open the PIP 3A4: Purchase Order Request and Confirmation Example” on page 3-16.</p>

Folders in the RosettaNet Tutorial Application

The following table describes the folders in the \tptutorialapps\rosettanet directory:

Table 3-2 Folders in \tptutorialapps\rosettanet

Directory	Description
PIP0A1Schema	Schema project for the PIP0A1 schema.
PIP3A4Schema	Schema project for the PIP3A4 schema.
PIP3B2Schema	Schema project for the PIP3B2 schema.
pips	Contains business process folders for each example PIP implementation (PIP0A1Processes, PIP3A4Processes, and PIP3B2Processes) and the private administrator alert notification (privateAdmin).
Schemas	Contains standard WebLogic Integration schemas.
Modules	Empty by default. Standard WebLogic Workshop folder for compiled Java code.
Libraries	Empty by default. Standard WebLogic Workshop folder for compiled Java code.
Security Roles	Empty by default. Standard WebLogic Workshop folder for defining security roles for this application.

Note: The PIP schemas were converted from the RosettaNet DTDs to W3C XSDs, as described in [“Note About Converting RosettaNet DTDs to W3C Schemas” on page 3-4](#). The PIP schemas are stored in separate schema projects to avoid element name collisions.

RosettaNet Design Patterns

RosettaNet PIPs follow one of the following design patterns:

Table 3-3 RosettaNet PIP Design Patterns

Directory	Description
Asynchronous single-action activity	<p>Involves a single action from both trading partners:</p> <ul style="list-style-type: none"> Initiator sends a business message to the participant. The participant sends a receipt acknowledgement to the initiator. <p>The PIP 3B2 (Advance Shipment Notice) example, which implements this design pattern, is described in “Step 2: Open the PIP 3B2: Advance Shipment Notice Example” on page 3-9.</p>
Asynchronous two-action activity	<p>Involves two actions from both trading partners:</p> <ul style="list-style-type: none"> Initiator sends a business message to the participant. The participant sends a receipt acknowledgement to the initiator. The participant sends a business message to the initiator. The initiator sends a receipt acknowledgement to the participant. <p>The PIP3A4 (Purchase Order Request and Confirmation) example, which implements this design pattern, is described in “Step 3: Open the PIP 3A4: Purchase Order Request and Confirmation Example” on page 3-16.</p>
Synchronous one-action / two-action activity	<p>Synchronous versions of the above design patterns, in which an immediate response is required. The current release of WebLogic Integration does not support synchronous design patterns.</p>

The example PIP implementations provided in this tutorial allow you to quickly and easily build any PIPs that following the same design patterns, as described in [“Implementing New PIPs Based on the Example PIPs” on page 3-26.](#)

For more information about RosettaNet design patterns and conversation choreography, see the following documents:

- “PIP Design Patterns” in [Introducing RosettaNet Solutions](http://edocs.bea.com/wli/docs81/tpintro/rosettanet.html) (available at <http://edocs.bea.com/wli/docs81/tpintro/rosettanet.html>.) in *Introducing Trading Partner Integration*.
- *RosettaNet Implementation Framework Core Specification* (version V02.00.01) at <http://www.rosettanet.org/rnif>.

Tutorial Steps

This section describes the following tutorial steps:

- [Step 1: Open the RosettaNet Example Application](#)
- [Step 2: Open the PIP 3B2: Advance Shipment Notice Example](#)
- [Step 3: Open the PIP 3A4: Purchase Order Request and Confirmation Example](#)

Before you proceed through these steps, make sure that you have reviewed the material in [“Before You Begin”](#) on page 3-2.

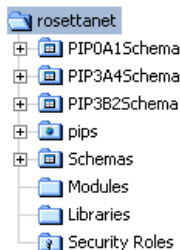
Step 1: Open the RosettaNet Example Application

The RosettaNet example application contains all the PIP implementations described in [“PIPs Implemented In These Examples”](#) on page 3-4.

To open the RosettaNet example application:

1. Start WebLogic Workshop, if you have not already done so.
2. From the File menu, choose **Open→Application**.
3. Navigate to the `tptutorialapps` folder that you created in [“Step 3: Install the Tutorial Files”](#) on page 1-9.
4. Open the `rosettanet` folder
5. Open the `rosettanet.work` file.

