BEA WebLogic® Integration

Tutorials for Trading Partner Integration

Version 10.2
Document Revised: March 2008

Contents

1. Introduction

2. Tutorial: Building RosettaNet Solutions

3. Tutorial: Building ebXML Solutions

CHAPTER 1

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 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
Chapter 3, "Tutorial: Building ebXML Solutions"	Describes how to build the ebXML (Electronic Business using eXtensible Markup Language) business process examples.	http://www.ebXML.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 10.2 installed on your system. For more information, see Installing Guide.
- It is recommended that you complete the following tutorials so that you know how to build business processes and create data transformations in BEA WorkSpace Studio:
  - Tutorial: Building Your First Business Process
  - Tutorial: Building Your First Data Transformation
- You should be familiar with basic trading partner integration concepts, as described in Introducing Trading Partner Integration.
- You should first create the `tptutorial` 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

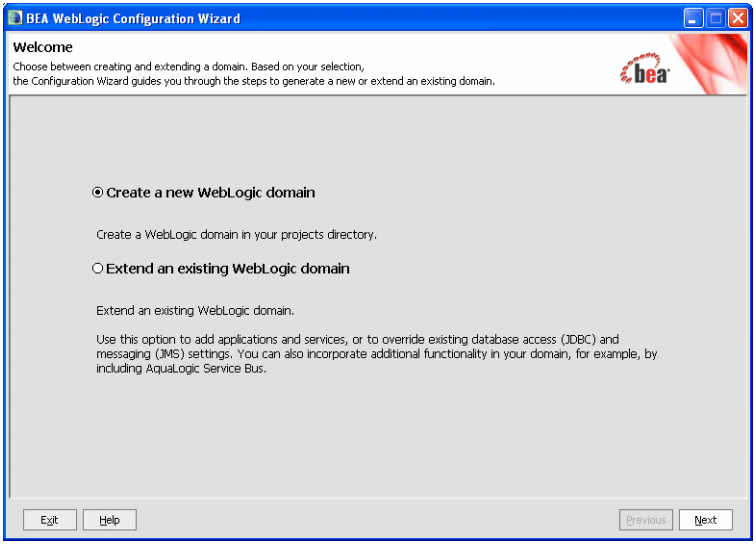
The trading partner integration tutorials require a WebLogic Integration domain that you must create using the BEA WebLogic Platform Configuration Wizard. The domain name used in this document is `tptutorial`, but you can use any valid domain name you want.

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. To Start the Configuration Wizard, from the **Start menu**, click **All Programs > BEA > Tools > Configuration Wizard** to start the BEA WebLogic Configuration Wizard. This displays the Welcome page in the BEA WebLogic Configuration Wizard dialog box (see Figure 1-1).

Figure 1-1 BEA WebLogic Configuration Wizard.



2. Select **Create a new WebLogic domain** and click **Next**. This displays the Select Domain Source page in the Configuration Wizard dialog box.

   As you proceed through the Configuration Wizard, several pages will appear in a sequence. You need to specify your settings on each page and click **Next** to proceed to the subsequent page. Table 1-2 lists the pages and the options that you need to select to create the domain successfully.

   Note: These instructions assume mostly default selections. For more information about advanced configuration options, see Creating WebLogic Domains Using the Configuration Wizard.

Table 1-2 Configuring the Domain Using the Configuration Wizard

Page in the Configuration Wizard Dialog Box	Recommended Action
Select a Domain Source	Select **Generate a domain configured automatically to support the following BEA Products** option for the following BEA Products: • WebLogic Server (Required) • Workshop for WebLogic 10.2 • WebLogic Integration Click **Next** to proceed.
Configure Administrator Username and Password	Specify the following mandatory credentials: User name = `weblogic` User password = `weblogic` Confirm user password = `weblogic` Click **Next** to proceed.
Configure Server Start Node and JDK	Select **Development Mode** in the WebLogic Domain Startup Mode column. Select **Sun SDK 1.5.0_11 @ C:\bea\jdk150_11** in the BEA Supplied JDKs column. Click **Next** to proceed.
Customize Environment and Service Settings	Click **No**, to retain the settings defined in the domain source and proceed directly to creating your domain. Click **Next** to proceed.

3. On the Create WebLogic Domain page, specify the following values for each field and click **Create**:
   - **Domain name:** tptutorial
   - **Domain Location:** `C:\bea\user_projects\domains`
   - **Application Location:** `C:\bea\user_projects\applications`

   After the domain is created successfully, the Creating Domain page is displayed.
4. Select the **Start Admin Server** check box and click **Done** to proceed.

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 in *Using The WebLogic Integration Administration Console*.

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 in *Using The WebLogic Integration Administration Console*.

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`	`000000001`	Default local trading partner. In the tutorials, this trading partner is usually the *initiator* 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`	`000000002`	In the tutorials, this trading partner is usually the *participant* 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.

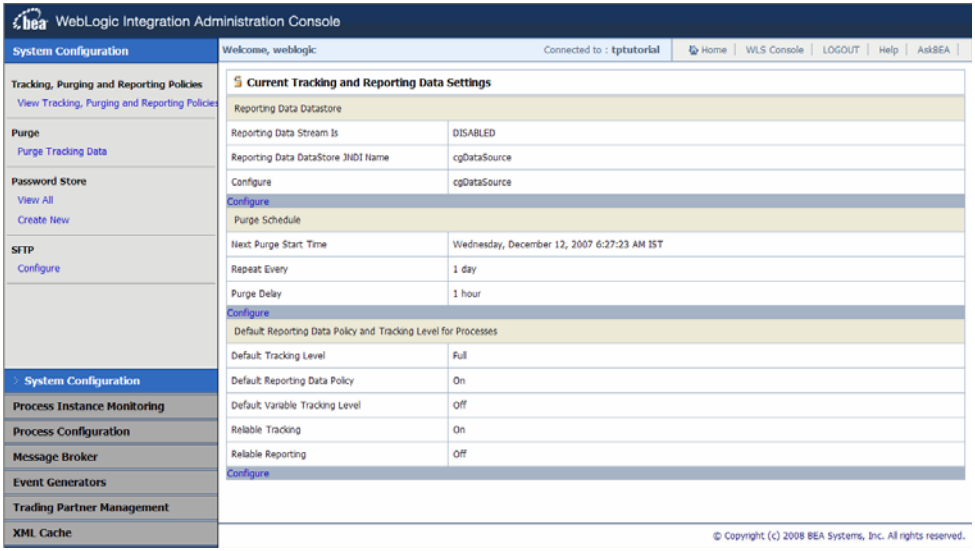
Note: If you make any changes to the preconfigured settings in the TPM repository, you should adjust the instructions in the tutorials accordingly. For configuration instructions, see Trading Partner Management in *Using The WebLogic Integration Administration Console*.

To view the default trading partner information:

1. Start the WebLogic Integration Administration Console.
2. In a browser, enter the following URL: `http://localhost:7001/wliconsole`.
3. When prompted, specify the username and password.

   The WebLogic Integration Administration Console displays the home page (see Figure 1-2).

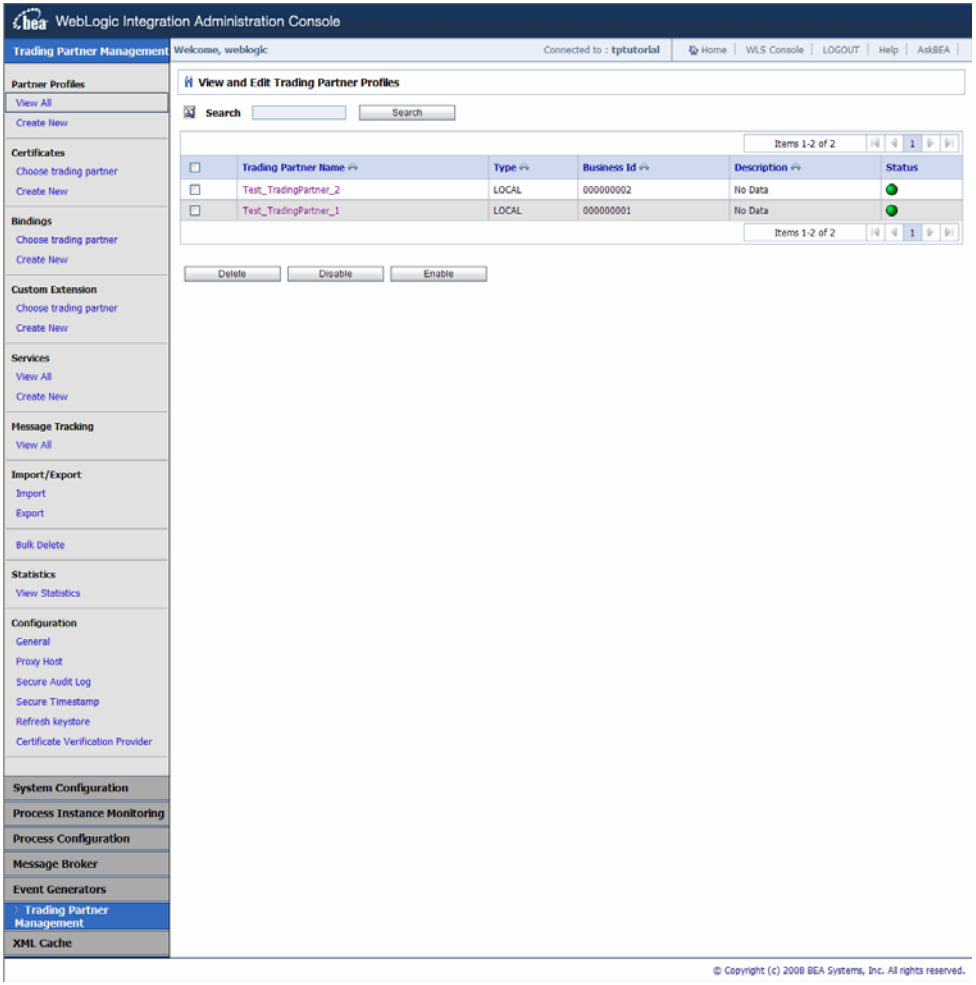
Figure 1-2 WebLogic Integration Administration Console



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

4. Select the **Trading Partner Management** module to see the Trading Partner Management home page and the profiles for the preconfigured trading partners (see Figure 1-3).

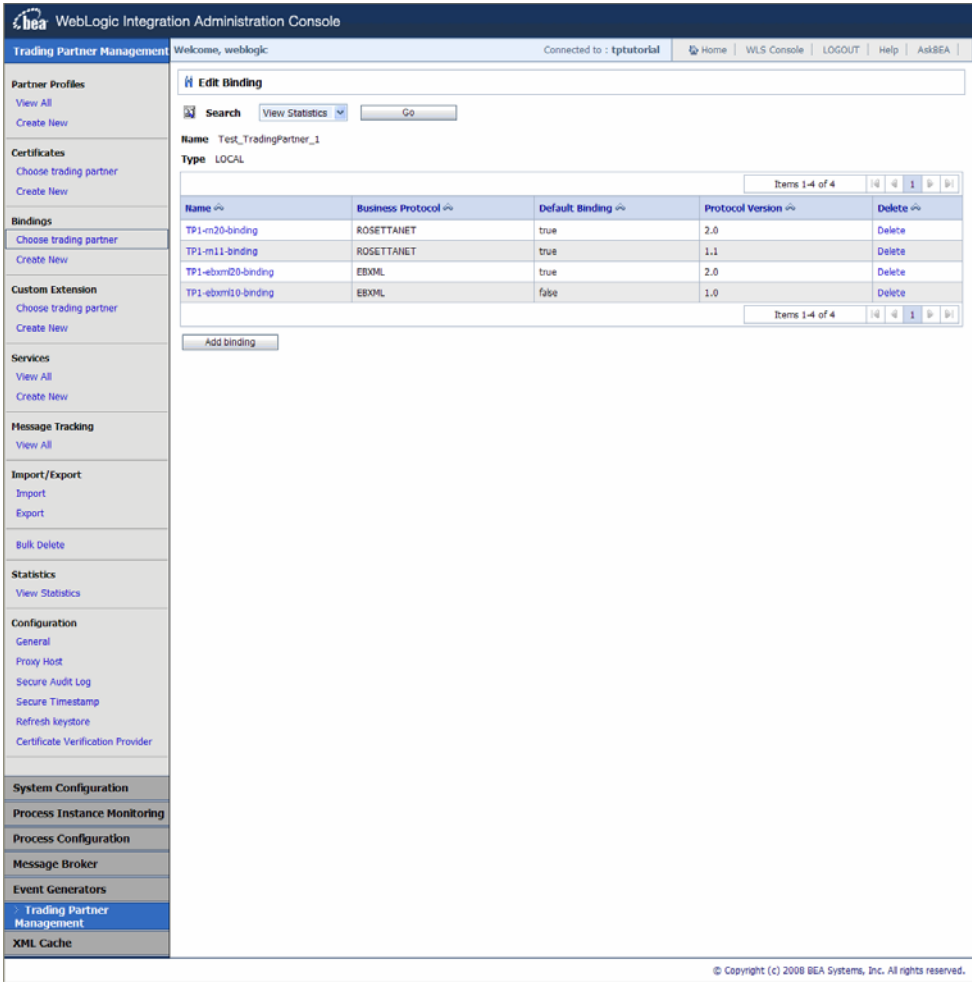
Figure 1-3 View and Edit Trading Partner Profiles



5. From the left panel, select **Bindings** > **Choose trading partner**.

6. Select `Test_TradingPartner_1` name from the drop-down menu in the Choose Trading Partner page, and click **Go** button to see the preconfigured bindings for the default trading partner (see Figure 1-4).

Figure 1-4 Edit Binding



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 10.2 page at the following URL:

   `https://codesamples.projects.dev2dev.bea.com/servlets/Scarab/id/S417`

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\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 RosettaNet Solutions"
- Chapter 3, "Tutorial: Building ebXML 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.

CHAPTER 2

Tutorial: Building RosettaNet Solutions

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

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, both with notification of failure interactions) that form the basis of most PIP implementations.
- To describe example BEA WorkSpace Studio public business processes that implement two RosettaNet Partner Interface Processes (PIPs)—PIP3B2 (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

To use this tutorial, you must have:

- Installed WebLogic Platform 10.2 on your system according to the instructions in Installation Guide.
- 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.
- For an introduction to RosettaNet solutions using WebLogic Integration, see Introducing RosettaNet Solutions.
- For more information about using the RosettaNet control in initiator business processes, see RosettaNet Control.

To learn more about the RosettaNet business protocol and how to implement RosettaNet solutions, see:

- The RosettaNet web site, which is available at:

  `http://www.rosettanet.org`

- *RosettaNet Implementation Framework: Core Specification* (RNIF Specification), available at: `http://www.rosettanet.org`

Note About Obtaining RosettaNet W3C XSD Schemas

The RosettaNet W3C XSD schemas for selected PIPs are included in the RosettaNet Self-Test Kit (STK) which can be downloaded from the Developer Tools area on the RosettaNet Ready Web site, available at: `http://www.rosettanet.org`.

If the schema for the PIP you are using is not available on this web site, it is possible to implement RosettaNet solutions in WebLogic integration using RosettaNet message definitions specified as DTD files. However, it is recommended to use W3C XSD files instead, since many of the WebLogic Integration tools such as the XQuery mapping tools (used to define data transformations) only support XSD schemas.

