BEA WebLogic Integration™

Learning to Use BEA WebLogic Integration

Version 2.1
Document Date: October 2001

Learning to Use BEA WebLogic Integration

Part Number	Date	Software Version
N/A	October 2001	2.1

Contents

About This Document

This document describes a sample integrated application. The sample application deploys a supply-chain hub, which connects business partners, automates a number of business processes, and integrates back-end enterprise information systems. Readers learn how to set up and run the sample application, and understand how the integrated solution is architected and developed using WebLogic Integration.

This document is one in a series of four documents that provide an overview of WebLogic Integration, and that explain how the functionality provided by WebLogic Integration is used at various stages in the design, development, and deployment of integrated solutions. Readers should start with these documents to gain a comprehensive understanding of the functionality provided by WebLogic Integration. The other documents in the series are:

- *Introducing BEA WebLogic Integration*—Provides an overview of WebLogic Integration. It outlines the integration problems today's e-businesses face, with their collections of fragmented, heterogeneous business systems. It also describes the application integration, B2B integration, business process management, and data integration functionality WebLogic Integration provides to solve e-business integration problems.
- *Designing BEA WebLogic Integration Solutions*—Describes how to design and architect WebLogic Integration solutions. Readers learn about good design principles following recommended best practices.
- *Deploying BEA WebLogic Integration Solutions*—Explains how to move an integrated solution from a development to a production environment. Readers learn about configuring, scaling, porting, and performance tuning integrated applications.

Once you are familiar with the contents of these overview documents, you can proceed to the detailed documentation about the functionality provided by WebLogic Integration.

This document is organized as follows:

- Chapter 1, “Introduction,” describes the business problem and an overview of how the WebLogic Integration sample application addresses it.
- Chapter 2, “Setting Up and Running the Sample,” describes how to set up and run the sample implementation.
- Chapter 3, “Understanding the Sample,” explains how the sample works—that is, how the WebLogic Integration components work together to solve an enterprise integration problem.This chapter is a tutorial that describes how the sample defines and manages business partners, automates a number of business processes, and integrates back-end enterprise information systems.
- Appendix A, “DTDs,” defines the Document Type Definitions (DTDs) used in the sample implementation.

Who Should Read This Document

This document is intended for the following users:

- Business Analysts—Define the goals of the integrated solution, and work with technical analysts to ensure that those goals are implemented successfully. Business analysts have expertise in business and economics, and have intimate knowledge about they way a company runs its business. Business analysts want to improve the existing business processes by deploying technologies that leverage the company’s information technology assets to provide competitive advantages.
- Technical Analysts—Responsible for evaluating, deploying, and administering a WebLogic Integration solution that includes the application server and application integration, business process management, B2B integration, and data integration functionality. Technical analysts typically have end-to-end knowledge of the entire system, and work with business analysts to ensure that the goals of the integrated solution are met.
- Software Engineers—Responsible for implementing a WebLogic Integration solution. Software Engineers have detailed knowledge of the underlying code that drives the integrated solution, they ensure that all the software components

are linked together so the solution runs error free, and they address software application issues such as scaling, porting, and performance tuning.

This document is intended mainly for application developers who are interested in building solutions to address enterprise integration challenges. It assumes a familiarity with the WebLogic Integration platform and Java programming.

e-docs Web Site

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

How to Print the Document

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

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

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

Related Information

The following BEA WebLogic Integration documents contain information that supplements the information in this document and is relevant to understanding how to develop and deploy WebLogic Integration applications:

- *Introducing BEA WebLogic Integration*
- *Deploying BEA WebLogic Integration Solutions*
- *Designing BEA WebLogic Integration Solutions*

See the entire suite of WebLogic Integration documents at `http://edocs.bea.com/`.

Contact Us!

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

In your e-mail message, please indicate that you are using the documentation for the BEA WebLogic Integration 2.1 release.

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

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

- Your name, e-mail address, phone number, and fax number
- Your company name and company address
- Your machine type and authorization codes

- The name and version of the product you are using
- A description of the problem and the content of pertinent error messages

Documentation Conventions

The following documentation conventions are used throughout this document.

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

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

`monospace boldface text`	Identifies significant words in code. *Example*: `void` **`commit`** `( )`
`monospace italic text`	Identifies variables in code. *Example*: `String` *`expr`*

Convention	Item
UPPERCASE TEXT	Indicates device names, environment variables, and logical operators. *Examples*: LPT1 SIGNON OR
`{ }`	Indicates a set of choices in a syntax line. The braces themselves should never be typed.

`[ ]`	Indicates optional items in a syntax line. The brackets themselves should never be typed.
	Example:
	`buildobjclient [-v] [-o name ] [-f` *`file-list`*`]...`
	`[-l` *`file-list`*`]...`

`|`	Separates mutually exclusive choices in a syntax line. The symbol itself should never be typed.

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

. . .	Indicates the omission of items from a code example or from a syntax line. The vertical ellipsis itself should never be typed.

CHAPTER

1 Introduction

This section describes the example business problem that serves as the background for the WebLogic Integration sample. It includes the following topics:

- Scope of the Sample
- Background to the Scenario
- Deploying an Integrated Solution

Scope of the Sample

BEA WebLogic Integration™ is a single platform that delivers application server, application integration, business process modeling, and business-to-business (B2B) integration functionality for the enterprise.

The WebLogic Integration sample demonstrates how to develop a new application, integrate it with existing systems, streamline complex business processes, and connect business partners using the WebLogic Integration platform. The sample code documented here is available in the `/samples/wlis` directory of your WebLogic Integration installation.

The sample implementation addresses only part of the complete supply chain automation and integration challenge, but it demonstrates how you can use the flexible WebLogic Integration platform to develop solutions according to your requirements, thereby improving efficiency within your company and across the value chain.

Background to the Scenario

General Control Systems (GCS) is an Illinois-based division of a large enterprise that makes a variety of control systems for factories and office buildings.

EnergyMiser 76 History

GCS has produced the EnergyMiser 76, a monitoring and control system for reducing electrical consumption, since the oil crisis of the 1970s. The company supplies these energy control systems for three other OEM companies. GCS procures the metal boxes used for the EnergyMiser 76 from Midwest Metals, a medium-size firm in Chicago. Midwest Metals has been the exclusive supplier of the silk-screened boxes for 25 years.

GCS's value-added resellers (VAR) all use EDI systems to automate their supply chains. GCS has been using EDI in both its supply and demand chains for 15 years, to process orders from VARs and to automate procurement from Midwest Metals. Inventories of the silk-screened boxes are controlled by an automated replenishment system that links the EDI systems of GCS and Midwest Metals. The supply-chain mechanism has been working smoothly because demand for the EnergyMiser 76 model has been steady, making it easy for GCS to plan its inventory needs.

The corporate parent of GCS recently started an initiative to use BEA WebLogic Integration as its business-to-business (B2B) infrastructure. Until now, GCS has been unable to justify a move to this system because it has a working system in place, and neither suppliers nor customers have requested XML-based B2B transactions.

Increase in Demand for Product

Suddenly, in Q3, GCS is overwhelmed by an unprecedented glut of orders for the EnergyMiser 76 from its VARs. The automated replenishment system goes into operation and an automated EDI 850 message (purchase order) is sent to Midwest Metals for a larger than usual order of silk-screened boxes. The president of Midwest

Metals calls the GCS procurement department to let them know that his company is unable to fill the entire order in the specified time because his plant does not have enough capacity.

The Midwest Metals president further explains that he cannot increase the capacity of his plant unless he has a long-term, open purchase order for the increased capacity. The GCS manager explains that the spike in demand is due to unusually large orders from customers in California. However, because of the uncertainties of the Californian energy situation, it is unclear whether the current orders indicate a long-term increase in demand or a temporary spike.

Understanding the situation, the procurement manager buys the maximum allotment of supplies from Midwest Metals and begins to look for alternative sources. Remembering the new corporate initiative to improve the B2B integration infrastructure, he contacts the IT Department to discuss short-term plans to resolve the current crisis and long-term plans to gain more control and visibility into the supply chain to avoid future shortages or other surprises. He realizes that this situation is an indication of how the GCS legacy system is no longer sufficient to meet today's automated and dynamic marketplace.

Deploying an Integrated Solution

GCS is the channel master in this value chain. To improve efficiency in its marketplace and drive competitive advantage, GCS deploys a supply-chain hub using BEA WebLogic Integration.

The sample application is a supply-chain hub that connects business partners, automates a number of business processes, and integrates back-end enterprise information systems. This sample application does not address the GCS demand chain or GCS integration with its VARs. See "The End-to-End EDI Sample" in *Using EDI with WebLogic Integration* for a sample EDI integration application.

Short-Term and Long-Term Advantages

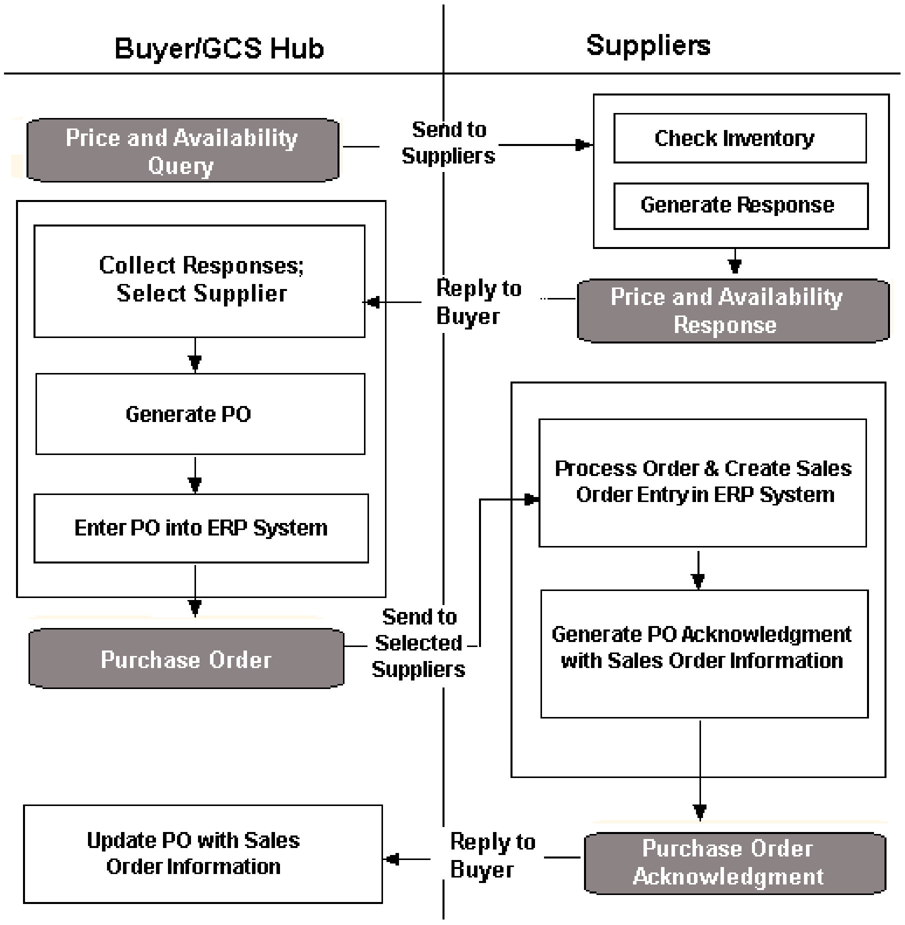
Deployment of the hub has both short-term and long-term advantages. It resolves the current crisis by facilitating procurement of supplies from partners other than Midwest Metals in an approved vendor list (AVL), and it establishes a system that streamlines the supply chain for future business.