The Application tab displays the contents of the RosettaNet tutorial application, which is described in [“Folders in the RosettaNet Tutorial Application”](#) on page 3-6.



6. On the Application tab, right-click the root `rosettanet` folder and choose Properties.

7. In the Application Properties dialog, verify that the Server Home Directory points to the WebLogic Integration domain that you created in “[Step 1: Create a New WebLogic Integration Domain](#)” on page 1-3 (such as `c:\bea\user_projects\domains\tp tutorial`), and then click the OK button.
8. On the Application tab, click the `pips` folder to display folders containing the example business processes.

Step 2: Open the PIP 3B2: Advance Shipment Notice Example

This topic describes the example implementation of PIP 3B2: Advance Shipment Notice. It contains the following sections:

- [About the PIP 3B2 Example](#)
- [Components of the PIP 3B2 Example](#)
- [Walkthrough of the Initiator Business Process](#)
- [Walkthrough of the Participant Business Process](#)
- [Walkthrough of the Private Business Processes](#)
- [Running the PIP 3B2 Example](#)

For detailed information about PIP 3B2, see <http://www.rosettanet.org/PIP3B2>.

About the PIP 3B2 Example

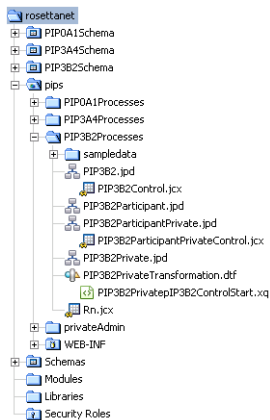
PIP 3B2 is an example of the asynchronous one-way action activity design pattern described in “[RosettaNet Design Patterns](#)” on page 3-7. The `PIP3B2Processes` folder contains business process definitions for PIP 3B2. The `PIP3B2Schemas` schema project contains the schema file for the PIP 3B2 message, converted from the PIP 3B2 DTD file that was provided at <http://www.rosettanet.org/PIP3B2>. The `sampledata` directory contains sample XML documents that you can use when “[Running the PIP 3B2 Example](#)” on page 3-16.

The PIP 3B2 Advance Shipment Notice involves the following steps:

1. The Shipper (initiator) constructs an Advance Shipment Notice message and sends it to the Receiver (participant).
2. The Receiver validates the message and sends a receipt acknowledgement to the Shipper.
3. The Receiver submits the message to the backend system for further processing.

Components of the PIP 3B2 Example

When you open the `PIP3B2Processes` folder in WebLogic Workshop, the application tab displays the contents of the folder.



The following table summarizes the components of the PIP 3B2 example:

Table 3-4 Components of the PIP 3B2: Advance Shipment Notice Example

Role / Component	Description
Shipper (Initiator)	RosettaNet role name for the initiator.
<code>PIP3B2.jpd</code>	Public initiator business process that sends the message, waits for the acknowledgement, and handles failures. Uses <code>Rn.jcx</code> , a RosettaNet control instance.
<code>Rn.jcx</code>	<p>RosettaNet control instance used to exchange messages with the Receiver via RNIF. For more information about the RosettaNet control, see RosettaNet Control in the WebLogic Workshop Online Help.</p> <p>The annotations on this control instance in <code>PIP3B2.jpd</code> include the PIP name and version, role names, and the ‘to-selector’ that extracts the business ID of the remote partner from the Advance Shipment Notice document.</p>

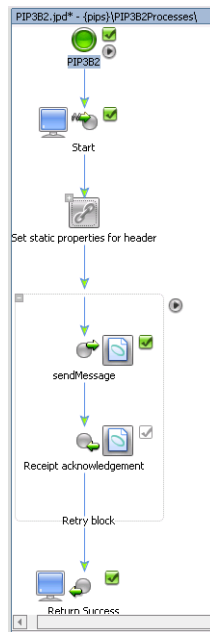
Table 3-4 Components of the PIP 3B2: Advance Shipment Notice Example (Continued)