If you want to use the graphical XQuery mapping tools, you need to convert the RosettaNet DTD files to W3C XSD files (using a tool like XML Spy Enterprise Edition) and then import the XSD files into your project. Make sure that you refer to the RosettaNet Message Guidelines for the PIP which DTD file you are converting and add the appropriate validation rules for Service Content Validation to your XSD file after the conversion. For instructions on how to convert the files using XML Spy, see “Converting RosettaNet DTD Schemas to XSD Schemas” on page 2-39.

In this tutorial, all the XSDs necessary to complete the examples are included in the tutorial files.

Tutorial Overview

The RosettaNet example includes business process definitions (.java), schemas, and other files that illustrate how to implement RosettaNet PIPs using WebLogic Integration 10.2. 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:

- 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 2-1 RosettaNet Examples

PIP	Description
0A1	Implements PIP0A1: Notification of Failure. 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 BEA WorkSpace Studio. For more information, see Building RosettaNet Participant Business Processes.
3B2	Implements PIP3B2: Notify of Advance Shipment. Described in "Step 3: Open the PIP3B2: Notify of Advance Shipment Example" on page 2-17.
3A4	Implements PIP3A4: Request Purchase Order. Described in "Step 4: Open the PIP3A4: Request Purchase Order Example" on page 2-26.

Folders in the RosettaNet Tutorial Application

The following table describes the folders you see in your Package Explorer pane when you open the in the `\tptutorialapps\rosettanet` directory:

Table 2-2 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.

Table 2-2 Folders in `\tptutorialapps\rosettanet (Continued)`

Directory	Description
`pips`	Contains business process folders for each example PIP implementation (`PIP0A1Processes`, `PIP3A4Processes`, and `PIP3B2Processes`) and the private administrator alert notification (`privateAdmin`).
`rosettanet`	Contains the EAR folder
`System`	Schema project for standard WebLogic Integration system schemas.

RosettaNet Design Patterns

RosettaNet PIPs follow one of the following design patterns:

Table 2-3 RosettaNet PIP Design Patterns

Directory	Description
Asynchronous single-action activity	Involves a single action message and a receipt acknowledgment: • Initiator sends a business message to the participant. • The participant sends a receipt acknowledgement to the initiator. The PIP3B2 (Notify of Advance Shipment) example, which implements this design pattern, is described in "Step 3: Open the PIP3B2: Notify of Advance Shipment Example" on page 2-17.

Table 2-3 RosettaNet PIP Design Patterns (Continued)

Directory	Description
Asynchronous two-action activity	Involves actions messages and receipt acknowledgments from both trading partners: • 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. The PIP3A4 (Request Purchase Order) example, which implements this design pattern, is described in "Step 4: Open the PIP3A4: Request Purchase Order Example" on page 2-26.
Synchronous one-action / two-action activity	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. **Note:** Synchronous RosettaNet design patterns usually do not implement the receipt acknowledgments described in the asynchronous single-action and two-action activity design patterns.

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 2-38.

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

- "PIP Design Patterns" in Introducing RosettaNet Solutions.
- *RosettaNet Implementation Framework Core Specification* (version V02.00.01) at http://www.rosettanet.org.

Tutorial Steps

This section describes the following tutorial steps:

- “Step 3: Open the PIP3B2: Notify of Advance Shipment Example” on page 2-17
- “Step 4: Open the PIP3A4: Request Purchase Order Example” on page 2-26

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

Step 1: Open the RosettaNet Example Application

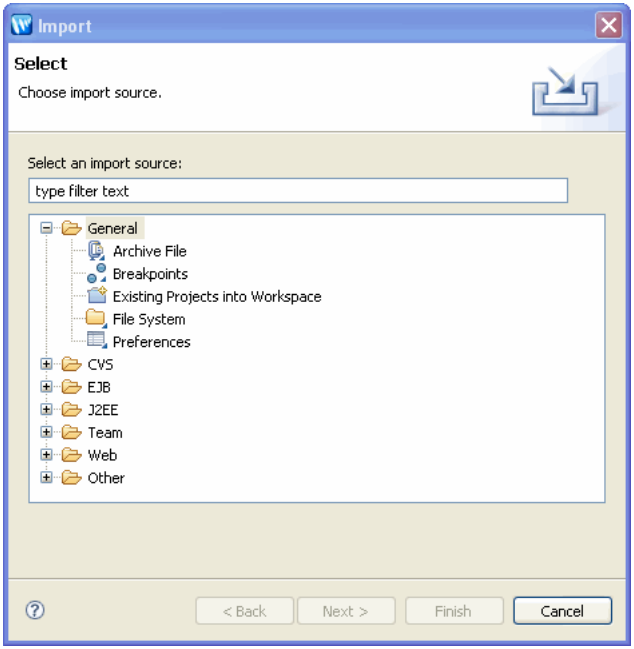
The RosettaNet example application contains all the PIP implementations described in “PIPs Implemented In These Examples” on page 2-4.

To open the RosettaNet example application:

1. Start WorkSpace Studio, if you have not already done so.
2. From the File menu, choose **Import**

   The **Import** wizard appears.
3. Expand General and select **Existing Projects into Workspace** (see Figure 2-1).

Figure 2-1 Import

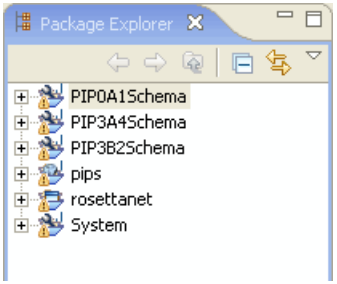


4. Click **Next**.
5. Check **Select root directory**, and click **Browse**, then navigate to the `tptutorialapps` folder that you created in "Step 3: Install the Tutorial Files" on page 1-9.
6. Select the `rosettanet` folder and click **OK**.
7. Ensure all the files are selected and click **Finish**.

   Note: If you are prompted to select a WebLogic Integration domain and server, select the WebLogic Integration domain that you created in "Step 1: Create a New WebLogic Integration Domain" on page 1-2 (such as `c:\bea\user_projects\domains\tptutorial`)

   The Package Explorer pane displays the contents of the RosettaNet tutorial application (see Figure 2-2), which is described in "Folders in the RosettaNet Tutorial Application" on page 2-4.

Figure 2-2 RosettaNet Tutorial Application



Step 2: Open the PIP0A1: Notification of Failure Example

This topic describes the example implementation of PIP0A1: Notification of Failure. It contains the following sections:

- About the PIP0A1 Example
- Components of the PIP0A1 Example
- Walkthrough of the Failure Notifier Business Process
- Walkthrough of the Report Administrator Business Process

For detailed information about PIP0A1, see http://www.rosettanet.org.

About the PIP0A1 Example

PIP0A1 is an example of how to implement the PIP0A1 Notification of Failure.
The `PIP0A1Processes` folder contains business process definitions for PIP0A1.
The `PIP0A1Schemas` schema project contains the schema file for the PIP0A1 message.

The PIP0A1 demonstrates how to set up business logic, properly process error messages, and then, based on the business logic and error messages, open the correct channel to use when sending notifications of failure to the appropriate failure administrator. In this tutorial, the PIP0A1 Notification of Failure scenarios are:

- The Buyer side of PIP 3A4 example found the Purchase Order Syntactically correct and sent a Receipt Acknowledgment. However, when while the private process is processing, it finds a disagreement/error with the Purchase Order Response. Since the Seller process would have completed its instance, a new instance of PIP0A1 is started to notify the Seller about the disagreement/error.

- The Receiver side of PIP3B2 finds a disagreement/error while processing the Advance Shipment Notification in its Private process. However, the participant side has already sent a Receipt Acknowledgment after syntactically verifying the Advance Shipment Notification from the Shipper. Since the Shipper instance would have completed upon Acknowledgment receipt, a new instance of PIP0A1 is started to notify the Shipper about the disagreement/error.

Note: For demonstration purposes, the business processes in the tutorial examples are purposely configured to trigger Notification of Failure.

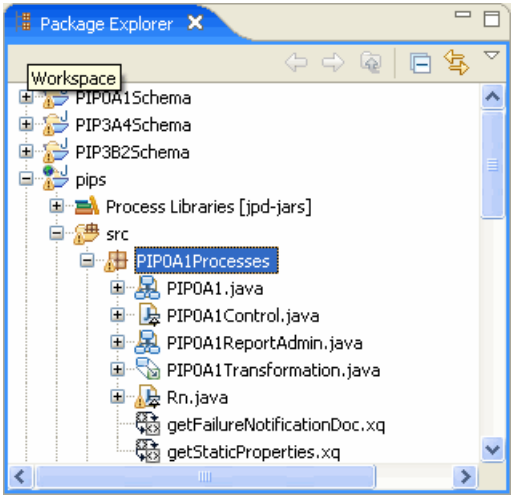
The following steps provides a brief overview of the PIP A01 business logic:

1. The PIP0A1 Failure Notifier business process (`PIP0A1.java`) receives a Notification of Failure request from either the PIP3A4 Seller or the PIP3B2 Receiver side.
2. The Failure Notifier processes the Notification of Failure request and sends a Notification of Failure message to the appropriate Report Administrator (in this case the `PIP0A1ReportAdmin.java`). The Failure Notifier process also writes a notification on the WebLogic Server Console to show that it has sent the message.
3. The Report Administrator process validates the Notification of Failure message and sends a receipt acknowledgement to the Failure Notifier. The process also writes a notification on the WebLogic Server Console to show that it has received the message.

Components of the PIP0A1 Example

When you open the `PIP0A1Processes` folder in WorkSpace Studio, the Package Explorer pane displays the contents of the folder (see Figure 2-3).

Figure 2-3 PIP0A1 Processes



The following table summarizes the components of the PIP0A1 example:

Table 2-4 Components of the PIP0A1: Notification of Failure Example

Role / Component	Description
`PIPA01.java`	The initiator business process that receives the Notification of Failure from the PIP3A4 initiator or the PIP3B2 participant. It then sends a Notification of Failure to the failure administrator business process by using an instance of the RosettaNet control.
`PIPA01Control.java`	Process control definition file that wraps `PIPA01.java` for use in other processes. Automatically generated by right-clicking the `PIPA01.java` file in the Package Explorer tab and choosing **Generate** > **Process Control**.
`Rn.java`	RosettaNet control definition file used to exchange messages with the failure administrator business process. For more information about the RosettaNet control, see RosettaNet Control.

Table 2-4 Components of the PIP0A1: Notification of Failure Example (Continued)

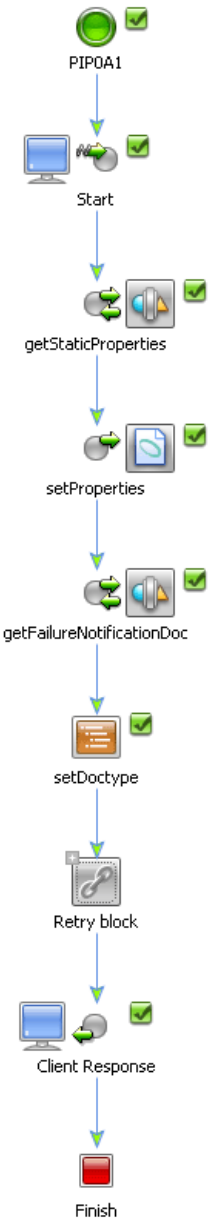
`PIP0A1ReportAdmin.java`	Business process that demonstrates the back end business logic used to process error messages and notifications.
`PIP0A1Transformation.java`	Transformation control definition file which is used to generate XQueries. To learn more about transformation and XQueries, see Guide to Data Transformation.
`getStaticProperties.xq`	XQuery that was generated when the getStaticProp Control Send with Return node was configured.
`getFailureNotificationDoc.xq`	XQuery that was generated when the getFailureNotificationDoc Control Send with Return node was configured.

Walkthrough of the Failure Notifier Business Process

This section describes the example initiator business process (`PIP0A1.java`). To view this business process:

1. In the Package Explorer pane in WorkSpace Studio, expand **pips > src**, and open the `PIP0A1Processes` folder.
2. Double click the `PIP0A1.java` file.
3. If necessary, collapse the Retry block paths to see only the main (success) path.

Figure 2-4 PIP0A1 Business Process



Success Process Path

The success path of the business process is as follows:

1. The Client Request node named **Start** receives a notification of failure message.
2. The Control Send with Return node named **getStaticProperties** extracts the `RosettaNet Context` properties from the incoming message. For more information about `RosettaNet Context`, see Interface RosettaNet Control.
3. The **setProperties** Control Send node maps the extracted properties to a document based on the PIP0A1 Schema
4. The Control Send with Return node named **getFailureNotification** uses the PIPA01 transformation query to map the `RosettaNet Context` properties to a document based on the PIPA01 Schema.

   To view the fields that are being mapped, in the Package Explorer pane, double click on the `getFailureNotification.xq` file.
5. The Perform node named **setDocType** sets the DOCTYPE DTD value in the transformation output.

   Note: This step is necessary since the WebLogic Integration transformation output by default does not contain the DOCTYPE property.
6. The group named **Retry block** contains a Control Send node named **Send Notification of Failure** and a Control Receive node named **Receipt acknowledgement**. This block sends a notification of failure to the `PIP0A1ReportAdmin` business process and waits for an acknowledgement that the message was received.

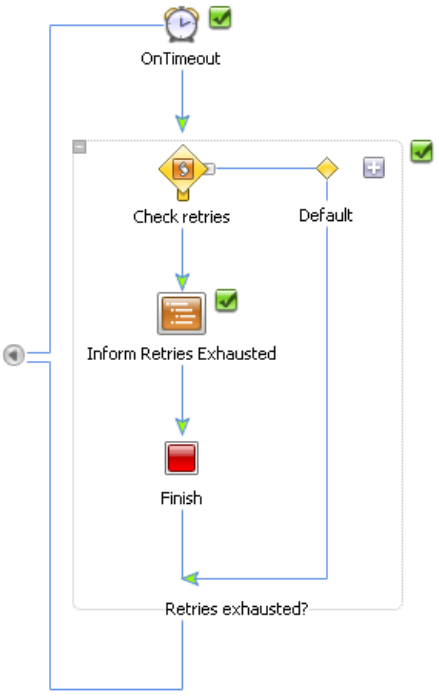
   For demonstration purposes, the Send Notification of Failure node also prints a message to the WebLogic Server Console after it has sent the message to the `PIP0A1ReportAdmin` process.
7. The Client Response node at the end of the business process is a place holder for any business logic that would take place after the Notification of Failure message is sent to the `PIP0A1ReportAdmin` business process.

Failure Paths

The OnTimeout path handle failure management, such as in the event of a network failure. To view the failure path:

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

Figure 2-5 OnTimeout Path

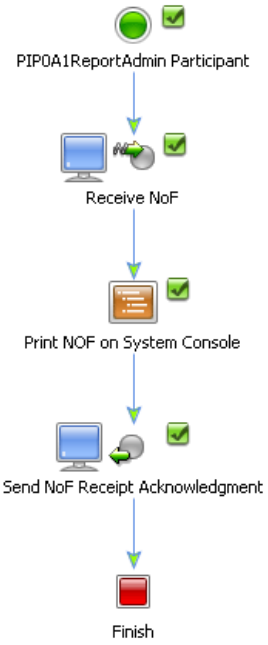


This path represents a standard BPM OnTimeout construct. The timeout value is set to two hours (the standard PIP retry interval) and three retries (the standard PIP retry count). Thus, if for any reason the acknowledgement fails to arrive within two hours, the group will be retried and the **sendNotification of Failure** step will be executed again. The OnTimeout path contains a condition that determines whether retries have been exhausted and, if so, logic could be added to the path which takes the appropriate action.