Deploying a supply-chain hub allows GCS to automate several processes. In this sample application, GCS engages in a conversation with selected business partners. The following sequence of events summarizes the transaction among the business partners:

1. A price and availability query is issued by GCS to approved secondary suppliers.
2. GCS collects the responses from the suppliers and presents them to the ordering manager for selection of a supplier.
3. A purchase order is generated for the selected supplier.
4. The purchase order is automatically entered into the company’s ERP system.
5. The selected supplier returns a purchase order acknowledgment to GCS.
6. The purchase order record is automatically updated with information from the purchase order acknowledgment (for example, the supplier’s sales order number).

The following figure illustrates the process flow for the price and availability query exchanged by the GCS hub and its suppliers.

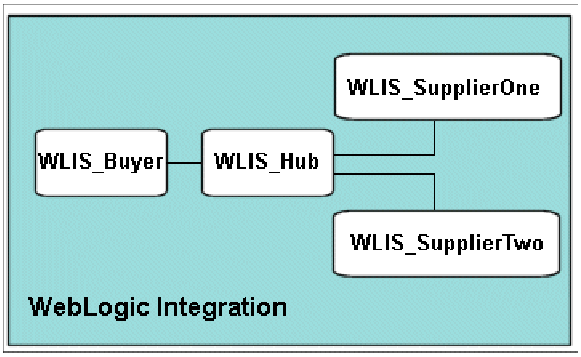
Figure 1-1 Process Flow of the Sample Application


Buyer/GCS Hub
Suppliers
Price and Availability Query
Send to Suppliers
Check Inventory
Generate Response
Collect Responses; Select Supplier
Reply to Buyer
Price and Availability Response
Generate PO
Enter PO into ERP System
Process Order & Create Sales Order Entry in ERP System
Purchase Order
Send to Selected Suppliers
Generate PO Acknowledgment with Sales Order Information
Update PO with Sales Order Information
Reply to Buyer
Purchase Order Acknowledgment

Solution Architecture

The WebLogic Integration sample implements the solution by deploying four business entities: a hub, a buyer, and two supplier trading partners. In a production environment, each entity can run on a separate WebLogic Integration server. However to simplify the procedure for running the sample, this scenario uses a collocated approach in which a single WebLogic Integration server hosts all four entities. The following figure illustrates the implementation.

Figure 1-2 Implementation Architecture for WebLogic Integration Sample



The functions performed by the business entities that make up the sample are described in the following table.

Table 1-1 Functions Performed by Business Entities in Sample

Business Entity	Function
WLIS_Hub	Routes the communication between the buyer and suppliers, providing business-to-business integration
WLIS_Buyer	■ Coordinates business processes among suppliers and internal functions (for example, back-end database updates), using workflow templates ■ Provides connectivity to the buyer's database system, using an application view ■ Handles data presentation and the user interface through HTML and JSP pages

Table 1-1 Functions Performed by Business Entities in Sample

Business Entity	Function
WLIS_SupplierOne	■ Responds to requests from the buyer and invokes internal programs (for example, data transformation and persistence), using workflow templates ■ Performs data translations to facilitate the exchange of information among applications
WLIS_SupplierTwo	■ Responds to requests from the buyer and invokes internal programs (for example, data transformation and persistence), using workflow templates ■ Performs data translations to facilitate the exchange of information among applications

CHAPTER

2 Setting Up and Running the Sample

This section provides instructions for setting up your environment and running the WebLogic Integration sample application. The instructions are provided in the following sections:

- Preparing to Run the Sample
- Running the Sample

For a description of the architecture behind this sample, see "Solution Architecture" on page 1-6. For a summary of the process illustrated by the sample application, see Figure 1-1, "Process Flow of the Sample Application."

Preparing to Run the Sample

Before running the sample, complete the following steps:

1. Install WebLogic Integration with the following option: WebLogic Integration Full Installation with Samples. For installation instructions, see *Installing BEA WebLogic Integration*.

   The WebLogic Integration sample is installed in the `/samples/wlis` directory in your WebLogic Integration installation.

2. Make sure the proxy settings on your browser are set so that you can connect to your local WebLogic Server. For more information, see “Web Browser Configuration Requirements” in “WebLogic Integration Administration and Design Tools” in *Starting, Stopping, and Customizing BEA WebLogic Integration*.

Note: The sample uses the database you specified when you installed WebLogic Integration. If you want to change the database, you can do so by running the WebLogic Integration Database Configuration wizard. See “Using the Database Configuration Wizard” in *Starting, Stopping, and Customizing BEA WebLogic Integration* for instructions on running the Configuration wizard.

Running the Sample

Note: This sample can be run by only one user at a time per instance of WebLogic Integration.

To run the WebLogic Integration sample, complete the following eight-step procedure:

- Step 1. Configure and Invoke the Launcher Web Page
- Step 2: Choose the WebLogic Integration Sample
- Step 3. Start the Sample
- Step 4. Send the QPA Request
- Step 5. Check for QPA Responses
- Step 6. Create the Purchase Order
- Step 7. Check the Purchase Order
- Step 8. Check for Purchase Order Acknowledgment

Step 1. Configure and Invoke the Launcher Web Page

The `\samples\bin` directory in your WebLogic Integration installation contains scripts used to configure and run all the samples for the WebLogic Integration product.

About the Scripts

The `RunSamples` script is a driver script that invokes other scripts in the appropriate order until the launcher Web page is displayed. The launcher Web page contains links to start several WebLogic Integration samples and administration consoles. See "Configuring and Starting the Samples Domain" in "Getting Started" in *Starting, Stopping, and Customizing BEA WebLogic Integration* for details about the `RunSamples` script.

This Step (Configure and Invoke the Launcher Web Page) describes two substeps. These substeps describe alternative procedures to configure and invoke the launcher Web page:

- Step 1A: Invoke the RunSamples Script
- Step 1B: Invoke the Start Server and Launcher Scripts

You must execute the `RunSamples` script, described in Step 1A, the first time you run a sample in the samples domain. The `RunSamples` script both configures the samples database, and starts WebLogic Integration in the samples domain.

If you have executed the `RunSamples` script once, and the database is properly configured, the next time you start the samples domain, you can choose to start a sample by executing the `RunSamples` script again, as described in Step 1A.

Note: When you execute the `RunSamples` command after the samples database has already been properly configured, you will be prompted with the following message:

```
The WebLogic Integration repository has already been created
and populated, possibly from a previous run of this
RunSamples script. Do you want to destroy all the current data
in the repository and create and populate the WebLogic
Integration repository again?
Y for Yes, N for No
```

When you enter N, the command bypasses the database configuration tasks. The command starts WebLogic Integration in the samples domain, launches your default Web browser, and displays the Samples Launcher page.

Alternatively, you can use the Start Server and Launcher scripts, described in Step 1B.

Step 1A: Invoke the RunSamples Script

Note: You must complete the tasks described in this step if you have not invoked the `RunSamples` script before, or if you have changed the database with which you want to run the sample.

The following sections provide instructions for running the `RunSamples` script on a Windows and UNIX system.

Invoking the RunSamples Script on Windows

To run the `RunSamples` script on a Windows system, do one of the following:

- Using menus, invoke the `RunSamples` script as follow:

  Choose Start→Programs→BEA WebLogic E-Business Platform→WebLogic Integration 2.1→Samples→RunSamples.

- From the command line, invoke the `RunSamples` script as follows:

  a. Open a command window.

  b. Go to the `\bin` directory in the samples domain. For example, if you installed the product in the default location, enter the following:

  ```
  cd \bea\wlintegration2.1\samples\bin
  ```

  c. Execute the `RunSamples` script by entering:

  ```
  RunSamples
  ```

After the `RunSamples` script finishes running, a Web browser, which displays the launcher Web page provided with the samples, is displayed.

Invoking the RunSamples Script on UNIX

To run the `RunSamples` script on a UNIX system:

1. Go to the `bin` directory in the samples domain. For example, if you installed the product in the default location, enter the following:

   ```
   cd BEA_Home/wlintegration2.1/samples/bin
   ```

2. Execute the `RunSamples` script by entering:

   ```
   . ./RunSamples
   ```

   After the `RunSamples` script finishes running, a Web browser, which displays the launcher Web page provided with the samples, is displayed.

Step 1B: Invoke the Start Server and Launcher Scripts

If you have previously run the `RunSamples` script, as described in Step 1A, the database has been created and populated with the sample data. In this case, when you want to run a sample, you can complete the tasks in this step (as an alternative to Step 1A). The procedures in Step 1B start WebLogic Server and invoke the launcher Web page on Windows and UNIX systems.

Invoking the Start Server and Launcher Scripts on Windows

To run the scripts on a Windows system, do one of the following:

- Using menus, invoke the Start Server script, and then the Samples Launcher script, as follows:

  a. Choose Start→Programs→BEA WebLogic E-Business Platform→WebLogic Integration 2.1→Samples→Start Server.

     When the following message is displayed, the `Start Server` script has completed successfully:

     ```
     StartServer execution successful
     ```

  b. Choose Start→Programs→BEA WebLogic E-Business Platform→WebLogic Integration 2.1→Samples→Samples Launcher.

     A Web browser, which displays the launcher Web page provided with the samples, is displayed.

- From the command line, invoke the Start Server script, and then the Samples Launcher script, as follows:

  a. Open a command window.

  b. Go to the `bin` directory in the samples domain. For example, if you installed the product in the default location, enter the following:

  ```
  cd \bea\wlintegration2.1\samples\bin
  ```

  c. Start the server by entering:

  ```
  startServer
  ```

  When the following message is displayed, the `startServer` script has completed successfully:

  ```
  StartServer execution successful
  ```

  d. Start the Web browser by entering:

  ```
  launchBrowser
  ```

  A Web browser, which displays the launcher Web page provided with the samples, is displayed.