PIP3B2Control.jcx	Process control that wraps PIP3B2.jpđ for use in other processes. Automatically generated by right-clicking the PIP3B2.jpđ file in the Application tab and choosing Generate Process Control .
PIP3B2Private.jpđ	Private business process that invokes the PIP3B2.jpđ business process via a process control. This private business process can utilize the full power of WebLogic Integration to access the backend systems, assemble the information, and transform the request data into the data format required for PIP 3B2.
PIP3B2PrivateTransformation.dtf and PIP3B2PrivatePIP3B2ControlStart.xq (the associated .xq file)	Example transformation file. Automatically generated when a transformation is defined in a process node.
Receiver (Participant)	RosettaNet role name for the participant.
PIP3B2Participant.jpđ	Public business process that receives the RNIF message and sends the receipt acknowledgement to the Shipper. The RosettaNet annotations include the PIP name, version, and role. For more information, see @jpd:rosettanet Annotation in the WebLogic Workshop online help.
PIP3B2ParticipantPrivate.jpđ	Private business process that represents the backend processing of the shipment notice.
PIP3B2ParticipantPrivateControl.jcx	Process control that wraps the private process for use in PIP3B2Participant.jpđ. Automatically generated by right-clicking the PIP3B2ParticipantPrivate.jpđ file in the Application tab and choosing Generate Process Control .

Walkthrough of the Initiator Business Process

This section describes the example initiator business process (PIP3B2.jpđ) for PIP 3B2. To view this business process:

1. On the Application tab in WebLogic Workshop, open the `PIP3B2Processes` folder.
2. Open the `PIP3B2.jpdl` file.
3. If necessary, collapse the OnTimeout and On Error Message paths to see only the main (success) path.



Success Path

The success path is simple and straightforward:

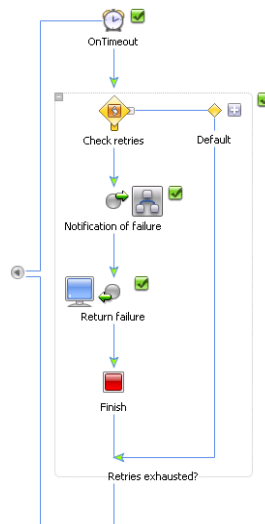
1. The process starts with a client request that expects the Advance Shipment Notice XML document (in this tutorial, the client request comes from the private process).
2. The business message is sent to the Receiver (participant) via the `sendMessage` method on the RosettaNet control.
3. The Receipt acknowledgement node awaits the receipt acknowledgement callback (`rn_onAck` method) from the Receiver.
4. Once the acknowledgement is received, the process responds to the private process and finishes.

Because PIP 3B2 implements the one-step action activity design pattern, no business data is returned from the Receiver—only a receipt acknowledgement. The response to the private process merely indicates a successful completion and returns no data.

Failure Paths

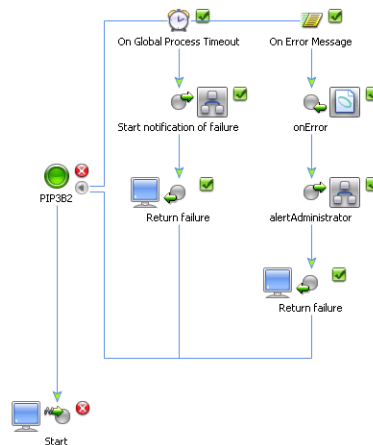
The OnTimeout and On Error Message paths handle failure management, such as in the event of a network failure. To view the failure paths:

1. Expand the OnTimeout path next to the group of nodes named `Retry block`.



This path represents a standard BPM OnTimeout construct. The timeout value is set to two hours (the standard PIP retry interval) and three retries (the standard PIP retry count). Thus, if for any reason the acknowledgement fails to arrive within two hours, the group will be retried and the `sendMessage` step will be executed again. The OnTimeout path contains a condition that determines whether retries have been exhausted and, if so, the PIP 0A1 (Notification of Failure) subprocess is triggered (via a Control Send node), and the process responds to the private process with an error and finishes.

2. Expand the On Error Message path next to the PIP3B2 node at the top of the process.

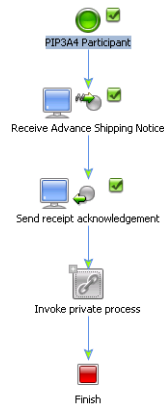


The Global Process Timeout is configured for the standard PIP timeout value of 24 hours. If the whole process fails to complete within this time, the PIP 0A1 (Notification of Failure) subprocess is triggered and an error is returned to the private process.