Walkthrough of the Report Administrator Business Process

The participant business process of the PIP0A1 example is named `PIPA01ReportAdmin.java`. To view the participant business process in WorkSpace Studio, in the Package Explorer pane, double click on `PIPA01ReportAdmin.java`.

Figure 2-6 Report Administrator Business Process



The Report Administrator business process includes the following steps:

1. The Notification of Failure message arrives at the Client Request node named **Receive NoF**.
2. The Perform node named **Print NOF on System Console** prints an acknowledgement on the WebLogic Server Console window. This node is a place holder for any business logic that would take place in a PIP0A1 Report Administrator business process used in production mode.
3. The Client Response node named **Send NoF Receipt Acknowledgment** sends an acknowledgement back to the Failure Notifier process.

The following two examples in this tutorial, both utilize the PIP0A1 notification of failure example. To see how the example works, complete either "Running the PIP3B2 Example" on page 2-25 or "Running the PIP3A4 Example" on page 2-37.

Step 3: Open the PIP3B2: Notify of Advance Shipment Example

This topic describes the example implementation of PIP3B2: Notify of Advance Shipment. It contains the following sections:

For detailed information about PIP3B2, see http://www.rosettanet.org.

About the PIP3B2 Example

PIP3B2 is an example of the asynchronous single action activity design pattern described in "RosettaNet Design Patterns" on page 2-5. The `PIP3B2Processes` folder contains business process definitions for PIP3B2. The `PIP3B2Schemas` schema project contains the schema file for the PIP3B2 message. The `sampledata` directory contains sample XML documents that you can use in section "Running the PIP3B2 Example" on page 2-25.

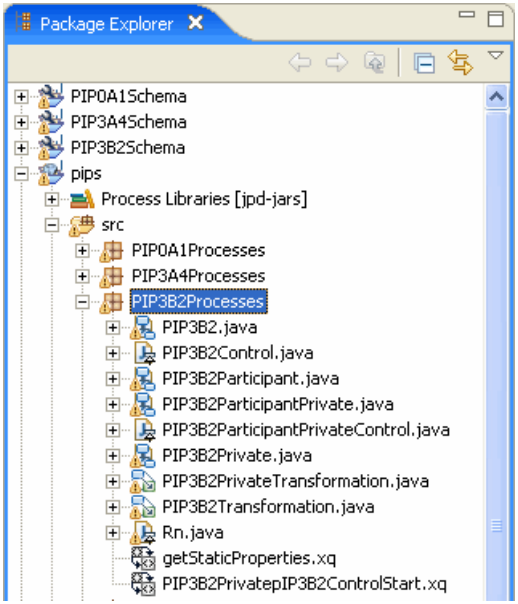
The following steps provides a brief overview of the PIP3B2 business logic:

1. The Shipper (initiator process `PIP3B2.java`) constructs an Notify of Advance Shipment message and sends it to the Receiver (participant process- `PIP3B2Participant.java`).
2. The Receiver validates the message and sends a receipt acknowledgement to the Shipper.
3. The Receiver submits the message to the backend system (`PIP3B2ParticipantPrivate.java`) for further processing.
4. The backend system encounters a problem during processing and invokes the PIP0A1 example.

Components of the PIP3B2 Example

When you open the `PIP3B2Processes` folder in WorkSpace Studio, the Package Explorer pane displays the contents of the folder (see Figure 2-7).

Figure 2-7 PIP3B2 Processes



The following table summarizes the components of the PIP3B2 example:

Table 2-5 Components of the PIP3B2: Notify of Advance Shipment Example

Role / Component	Description
Shipper (Initiator)	RosettaNet role name.
`PIP3B2.java`	Public initiator business process that sends the message, waits for the acknowledgement, and handles failures. Uses `Rn.java`, a RosettaNet control instance.
`Rn.java`	RosettaNet control definition file which is used to exchange messages with the Receiver via RNIF. For more information about the RosettaNet control, see RosettaNet Control. The annotations on this control instance in `PIP3B2.java` include the PIP name and version, role names.

Table 2-5 Components of the PIP3B2: Notify of Advance Shipment Example (Continued)

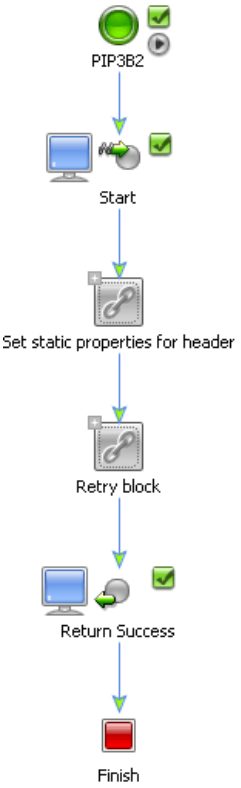
`PIP3B2Control.java`	Process control definition file that wraps `PIP3B2.java` for use in other processes. Automatically generated by right-clicking the `PIP3B2.java` file in the Package Explorer pane, and choosing **Generate** > **Process Control**.
`PIP3B2Private.java`	Private business process that invokes the `PIP3B2.process` 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 PIP3B2.
`PIP3B2PrivateTransformation.dtf` and `PIP3B2PrivatepIP3B2ControlStart.xq` (the associated `.xq` file)	Example transformation file. Automatically generated when a transformation is defined in a process node.
`PIP3B2Transformation.java` and `getStaticProperties.xq` (the associated `.xq` file)	Example transformation file. Automatically generated when a transformation is defined in a process node.
Receiver (Participant)	RosettaNet role name.
`PIP3B2Participant.java`	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.
`PIP3B2ParticipantPrivate.java`	Private business process that represents the backend processing of the shipment notice.
`PIP3B2ParticipantPrivateControl.java`	Process control that wraps the private process for use in `PIP3B2Participant.java`. Automatically generated by right-clicking the `PIP3B2ParticipantPrivate.java` file in the Package Explorer pane, and choosing **Generate** > **Process Control**.

Walkthrough of the Shipper Business Process

This section describes the example initiator business process (`PIP3B2.java`) for PIP3B2. To view this business process:

1. On the Package Explorer pane in WorkSpace Studio, expand **pips > src** and open the `PIP3B2Processes` folder.
2. Double-click the `PIP3B2.java` file.
3. If necessary, collapse the OnTimeout and On Error Message paths, as well as the Set static properties for header and Retry block groups to see only the main (success) path.

Figure 2-8 PIP3B2 Business Process



Success Path

The success path is the main path of the business process:

1. The Client Receive node named **Start** invokes the process when the Notify of Advance Shipment XML document (in this tutorial, the client request comes from the private process) is received.
2. The **Set static properties for header** group contains a Control Send with Return node named **getStaticProperties** and a Control Send node named **setStaticProperties**. These two nodes extract the RosettaNet properties from the message and then map them to a document based on the PIP3B2 Schema.
3. The **Retry block group** contains a two nodes. The Control Send node named **sendMessage** sends a message to the Receiver (participant) via the `sendMessage` method on the RosettaNet control. The Control Receive node named **Receipt acknowledgement** waits for the receipt acknowledgement callback (`rn_onAck` method) from the Receiver.
4. Once the acknowledgement is received, the process responds to the private process via the Client Response node named **Return Success**.

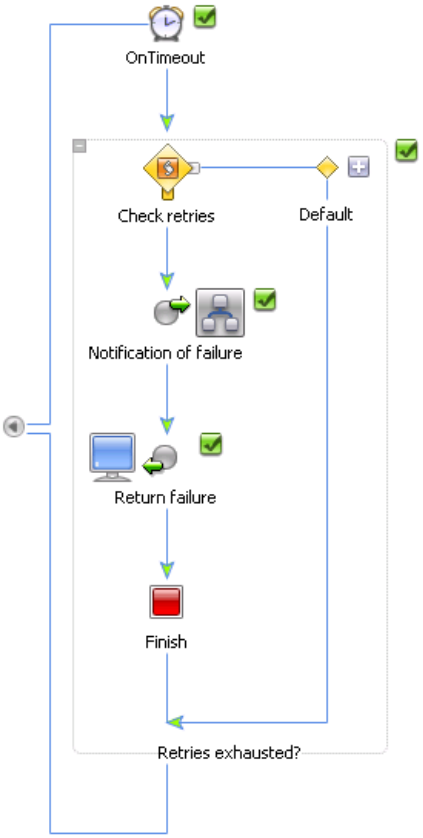
Because PIP3B2 implements the single-action activity design pattern, no business action is returned from the Receiver—only a business signal (Receipt Acknowledgment or Exception). The response to the private process merely indicates a successful completion and returns no data.

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**.

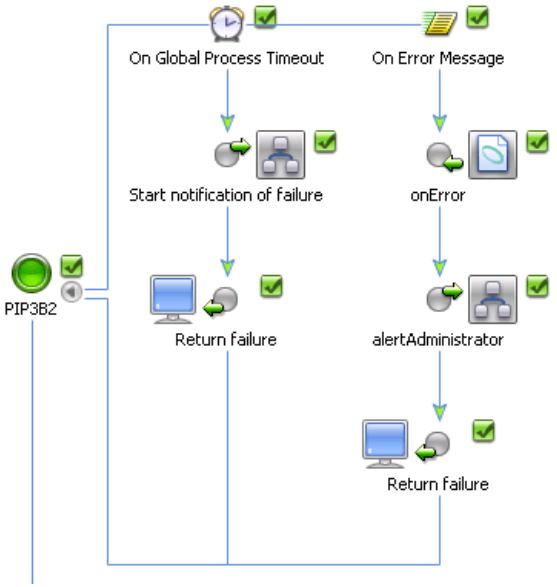
Figure 2-9 OnTimeout Path



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.

Figure 2-10 OnError Message Path



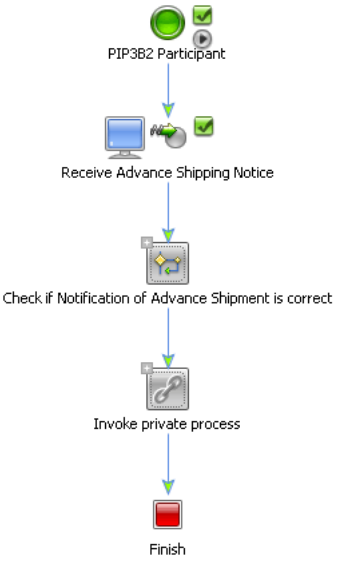
The Global Process Timeout is configured for the standard PIP timeout value of 24 hours. If the whole process fails to complete within this time, the PIP0A1 (Notification of Failure) subprocess is triggered and an error is returned to the private process.

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 PIP0A1 (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 Receiver Business Process

To view the participant business process in WorkSpace Studio, open the `PIP3B2Participant.java` file.

Figure 2-11 PIP3B2Participant Business Process



The participant business process includes the following nodes:

1. The Client Request node named **Receive Advance Shipping Notice** receives the Notify of Advance Shipment message.
2. The **Check if Notification of Advance Shipment** Decision node validates the message and:
   - Sends a receipt acknowledgement through the **Send receipt acknowledgement** Client Response node if the message is correct.
   - Sends an error message through the **send Exception** Client Response node if there is a problem with the message.
3. In the **Invoke private process** group, the Control Send node named **processShipment**, passes the Notify of Advance Shipment to a private business process for further processing while the **get Private Response** Control Receive node waits for an acknowledgement from the private process.

Walkthrough of the Private Business Processes

The private processes on both sides are place holders for processes which in production environments would interact with 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.

In this example, the `PIP3B2ParticipantPrivate.java` is purposely configured to trigger an error and invoke the PIP0A1 example when you run the process.

Running the PIP3B2 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.

To run the PIP3B2 example:

1. If WebLogic Server is not already running, from the BEA Workshop menu, choose **Window > Show View > Other > Server > Servers**, and click **OK**. A **Server** view is displayed in which the Server and its state are shown.
2. In the **Package Explorer,** expand **Pips > PIP3B2Processes**, select and right-click on **PIP3B2Participant.java**, click **Run As**, and click **Run On Server**.
3. In the Define a New Server dialog box, select either a **Choose an existing server** option or **Manually define a server** (if there is no server defined), and click **Next**.
4. In the **BEA WebLogic v10.0 Server** dialog box, to manually define a server, click **Browse**, and select the WebLogic Integration domain that you created in "Step 1: Create a New WebLogic Integration Domain" on page 1-2 (such as `c:\bea\user_projects\domains\tptutorial`), and then click **OK**.
5. Click **Finish**.

   The samples domain integration server is started, and the RequestQuote application is deployed on it. When WebLogic Server is running, the following indicator is visible in the Servers view:

6. After the application is deployed, the Test Browser is displayed.
7. Click the Test Form tab.
8. In the Test Form page, Click **Browse**, beside the xml pro (file value) field. and navigate to `[file location]\rosettanet\pips\WebContent\PIP3B2Processes\sampledata\3B2AdvanceShipmentNotificationMessageBase.xml`

   Where `[file location]` is the directory in which you installed the tutorial files as described in "Step 3: Install the Tutorial Files" on page 1-9.
9. Click **`onMessage`**.
10. In a few moments, the end-to-end choreography will execute. The client response will echo back the XML message.Bring up your WebLogic Server Console
11. In a few moments, you should see the following messages on the console:
    - `>>>>> PIP0A1.jpd: Sent Notification of Failure`
    - `>>>>> PIP0A1ReportAdmin.jpd: GOT Notification of Failure.`
12. Optionally, you can open the WebLogic Integration Administration Console and observe message tracking and process tracking entries.

Step 4: Open the PIP3A4: Request Purchase Order Example

This topic describes the example implementation of the PIP3A4: Request Purchase Order. It contains the following sections:

- About the PIP3A4 Example
- Components of the PIP3A4 Example
- Walkthrough of the Seller Business Process
- Walkthrough of the Buyer Business Process
- Walkthrough of the Private Business Processes
- Running the PIP3A4 Example

For detailed information about PIP3A4, see http://www.rosettanet.org.

About the PIP3A4 Example

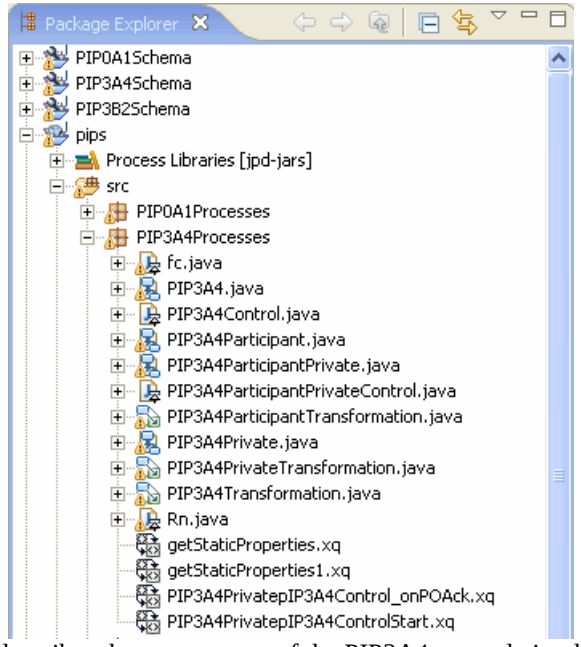
PIP3A4 is an example of the asynchronous two-action activity design pattern described in "RosettaNet Design Patterns" on page 2-5. The `PIP3A4Processes` folder contains business process definitions for PIP3A4. The `PIP3A4Schemas` schema project contains the schema file for the PIP3A4 message. The `sampledata` directory contains sample XML documents that you can use when "Running the PIP3A4 Example" on page 2-37.