Invoking the Start Server and Launcher Scripts on UNIX

To run the scripts on a UNIX system:

1. Go to the `bin` directory in the samples domain. For example, if you installed the product in the default location, enter the following:

   ```
   cd BEA_Home/wlintegration2.1/samples/bin
   ```

2. Start the server by entering:

   ```
   . ./startServer
   ```

   When the following message is displayed, the `startServer` script has completed successfully:

   ```
   StartServer execution successful
   ```

3. Start a Web browser using the following URL:

   ```
   http://localhost:7001/index.html
   ```

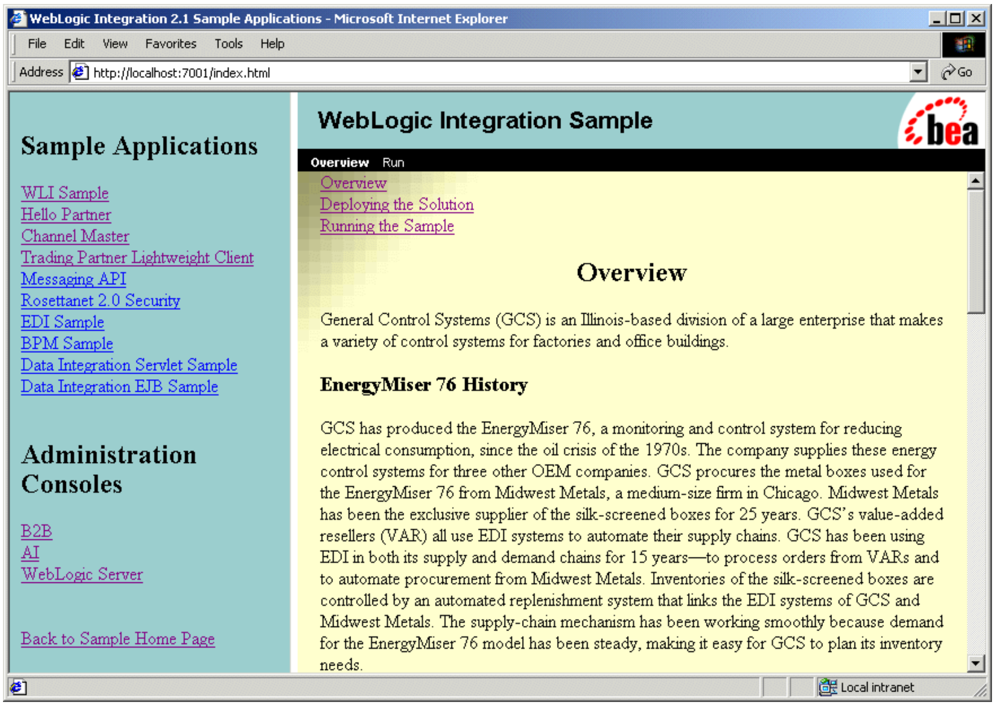
   The launcher Web page provided with the samples, is displayed.

Step 2: Choose the WebLogic Integration Sample

You can start several sample applications and administration consoles from the launcher Web page. Click the `WLI Sample` link under Sample Applications on this Web page to invoke the sample application that addresses the supply chain problem described in Chapter 1, "Introduction."

As shown in the following figure, the sample overview page is displayed in your browser.

Figure 2-1 WebLogic Integration Sample Overview Window


WebLogic Integration 2.1 Sample Applications - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Address http://localhost:7001/index.html
Go
Local intranet

Sample Applications

WLI Sample
Hello Partner
Channel Master
Trading Partner Lightweight Client
Messaging API
Rosettanet 2.0 Security
EDI Sample
BPM Sample
Data Integration Servlet Sample
Data Integration EJB Sample

Administration Consoles

B2B
AI
WebLogic Server

Back to Sample Home Page

WebLogic Integration Sample

Overview Run

Overview
Deploying the Solution
Running the Sample

Overview

General Control Systems (GCS) is an Illinois-based division of a large enterprise that makes a variety of control systems for factories and office buildings.

EnergyMiser 76 History

GCS has produced the EnergyMiser 76, a monitoring and control system for reducing electrical consumption, since the oil crisis of the 1970s. The company supplies these energy control systems for three other OEM companies. GCS procures the metal boxes used for the EnergyMiser 76 from Midwest Metals, a medium-size firm in Chicago. Midwest Metals has been the exclusive supplier of the silk-screened boxes for 25 years. GCS's value-added resellers (VAR) all use EDI systems to automate their supply chains. GCS has been using EDI in both its supply and demand chains for 15 years—to process orders from VARs and to automate procurement from Midwest Metals. Inventories of the silk-screened boxes are controlled by an automated replenishment system that links the EDI systems of GCS and Midwest Metals. The supply-chain mechanism has been working smoothly because demand for the EnergyMiser 76 model has been steady, making it easy for GCS to plan its inventory needs.

The overview window contains a brief description of the business problem and the solution offered by the WebLogic Integration sample.

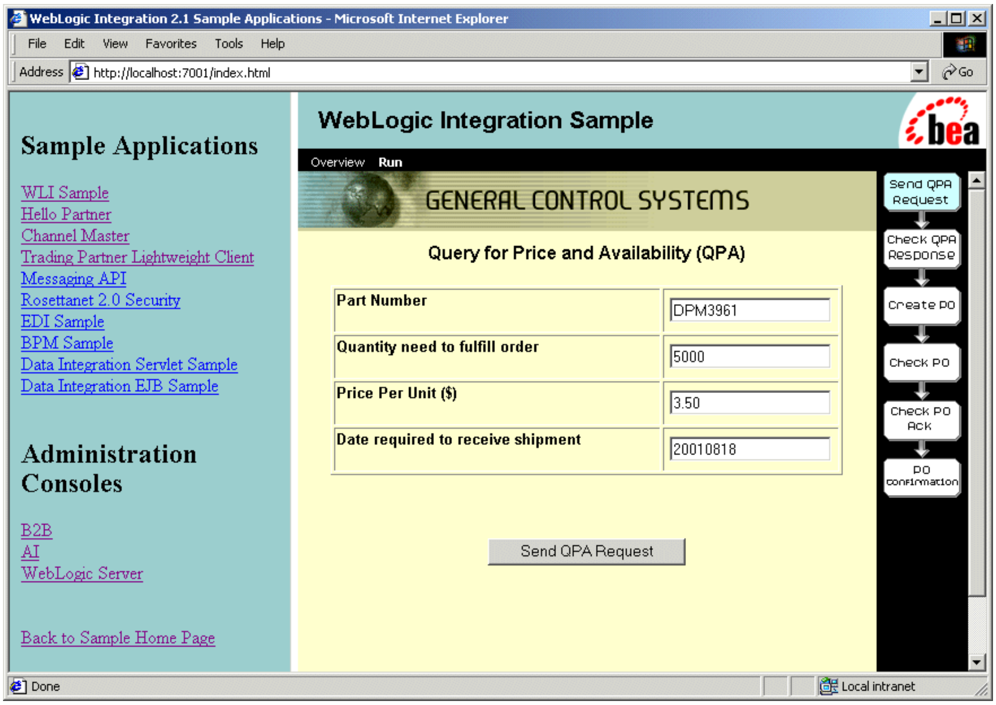
Step 3. Start the Sample

Start the sample by doing either of the following:

- Click Run in the black horizontal bar at the top of the Overview window.
- Click Start the Sample at the bottom of the Overview window.

The Send QPA Request window, shown in the following figure, is displayed.

Figure 2-2 Send QPA Request Window



Query for Price and Availability (QPA)

Part Number	DPM3961
Quantity need to fulfill order	5000
Price Per Unit ($)	3.50
Date required to receive shipment	20010818

Note the following details about this window:

- The window contains the Query for Price and Availability (QPA) form which, in turn, contains the QPA request data to be sent from the buyer (GCS) to two suppliers.

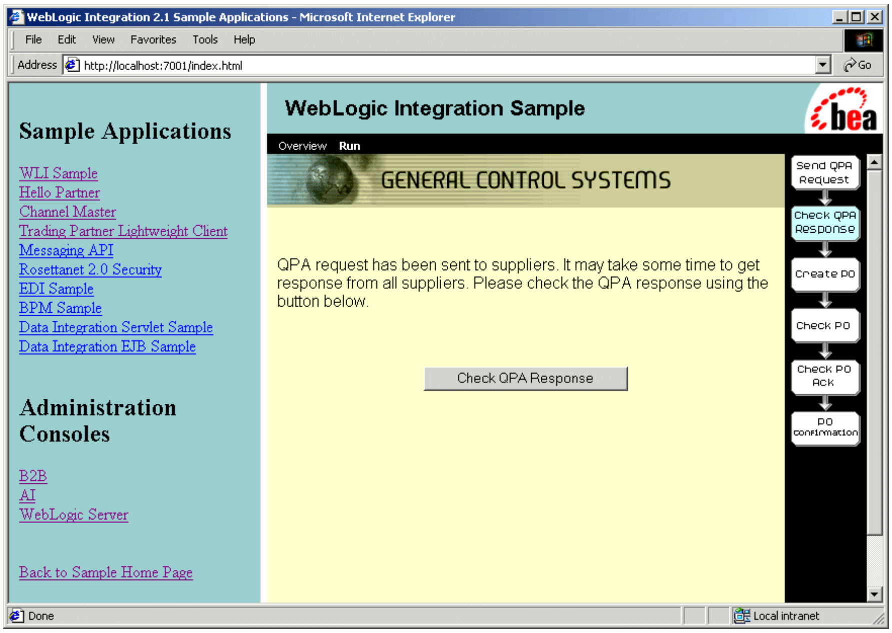
  You cannot modify data in the fields in this form.

- The status bar is displayed on the right side of the window. This bar is a compact flow chart in which each box represents both a step in the procedure you follow, as you progress through the sample application, and a window that corresponds to that step. The appropriate status box is highlighted in each window as you progress through the sample. Use this status bar to track your progress.

Step 4. Send the QPA Request

Click Send QPA Request to trigger the QPA process. The Check QPA Response window, shown in the following figure, is displayed.

Figure 2-3 Check QPA Response Window



QPA request has been sent to suppliers. It may take some time to get response from all suppliers. Please check the QPA response using the button below.

This window indicates that the QPA request has been sent to suppliers and that the buyer is waiting for responses.

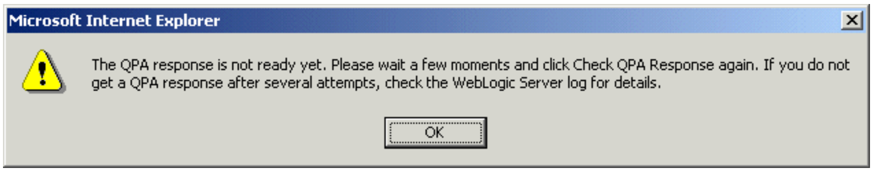
Click Check QPA Response in the Check QPA Response window to go to the next window.

Step 5. Check for QPA Responses

Which window is displayed next depends on whether or not the buyer has received QPA responses from the suppliers:

- If the buyer has not received QPA responses from the suppliers, the following dialog box is displayed.

Figure 2-4 QPA Response Dialog Box



The QPA response is not ready yet. Please wait a few moments and click Check QPA Response again. If you do not get a QPA response after several attempts, check the WebLogic Server log for details.