Finally, the Receiver might reject the message for a variety of reasons (for example, the message fails validation). The message rejection is represented as the `onError` callback on the RosettaNet control. You can see that the global On Error Message handler is prepared to receive the error at any time and, if it does, it starts a subprocess to notify the administrator of the problem, returns an error to the private process, and exits. Notice that, if we cannot successfully send the message (for example, due to network problems), the business process starts the PIP 0A1 (Notification of Failure) to try to notify the remote partner of the problem. Similarly, upon receipt of an error from the Receiver, the business process should notify the local administrator as well as the remote administrator.

Walkthrough of the Participant Business Process

The participant business process (`PIP3B2Participant.jpd`) on the Receiver side is simpler than the initiator business process. To view the participant business process in WebLogic Workshop, open the `PIP3B2Participant.jpd` file.



The participant business process includes the following steps:

1. The Advance Shipment Notice message arrives at the Client Request node (Receive Advance Shipping Notice).
2. The Client Response node (Send receipt acknowledgement) sends a receipt acknowledgement.
3. The business process passes the Advance Shipment Notice to a private process for further processing.

Walkthrough of the Private Business Processes

The private processes on both sides are merely stubbed-out processes that don't carry out any specific work. You would typically customize these processes to tie them with the backend applications. For example, the shipment notice can be inserted into an ERP application via the Application View control and BEA adapters, sent on a message queue, and written out to file or database.

The public/private process pattern used in this example is not mandatory and is merely a recommended approach. You are free to partition your processes as best suits your environment. It is entirely possible to not use the private processes and implement all the necessary backend integration as well as RosettaNet interactions in a single process. However, separating processes into public (those that only deal with RosettaNet choreography) and private (those that deal with backend system integration) may improve the reuse and maintainability of your application.

Running the PIP 3B2 Example

By default, WebLogic Integration runs in Test (development) mode, which allows you to run the PIP3B2 example business processes on the same machine (collocated). In a production environment, each trading partner would run its respective business process on its own separate WebLogic Integration server. For more information about the Test and Production modes, see “Configuring the Mode and Message Tracking” in [Trading Partner Management](http://edocs.bea.com/wli/docs81/manage/tpm.html) (available at <http://edocs.bea.com/wli/docs81/manage/tpm.html>) in *Managing WebLogic Integration Solutions*.

To run the PIP 3B2 example:

1. With the application open in WebLogic Workshop, press F7 to build the application.
2. Open `PIP3B2Participant.jpd` and click Start.

Note: Do not actually try to execute it from the test browser. This step is needed to “prime” the participant process.

3. Return to WebLogic Workshop, open `PIP3B2Private.jpd` and click Start.
4. In the test browser, specify the `3B2AdvanceShipmentNotificationMessageBase.xml` file in the `sampledata` subdirectory of the `PIP3B2Processes` folder, and click `clientRequest`.

In a few moments, the end-to-end choreography will execute.

5. Optionally, you can open the WebLogic Integration Administration Console and observe message tracking and process tracking entries.

Step 3: Open the PIP 3A4: Purchase Order Request and Confirmation Example

This topic describes the example implementation of the PIP 3A4: Purchase Order Request and Confirmation. It contains the following sections:

- [About the PIP 3A4 Example](#)
- [Components of the PIP 3A4 Example](#)
- [Walkthrough of the Initiator Business Process](#)
- [Walkthrough of the Participant Business Process](#)
- [Walkthrough of the Private Business Processes](#)

- [Running the PIP 3A4 Example](#)

For detailed information about PIP 3A4, see <http://www.rosettanet.org/PIP3A4>.

About the PIP 3A4 Example

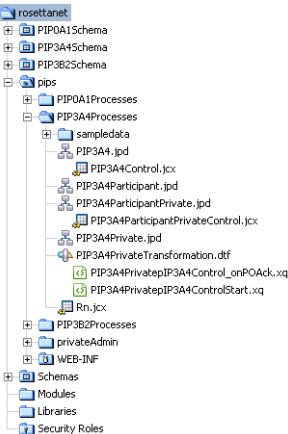
PIP 3A4 is an example of the asynchronous two-way action activity design pattern described in “[RosettaNet Design Patterns](#)” on page 3-7. The `PIP3A4Processes` folder contains business process definitions for PIP 3A4. The `PIP3A4Schemas` schema project contains the schema file for the PIP 3A4 message, converted from the PIP 3A4 DTD file that was provided at <http://www.rosettanet.org/PIP3A4>. The `sampledata` directory contains sample XML documents that you can use when “[Running the PIP 3A4 Example](#)” on page 3-25.