The PIP3A4 Request Purchase Order 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, but encounters an error during processing which triggers the PIP A01 example.

Components of the PIP3A4 Example

When you open the `PIP3A4Processes` folder in WorkSpace Studio, the Package Explorer pane displays the contents of the folder.



The following table describes the components of the PIP3A4 example implementation.

Table 2-6 Components of the PIP3A4 (Request Purchase Order) Example

Role / Component	Description
Buyer (Initiator)	RosettaNet role name.
`PIP3A4.java`	Public business process that exchanges messages with the seller, waits for the acknowledgement, and handles failures. Uses the RosettaNet control instance (`Rn.java`).
`Rn.java`	RosettaNet control definition file used to exchange messages with the Seller via RNIF. For more information about the RosettaNet control, see RosettaNet Control. The annotations on this control instance in `PIP3A4.java` include the PIP name, version, and role names.

Table 2-6 Components of the PIP3A4 (Request Purchase Order) Example (Continued)

`PIP3A4Control.java`	Process control definition file that wraps `PIP3A4.java` for use in other processes. Automatically generated by right-clicking the `PIP3A4.java` file in the Package Explorer pane and choosing **Generate > Process Control**.
`PIP3A4Private.java`	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 PIP3A4.
`PIP3A4PrivateTransformation.dtf` and the associated `.xq` files `PIP3A4Transformation.dtf` and the associated `getStaticProperties.xq` file	Example transformation file. Automatically generated when a transformation is defined in a process node.
Seller (Participant)	RosettaNet role name.
`PIP3A4Participant.java`	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 Annotation Type RosettaNet.
`PIP3A4ParticipantPrivate.java`	Private business process that represents the backend processing of the purchase order and the creation of the PO Confirmation.
`fc.java`	File control definition file which is used to read the PO Confirmation sample file from the local file system.

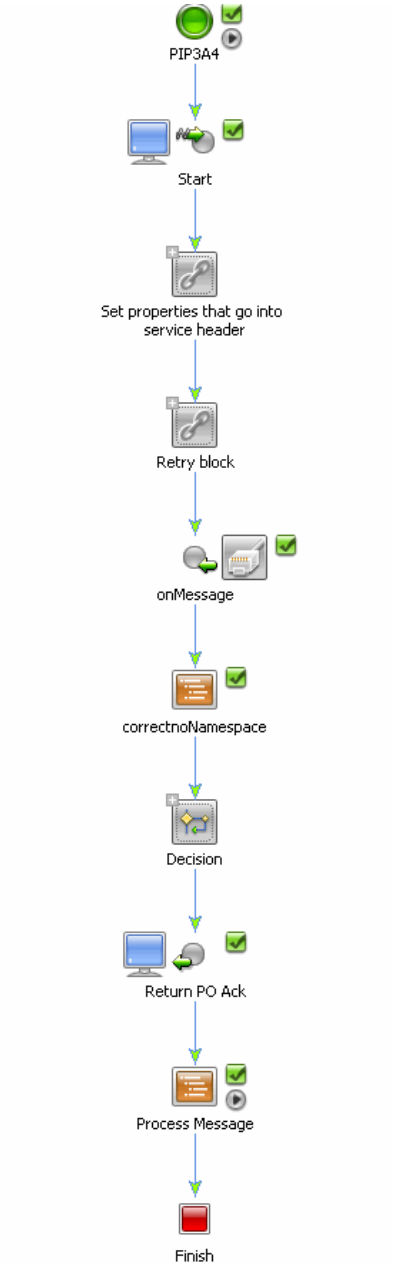
Table 2-6 Components of the PIP3A4 (Request Purchase Order) Example (Continued)

`PIP3A4ParticipantPrivateControl.java`	Process control that wraps the private process for use in `PIP3A4Participant.java` Automatically generated by right-clicking the `PIP3A4ParticipantPrivate.java` file in the Package Explorer pane and choosing **Generate > Process Control**.
`PIP3A4ParticipantTransformation.java` and the associated `getStaticProperties1.xq` file	Example transformation file. Automatically generated when a transformation is defined in a process node.

Walkthrough of the Seller Business Process

This section describes the example initiator business process (`PIP3A4.java`) for PIP3B2. To view this business process:

1. In WorkSpace Studio, open the `PIP3A4.java` file.
2. If necessary, collapse the OnTimeout and On Error Message paths, as well as any groups to see only the main (success) path.


PIP3A4
Start
Set properties that go into service header
Retry block
onMessage
correctnoNamespace
Decision
Return PO Ack
Process Message
Finish

Success Path

The success path business logic is as follows:

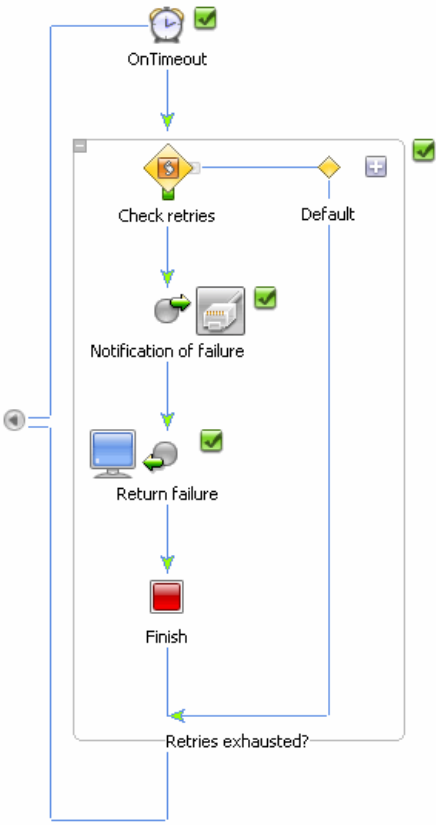
1. The Client Request node named **Start** receives the Purchase Order Request XML document message (in this tutorial, the client request comes from the private process).
2. The **Set properties that go into service header** group contains a Control Send with Return node named **getStaticProperties** and a Control Send node named **setStaticProperties**. These two nodes extract the RosettaNet properties from the message and then map them to a document based on the PIP3B2 Schema.
3. The **Retry block group** contains two nodes. The Control Send node named **sendMessage** sends a message to the Seller (participant) via the `sendMessage` method on the RosettaNet control. The Control Receive node named **Receipt acknowledgement** waits for 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 at the Control Receive node named **onMessage**.
5. The **Correctnonamespace** Perform Node adds the correct name space to the received XML Bean. (This step is necessary since Schemas in the project folder cannot have the same name space.)
6. After receiving the Purchase Order Confirmation, the **Decision** node validates the message and:
   - Sends a receipt acknowledgement through the **Send PO Ack receipt acknowledgement** Control Send node if the message is correct.
   - Sends an error message through the **sendError** Control Send node if there is a problem with the message.
7. The business process responds to the private process with the PO Confirmation through the **Return PO ack** Client Response node.
8. The **Process Message** Perform node is purposely configured to trigger an error and invoke the PIP0A1 example.

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

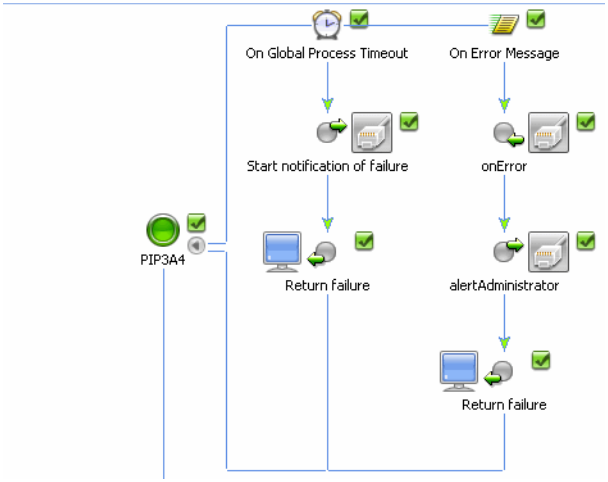
The OnTimeout and On Error Message paths handle failure management, such as in the event of a network failure. To view the failure paths:

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.

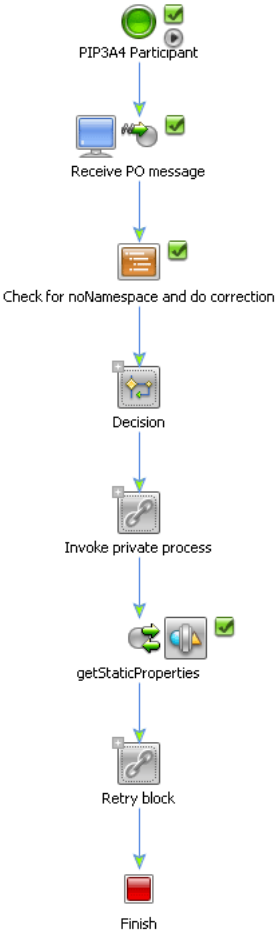


The Global Process Timeout is configured for the standard PIP timeout value of 24 hours. If the whole process fails to complete within this time, the PIP0A1 (Notification of Failure) subprocess is triggered and an error is returned to the private process.

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 PIP0A1 (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 Buyer Business Process

The participant business process (`PIP3A4Participant.java`) on the Seller side is less complex than the initiator business process. To view the participant business process in WorkSpace Studio, open the `PIP3A4Participant.java` file.



The participant business process includes the following steps:

1. The Client Request node **Receive PO message** receives the Purchase Order Request message.
2. The **Check for noNamespace and do the correction** Perform node adds the correct name space to the received XML Bean. (This step is necessary since Schemas in the project folder cannot have the same name space.)
3. The **Decision** node validates the PO:

  - Sends a receipt acknowledgement to the Buyer through the **Send receipt acknowledgement** Client Response node if the PO is correct.
  - Sends an error message through the send Exception Client Response node if there is a problem with the PO.

4. In the **Invoke Private Process group**, the **processOrder** Control Send node passes the PO Request to a private business process for further processing, while the **onPOAck** Control Receive node waits for the response back from the private process. The private process constructs the response that was sent after reading the confirmation using a File Control.
5. The Controls Send with Return node named **getStaticProp** extracts the `RosettaNet Context` properties from the incoming message. These properties are needed only in case of errors. These properties can be saved for later use and contains the information that is needed to construct the PIP0A1 Notification of Failure. For example: from / to / message id / etc.
6. In the **Retry block** group, the **Send PO Acknowledgement** Client Response node sends the Purchase Order Confirmation message to the buyer using the `sendReply` callback method, while the Receive receipt acknowledgement **Client Request** node waits for the receipt acknowledgement.

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 Shipper Business Process" on page 2-20. 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 place holders for backend business logic. The participant private process actually echoes the received message. 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, 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 PIP3A4 Example

To run the PIP3A4 example:

1. This example uses a File Control to read the PO Confirmation message from your local file system. Before you can run the example, you have to set the directory path in the `fc.java` file so that it can find the correct sample data file:

   a. In the Package Explorer pane, double-click the `fc.java` file.

   b. In the Source view, set the `directory-name` attribute to:

   ```
   [file location]\tptutorialapps\rosettanet\PIP3A4Processes
   ```

   Where `[file location]` is the directory in which you installed the tutorial files as described in "Step 3: Install the Tutorial Files" on page 1-9.

2. If WebLogic Server is not already running, from the BEA WorkSpace Studio menu, choose **Window > Show View > Other Server > Servers**, and click **OK**. A Server view is displayed in which the Server and its state are shown.

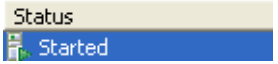
3. In the Package Explorer, expand PIP3A4Processes and select and right-click on `PIP3A4Participant.java`, click **Run As**, and click **Run On Server**.

4. In the Define a New Server dialog box, select either a **Choose an existing server** option or **Manually define a server** (if there is no server defined), and click **Next**.

5. In the **BEA WebLogic v10.0 Server** dialog box, to manually define a server, click **Browse**, and select the WebLogic Integration domain that you created in "Step 1: Create a New WebLogic Integration Domain" on page 1-2 (such as `c:\bea\user_projects\domains\tptutorial`), and then click **OK**.

6. Click **Finish**.

   The samples domain integration server is started, and the RequestQuote application is deployed on it. When WebLogic Server is running, the following indicator is visible in the Servers view:

7. After the application is deployed, the Test Browser is displayed.

8. Click the Test Form tab.

9. In the Test Form page, Click **Browse**, beside the xml pro (file value) field. and navigate to `[file location]\rosettanet\pips\WebContent\PIP3A4Processes\sampledata\3A4PurchaseOrderRequestMessageBase_0010.xml`

   Where `[file location]` is the directory in which you installed the tutorial files as described in "Step 3: Install the Tutorial Files" on page 1-9.

10. Click **`onMessage`**.

    In a few moments, the end-to-end choreography will execute. The client response will echo back the XML message.

11. Bring up your WebLogic Server Console

    In a few moments, you should see the following messages on the console:

- `>>>>> PIP0A1.jpd: Sent Notification of Failure`
- `>>>>> PIP0A1ReportAdmin.jpd: GOT Notification of Failure.`

12. 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

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 PIP3A4. 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 PIP3A4, see http://www.rosettanet.org.

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 2-39.
3. Copy the example PIP implementation associated with the design pattern that you want to use.
4. In WorkSpace Studio, 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`, `to-role`, and `from-role` attributes (and others if needed) in the Interface RosettaNetControl.
   - For *public* participant business processes, you change the `pip-name`, `pip-role` and `pip-version` attributes (and others if needed) in the Annotation Type RosettaNet.
6. Rename the Process and control 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

The RosettaNet W3C XSD schemas for selected PIPs are included in the RosettaNet Self-Test Kit (STK) which can be downloaded from the Developer Tools area on the RosettaNet Ready Web site, available at: `http://www.rosettanet.org`.

If the schema for the PIP you are using is not available on this web site, it is possible to implement RosettaNet solutions in WebLogic integration using RosettaNet message definitions specified as DTD files. However, it is recommended to use W3C XSD files instead, since many of the WebLogic Integration tools such as the XQuery mapping tools (used to define data transformations) only support XSD schemas.

If you want to use the graphical XQuery mapping tools, you need to convert the RosettaNet DTD files to W3C XSD files (using a tool like XML Spy Enterprise Edition) and then import the XSD files into your project. Make sure that you refer to the RosettaNet Message Guidelines for the PIP which DTD file you are converting and add the appropriate validation rules for Service Content Validation to your XSD file after the conversion.