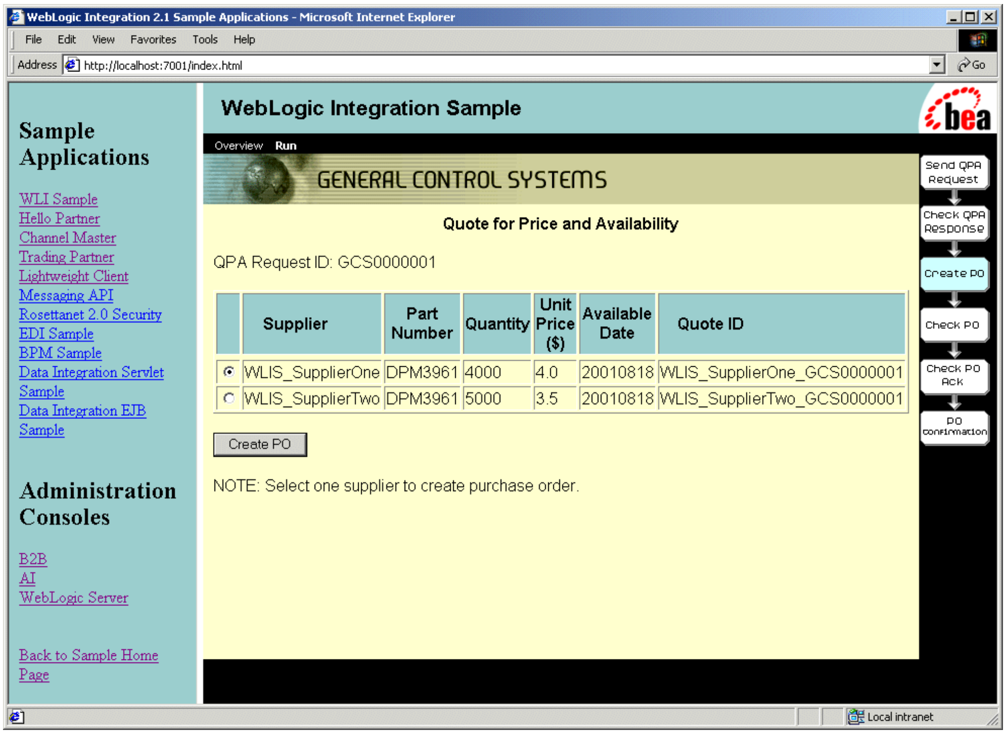
  a. Click OK to close the dialog box. The Check QPA Response window remains displayed in the browser.

  b. Wait a few moments and click Check QPA Response again.

  c. Repeat steps a and b until the QPA Response dialog box is no longer displayed, indicating that the buyer has received responses from the suppliers. When the responses are received, the Create PO window is displayed, as shown in Figure 2-5. If you do not get a response, check the WebLogic Server log for error information.

- If the buyer has received the QPA responses from the suppliers, the Create PO window, shown in the following figure, is displayed.

Figure 2-5 Create PO Window



Quote for Price and Availability

QPA Request ID: GCS0000001

	Supplier	Part Number	Quantity	Unit Price ($)	Available Date	Quote ID
●	WLIS_SupplierOne	DPM3961	4000	4.0	20010818	WLIS_SupplierOne_GCS0000001
○	WLIS_SupplierTwo	DPM3961	5000	3.5	20010818	WLIS_SupplierTwo_GCS0000001

Create PO

NOTE: Select one supplier to create purchase order.

This window displays a form summarizing each supplier's response to the price and availability query.

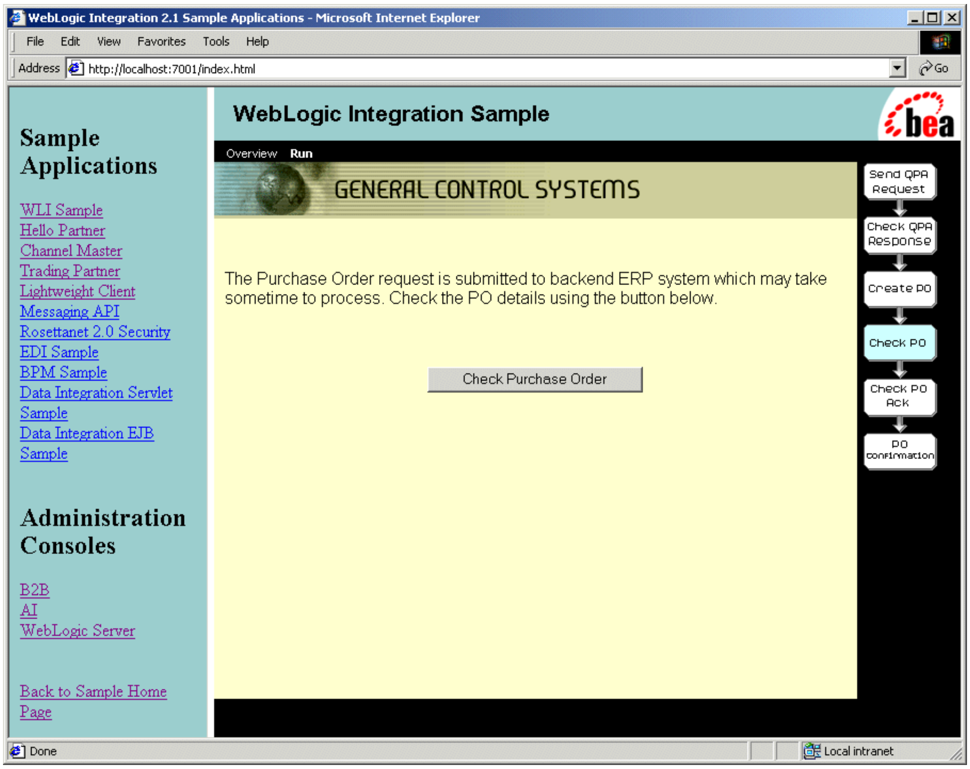
Step 6. Create the Purchase Order

Complete the following procedure to create the purchase order for the silk-screened metal boxes:

1. Select either WLIS_SupplierOne or WLIS_SupplierTwo on the form displayed in Figure 2-5.
2. Click Create PO to submit your request to the back-end ERP system, in which the specified purchase order is generated and sent to the selected supplier.

The Check PO window is displayed. It displays a message indicating that the purchase order request has been submitted, as shown in the following figure.

Figure 2-6 Check PO Window



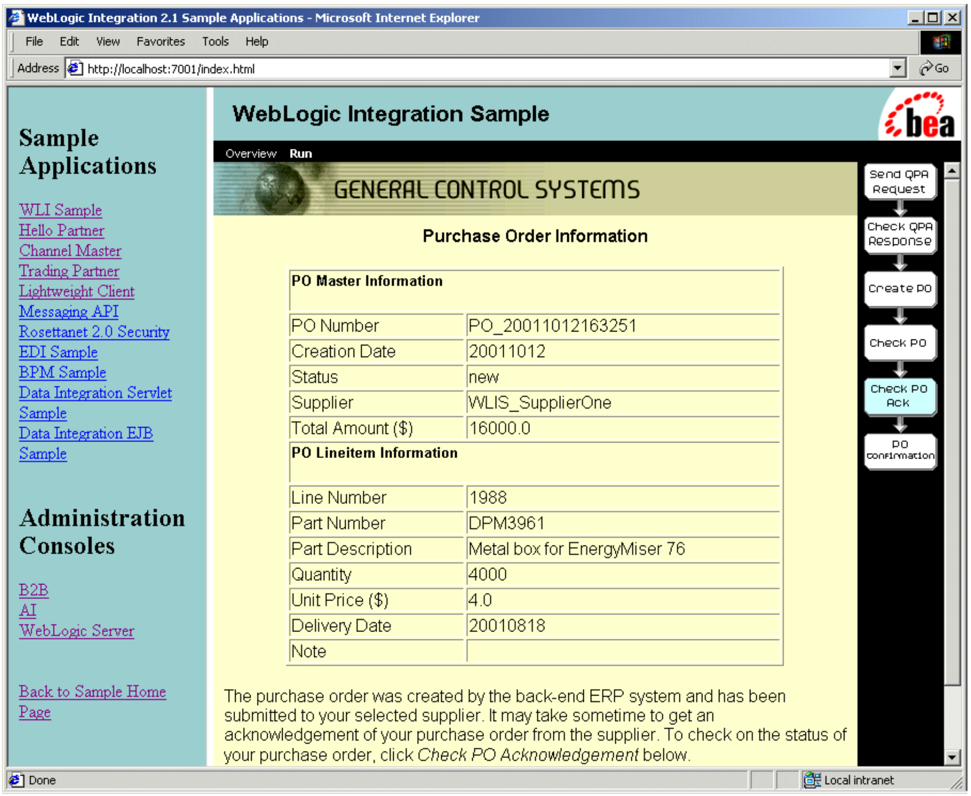
The Purchase Order request is submitted to backend ERP system which may take sometime to process. Check the PO details using the button below.

To check the status of the purchase order you have submitted, click Check Purchase Order.

Step 7. Check the Purchase Order

When the purchase order has been submitted to the selected supplier, the system displays its contents, as shown in the following figure.

Figure 2-7 Check PO Ack Window



Purchase Order Information

PO Master Information	
PO Number	PO_20011012163251
Creation Date	20011012
Status	new
Supplier	WLIS_SupplierOne
Total Amount ($)	16000.0
PO Lineitem Information	
Line Number	1988
Part Number	DPM3961
Part Description	Metal box for EnergyMiser 76
Quantity	4000
Unit Price ($)	4.0
Delivery Date	20010818
Note	

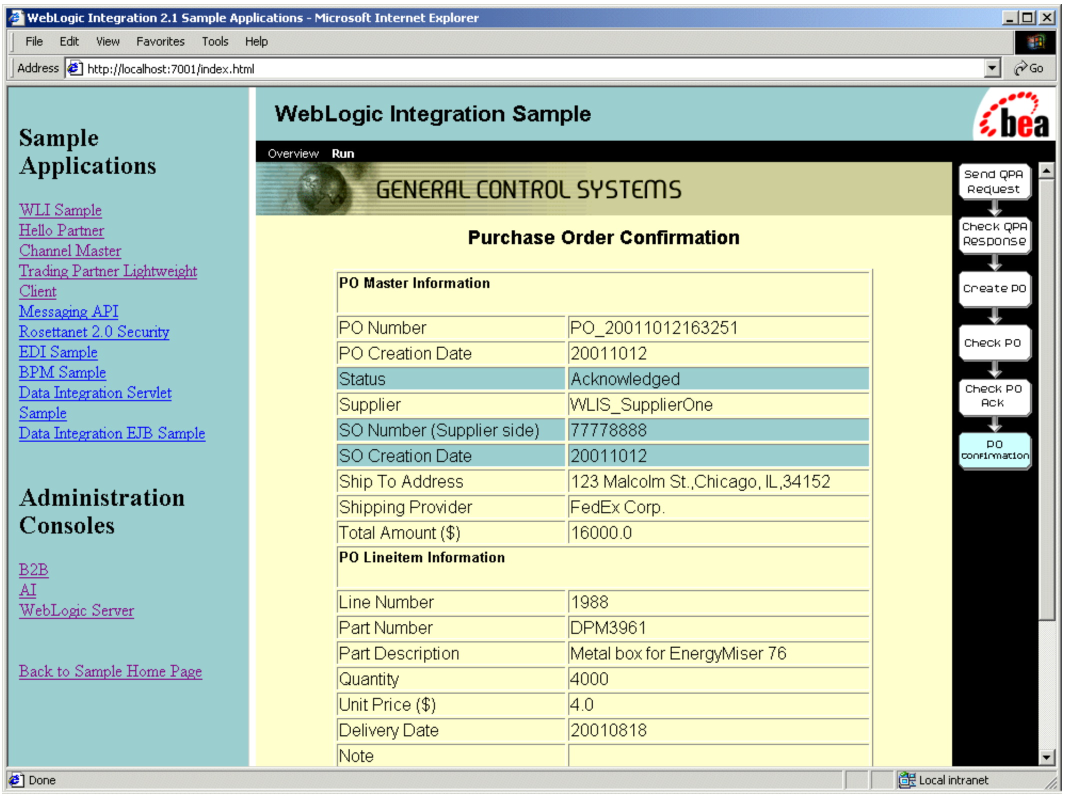
The purchase order was created by the back-end ERP system and has been submitted to your selected supplier. It may take sometime to get an acknowledgement of your purchase order from the supplier. To check on the status of your purchase order, click *Check PO Acknowledgement* below.