The PIP 3A4 Purchase Order Request and Confirmation involves the following steps:

1. The Buyer (initiator) constructs a Purchase Order Request message and sends it to the Seller (participant).
2. The Seller validates the message and sends a receipt acknowledgement to the Buyer.
3. The Seller then submits the order request message to the backend system for further processing.
4. The Seller’s backend system generates a Purchase Order Confirmation (which confirms the availability of requested products and the expected ship dates).
5. The Seller sends the confirmation to the Buyer.
6. The Buyer acknowledges the receipt of the message to the Seller.
7. The Buyer passes the Purchase Order Confirmation to its backend system for processing.

Components of the PIP 3A4 Example

When you open the `PIP3A4Processes` folder in WebLogic Workshop, the application tab displays the contents of the folder.



The following table describes the components of the PIP 3A4 example implementation.

Table 3-5 Components of the PIP 3A4 (Purchase Order Request and Confirmation) Example

Role / Component	Description
Buyer (Initiator)	RosettaNet role name for the initiator.
PIP3A4 . jpdl	Public business process that exchanges messages with the seller, waits for the acknowledgement, and handles failures. Uses the RosettaNet control instance (Rn . jcx).
Rn . jcx	<p>RosettaNet control instance used to exchange messages with the Seller via RNIF. For more information about the RosettaNet control, see RosettaNet Control in the WebLogic Workshop Online Help.</p> <p>The annotations on this control instance in PIP3A4 . jpdl include the PIP name and version, role names, and the ‘to-selector’ that extracts the business ID of the remote partner from the Purchase Order Request document.</p>

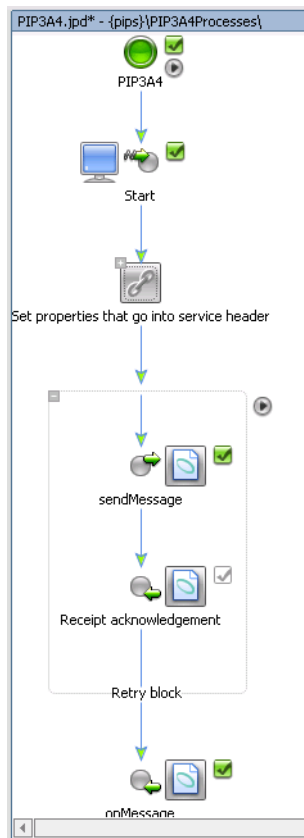
Table 3-5 Components of the PIP 3A4 (Purchase Order Request and Confirmation) Example (Continued)

PIP3A4Control.jcx	Process control that wraps PIP3A4.jpd for use in other processes. Automatically generated by right-clicking the PIP3A4.jpd file in the Application tab and choosing Generate Process Control .
PIP3A4Private.jpd	Private business process that invokes the PIP3A4 process via a process control. This business process can utilize the full power of WebLogic Integration to access the backend systems, assemble the information, and transform it into the required format for PIP 3A4.
PIP3A4PrivateTransformation.dtf and the associated .xq files	Example transformation file. Automatically generated when a transformation is defined in a process node.
Seller (Participant)	RosettaNet role name for the participant.
PIP3A4Participant.jpd	Public business process that receives the message and sends the receipt acknowledgement, responds with a message and receives acknowledgement. The RosettaNet annotations include the PIP name, version, and role. For more information, see @jpd:rosettanet Annotation in the WebLogic Workshop online help.
PIP3A4ParticipantPrivate.jpd	Private business process that represents the backend processing of the purchase order and the creation of the PO Confirmation.
PIP3A4ParticipantPrivateControl.jcx	Process control that wraps the private process for use in PIP3A4Participant.jpd. Automatically generated by right-clicking the PIP3A4ParticipantPrivate.jpd file in the Application tab and choosing Generate Process Control .

Walkthrough of the Initiator Business Process

This section describes the example initiator business process (PIP3A4 . jpd) for PIP 3B2. To view this business process:

1. In WebLogic Workshop, open the PIP3A4 . jpd file.
2. If necessary, collapse the OnTimeout and On Error Message paths to see only the main (success) path.



Success Path

The success path is simple and straightforward:

1. The process starts with a client request that expects the Purchase Order Request XML document (in this tutorial, the client request comes from the private process).