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 WorkSpace Studio 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 WorkSpace Studio. To fix the file, open it in WorkSpace Studio, 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. The name collisions can be avoided by using explicit `targetNamespace`. For example for the 3A4 Purchase Order Request W3C Schema in this tutorial we use: `<xs:schema id="Pip3A4PurchaseOrderRequest" targetNamespace="Pip3A4PurchaseOrderRequest" xmlns="Pip3A4PurchaseOrderRequest">`

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

CHAPTER 3

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. WorkSpace Studio 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 WorkSpace Studio 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

To use this tutorial, you must have:

- Installed WebLogic Platform 10.2 on your system according to the instructions in Installation Guide.
- 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*.
- For an introduction to ebXML solutions using WebLogic Integration, see Introducing ebXML Solutions in *Introducing Trading Partner Integration*.
- For more information about using the ebXML control in initiator business processes, see ebXML Control.
- For information about using participant business processes, see the following topics:
  - Building ebXML Participant Business Processes.
  - ebXML Annotation.

Tutorial Overview

In this tutorial, the first couple of examples contains detailed instructions of how to use the WorkSpace Studio 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 3-4

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 3-9

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 WorkSpace Studio Test Browser.

"Step 3: Selecting the Trading Partner Information Dynamically Through Typed XML" on page 3-26

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 3-35

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 3-42

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 3-51

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 3-58

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 3-63

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 3-68

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 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 WorkSpace Studio 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 3-4
- "Importing the Tutorial Sample Data" on page 3-6
- "Importing the Tutorial Schemas" on page 3-7

Creating the Business Process Application

WebLogic Integration extends the WorkSpace Studio 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 **BEA WorkSpace Studio** by choosing, **Start** > **All Programs** > **BEA** > **WorkSpace Studio 1.1.**

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

2. From the BEA WorkSpace Studio menu, click **File** > **New** > **Project**.

   The Select a wizard dialog box is displayed.

3. Expand WebLogic Integration and select **Process Application** and click **Next**.

   The Process Application dialog box is displayed.

4. In the Process Application dialog box, enter the following:

   a. In the **EAR Project Name** field, enter `ebXML`.

   b. In the **Web Project Name** field, `ebXMLWeb`.

   c. In the **Utility Project Name** field, enter `Schemas`.

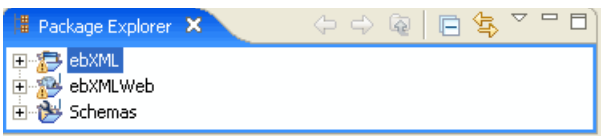
   d. Select **Add WebLogic Integration System and Control Schemas in Utility Project** check box. This adds the system schemas to the **Utility Project/schemas** folder.

   e. Click **Finish**.

5. In the displayed **Open Associated Perspective?** dialog box, click **Yes** to switch to Process Perspective.

   Your ebXML process application is created and displayed in the Package Explorer pane (see Figure 3-1).

Figure 3-1 ebXML Application



The components we will work with in this tutorial includes the following:

ebXML—This contains the JAR files and deployment descriptors build files and auto-generated files. J2EE Applications and their components are deployed on the WebLogic Server as EAR files.

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.)

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.

6. 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:

7. Go to **ebXMLWeb** > **src** > **processes**.
8. Right-click on the **processes** folder and select **Delete** from the drop-down menu.
9. 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.

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).

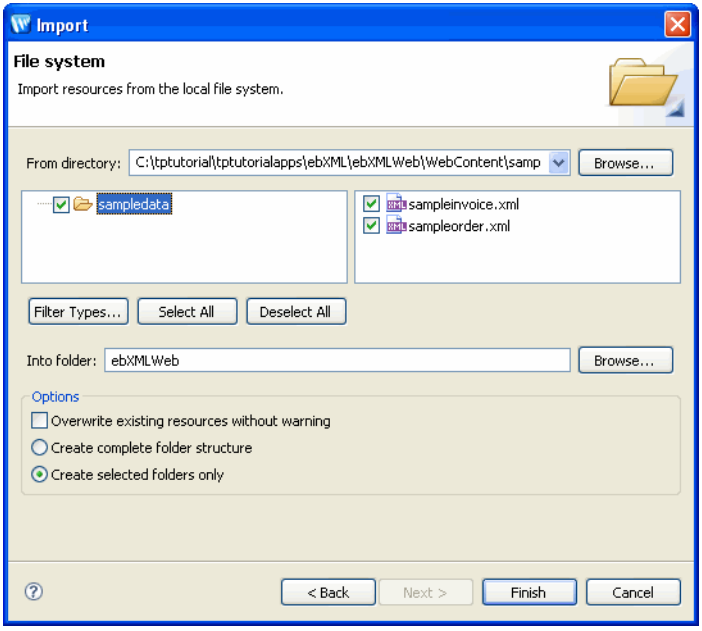
To Import the Sample Data Into Your Project Application

1. In the Package Explorer pane, right click on **ebxmlWeb** and select **Import**.

   The Import dialog box is displayed.

2. In the Import dialog box, select **General** > **File System**, and click **next**.
3. In File System window, click **Browse** next to From directory: and navigate to `[unzip location]\tptutorialapps\ebxml\ebxmlWeb\webContent\sampledata` 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. Check **Sampledata** check-box (see Figure 3-2).

Figure 3-2 Import File System



5. Click **Finish**.

   sampleinvoice.xml and **sampleorder.xml** is added to your Package Explorer pane under **ebXMLWeb**.

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.

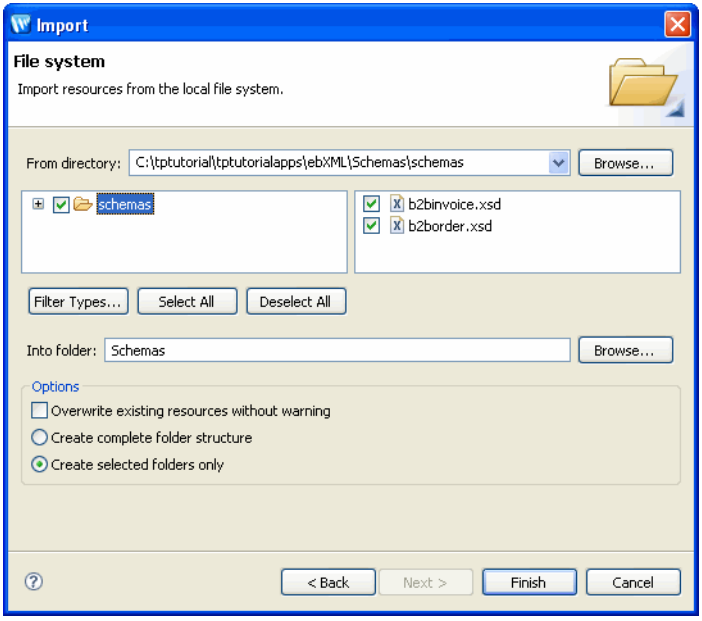
To Import the Tutorial Schemas

1. In the Package Explorer pane, right-click on **Schemas**.
2. Select **Import** from the drop-down menu.

   The **Import** dialog box is displayed.
3. In the **Import** dialog box, select **General > File System** and click **Next**.

4. In File System window, click **Browse** next to From directory: and navigate to: `[unzip location]\tptutorialapps\ebxml\Schemas\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).
5. Check **Schemas** check-box (see Figure 3-3).

Figure 3-3 Import Schemas



6. Click **Finish**.

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.

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 3-4 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

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

1. In the Package Explorer pane, expand **ebXMLWeb** and select **src**.
2. Right-click the **src** folder, then select **New** > **Package**.

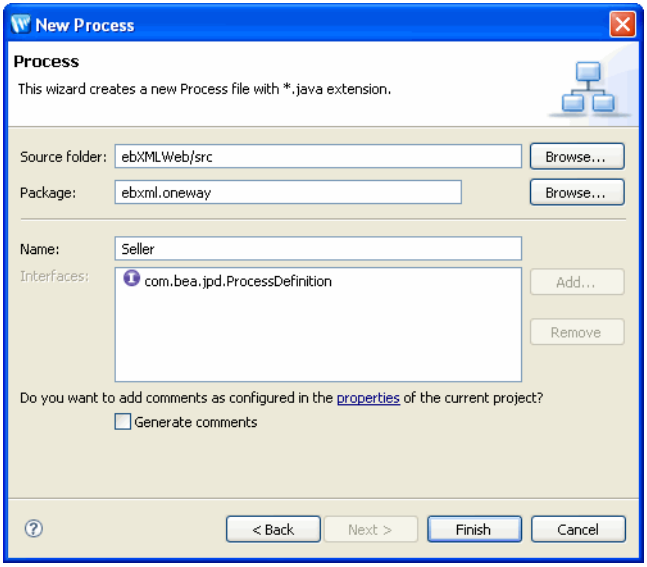
   The New Java Package dialog box is displayed.
3. In the New Java Package dialog box, enter **ebxml.oneway** as the name of the new package.
4. Click **Finish**.

   The **ebxml.oneway** package appears under **ebXMLWeb/src** directory in the Project Explorer.
5. Right-click **ebxml.oneway** package.
6. Select **New > Other**.

   The Select a Wizard dialog box is displayed.
7. Expand **WebLogic Integration** and select **ebXML Participant Process**.
8. Click **Next**.

   The New Process dialog box is displayed.
9. In the New Process dialog box, enter **Seller** in the **Name** field (see Figure 3-4).

Figure 3-4 New Process Dialog Box



10. Click **Finish**.

    A new ebXML participant process file is created in your **ebxml.oneway** package in the **Package Explorer** pane and is displayed in Design view (see Figure 3-5).

Figure 3-5 Seller ebXML participant process

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.

11. 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. If the Data Palette is not visible in BEA WorkSpace Studio, choose **Window > Show View > Data Palette** from the BEA WorkSpace Studio menu.

2. Click  on the **Data Palette**. A drop-down list of controls that represent the resources with which your business process can interact is displayed. Instances of controls already available in your project are displayed in the **Controls** tab.

3. Select **File** from the **Integrations Controls** drop-down menu.

   The **Insert Control: File** window opens.

4. In the **Insert File: Control** dialog box do the following:

   – In the **Field Name**, type the variable name as **File,** this is used to access the new File control instance from your business process. The name you enter must be a valid Java identifier.

   – Click **Next**.

5. In the **Create Control** dialog box enter the following details:

   – In the **Name** field, enter **File** as the name of the new control file that will be created.

   – Click **Next**.

6. The **Insert control - File** dialog-box appears.

7. In the **Insert control - File** dialog-box enter the following.

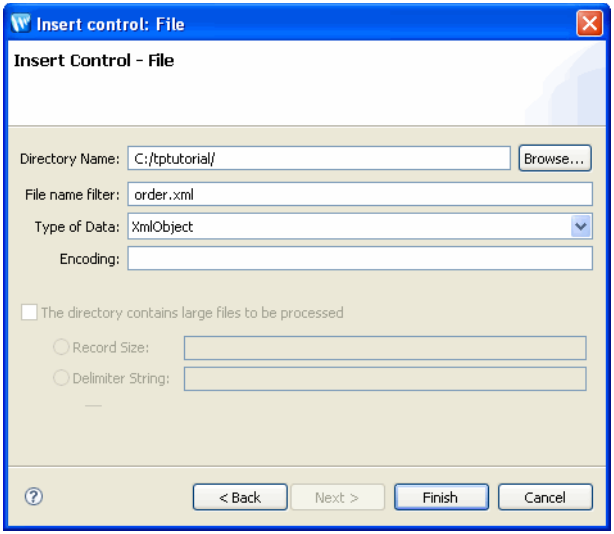
   – In the **Directory Name** field, enter **/tptutorial** as the directory-name. This is the directory to which the message received by the Seller.java will be written. In this tutorial, we always use the tptutorial directory that you created in the "Creating the

Read and Write Directories" on page 3-8 section as the directory to read from and write to.

– In the **File name filter** field, enter **order.xml** as the file name This is the name of the file to which the message received by the Seller.java will be written.

– Select **XmlObject** as the type of data contained in the received message from the **Type of Data** drop-down menu (see Figure 3-6).

Figure 3-6 File Control Properties



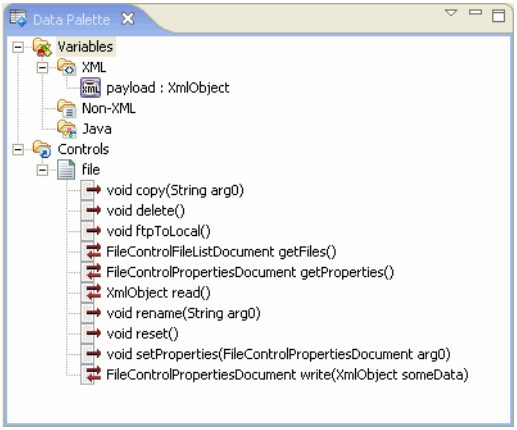
8. Click **Finish**.

   The **File** control instance is added to your list of controls in the **Data** palette, and a control file corresponding to the File control (**File.java**) is added to the Package Explorer pane. For more information about File controls, see File Control.

9. In the Data palette, under **Controls** expand the **File** control by clicking on the + next to it.

   The methods associated with the **File** control are displayed (see Figure 3-7).

Figure 3-7 File Control Methods



10. Select the `FileControlPropertiesDocument write(XmlObject someData)` method.

11. 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.

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

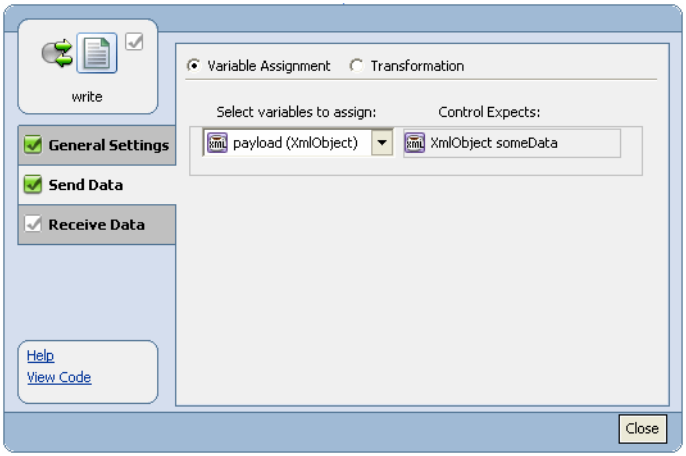
1. Double-click the new **write** node.

   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 (see Figure 3-8).

Figure 3-8 Configure Send Data



4. Click **Close** to close the node builder.

5. 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 JPD Configuration pane 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 JPD Configuration pane for the `Seller` class.

3. In the JPD Configuration pane, 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 ebxml Annotation.

We are now ready to test the Seller business process. WorkSpace Studio 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.