The preceding figure displays the PO information that you, as the buyer, submitted to the selected supplier. On receipt of the purchase order, the supplier returns an acknowledgment message.To view this message, click Check PO Acknowledgement.

Step 8. Check for Purchase Order Acknowledgment

When the supplier acknowledges receipt of the purchase order, a PO Confirmation page, as shown in the following figure, is displayed to the buyer.

Figure 2-8 PO Confirmation Window



Purchase Order Confirmation

PO Master Information	
PO Number	PO_20011012163251
PO Creation Date	20011012
Status	Acknowledged
Supplier	WLIS_SupplierOne
SO Number (Supplier side)	77778888
SO Creation Date	20011012
Ship To Address	123 Malcolm St.,Chicago, IL,34152
Shipping Provider	FedEx Corp.
Total Amount ($)	16000.0
PO Lineitem Information	
Line Number	1988
Part Number	DPM3961
Part Description	Metal box for EnergyMiser 76
Quantity	4000
Unit Price ($)	4.0
Delivery Date	20010818
Note	

The preceding figure shows the purchase order confirmation sent from the supplier to the buyer (GCS). Note that the supplier has added shipping information to the original purchase order sent by the buyer and confirmed by the supplier, as shown in the PO Master Information sections of Figure 2-7 and Figure 2-8, respectively.

You have now completed running the WebLogic Integration sample application. See Chapter 3, "Understanding the Sample," to find out how this application works.

CHAPTER

3 Understanding the Sample

Note: We strongly recommend that you run the sample before reading this section. For instructions, see Chapter 2, "Setting Up and Running the Sample." The sample code described in this section is available in the `\samples\wlis` directory of your WebLogic Integration installation.

The sample application automates a number of business processes, integrates back-end enterprise information systems (EIS), and deploys a supply-chain hub that connects business partners. This section describes how this sample integrated solution addresses the supply-chain challenge experienced by our hypothetical enterprise, General Control Systems (GCS). Specifically, it includes the following topics:

- Overview
- B2B Integration
- Business Process and Workflow Modeling
- Application and Data Integration

Overview

In our sample scenario, General Control Systems (GCS) decides to implement a WebLogic Integration solution to its supply-chain challenge. What is the first step?

GCS begins by analyzing how it will use WebLogic Integration to perform the following tasks:

- Model the business processes that drive transactions among participating trading partners
- Define and manage business-to-business (B2B) integration among business partners in the scenario
- Integrate new Web-based front-end systems with existing enterprise information systems
- Handle heterogeneous data formats

The remainder of this section explains how WebLogic Integration enables GCS to perform these tasks.

Model Business Processes

WebLogic Integration provides the tools required to manage business processes:

- WebLogic Integration process engine—Collection of EJBs that provide services to create and maintain workflow templates, start and terminate running workflows, control and manage workflow instances at run time, and so on
- WebLogic Integration Studio—Graphical design tool for creating business processes and monitoring them at run time

Manage B2B Integration

WebLogic Integration supports communication with external trading partners over the Internet. Specifically, it supports the XOCP, RosettaNet (1.1 and 2.0), and Ariba cXML business protocols for sending and receiving XML messages.

A B2B integration BPM plug-in extends the WebLogic Integration Studio so the Studio can be used to define collaborative workflows, that is, workflows that exchange messages with external trading partners.

Integrate New and Existing Systems

WebLogic Integration supports the integration of new Web-based front-end systems with existing enterprise information system (EIS) applications through the use of *adapters*. Application integration adapters can be broadly categorized as follows:

- Service Adapters—Provide synchronous request and response integration from WebLogic Integration into an EIS application.
- Event Adapters—Provide unidirectional asynchronous integration from the EIS application into WebLogic Integration.

Application integration adapters are configured using a Web browser interface. Business events that flow through application integration adapters are defined as XML Schema Definitions (XSDs) in the WebLogic Integration repository.

An application integration BPM plug-in extends the WebLogic Integration Studio so the Studio can be used to define processes that:

- Send events to application integration service adapters
- Consume events generated by application integration adapters

Handle Heterogeneous Data Formats

XML is the message format used in WebLogic Integration systems. The WebLogic Integration framework, however, makes it easy to integrate WebLogic Integration applications with environments in which a binary message format is used.

WebLogic Integration supports the following types of data integration:

- Binary-to-XML/XML-to-binary translation: WebLogic Integration performs bidirectional translation between XML and binary formats. This type of translation alters only the data representation (XML or binary) of a message; the data structure and content of a message are unchanged.
- XML-to-XML transformation: Using XSLT, WebLogic Integration can map the data structure or the content (or both) of an XML message from a source message to a target message.

XSL stylesheets can be created manually or by using the Contivo Analyst mapping tool that is bundled with the WebLogic Integration product. Stylesheets are stored in the WebLogic Integration repository.

Transformations are executed at run time: an input message is assigned to a workflow variable; the process engine references the XSL stylesheet in the repository; and the output message is assigned to another workflow variable.

The following sections describe how the components of WebLogic Integration work together to build and deploy this sample application.

B2B Integration

The definition and management of the relationships between the trading partners that participate in the e-business transactions (in the roles of the buyer and suppliers for this scenario) is fundamental to the development of this integrated solution. The data to define and administer trading partners is stored in the WebLogic Integration repository, and WebLogic Integration provides tools and processes for the effective management of dynamic and diverse trading partner relationships.

An exhaustive discussion of how to configure your B2B integration environment is beyond the scope of this document. However, this section briefly describes the WebLogic Integration repository data that is used in the sample application, how it is loaded for the sample, and how you can view the data and monitor the progress of the sample's business conversations.

For information about B2B integration and instructions for configuring the WebLogic Integration environment to support it, see *Introducing B2B Integration* and *Administering B2B Integration*, respectively.

This section includes the following topics:

- Loading the Repository Data
- Understanding the Repository Data
- Using the WebLogic Integration B2B Console

Loading the Repository Data

The data required by the sample for integrating the trading partners is bulk loaded into the WebLogic Integration repository when you run the `RunSamples` script during the sample setup (see “Running the Sample” on page 2-2).

The `RunSamples` script loads the repository with the B2B configuration data contained in the following XML files:

- `SystemRepData.xml`—located in the `\dbscripts` directory in your WebLogic Integration installation directory, for example:

  `D:\bea\wlintegration2.1\dbscripts`

  The `SystemRepData.xml` file contains system data. The elements used by this sample include:

  - Business protocol definitions
  - Logic plug-ins

- `BulkLoaderData.xml`—located in the `\samples\wlis\lib` directory in your WebLogic Integration installation directory, for example:

  `D:\bea\wlintegration2.1\samples\wlis\lib`

  This `BulkLoaderData.xml` file contains data specific to the WebLogic Integration sample. It describes the following elements:

  - Trading partners
  - Conversation definitions
  - Collaboration agreements

For details about each of these data elements, see the next section, “Understanding the Repository Data.”

Understanding the Repository Data

This section highlights important information about the following data elements that are bulk loaded to the WebLogic Integration repository for the sample application:

- Business Protocol Definitions
- Logic Plug-Ins
- Trading Partners
- Conversation Definitions
- Collaboration Agreements

Note: As described in “Loading the Repository Data” on page 3-5, data from two XML files is imported into the WebLogic Integration repository to support the sample application. You can bulk load configuration data or enter it through the WebLogic Integration B2B Console. You can also access and configure bulk-loaded data using the B2B Console. For more information, see “Using the WebLogic Integration B2B Console” on page 3-14.

Business Protocol Definitions

The `SystemRepData.xml` file contains system data, including definitions for all the business protocols supported by WebLogic Integration: XOCP, RosettaNet, and cXML. The WebLogic Integration sample application uses only XOCP. The following excerpt from the `SystemRepData.xml` file shows the XOCP business protocol definitions.

Listing 3-1 XOCP Business Protocol Definitions in the SystemRepData.xml File

```
<!-- XOCP BUSINESS PROTOCOL DEFINITIONS -->
<business-protocol-definition
      name="XOCP-SPOKE"
      business-protocol-name="XOCP"
      protocol-version="1.1"
      endpoint-type="SPOKE">
      <java-class>com.bea.b2b.protocol.xocp.XOCPSpokeProtocol</java-class>
      <decoder>XOCP-Decoder</decoder>
      <encoder>XOCP-Encoder</encoder>

</business-protocol-definition>
```

```
<business-protocol-definition
      name="XOCP-Hub"
      business-protocol-name="XOCP"
      protocol-version="1.1"
      endpoint-type="HUB">
      <java-class>com.bea.b2b.protocol.xocp.XOCPHubProtocol</java-class>
      <decoder>XOCP-Decoder</decoder>
      <system-router>XOCP-System-Router</system-router>
       :
       :
      <system-router>XOCP-Router-Enqueue</system-router>
      <system-filter>XOCP-System-Filter</system-filter>
       :
      <encoder>XOCP-Encoder</encoder>
       :
       :
</business-protocol-definition>
```

In the preceding listing, note the following:

- WebLogic Integration supports two definitions for the XOCP business protocol: XOCP-HUB and XOCP-SPOKE.
- An endpoint-type is provided for each definition.
- Each business protocol definition is implemented by a specified Java class.

For details about configuring business protocol definitions, see "Configuration Requirements" in *Administering B2B Integration* and "Configuring Trading Partners" in *Online Help for the WebLogic Integration B2B Console*.

Logic Plug-Ins

Logic plug-ins are Java classes that can intercept and process business messages at run time. Each business protocol is associated with standard router and filter logic plug-ins.

The `SystemRepData.xml` file contains system data, including the WebLogic Integration logic plug-ins for XOCP, RosettaNet, and cXML business protocols. This sample uses the XOCP logic plug-ins only.