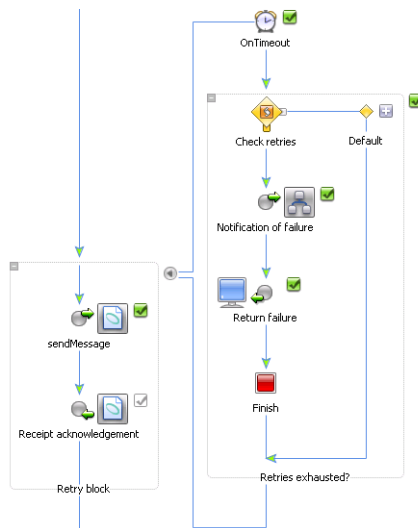
2. The message is sent to the Seller (participant) via the `sendMessage` method on the RosettaNet control.
3. The Receipt acknowledgement node awaits the receipt acknowledgement callback (`rn_onAck` method) from the Receiver.
4. Once the acknowledgement is received, the business process waits for the Purchase Order Confirmation.
5. After receiving the Purchase Order Confirmation, the business process acknowledges its receipt.
6. The business process responds to the private process with the PO Confirmation and finishes.

Note: WebLogic Integration allows the receipt acknowledgement and business message to be received in either order, in conformance with RosettaNet specifications. If the initiator receives the response message before the receipt acknowledgement, the response message is queued for later processing.

Failure Paths

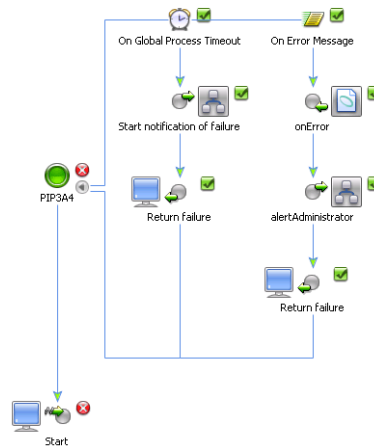
The OnTimeout and On Error Message paths handle failure management, such as in the event of a network failure. To view the failure paths:

1. Expand the OnTimeout path next to the group of nodes named `Retry` block.



This path represents a standard BPM OnTimeout construct. The timeout value is set to two hours (the standard PIP retry interval) and three retries (the standard PIP retry count). Thus, if for any reason the acknowledgement fails to arrive within two hours, the group will be retried and the `sendMessage` step will be executed again. The OnTimeout path contains a condition that determines whether retries have been exhausted and, if so, the PIP 0A1 (Notification of Failure) subprocess is triggered (via a Control Send node), and the process responds to the private process with an error and finishes.

2. Expand the On Error Message path next to the PIP3A4 node at the top of the process.



The Global Process Timeout is configured for the standard PIP timeout value of 24 hours. If the whole process fails to complete within this time, the PIP 0A1 (Notification of Failure) subprocess is triggered and an error is returned to the private process.

Finally, the Receiver might reject the message for a variety of reasons (for example, the message fails validation). The message rejection is represented as the `onError` callback on the RosettaNet control. You can see that the global On Error Message handler is prepared to receive the error at any time and, if it does, it starts a subprocess to notify the administrator of the problem, returns an error to the private process, and exits. Notice that, if we cannot successfully send the message (for example, due to network problems), the business process starts the PIP 0A1 (Notification of Failure) to try to notify the remote partner of the problem. Similarly, upon receipt of an error from the Receiver, the business process should notify the local administrator as well as the remote administrator.

Walkthrough of the Participant Business Process

The participant business process (`PIP3A4Participant.jpdl`) on the Seller side is simpler than the initiator business process. To view the participant business process in WebLogic Workshop, open the `PIP3A4Participant.jpdl` file.



The participant business process includes the following steps:

1. The Purchase Order Request message arrives at the Client Request node (Receive PO message).
2. The Client Response node (Send receipt acknowledgement) sends a receipt acknowledgement to the Buyer.
3. The business process passes the Purchase Order Request to a private process for further processing.
4. When the private process returns with a Purchase Order Confirmation, the business process sends it to the Buyer using the `sendReply` callback on the Client Response node (Send PO Acknowledgement), and waits for the receipt acknowledgement, after which the business process finishes.