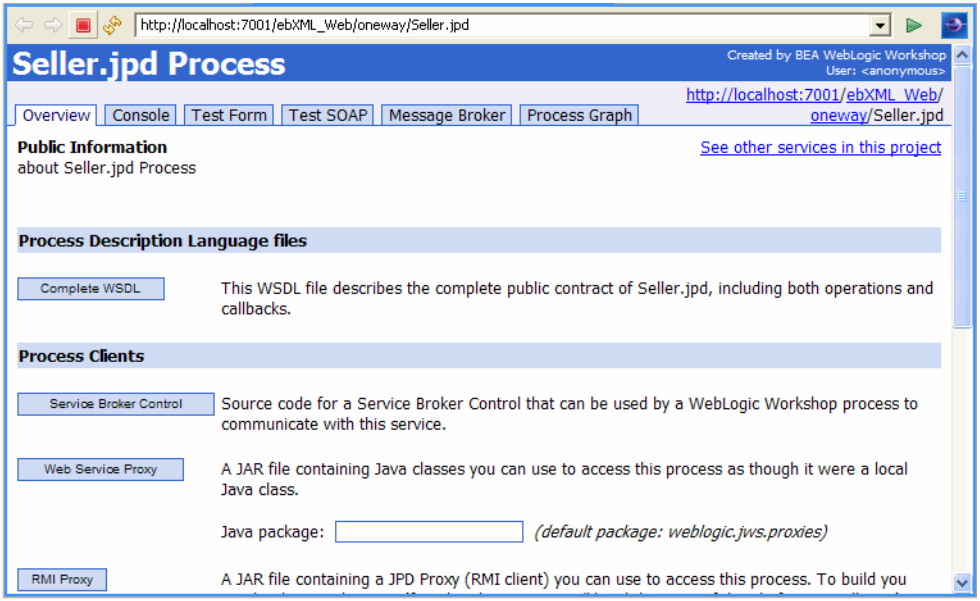
To Test the Seller Process

1. If the **Server** view is not visible in BEA WorkSpace Studio menu, choose **Window** > **Show View** > **Other** > **Server** > **Servers**, and click **Ok**. A **Server** view is displayed.
2. On the **Package Explorer** pane, select and right-click the **Seller.java** business process.
3. Click **Run As**, and **Run On Server**.
4. In the **Define a New Server** dialog box, select **Choose an existing server** option and click **Next**.
5. In the BEA WebLogic v10.0 Server dialog box, to manually define a server choose the server which you created when you set up your domain in "Step 1: Create a New WebLogic Integration Domain" on page 1-2.
6. Click **Finish**.
7. The server is started, and the application is deployed on it. When WebLogic Server is running, the following indicator is visible in the **Server** view (see Figure 3-9).

Figure 3-9 Sever Status

8. After the application is deployed, the Test Browser is displayed (see Figure 3-10).

Figure 3-10 Seller.jpd Process Test Browser



9. Click **Test Form** in the Test Browser window.
10. To enter test data, do one of the following:
    - Click **Browse**, and navigate to `C:\myappstptutorialapps\ebXML\ebXMLWeb\WebContent\sampledata\sampleorder.xml`.
    - Cut and paste the content of `C:\myapps\tptutorialapps\ebXML\ebXMLWeb\WebContent\sampledata\sampleorder.xml`, except for the first line of the file, into the xml variable (payload) field.
11. Click **request**.

    The Test Form page refreshes to display a summary of your request parameters.
12. Click **Refresh** to refresh the summary.
13. Scroll down to the **Operation request** section to see the content of the sampleorder.xml file.
14. Using a file browser or command line tool, navigate to your `C:\tptutorial` 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 3-10.

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.

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.

Building the Buyer Business Process

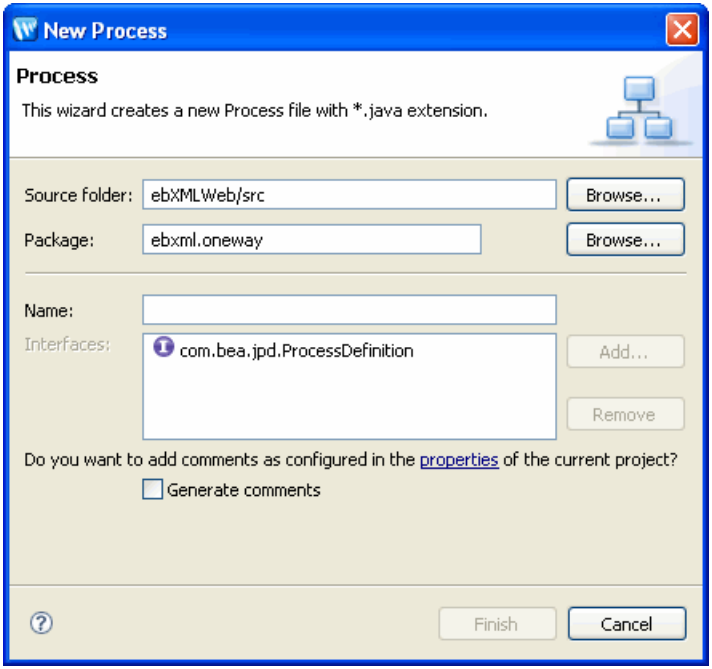
The Buyer business process is the initiator of our ebXML conversation. After creating the Buyer business process, we will use an ebXML control to communicate with the Seller participator business process. This section contains the following procedures:

To Create the Buyer Business Process File

1. In the Package Explorer pane, right-click on **ebxml.oneway** and select **New** > **Process**.

   The New Process dialog box opens (see Figure 3-11).

Figure 3-11 New Process

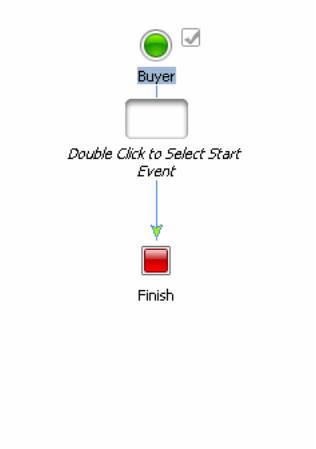


Since we are creating a initiator business process file this time and will use a ebXML file control to communicate with the participator process, we will create a default process file rather than an ebXML participant process file.

2. Enter **Buyer** in the **Name** field.
3. Click **Finish**.

   A new **Buyer.java** file is added to your Package Explorer pane and displayed in Design view (see Figure 3-12).

Figure 3-12 Buyer Business Process



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

4. Double-click on the *Start Event* place holder.

   The Starting Event selection window is displayed.

5. Select **Invoked via a Client Request** and click **Close**.

   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 **untyped XML** types and select **XmlObject**.

   e. Click **OK**.

      The XmlObject parameter type is added to the pane.

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 name **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 **Close**.

   Your Client Request node is completed. To learn more about Client Request nodes, see Receiving Messages From Clients. To learn more about variables and data types, see Working with Data Types.

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. Click **Menu** on the **Data Palette** and from the drop-down list choose **Integration Controls** to display the list of controls used for integrating applications.

Note: If the Data Palette view is not visible in BEA WorkSpace Studio, click **Window > Show View > Data Palette** from the menu bar.

2. Select **ebXML**.

   The **Insert Control: ebXML** window opens.

3. In the **Insert control: ebXML** dialog box enter **sellerControl** as the name of the variable to use for the control**.** This name must be a valid Java identifier.
4. Click **Next**.

   The **Create Control** wizard appears.

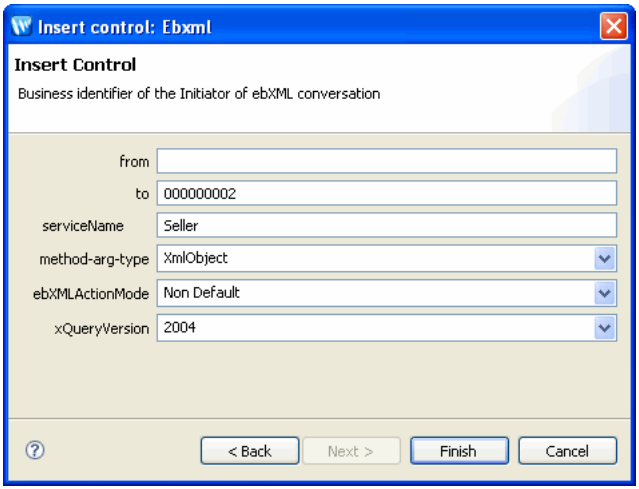
5. Enter **SellerControl** as the name of the new control file that will be created.

6. Click Next.

   The **Insert control: Ebxml** dialog box appears.

7. In the **Insert control: Ebxml** dialog box, specify the following information:
   - Leave the **from** field blank. This will default to the 000000001 at runtime, the default trading partner.
   - Enter **000000002** in the **to** field. This sets the participating trading partner id as a static value.
   - Enter servicename as **Seller**.
   - select **XmlObject** as the type of data contained in the received message from the **message-arg-type** drop-down menu.
   - Accept the default setting for ebXMLActionMode and xQueryVersion (see Figure 3-13).

Figure 3-13 Ebxml Control Properties



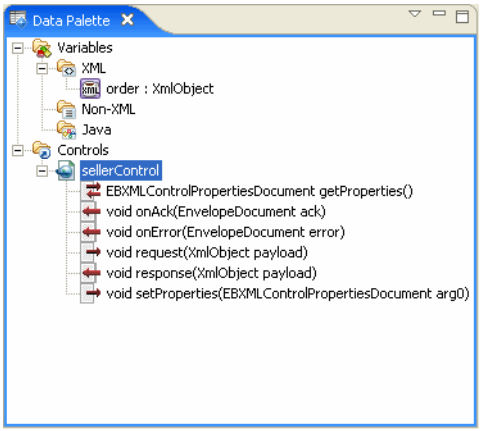
   - Click **Finish**.

The **ebXML** control instance is added to your list of controls in the Data palette, and a control file (SellerControl.java) corresponding to the **ebXML** control is added to the Package Explorer pane. You can double-click on the control file (SellerControl.java) to open it in Design view and review the values which you just entered. For more information about ebXML controls, see ebXML Control.

8. In the Data palette, expand the **ebXML** control (SellerControl) by clicking on the + next to it.

   The methods associated with the **ebXML** control are displayed (see Figure 3-14).

Figure 3-14 ebXML Control Methods



9. 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.

10. 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.

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 **Close** to close the node builder.

5. 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. If the Server view is not visible in BEA WorkSpace Studio, choose **Window** > **Show View** > **Other** > **Server** > **Servers**, and click **Ok**. A Server view is displayed.
2. On the Package Explorer pane, select and right-click the **Buyer.java** business process.
3. Click **Run As**, and **Run On Server**.
4. In the **Define a New Server dialog box**, select **Choose an existing server** option and click Next.
5. In the BEA WebLogic v10.0 Server dialog box, to manually define a server choose the server which you created when you set up your domain in "Step 1: Create a New WebLogic Integration Domain" on page 1-2.
6. Click **Finish**.
7. The server is started, and the application is deployed on it.
8. After the application is deployed, the Test Browser is displayed (see Figure 3-15).

Figure 3-15 Buyer.Jpd Test Browser

9. Click **Test Form** in the Test Browser window.
10. To enter test data, do one of the following:
    - Click **Browse**, and navigate to `C:\myapps\tptutorialapps\ebXML\ebXMLWeb\WebContent\sampledata\sampleorder.xml`.
    - Cut and paste the content of `C:\myapps\tptutorialapps\ebXML\ebXMLWeb\WebContent\sampledata\sampleorder.xml` into the xml variable (payload) field.
11. Click **startBuyer**.

    The Test Form page refreshes to display a summary of your request parameters.
12. Click **Refresh** to refresh the summary.
13. 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.

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 3-9. 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

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

1. In the Package Explorer pane, go to **ebXMLWeb > src**.
2. Right-click **src**, and select **New** > **Package**.

   The New Package dialog box is displayed.
3. Enter the name **ebxml.oneway.selector**.
4. Click **Finish**.

The **ebxml.oneway.selector** package appears under **ebXMLWeb/src** directory in the Package Explorer.

5. Right-click **ebxml.oneway.selector** package, then select **New** > **Other**.
6. In the New dialog box, expand **WebLogic Integration** and select **ebXML Participant Process**.
7. Click **Next**.

   The New Process dialog box is displayed.
8. In the New Process dialog box, enter **SelectorSeller** in the **Name** field.
9. Click **Finish**.

   A new ebXML participant process file is created in your **ebxml.oneway.selector** package in the Package Explorer pane and is displayed in Design view.
10. Delete the **Respond to request** node: right-click on the node and select **Delete** from the drop-down menu.

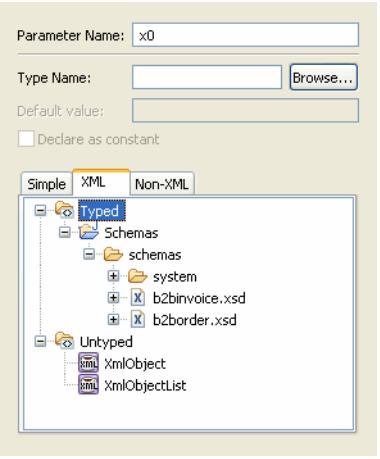
The next step involves configuring the Receive request node to accept XML data of a type that is specified in the Schemas which you imported into your application in "Importing the Tutorial Schemas" on page 3-7.

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**.
   b. Select **XmlObject payload**.
   c. Click **Remove**
   d. Click **Add**.
   e. If not already selected, select the **XML** option.
   f. Expand **Typed** and go to **Schemas > schemas** and select **b2border.xsd** (see Figure 3-16).

Figure 3-16 Selecting Schemas



g. Expand the **b2border.xsd** by clicking on the + next to it.

h. Select **Order** from the expanded list.

i. Click **OK**

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

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 **Close**.

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 3-12 to create a new **File** control. To save time, you can also drag and drop **File.java** from the **oneway** package in the **Package Explorer** pane directly on to the **Data Palette** under Controls**.**

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 **Close** to close the node builder.

5. 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. Right-click **SellectorSeller.java** and select **Run As > Run on Server**. (For more detailed instructions refer back to, "To Test the Seller Process" on page 3-16.)

2. After the application is deployed, the Test Browser is displayed.

3. To enter test data, do one of the following:

   – Click **Browse**, and navigate to `C:\myapps\tptutorialapps\ebXML\ebXMLWeb\WebContent\sampledata\sampleorder.xml`.

   – Cut and paste the content of `C:\myapps\tptutorialapps\ebXML\ebXMLWeb\sampledata\WebContent\sampleorder.xml` into the xml variable (payload) field.

4. Click **processOrder**

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

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

1. Right-click **ebxml.oneway.selector** package, and select **New > Process**.
2. Enter **SelectorBuyer** in the **Name** field.
3. Click **Finish**.

   A new **SelectorBuyer.java** file is added to your Package Explorer pane and displayed in Design view.
4. The **SelectorBuyer.java** file contains a Start node, a Start Event place holder and a Finish node.
5. Double-click on the Start Event place holder.

   The Starting Event selection window is displayed.
6. Select Invoked via a Client Request and click **Close**.

   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. Click **Add**.
   b. If not already selected, select the **XML** option.
   c. Expand **Typed** and go to **Schemas > schemas** and select **b2border.xsd**.
   d. Expand the **b2border.xsd** by clicking on the + next to it.
   e. Select **Order** from the expanded list.
   f. Click **OK**

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

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

To Create the ebXML Control

1. Click on the **Data Palette** and from the drop-down list choose **Integration Controls** to display the list of controls used for integrating applications.
2. Select **ebXML**.

   The **Insert Control: Ebxml** dialog box displays.

3. In the **Insert control: ebXML** dialog box enter**,** enter **selectorSellerControl** as the name of the variable to use for the control**.** This name must be a valid Java identifier.
4. Click **Next**.