Table 3-1 XOCP-Specific Logic Plug-Ins

Logic Plug-In	Description
XOCP-Router	This router logic plug-in processes incoming business messages and generates the message-context document that is used in the processing. By default, this is the first logic plug-in in the router chain. The XOCP router logic plug-in for a hub delivery channel can modify the list of trading partner recipients, based on XPath router expressions. For more information, see "Routing and Filtering Business Messages" in *Programming Logic Plug-Ins for B2B Integration*.
XOCP-Router-Enqueue	This router enqueue logic plug-in adds business messages to the router message queue. By default, this logic plug-in is the last one in the router chain.
XOCP-Filter	This filter logic plug-in processes outgoing business messages and generates the message-context document that is used in processing outgoing messages. By default, this is the only logic plug-in in the filter chain. The XOCP filter logic plug-in for a hub delivery channel can use any XPath filter expressions defined to determine whether or not to send a message. For more information, see "Routing and Filtering Business Messages" in *Programming Logic Plug-Ins for B2B Integration*.
XOCP-Encoder	The encoder forwards the message to the B2B transport service.
XOCP-Decoder	The decoder processes the XOCP headers, identifies the sending trading partner, enlists the sending trading partner in a conversation, prepares a reply to return to the sender, and forwards the message to the B2B scheduling service.

For details about developing logic plug-ins and processing messages through the WebLogic Integration B2B engine, see *Programming Logic Plug-Ins for B2B Integration*.

Trading Partners

The WebLogic Integration sample scenario involves three business partners: a buyer (General Control Systems) and two suppliers. For each business partner, a trading partner is configured in the `BulkLoaderData.xml` file. The following trading partners are defined for the sample: WLIS_Buyer, WLIS_SupplierOne, and WLIS_SupplierTwo.

Because these trading partners communicate using the XOCP business protocol, General Control Systems must define its WebLogic Integration system as hub-and-spoke configuration. (See "Getting Started with B2B integration" in *Introducing B2B Integration* for details about configuring B2B integration.) To that end, the `BulkLoaderData.xml` file defines a fourth trading partner: WLIS_Hub.

The WLIS_Hub trading partner acts as an intermediary. It is responsible for mediating messages between the spoke trading partners: WLIS_Buyer, WLIS_SupplierOne, and WLIS_SupplierTwo. The WLIS_Hub trading partner is not the sender or receiver of a business message, but it acts as the proxy buyer and proxy supplier, at different times during the transaction.

Each of the three trading partners—WLIS_Buyer, WLIS_SupplierOne, and WLIS_SupplierTwo—has a collaboration agreement with the WLIS_Hub trading partner. The WLIS_Hub trading partner is responsible for linking collaboration agreements. Such linking is essential, for example, when a message arrives as part of one collaboration agreement and must be routed to another trading partner as part of another collaboration agreement. Collaboration agreements that use the same delivery channel—the channel defined for the WLIS_Hub trading partner—are linked. For details about the collaboration agreements in this scenario, see "Collaboration Agreements" on page 3-12.

Each trading partner element is characterized by various attributes and subelements, some of which contain simple identification information, such as name, email, phone, and fax. Other trading partner configuration information is described in the following table.

Table 3-2 Trading Partner Configuration

Element	Description
Type	Note that `Type="LOCAL"` for all trading partners in the sample scenario because all run on the same instance of WebLogic Server.

Table 3-2 Trading Partner Configuration

Element	Description
Party Identifier	A unique party ID, which is used along with the trading partner name to identify the party in a collaboration agreement.The party ID has a unique name, a business ID, and a business ID type.
Business ID	Descriptive name for the business ID associated with a party ID. In this sample scenario the same business ID (business-id="999999999") is used for both the WLIS_Hub and WLIS_Buyer trading partners because both are owned by GCS. Each supplier trading partner has a unique business ID.
Delivery Channel	References a transport and a document exchange for each trading partner. Note that the delivery channel for the WLIS_Hub trading partner is configured with `routing-proxy="true"`, and the delivery channels for the other trading partners (each of which is configured as a spoke in this scenario) are configured with `routing-proxy=”false”`.
Document Exchange	Defines the business protocol (XOCP for this scenario) and run-time parameters.
Transport	Defines the transport protocol (HTTP) and end-point URI.

Note: See the `BulkLoaderData.xml` file for the sample’s trading partner configuration.

Conversation Definitions

The `BulkLoaderData.xml` file contains two XOCP conversation definitions, one each for the Query Price and Availability (QPA) and the Purchase Order (PO) conversations. There are two roles—buyer and supplier—for each conversation. Each role references the following:

- A BPM workflow template, such as:

  `wlpi-template=”WLIS_SupplierQPA”`

- A WebLogic Integration BPM organization, such as:

```
process-implementation wlpi-org="ORG1"
```

Note: The BPM component of WebLogic Integration was formerly known as WebLogic Process Integrator (WLPI). You may still see references to WebLogic Process Integrator or WLPI, as in the names of the template and organization shown here.

The workflow template, in turn, references the conversation definition to which it applies. (For example, see the reference to a conversation in Figure 3-12, which shows properties for one of the public workflows used in this sample).

For details about the workflows used to implement this sample, see "Business Process and Workflow Modeling" on page 3-17.

Organizations represent different business entities, geographical locations, or any other classifications that are relevant to the particular business of the company. For details about BPM organizations, see "Configuring the Security Realms" in *Programming BPM Client Applications*.

The following listing is an excerpt from the `BulkloaderData.xml` file. It defines the `WLIS_QPAConversation`.

Listing 3-2 Conversation Definition in the BulkLoaderData.xml File

```
...
<conversation-definition
        name="WLIS_QPAConversation"
        version="1.1"
        business-protocol-name="XOCP"
        protocol-version="1.1">
        <role
            name="Buyer"
            wlpi-template="WLIS_BuyerQPAPublic">
            <process-implementation wlpi-org="ORG1" />
        </role>
        <role
            name="Supplier"
            wlpi-template="WLIS_SupplierQPAPublic">
            <process-implementation wlpi-org="ORG1" />
            </role>
</conversation-definition>

...
```

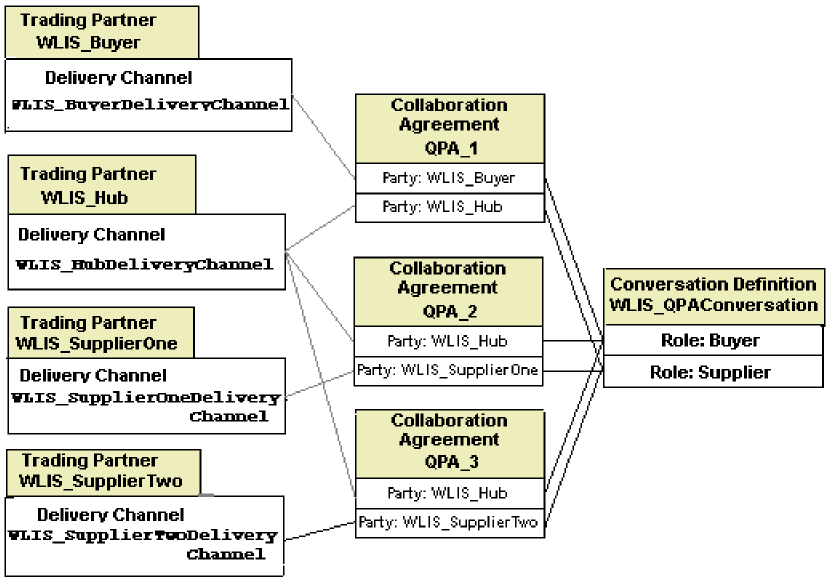
Collaboration Agreements

Six collaboration agreements are used in this sample: three in the QPA conversation and three in the PO conversation. For each conversation, collaboration agreements are defined between the following pairs of business entities:

- WLIS_Buyer and WLIS_Hub
- WLIS_SupplierOne and WLIS_Hub
- WLIS_SupplierTwo and WLIS_Hub

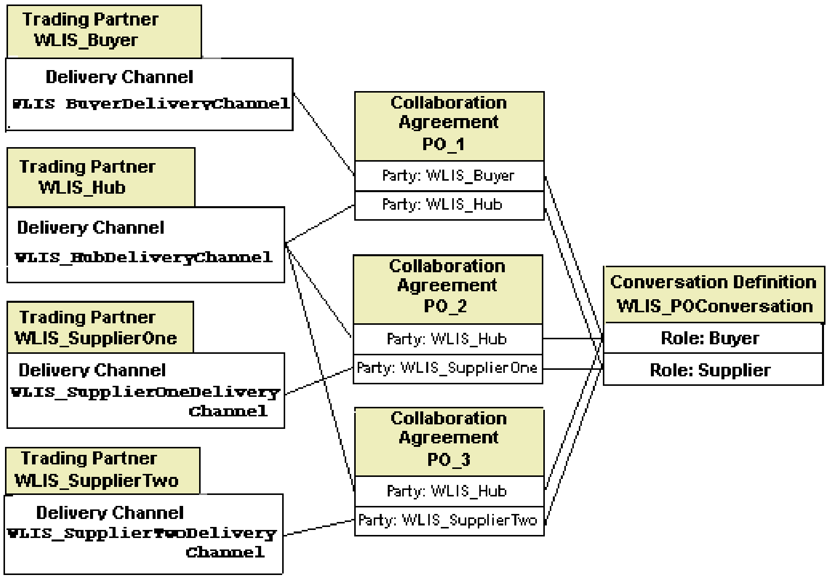
The following figure illustrates the collaboration agreements among trading partners who participate in the QPA conversation (`WLIS_QPAConversation`).

Figure 3-1 Collaboration Agreements Among Trading Partners in the QPA Conversation


Trading Partner WLIS_Buyer
Delivery Channel WLIS_BuyerDeliveryChannel
Trading Partner WLIS_Hub
Delivery Channel WLIS_HubDeliveryChannel
Trading Partner WLIS_SupplierOne
Delivery Channel WLIS_SupplierOneDelivery Channel
Trading Partner WLIS_SupplierTwo
Delivery Channel WLIS_SupplierTwoDelivery Channel
Collaboration Agreement QPA_1
Party: WLIS_Buyer
Party: WLIS_Hub
Collaboration Agreement QPA_2
Party: WLIS_Hub
Party: WLIS_SupplierOne
Collaboration Agreement QPA_3
Party: WLIS_Hub
Party: WLIS_SupplierTwo
Conversation Definition WLIS_QPAConversation
Role: Buyer
Role: Supplier

The following figure illustrates the collaboration agreements among trading partners who participate in the PO conversation (`WLIS_POConversation`).