Unlike the 3B2 participant business process, which merely receives the message, the 3A4 participant business process sends out a response, so it is necessary to account for failure scenarios, such as network failures or validation failure. Open the `onTimeout` path on the Retry Block to display the time-outs and retries. Note that this path is similar to the `onTimeout` path in the 3B2 initiator business process described in [“Walkthrough of the Initiator Business Process” on page 3-11](#). In addition, there is a global `onMessage` handler that handles errors returned by the Buyer.

Walkthrough of the Private Business Processes

As in the 3B2 example, the private processes on both sides are merely stubbed-out processes that don't carry out any specific work. The participant private process actually echoes the received message. You would typically customize these processes to tie them with the backed applications. For example, the shipment notice can be inserted into an ERP application via the Application View control and BEA adapters, sent on a message queue, written out to file or database.

The public/private process pattern used in this example is not mandatory and is merely a recommended approach. You are free to partition your processes as best suits your environment. It is entirely possible to not use the private processes and implement all the necessary backend integration as well as RosettaNet interactions in a single process. However, separating processes into public (those that only deal with RosettaNet choreography) and private (those that deal with backend system integration) may improve the reuse and maintainability of your application.

Running the PIP 3A4 Example

To run the PIP 3A4 example:

1. With the application open in WebLogic Workshop, press F7 to build the application.
Note: Open `PIP3A4Participant.jpdl` and click Start.
2. Do not actually try to execute it from the test browser. This step is needed to “prime” the participant process.
3. Return to WebLogic Workshop, open `PIP3A4Private.jpdl` and click Start.
4. In the test browser, specify the `3A4PurchaseOrderRequestMessageBase_0010.xml` file in the `sampledata` subdirectory of the `PIP3A4Processes` folder, and click `clientRequest`.
 In a few moments, the end-to-end choreography will execute. The client response will echo back the XML message.
5. Optionally, you can open the WebLogic Integration Administration Console and observe message tracking and process tracking entries.

Implementing New PIPs Based on the Example PIPs

This topic describes implementing new PIP based on the example PIPs in this tutorial. It contains the following topics:

- [About Implementing New PIPs](#)
- [Copying and Customizing PIP Implementations](#)
- [Converting RosettaNet DTD Schemas to XSD Schemas](#)

About Implementing New PIPs

You can implement a new PIP based on an existing PIP implementation with a similar design pattern. For example, PIP 3A2: Request Price and Availability, is an example of a message send with response design pattern. Therefore, the business process choreography is identical to that of PIP 3A4. The main differences are that the request and response message schemas are different and the annotations must be changed. You can duplicate and rename the 3A4 processes and change the schema type of the messages to easily create a PIP 3A2. implementation. For detailed information about PIP 3A4, see <http://www.rosettanet.org/PIP3A2>.

Copying and Customizing PIP Implementations

To implement a new PIP based on an existing PIP implementation, complete the following tasks:

1. Download the PIP distribution, including the specification and any DTDs, from the RosettaNet web site at <http://www.rosettanet.org/>.
2. Optionally, convert any DTDs to XSD files, as described in “[Converting RosettaNet DTD Schemas to XSD Schemas](#)” on page 3-27.
3. Copy the example PIP implementation associated with the design pattern that you want to use.
4. In WebLogic Workshop, import the schema for the new PIP into the project and then change the schema definition to the new PIP.
5. Change the RosettaNet annotations for the new PIP:
 - For *public* initiator business processes, you change the `pip` and `pip-version` attributes (and others if needed) in the `@jc:rosettanet` Tag, which is available at <http://edocs.bea.com/workshop/docs81/doc/en/integration/controls/controlsRosettaNetTag.html>.

- For *public* participant business processes, you change the `pip-name` and `pip-version` attributes (and others if needed) in the [@jpd:rosettanet Annotation](http://edocs.bea.com/workshop/docs81/doc/en/integration/reference/refJpdAnnotations_rosettanet.html), which is available at http://edocs.bea.com/workshop/docs81/doc/en/integration/reference/refJpdAnnotations_rosettanet.html.
- 6. Rename the JPD and JCX files and change the names of other components to be more descriptive of the new PIP implementation, if you want.
- 7. Change the implementation of any *private* business processes as needed.
- 8. Make any other changes as needed.