   The **Create Control** wizard appears.
5. Enter **SelectorSellerControl** as the name of the new java file that will be created.
6. Click **Next**.

   The **Insert control: Ebxml** dialog box appears.
7. In the **Insert control: Ebxml** dialog box, specify the following information:
   - Leave the **from** field blank. This will default to the 000000001 at runtime, the default trading partner.
   - Leave the **to** field blank. This will be determined at runtime by the XQuery selector which you will define in the next section.
   - Enter servicename as **SelectorSeller**.
   - select **XmlObject** as the type of data contained in the received message from the **message-arg-type** drop-down menu.

   Accept the default setting for ebXMLActionMode and xQueryVersion
8. Click **Finish**.

   The **ebXML** control instance is added to your list of controls in the **Data** palette, and a java file (SelectorSwllerControl.java) corresponding to the **ebXML** control is added to the Package Explorer pane. You can double-click on the java file to open it in Design view and review the values which you just entered.

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 **Source** view.
3. Rename **request** as **processOrder** in the Source view.

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.

4. Change the line `void processOrder(XmlObject payload);` to `void processOrder(OrderDocument payload);`
5. The tooltip editor displays a suggested package to import, select `Import'OrderDocument'(com.bea.tutoriat.order).`

   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 3-7 rather than generic untyped XML.
6. Select **processOrder** in the Source view.
7. In the Properties pane, expand EBXMLControl.EbxmlMethod and add the following value in the **toselector**:

```
"declare namespace ns0=\"bea.com/tutorial/order\"
data($payload/ns0:Supplier_ID)
```

Note: If the Properties pane is not visible in BEA WorkSpace Studio, choose **Window > Show View > Other > Workshop > Properties** from the BEA WorkSpace Studio menu bar.

8. Select **File > Save** or enter **Ctrl+S** to save your work.
9. Close the **SelectorSellerControl.java** 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.

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 and XQuery selector to extract the target trading partner ID form the payload message rather than declaring it statically. The next section describes how to test your business processes.

To Test the SelectorBuyer Process

1. If the **Server** view is not visible in BEA WorkSpace Studio menu, choose **Window** > **Show View** > **Other** > **Server** > **Servers**, and click **Ok**. A **Server** view is displayed.
2. On the Package Explorer pane, select and right-click the **SelectorBuyer.java** business process.
3. Click **Run As**, and **Run On Server**.
4. In the **Define a New Server** dialog box, select **Choose an existing server** option and click **Next**.
5. Click **Finish**.
6. After the application is deployed, the Test Browser is displayed.
7. Click **Test Form** in the Test Browser window.
8. To enter test data, do one of the following:
   – Click **Browse**, and navigate to `C:\myapps\tptutorialapps\ebXML\ebXMLWeb\WebContent\sampledata\sampleorder.xml`.
   – Cut and paste the content of `C:\myapps\tptutorialapps\ebXML\ebXMLWeb\WebContent\sampledata\sampleorder.xml` into the xml variable (payload) field.
9. Click **clientRequest**.
10. 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 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

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 3-9. This section contains the following tasks:

To Create the BinarySeller Business Process

1. In the Project Explorer, expand **ebXMLWeb** and select **src** folder.
2. Right-click **src** folder, then select **New > Java Package**.
3. Enter **ebxml.oneway.binary**, as the name.
4. Click **Finish**.
5. Create a new ebXML participant process named **BinarySeller** under the **ebxml.oneway.binary** Package.
6. Delete the **Respond to request** node.
7. Modify the **Receive request** node as follows:
   a. In the **General Setting** tab, remove the **XmlObject** parameter data type.
   b. Click **Add**.
   c. Select the **Non-XML** option.

d. Go to **Untyped** and select **RawData** parameter data type.

e. Create a new variable named **data** of type RawData.

f. Click **Close**.

8. Create a new **File** control with the following values specified:
   - Variable name: **file**
   - New control name: **File**
   - directory-name: `/tptutorial/binary-in`
   - Type of Data: **RawData**
9. Drag and drop the **write** method (FileControlPropertiesDocument write(RawData someData) of the **file** control after the Receive request node in the business process.
10. 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. On the Package Explorer pane, select and right-click the **BinarySeller.java** business process.
2. Click **Run As**, and **Run On Server** (For more detailed instructions refer back to, "To Test the Seller Process" on page 3-16).
3. Use any binary file (for example an image file) as binary payload test data.
4. 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

Use the instructions outlined in detail in "Building the Buyer Business Process" on page 3-18 to complete the following tasks:

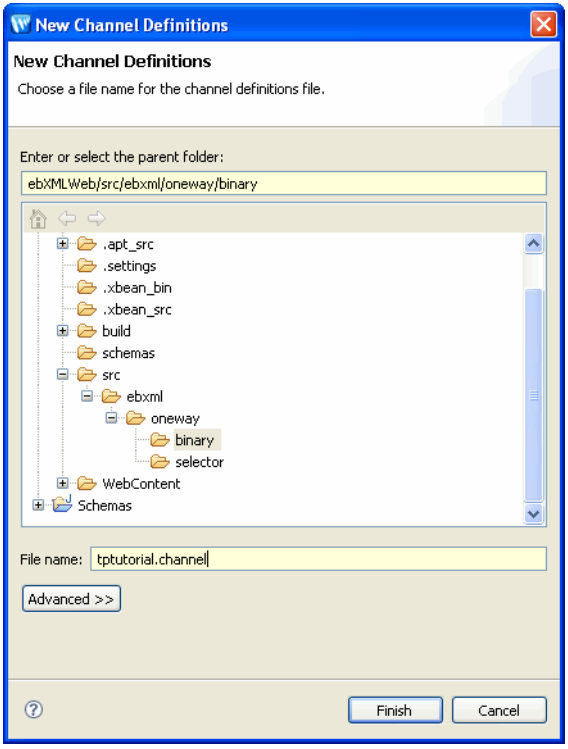
1. In the **ebxml.oneway.binary** package, create a new business process named **BinaryBuyer**.
2. Double-click the Start Event and select the **Subscribe to a Message Broker channel and start via an event (Timer, Email, File, Adapter, etc.)** option.
3. Click **Close**.

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 **Package Explorer** pane select **ebxml.oneway.binary**.
2. Right-click on the **ebxml.oneway.binary** folder.
3. Select **New > Channel Definitions**.
4. Enter `tptutorial.channel` as the file name (see Figure 3-17).

Figure 3-17 New Channel Definitions



5. Click **Finish**.

   The new channel file is created, added under the ebxml.oneway.binary Package in the Package Explorer pane, and displayed on screen. An application build is also started automatically to build the Schemas project.

6. In the channel file code, locate the following line:
   `<channel name="SampleRawDataChannel" messageType="rawData"/>`

7. Remove the comment tags `<!--` and `-->` preceding and following the line.

8. Select **File** > **Save** or enter **Ctrl+S** to save the channel file.

   The Schemas project builds again.

9. Close the Channel file.

The sample channels defined in the file are now available for selection in the subscription node.

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 **Close.**

You have now completed the configuration of your subscription node.

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 3-21 to create an ebXML with the following values:

   – Field name: **binarySellerControl**

   – Name: **BinarySellerControl**

   – ebxml-service-name: **BinarySeller** (must match the value in the participant process)

   – from: leave blank

   – to: **000000002**

   – method-arg-type: **RawData**

2. Click **Finish**.

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. If the **Server** view is not visible in BEA WorkSpace Studio menu, choose **Window > Show View > Other > Server > Servers**, and click **Ok**. A **Server** view is displayed.
2. On the **Package Explorer** pane, select and right-click the **BinaryBuyer.java** business process (For more detailed instructions refer back to, "To Test the Seller Process" on page 3-16).
3. After the application is deployed, the Test Browser is displayed, click **Test Form** in the Test Browser window.
4. Use an image file as the binary payload test data.
5. Click **subscription**.
6. 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 3-37 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 BinayBuyer 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.
3. After the Server is running, open the WebLogic Administration Console by selecting **Run > WebLogic Integration > WebLogic Integration Administration Console** from the WorkSpace Studio menu.
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 1-2 section.
5. Click **Event Generators** on the left pane.
6. Click **File > Create New** on the left pane.

The **Create a New File Event Generator** page opens.

7. Enter **TPTutorial1** 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**: <Base 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**:<Home 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.

To Test the Sending Raw Data Example.

1. On the Package Explorer pane, select and right-click the **BinaryBuyer.java** business process (For more detailed instructions refer back to, "To Test the Seller Process" on page 3-16).

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.

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 3-42
- "Building the RoundtripBuyer Business Process" on page 3-46

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 3-9 to complete the following sections:

- "To Create the Business Process File and Configure the Receive Request Node" on page 3-42
- "To Create the File Controls and Configure the Control Nodes" on page 3-43
- "To Configure the Respond to Request Node" on page 3-45
- "To Deploy the RoundtripSeller Business Process" on page 3-46

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

1. In the Package Explorer pane, expand **ebxmlWeb** and select **src**.
2. Right-click **src**, and select **New > Package**.

   The Java Package dialog box is displayed.

3. Enter **ebxml.roundtrip** in the Name field and click **Finish**.

   The **ebxml.roundtrip** Package appears under **ebxmlWeb\src**.

4. In the roundtrip Package, create a new ebXML participant process named **RoundtripSeller**.
5. Double-click the Receive request node and configure as follows:

   a. In the **General Setting**, tab rename the method to **processOrder**.

   b. Remove the **XmlObject payload** parameter type.

   c. Click **Add** and from the XML list select the parameter type **Order** (from b2border.xsd).

   d. In the **Receive Data** tab, create a new variable named **order** and select it as the variable to be assigned to the method.

   e. 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 3-51.

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.java** in the **ebxml.oneway** folder onto the Data Palette. This will create a new instance of the **File.java** control.

2. Drag and drop the `FileControlPropertiesDocument write(XmlObject someData)` method to the target that appears below the **Receive request** node.
3. Double-click on the **write** node.
4. Select Send Data and select Order variable.
5. Create a new directory on your hard drive named `C:\tptutorial\invoice`.
6. Copy the file `C:\myapps\tptutorialapps\ebxml\ebxmlWeb\WebContent\sampledata\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 `<Home Directory>\tptutorial\invoice` directory by using the following configurations:
   - **Field name: InvoiceFile**
   - **File name filter**: invoice.xml
   - **Type of Data**: XmlObject
   - **directory-name**: `/tptutorial/invoice`
9. Drag and drop the **XmlObject read()** method from Invoicefile control to the target that appears below the **write** node.
10. Double-click the **read** node.
11. Click **Receive Data**.

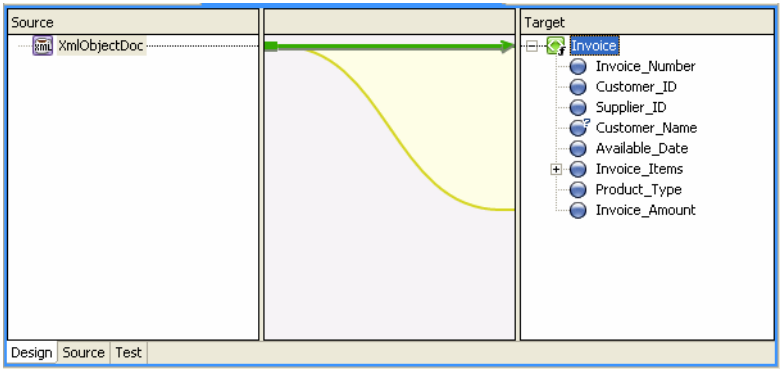
    The method expects untyped XML (XmlObject) but our invoice.xml file contains typed XML so we will need to do a simple transformation between the two.
12. Select the **Transformation** Option.
13. Click **Select Variable > Create New Variable...**.

    The **Create Variable** window opens.
14. Enter **invoice** as variable name.
15. From the XML list, 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** pane to the **Invoice** root in the **Target** pane (see Figure 3-18).

Figure 3-18 Transformation Tool



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

19. Close the Transformation Tool window by clicking on the x in the top of the Transformation Tool window.
20. Click **Yes** in the **Save Resource** 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.

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. Click **Add**, and from the XML list, select the parameter **Invoice** from the **b2binvoice.xsd** schema.
5. Click **Ok**.

6. Click **Send Data**.
7. Select **invoice** from the **Select variables to assign** drop-down list.
8. Click **Close**.
9. Select **File > Save** or enter **Ctrl+S** to save.

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. On the Package Explorer pane, select and right-click the **RoundtripSeller.java** business process.
2. Click **Run As**, and **Run On Server**.
3. The server is started, and the application is deployed on it.
4. After the application is deployed, the Test Browser is displayed.
5. 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 3-18 while completing the following tasks:

- "To Create the Business Process and Configure the Client Request Node" on page 3-47
- "To Create and Configure the ebXML Control" on page 3-47
- "To Create and Configure the ebXML Control Nodes" on page 3-48
- "To Create and Configure the Client Response Node" on page 3-49

To Create the Business Process and Configure the Client Request Node

1. In the **ebxml.roundtrip** folder, create a new business process named **RoundtripBuyer**.
2. Double-click the *Start Event*.
3. Select **Invoked via a Client Request**.
4. Click **Close**.
5. Double-click the **Client Request** node and configure as follows:
   a. Change the method name to **startBuyer**.
   b. Click **Add** and from the XML list select the parameter type **Order** (from b2border.xsd).
   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 **Close**.

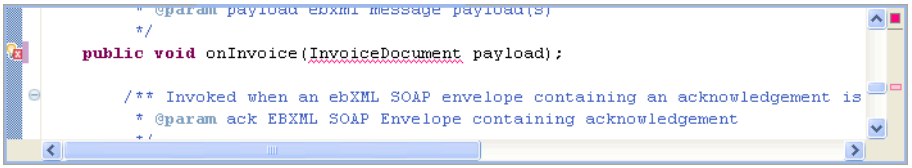
To Create and Configure the ebXML Control

1. Add a new ebXML control and configure it as follows:
   – **Field Name**: roundtripSellerControl
   – **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 **roundtripSellerControl** (ebXML control) and select **Edit**.
3. In the Source view, replace the following line `public void request(XmlObject payload);` with `public void processOrder(OrderDocument payload);`.
4. Double-click on the warning next to the `public void processOrder(OrderDocument payload);` and select `Import'OrderDocument'(com.bea.tutorial.order)`.

   The warning sign disappears.

5. In the Source view, replace the following line `public void response(XmlObject payload);` with `public void onInvoice(InvoiceDocument payload);`.
6. Double-click on the warning next to the `public void onInvoice(InvoiceDocument payload);`.

Figure 3-19



7. Select `Import'InvoiceDocument'(com.bea.tutorial.invoice)`.
8. Save your work by selecting **File > Save** or enter **Ctrl+S**
9. Close the ebXML Source view by clicking the x in the top right corner of the window.

You have now completed the roundtripSeller (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 roundtripSeller (ebXML) control.
2. Drag and drop the `void processOrder(OrderDocument payload)` method to the target that appears below the **Client Request** node.
3. Drag and drop the **order: OrderDocument** variable under the XML list on the Data Palette onto the **processOrder** node.
4. Drag and drop the `void onInvoice (InvoiceDocument payload)` 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**.
9. Click **OK**.
10. Click **Close**.