Figure 3-2 Collaboration Agreements Among Trading Partners in the PO Conversation


Trading Partner WLIS_Buyer
Delivery Channel
WLIS_BuyerDeliveryChannel
Trading Partner WLIS_Hub
Delivery Channel
WLIS_HubDeliveryChannel
Trading Partner WLIS_SupplierOne
Delivery Channel
WLIS_SupplierOneDelivery Channel
Trading Partner WLIS_SupplierTwo
Delivery Channel
WLIS_SupplierTwoDelivery Channel
Collaboration Agreement PO_1
Party: WLIS_Buyer
Party: WLIS_Hub
Collaboration Agreement PO_2
Party: WLIS_Hub
Party: WLIS_SupplierOne
Collaboration Agreement PO_3
Party: WLIS_Hub
Party: WLIS_SupplierTwo
Conversation Definition WLIS_POConversation
Role: Buyer
Role: Supplier

Notice the following details in the preceding figures:

- The WLIS_Hub trading partner is a party in each of the collaboration agreements. In the QPA_1 and PO_1 collaboration agreements, it is assigned the role of (proxy) supplier; in all other collaboration agreements, the role of (proxy) buyer.

  For example, when the WLIS_Hub trading partner receives a QPA message from WLIS_Buyer, it is acting as the proxy supplier in the QPA_1 collaboration agreement. It then changes roles and becomes the proxy buyer for the QPA_2 and QPA_3 collaboration agreements.

- Parties in the collaboration agreements are associated with roles in conversation definitions.You must configure each party in a collaboration agreement with a role name.

The following listing is an excerpt from the `BulkLoaderData.xml` file. It describes the collaboration agreement between WLIS_Hub and WLIS_Buyer.

Listing 3-3 Collaboration Agreement in the BulkLoaderData.xml file

```
...
<collaboration-agreement
name="WLIS_QPAConversation|1.1|WLIS_Buyer|WLIS_Hub"
global-identifier="WLIS_QPAConversation|1.1|WLIS_Buyer|WLIS_Hub
version="1.1"
status="ENABLED"
conversation-definition-name="WLIS_QPAConversation"
conversation-definition-version="1.1">
<party
        trading-partner-name="WLIS_Buyer"
        party-identifier-name="WLIS_BuyerPartyId"
        delivery-channel-name="WLIS_BuyerDeliveryChannel"
        role-name="Buyer"/>
<party
        trading-partner-name="WLIS_Hub"
        party-identifier-name="WLIS_HubPartyId"
        delivery-channel-name="WLIS_HubDeliveryChannel"
        role-name="Supplier"/>
</collaboration-agreement>

...
```

Using the WebLogic Integration B2B Console

WebLogic Integration allows you to bulk load configuration data or enter it through the WebLogic Integration B2B Console. You do not need to run the B2B Console when you run the WebLogic Integration sample, but you can use it to view the repository data that is bulk loaded for the sample (see “Loading the Repository Data” on page 3-5). You can also use the B2B Console to monitor the ongoing conversations while you are running the sample.

Start the WebLogic Integration B2B Console by completing the procedure appropriate for your platform:

Note: If you have not already done so, run the WebLogic Integration sample as described in “Setting Up and Running the Sample” on page 2-1.

- To launch the B2B Console on a Windows system, do one of the following:
  - Click the `B2B` link under `Administration Consoles` on the launcher Web page (`http://localhost:7001`) from which you started the sample.
  - Use menus, as follows:

    Choose Start→Programs→BEA WebLogic E-Business Platform→WebLogic Integration 2.1→Samples→B2B Console.
  - Invoke the `startB2bconsole` script from the command line, as follows:

    a. Open a command window.

    b. Go to the `bin` directory in the directory in which you installed WebLogic Integration. For example, if WebLogic Integration is installed in the default location, enter the following:

    ```
      cd \bea\wlintegration2.1\bin
    ```

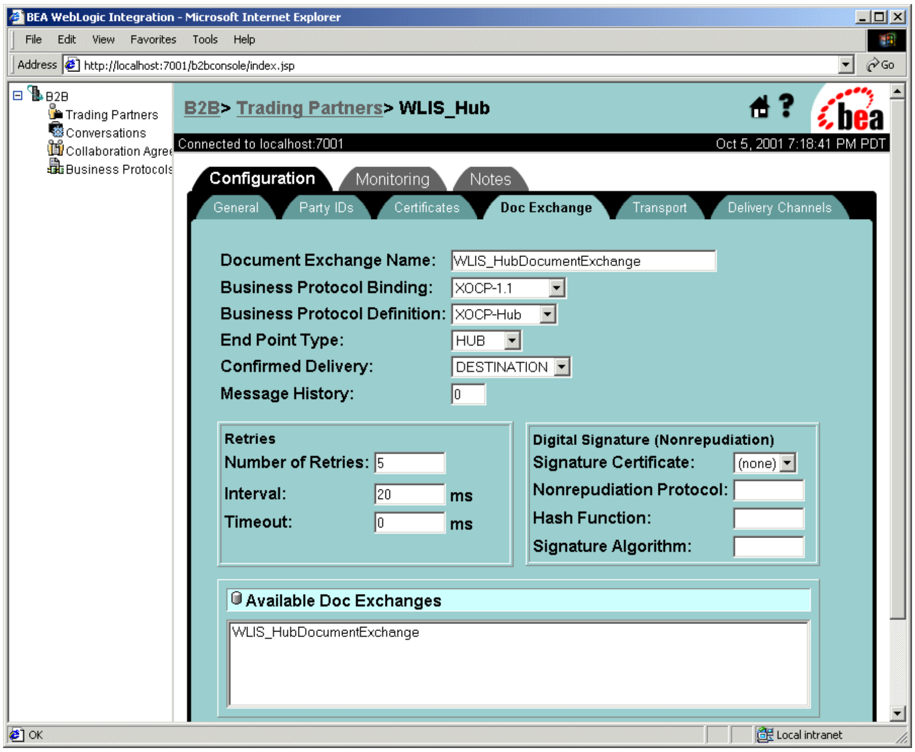
    c. Start the B2B Console by entering:

    ```
      startB2bconsole
    ```
- To launch the B2B Console on a UNIX system, do one of the following:
  - Click the `B2B` link under `Administration Consoles` on the launcher Web page (`http://localhost:7001`) from which you started the sample.
  - Start a Web browser with the following URL:

    ```
    http://localhost:7001/b2bconsole
    ```

The following figure shows the WebLogic Integration B2B Console with the WebLogic Integration sample data loaded.

Figure 3-3 WebLogic Integration B2B Console Displaying Sample Data


BEA WebLogic Integration - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Address http://localhost:7001/b2bconsole/index.jsp
Go
B2B
Trading Partners
Conversations
Collaboration Agree
Business Protocols
B2B> Trading Partners> WLIS_Hub
bea
Connected to localhost:7001
Oct 5, 2001 7:18:41 PM PDT
Configuration
Monitoring
Notes
General
Party IDs
Certificates
Doc Exchange
Transport
Delivery Channels
OK
Local intranet

Document Exchange Name: WLIS_HubDocumentExchange
Business Protocol Binding: XOCP-1.1
Business Protocol Definition: XOCP-Hub
End Point Type: HUB
Confirmed Delivery: DESTINATION
Message History: 0

Retries
Number of Retries: 5
Interval: 20 ms
Timeout: 0 ms

Digital Signature (Nonrepudiation)
Signature Certificate: (none)
Nonrepudiation Protocol:
Hash Function:
Signature Algorithm:

Available Doc Exchanges
WLIS_HubDocumentExchange

See *Online Help for the WebLogic Integration B2B Console* and *Administering B2B Integration* for details about using the WebLogic Integration B2B Console to configure B2B integration. See "Working with the Bulk Loader" in *Administering B2B Integration* for details about the Bulk Loader.

Business Process and Workflow Modeling

This section provides a brief introduction to the business process management (BPM) functionality provided by WebLogic Integration, followed by instructions for starting and using the Studio, and descriptions of the two business processes implemented by the WebLogic Integration sample: Query Price and Availability (QPA) and Purchase Order (PO). It includes the following topics:

- Introduction to BPM
- Using the WebLogic Integration Studio
- QPA Business Process
- PO Business Process

Introduction to BPM

Workflows that implement the roles assigned to trading partners in a conversation definition (see "Conversation Definitions" on page 3-10) are referred to as *collaborative workflows*.

A workflow template represents a workflow, and combines various workflow template definitions (versions) of its implementation. Workflow templates are designed and edited in the WebLogic Integration Studio. Several BPM plug-ins extend the functionality of the Studio:

- B2B integration plug-in—Supports B2B integration, that is, the design and management of collaborative workflows. The Studio allows you to assign properties to the workflows. These properties make the workflows usable in the B2B integration environment.
- Application integration plug-in—Allows you to design workflows that can integrate back-end and legacy enterprise information systems (EIS).
- Data integration plug-in—Allows you to design workflows that integrate heterogeneous data formats, by making it possible to share data among heterogeneous EIS applications.

In this sample scenario, trading partners implement both private and collaborative workflows. Private workflows work in conjunction with the collaborative workflows, and implement local processes for the trading partners. Local and private processes are not necessarily dictated by the conversation definition. For example, when a trading partner starts a conversation, that trading partner's private workflow can start the collaborative workflow to initiate the conversation.

The following sections describe the implementation of the business processes on the buyer-side and supplier-side of the sample business transaction. Key workflow design elements, tasks, and events are highlighted.

Using the WebLogic Integration Studio

The WebLogic Integration Studio allows you to design new workflows and monitor running workflows using a familiar flowchart paradigm. You are not required to run the Studio when you run the WebLogic Integration sample, but you may find it useful for viewing the details of any workflow or workflow node, and for learning how nodes are defined and configured for this sample. You can also use the Studio to monitor the workflows as you run the sample.

This section provides procedures for starting and using the Studio, and a list of components used by the sample to manage business processes.

Launching the Studio

Launch the Studio by completing the procedure appropriate for your platform:

- To launch the Studio on a Windows system, do one of the following:
  - Use menus, as follows:

    a. Choose Start→Programs→BEA WebLogic E-Business Platform→WebLogic Integration 2.1→Studio.

    b. Log on to the Studio (user: `wlpisystem`; password: `wlpisystem`).
  - Invoke the `Studio` script from the command line, as follows:

    a. Open a command window.

b. Go to the `bin` directory in the directory where you installed WebLogic Integration. For example, if WebLogic Integration is installed in the default location, enter the following:

```
cd \bea\wlintegration2.1\bin
```

c. Execute the `studio` command by entering:

```
studio
```

d. Log on to the Studio (user: `wlpisystem`; password: `wlpisystem`).

- To launch the Studio on a UNIX system, complete the following tasks:

  a. Go to the `bin` directory in the directory where you installed WebLogic Integration. For example, if WebLogic Integration is installed in the default location, enter the following:

  ```
  cd BEA_Home/wlintegration2.1/bin
  ```

  b. Start the Studio application by entering:

  ```
  . ./studio
  ```

  c. Log on to the Studio (user: `wlpisystem`; password: `wlpisystem`).

Viewing Workflow Templates in the Studio

To view a workflow template and its properties in the Studio, complete the following procedure:

1. In the left pane of the Studio, make sure ORG1 is selected in the Organization field.
2. In the left pane, double-click the Templates folder to display a list of workflow templates.
3. Expand the Templates folder to display the list of workflow template definitions. The template definitions of interest for this sample application are listed in Table 3-3, “Components Imported to the WebLogic Integration Repository.” They are imported, via the `workflow.jar` file, when you configure the sample, as described in “Step 1. Configure and Invoke the Launcher Web Page” on page 2-3.
4. Right-click a template definition, and select Open to open the workflow template in the Studio.