Converting RosettaNet DTD Schemas to XSD Schemas

RosettaNet PIPs use DTD files for message definitions. The graphical mapper tools and trading partner XQuery functionality used in WebLogic Workshop to design data transformations require W3C Schema (XSD) files. Although using XSD files is not required to implement PIPs in WebLogic Integration, if you want to use the graphical tools in WebLogic Workshop, you need to convert the RosettaNet DTD to an XSD file. You can use XMLSPY, included with the BEA WebLogic Platform distribution, to convert DTDs to schemas.

When converting the files, consider the following issues:

- When you convert the DTD to W3C schema, you might get an error message about the `xml:lang` name. To fix the problem, replace the following text:

```
<xs:extension base="xs:string">
<xs:attribute name="xml:lang" type="xs:string" />
</xs:extension>
```

with the following text:

```
<xs:extension base="xs:string">
<xs:attribute ref="xml:lang" />
</xs:extension>
```

and add the following import statement:

```
<xs:import namespace="http://www.w3.org/XML/1998/namespace"
schemaLocation="http://www.w3.org/2001/xml.xsd" />
```

Save the schema file, and then drag and drop it into a Workshop schema project.

- The schema generated by XMLSPY has elements that have both a name and reference, as shown in the following example.

```
<xs:element name="ActionIdentity" ref="ActionIdentity"/>
```

This creates problems in WebLogic Workshop. To fix the file, open it in WebLogic Workshop, press Ctrl+H, make sure that wildcard pattern matching is enabled, and then replacing all occurrences of:

```
name="*" ref
```

with

```
ref
```

as in the following example:

```
<xs:element ref="ActionIdentity"/>
```

- In some cases, two different schemas will contain identical element definitions, which results in schema compilation problems. To avoid problems, store each schema file in a *separate* schema project in your application.

After you build the schema projects, the PIP types will become available in the type system for use in parameter and variable types.

Index

B

- before you begin the tutorials 1-2
- business process applications, creating 2-5

C

- callbacks 2-51
- channel files, creating 2-34
- controls
 - ebXML 2-20, 2-28
 - email 2-53
 - file 2-12
 - RosettaNet 3-10, 3-18
 - TPM 2-51
- customer support contact information x

D

- documentation, where to find it ix
- DTDs, converting 3-26

E

- ebXML control 2-20, 2-28
- ebXML tutorial
 - callback 2-51
 - channel file 2-34
 - distributed setup 2-61
 - ebXML control 2-20, 2-28
 - email control 2-53
 - file control 2-12
 - file event generator 2-36, 2-58
 - importing sample data 2-8
 - importing schemas 2-8

- initiator (buyer) process 2-17
- participant (seller) process 2-10
- prerequisites 2-1
- public/private pattern 2-46
- source view parameters 2-15
- suggested reading 2-2
- test browser 2-16
- TPM control 2-51
- WebLogic Integration - Business Connect 2-64

- email control 2-53

F

- file control 2-12
- file event generator 2-36, 2-58

I

- installing tutorial files 1-9

P

- PIP 3A2 3-26
- PIP 3A4 example
 - about the PIP 3A4 example 3-17
 - components of 3-17
 - initiator business process 3-20
 - participant business process 3-23
 - private business processes 3-25
 - running 3-25
- PIP 3B2 example
 - about the PIP 3B2 example 3-9

- components of 3-10
- initiator business process 3-11
- participant business process 3-14
- private business processes 3-15
- running 3-16

- printing product documentation ix
- product support x

R

- related information ix
- RosettaNet control 3-10, 3-18
- RosettaNet tutorial
 - application, opening 3-8
 - converting DTDs to schemas 3-4
 - creating new PIPs using examples 3-26
 - design patterns 3-7
 - folders 3-6
 - goals 3-2
 - PIP 3A4 example 3-16
 - PIP3B2 example 3-9
 - PIPs, list of implemented 3-4
 - prerequisites 3-2
 - steps 3-8
 - suggested reading 3-3

S

- sample data, importing 2-8
- schemas
 - converting 3-26
 - importing 2-8
- support x

T

- technical support x
- test browser 2-16
- TPM control 2-51
- trading partner information 1-4
- tutorials
 - before you begin 1-2

- domain, creating 1-3
- installing files 1-9
- setting up 1-2
- trading partner information, default 1-4

W

- WebLogic Integration - Business Connect 2-64
- WebLogic Integration domain, creating 1-3
- WebLogic Server, starting 2-16
- WebLogic Workshop, starting 2-5