11. Create a new File control named BuyerFile and configure it to write out invoice.xml to the <Home Directory>\tptutorial\buyer directory by using the following configurations:
    - Field name: File
    - Name: BuyerFile
    - Directory-name: /tptutorial/buyer

    Note: Create a new directory on your hard drive named C:\tptutorial\buyer.

    - File name filter: invoice.xml
    - Type of Data: XmlObject
12. In the Data palette, under Controls expand the File control and select the `FileControlPropertiesDocument write(XmlObject someData)` method.
13. Drag the selected method into Design view and drop it below the OnInvoice node.
14. Double-click the new write node.

    The write node builder is invoked.
15. Click Send Data and select invoice variable.
16. Click Close
17. Right-click on the onInvoice node and select **Add Timeout Path**.
18. Select the OnTimeout Path and in the JPD Configuration pane configure the following properties:
    - duration: 10s
    - retry: 5

You have completed adding the 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 **Node 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. Click **Add** and select the parameter **Invoice** (under b2binvoice.xsd) from the XML list.

5. Click **Send Data**.
6. Select the **invoice** variable from the **Select variables to assign** drop-down list.
7. 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 **Annotations** pane, locate the **ebxml** attribute section.

   In the previous examples, we specified the ebXML attributes in the control definition (the Java 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.

3. In the **ServiceName** filed, enter **RoundtripSeller.**

   Note: This value has to be specified again although it exists in the controls definition 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, enter **declare namespace ns0="bea.com/tutorial/order"data($order/ns0:Supplier_ID)**.

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

To Test the Business Processes

1. On the Package Explorer pane, select and right-click the **RoundtripBuyer.java** business process.
2. Click **Run As**, and **Run On Server**.

   The server is started, and the application is deployed on it.

3. After the application is deployed, the Test Browser is displayed.
4. Select Test Form

5. In the Test Form page, click **Browse** and enter `<Home Directory>\myapps\tptutorialapps\ebXML\ebXMLWeb\sampledata\sampleorder.xml` into the xml variable (payload) field.
6. Click **startBuyer**.
7. Confirm that the Supplier_ID field in the test data has the correct trading partner ID (000000002).
8. Click the Refresh link to see the callback method.
9. 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.
10. Confirm that the `invoice.xml` file was written out correctly to the `C:\tptutorial\buyer` 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

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

1. In the Package Explorer pane, expand **ebxmlWeb** and select **src**.
2. Right-click **src**, and select **New > Package**.

   The **Java Package** dialog box is displayed.
3. Enter **ebxml.publicprivate** in the Name field and click **Finish**.

   The **ebxml.publicprivate** package appears under **ebxmlWeb\src**.

4. Click on the **RoundtripBuyer.java** in the **ebxml.roundtrip** folder and drag it to the **publicprivate** folder while holding down your **Ctrl** key.
5. Release your mouse button.

   A copy of the **RoundtripBuyer.java** 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.
6. Right-click on the new business process and select **Refactor > 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.**
7. Enter **PublicBuyer** and click **OK**.
8. Right-click on **PublicBuyer** and select **Generate > Process Control**.
9. Click **Ok**.

   A new control definition file (PublicBuyerPControl.java) is created in your publicprivate folder. This control can be used in other processes to invoke the PublicBuyer.java business process. You use this control in the PrivateBuyer business process to invoke the PublicBuyer process.
10. Right-click on the onInvoice node and select **Add Timeout Path**.
11. Select the OnTimeout Path and in the JPD Configuration pane, configure the following properties:
    - duration: 10s
    - retry: 5
12. Create a new File control named BuyerFile and configure it to write out invoice.xml to the <Home Directory>\tptutorial\publicprivate directory by using the following configurations:

    Note: Create a new directory on your hard drive named C:\tptutorial\publicprivate.
    - Field name: File
    - Name: PublicFile
    - Directory-name: /tptutorial/publicprivate
    - File name filter: publicinvoice.xml
    - Type of Data: XmlObject

13. In the Data palette, under Controls expand the File control and select the `FileControlPropertiesDocument write(XmlObject someData)` method.
14. Drag the selected method into Design view and drop it below the OnInvoice node.
15. Double-click the new write node.
16. The write node builder is invoked.
17. Click Send Data and select invoice variable.
18. Click Close.

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 3-18 while completing the following tasks:

1. Create a new business process in the **ebxml.publicprivate** folder named **PrivateBuyer**.
2. Double-click **Start Event** and select **Invoked via 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 under XML list.
   - **Receive data tab**: create a new XML variable named **order**.
4. Add a **Client Response** node from the Node Palette.
5. Configure the **Client Response** node according to the following:
   - **Method name**: onPrivateBuyerComplete
   - **Expected parameter type**: Invoice from the b2binvoice.xsd schema under XML list.
   - **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 **PublicBuyerPControl.java** from the Package Explorer pane onto the Data Palette.

   An instance of the **publicBuyerPControl** 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 from the Data Palette under the XML list onto the **startBuyer** node.
10. Drag and drop the **onBuyerComplete** method from the Controls list in the Data Palette onto the target that appears after the **startBuyer** node.
11. Drag and drop the **invoice** variable from the Data Palette under the XML list onto the **onBuyerComplete** node.
12. Click **Save**.
13. Right-click on the onBuyerComplete node and select Add Timeout Path.
14. Select the OnBuyerComplete Path and in the JPD Configuration pane, configure the following properties:
    - duration: 10s
    - retry: 5

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 3-9.

1. In the **ebxml.publicprivate** folder, create a new business process named **PrivateSeller**.
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 from the Node Palette 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. 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.java** from the **ebxml.oneway** folder in the Package Explorer pane onto the Data Palette.
7. Drag and drop the **FileControlPropertiesDocument write** method onto the target that appears below the **Client Request** node.
8. Double-click the write node.
9. Click **Send Data** and select the `order` variable from Select variable to assign.
10. Drag and drop **InvoiceFile.java** from the **ebxml.roundtrip** folder in the Package Explorer pane to the Data Palette.
11. Drag and drop the **read** method to the target that appears below the **write** node.
12. Double-click the **XMLObject read()** node.
13. Double-click the read node.
14. 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.
15. Select the **Transformation** Option.
16. Click **Select Variable** and select **invoice** from the drop-down menu.
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 **PrivateSellerTransformation.java** file, which was created in your Package Explorer 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.
22. Save your work.
23. Right-click on **PrivateSeller.java** 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 3-9 to complete the tasks in this section.

1. Drop and drag the **RoundtripSeller.java** file from the **ebxml.roundtrip** folder to the **ebxml.publicprivate** folder while holding down your ctrl key.
2. Right-click on the new process file and select **Refactor > Rename**.
3. Enter **PublicSeller**.
4. Double-click on **PublicSeller**.
5. Click the Source view tab.
6. In the Annotations pane, 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 **PrivateSellerPControl.java** file from the Package Explorer pane onto Data Palette.
12. Expand the **privateSellerPControl** 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 from the XML list in the Data Palette 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 from the XML list in the Data Palette onto the **onPrivateInvoice** node.
17. Right-click on the **onPrivateInvoice** node and select Add Timeout Path.
18. Select the OnTimeout Path and in the JPD Configuration pane, configure the following properties:
    - duration: 10s
    - retry: 5

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.java**.
2. In the **Data Palette**, select the **roundtripSellerControl**.
3. In the Annotaion pane, change the **ebxml-service-name** to **PublicSeller** to match the participant side.

Testing the Public/Private Pattern Example

1. Open **PublicSeller.java**.
2. Click **Run As > Run On Server**.

3. Switch back to WorkSpace Studio and open **PrivateBuyer.java**.
4. Click **Run As** > **Run On Server**.

   The server is started, and the application is deployed on it.
5. After the application is deployed, the Test Browser is displayed.
6. Click **Test Form**.
7. To enter test data, do one of the following:
   - Click **Browse**, and navigate to `C:\myapps\tptutorialapps\ebXML\ebXMLWeb\sampledata\sampleorder.xml`.
   - Cut and paste the content of `C:\myapps\tptutorialapps\ebXML\ebXMLWeb\sampledata\sampleorder.xml` into the xml variable (payload) field.
8. Refresh the browser after a few seconds. The log entry for `callback.onPrivateBuyerComplete` will have the content of the response with the Invoice document.
9. 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.
10. Confirm that the `invoice.xml` file was written out correctly to the `C:\tptutorial\publicprivate` 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

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

To Create the BuyerAlert Business Process

1. Create a new package named **tpm** under the root ebxmlWeb/src of your project.
2. Add a new process file named **BuyerAlert**.
3. Configure the business process to be invoked by a **Client Request** node.
4. Configure the **Client Request** node as follows:
   - In **General Settings** tab: add 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.
   - Click **Close**.
5. Add an ebXML control and configure it as follows:
   - **Field Name**: sellerControl
   - **Name**: SellerControl
   - **ebxml-service-name**: 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 JPD Configuration pane, 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 **Integration Controls > TPM** in the Controls section of the **Data Palette**.

2. Enter **tpm** as the variable name.

3. Click **Finish**.

   For more information about the TPM control, see TPM Control.

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 **Close**.

To Create and Add an Email Control to the Business Process

1. Select **Email** from the Integrations Controls drop-down menu in the Data Palette.

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

- **Field Name**: alert
- **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.

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' filed) 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.
2. After the Server is running, open the WebLogic Administration Console by selecting **Run > WebLogic Integration > WebLogic Integration 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 1-2 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. Select **BuyerAlert.java**.
2. Click **Run As > Run On Server.**
3. Test the **BuyerAlert** business process.
4. Specify any XML data as the payload.
5. Enter **000000002** for the string parameter.
6. 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 3-63
- "Reviewing the Participant Side of the Example" on page 3-67

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

1. Create the following two directories on your hard drive:
   - `C:\tptutorial\binary-out\dynamic\000000002`
   - `C:\tptutorial\binary-in\dynamic\000000001`
2. Create a new package under **ebxmlWeb/src**, and name it as **ebxml.oneway.binary.advance**.
3. Right-click on **ebxml.oneway.binary.advanced,** and select Import from the drop-down menu.

   The Import dialog box is displayed.
4. In the Import dialog box, select File System and click **Next**.
5. In the File System window, click **Browse** next to From directory: and navigate to the **advanced** folder in the `[unzip location]\tptutorialapps\ebxml\ebxmlWeb\src\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.
6. Click **Select All**.
7. Click **Finish**.
8. In the Package Explorer pane, expand the ebxml.oneway.binary.advanced folder.
9. Double-click on **DynamicBinaryBuyer.java**

   DynamicBinaryBuyer.java 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 Code in Perform Nodes.
- 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 Package Explorer pane, double-click on the `\Schemas\src\tptutorial.channel` file.

   The channel file opens in Design view.

3. Go to the Source view and 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.
2. After the Server is running, open the WebLogic Administration Console by selecting **Run > WebLogic Integration > WebLogic Integration Administration Console** from the WorkSpace Studio menu.
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 1-2 section.
4. Click **Event Generators** on the left pane.
5. Click **File > Create New** on the left pane.

   The **Create a New File Event Generator** page opens.

6. Enter **TPTutorial2** as the **Generator Name**.

7. Click **Submit**.

   The **File Event Generator Definition** page opens

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

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

9. 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.

10. Click **Submit.**

11. 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 **DynamicBinaryBuyer.java** to build and deploy.
2. Place a binary file in the `C:\tptutorial\binary-out\dynanmic\000000002` 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

1. Double-click on **DynamicBinarySeller.java**

   The **DynamicBinarySeller.java** 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 JPD Configuration pane 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 tab.

7. In the Data Palette, select the **baseDirectory** variable. In the Annotations pane , 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:

   – **setProperties**—invokes the setProperties 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 Package Explorer pane.

   – **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. Select **DynamicBinarySeller.java** to run and deploy the process.

2. Drop a file into the `C:\tptutorial\binary-out\dynamic\000000002` directory and watch it appear in a few moments in the `C:\tptutorial\binary-in\dynamic\000000001` directory.

Step 9: Creating a Distributed Setup

In all of the examples so far, the initiator (000000001) and the participant (000000002) are both located in the same WebLogic Integration 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 000000001 operates in one WebLogic Integration instance and the 000000002 in another.

1. Install and configure a second WebLogic instance on another machine.

   a. Complete the procedure described in "Step 1: Create a New WebLogic Integration Domain" on page 1-2.

2. In the **WebLogic Integration Administration Console**, click **Trading Partner Management** on the left pane.

3. For the first WebLogic Integration Server, modify the Test_Trading Partner_2 (000000002) trading partner information as follows:

   a. In the **View and Edit Trading Partner Profiles**, change Test_TradingPartner_2(000000002) to **remote** instead of **local**, to do this perform the following:

   – Click **Test_TradingPartner_2**.

   – Under General Information, click **Edit Profile**.

   – In Edit Trading Partner Profile page, in the Type option, select **Remote** from the drop-down menu option.

   – Click **Submit**.

   b. Modify the ebXML 2.0 binding by changing the **EndPoint URL** as follows:

   – Go to **Bindings** on the left pane and select **Choose trading partner**.

   – In the Choose Trading Partner page, select Test_TradingPartner_2 from the drop-down option and, then click **Go**.

   c. In the Edit Binding page, select **TP2-ebxmlL20-binding**.

   d. The View Binding Details page appears, click **Edit Binding**

e. In the Edit Binding page, enter the correct server/port of the second WebLogic Integration server in the**EndPoint URL**.

Note: The examples use ebXML 2.0 binding by default, modify this to other bindings if you use them.

4. For the second WebLogic Integration server modify the Test_TradingPartner_1 (000000001) 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.

5. On each of the servers, start WorkSpace Studio and open the tutorial application.

6. 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, 000000001 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 3-9. It contains the following sections:

Configuring the Participant Side

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

1. If not already running, start your WebLogic Server.
2. After the Server is running, open the WebLogic Administration Console by selecting **Run > WebLogic Integration > WebLogic Integration Administration Console** from the WorkSpace Studio.
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 1-2 section.
4. Click **Trading Partner Management** on the left pane.
5. In the **Services** on the left pane, click **View All**.

   A list of all participant processes which are currently deployed on your server is displayed.
6. Click on `ebxmlWeb\ebxml\oneway\Seller.java`

   The **View And Edit Service** window opens.
7. Click **Add Service Profile**.

   The **Add Service Profile** window opens.
8. In the **Name** row, select **Test_TradingPartner_2** as the **LOCAL** trading partner. This is the partner actually hosting the Seller service.
9. In the **Name** row, select **Test_TradingPartner_1** as the **REMOTE** trading partner. This is the partner who will use the service.
10. Select **ebXML 2.0** bindings for both the **LOCAL** and **REMOTE** binding.
11. 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. In the **Services** pane, click **Create New**.
2. Next to the name filed, click **Browse**.

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

3. Click on **ebxml.oneway.SellerControl**.
4. Click **Add Service**.
5. Click **Add Service Profile**.
6. In the **Name** row, select **Test_TradingPartner_1** as the **LOCAL** trading partner. This is the initiator partner on the local system.
7. In the **Name** row, select **Test_TradingPartner_2** as the **REMOTE** trading partner. This is the remote partner who hosts the actual service.
8. Select **ebXML 2.0** bindings for both the **LOCAL** and **REMOTE** binding.
9. 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.