Note: You can also expand a particular workflow template definition to display folders containing the Tasks, Decisions, Events, Joins, Starts, Dones, and Variables for that workflow template definition.

5. Double-click any node in the Studio to display the Properties dialog box for that node.

See *Using the WebLogic Integration Studio* for complete details about the Studio tools and functionality.

BPM Components Used in the Sample

Workflow templates and other data required by the Studio and process engine for this sample application are loaded into the WebLogic Integration repository, via the `workflow.jar` file, when you configure the sample. Imported components include those listed in the following table.

Table 3-3 Components Imported to the WebLogic Integration Repository

Component	Name

Business Operations	`binary to file`
	`file to binary`
	`xmlToFile`
	`create POAck from PO`

Template Definitions	`WLIS_BuyerPOPrivate`
	`WLIS_BuyerPOPublic`
	`WLIS_BuyerQPAPrivate`
	`WLIS_BuyerQPAPublic`
	`WLIS_SupplierOnePOPrivate`
	`WLIS_SupplierOneQPAPrivate`
	`WLIS_SupplierPOPublic`
	`WLIS_SupplierQPAPublic`
	`WLIS_SupplierTwoPOPrivate`
	`WLIS_SupplierTwoQPAPrivate`

XML Repository Folders	`com.bea.wlxt.MFL` `wlis`

Table 3-3 Components Imported to the WebLogic Integration Repository

Component	Name

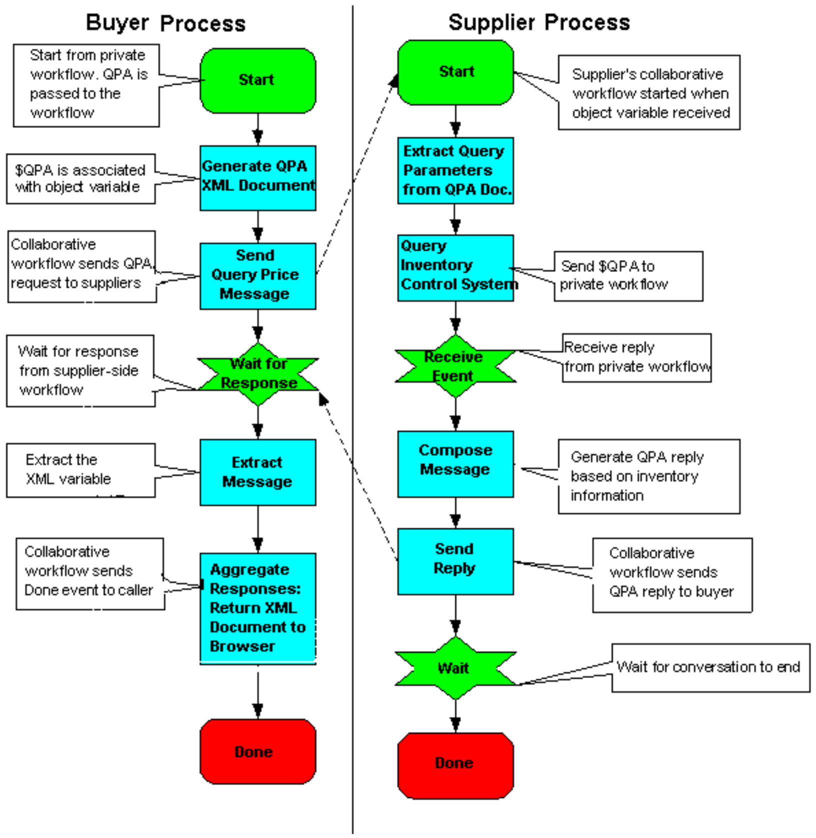
XML Repository Entities	`PO.mfl`
	`PO.dtd`
	`POAck.mfl`
	`POAck.dtd`

Event Keys	`PORequest`
	`AggregatedQPAResponse`
	`PurchaseOrderAcknowledgement`

QPA Business Process

Due to the supply shortage of metal boxes, GCS (the buyer trading partner) sends a QPA message for such boxes to selected suppliers. The following diagram illustrates the flow of events for the QPA business process.

Figure 3-4 Process Flow in the QPA Business Process


Buyer Process
Start from private workflow. QPA is passed to the workflow
Start
$QPA is associated with object variable
Generate QPA XML Document
Collaborative workflow sends QPA request to suppliers
Send Query Price Message
Wait for response from supplier-side workflow
Wait for Response
Extract the XML variable
Extract Message
Collaborative workflow sends Done event to caller
Aggregate Responses: Return XML Document to Browser
Done
Supplier Process
Start
Supplier's collaborative workflow started when object variable received
Extract Query Parameters from QPA Doc.
Query Inventory Control System
Send $QPA to private workflow
Receive Event
Receive reply from private workflow
Compose Message
Generate QPA reply based on inventory information
Send Reply
Collaborative workflow sends QPA reply to buyer
Wait
Wait for conversation to end
Done

The following sequence summarizes the events illustrated in the preceding figure:

1. The buyer trading partner creates a QPA.
2. The QPA is sent to suppliers.
3. The suppliers process the QPA and generate responses.
4. The buyer trading partner collects and aggregates responses from the suppliers.

Note: The preceding figure shows a high-level view of the QPA business process. Each side of the process is implemented by a public (collaborative) and a private workflow. The following sections describe these workflows:

- Overview of the QPA Implementation
- Buyer-Side Implementation
- Supplier-Side Implementation

Overview of the QPA Implementation

In this scenario, each trading partner implements a private and a public workflow for the QPA process. The following five workflow templates are used in this sample's QPA process.

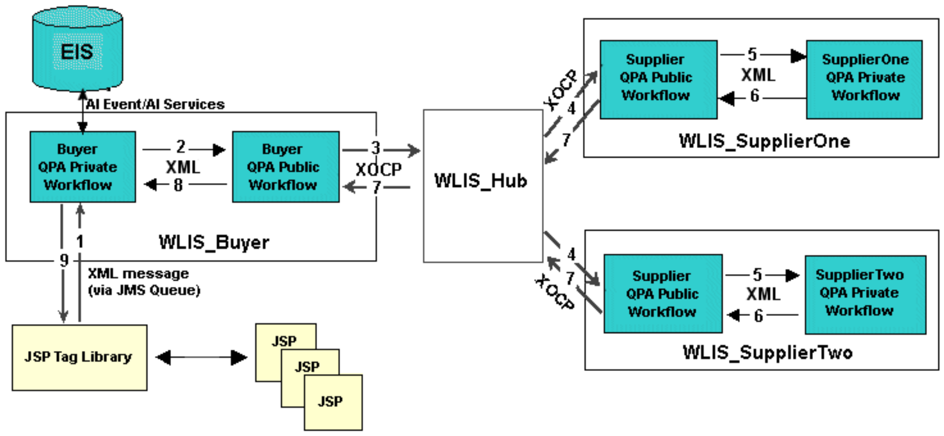
Table 3-4 Workflows for the Sample QPA Process

Role	Public/Private	Workflow Name
Buyer	Private	`WLIS_BuyerQPAPrivate`
Buyer	Public	`WLIS_BuyerQPAPublic`
Supplier	Public	`WLIS_SupplierQPAPublic` **Note:** Both suppliers in the scenario use the same public workflow.
Supplier	Private	`WLIS_SupplierOneQPAPrivate`
Supplier	Private	`WLIS_SupplierTwoQPAPrivate`

WebLogic Integration manages the business conversations and collaboration agreements between business partners, and it automates the business message exchange between the buyer and suppliers. The workflows are referenced in the collaboration agreements and conversations, as described in "Understanding the Repository Data" on page 3-6.

This sample uses JSPs and JSP tag libraries to initiate the QPA process and display QPA request and response data. The following figure illustrates the data flow among the trading partners involved in the QPA business transaction.

Figure 3-5 Data Flow in the QPA Business Process


EIS
AI Event/AI Services
Buyer QPA Private Workflow
2
XML
8
Buyer QPA Public Workflow
3
XOCP
7
WLIS_Buyer
1
9
XML message (via JMS Queue)
JSP Tag Library
JSP
JSP
JSP
WLIS_Hub
XOCP
4
7
Supplier QPA Public Workflow
5
XML
6
SupplierOne QPA Private Workflow
WLIS_SupplierOne
4
7
XOCP
Supplier QPA Public Workflow
5
XML
6
SupplierTwo QPA Private Workflow
WLIS_SupplierTwo

The buyer-side and supplier-side implementations are discussed in more detail in "Buyer-Side Implementation" on page 3-26 and "Supplier-Side Implementation" on page 3-47.

The following sequence of events summarizes the data flow among trading partners and workflows:

1. The JSP containing the QPA form (see Figure 2-2) sends the QPA request to a JMS queue and triggers the `WLIS_BuyerQPAPrivate` workflow.
2. The `WLIS_BuyerQPAPrivate` workflow invokes the `WLIS_BuyerQPAPublic` workflow, passing it the QPA request XML document. It then initiates the QPA conversation.
3. Based on the collaboration agreement between the WLIS_Buyer and the WLIS_Hub trading partners, the `WLIS_BuyerQPAPublic` workflow packs the QPA request XML into an XOCP message and sends it to the WLIS_Hub trading partner.

   Note: When the WLIS_Hub trading partner receives a message, it is acting as the proxy supplier in the collaboration agreement.
4. The WLIS_Hub trading partner routes the message to the destination trading partners, WLIS_SupplierOne and WLIS_SupplierTwo, based on registered collaboration agreements between itself and each supplier.

Note: In this step, the WLIS_Hub trading partner changes roles and becomes the proxy buyer in the collaboration agreements between itself and the suppliers.

Each supplier's public workflow is triggered by receiving the XOCP message. In this scenario, WLIS_SupplierOne and WLIS_SupplierTwo share the public workflow (`WLIS_SupplierQPAPublic`). The public workflow extracts the QPA request XML document from the message.

5. The `WLIS_SupplierQPAPublic` workflow invokes a private workflow for each supplier and passes the QPA request XML document to the private workflows.
6. Each supplier's private workflow creates its own QPA response (an XML document), and attaches it to the return variable of the public workflow.
7. The `WLIS_SupplierQPAPublic` workflow extracts the QPA response XML document, packs it into an XOCP message, and sends it to the buyer.

   Note that the WLIS_Hub trading partner acts as a routing proxy for WLIS_Buyer. When the supplier trading partners send response messages to WLIS_Hub (based on collaboration agreements between WLIS_Hub and each supplier trading partner), WLIS_Hub acts as the proxy buyer.

   WLIS_Hub then changes roles to that of proxy supplier, and routes the response messages to the buyer (WLIS_Buyer), based on the collaboration agreement between WLIS_Hub and WLIS_Buyer.
8. The buyer's public workflow (`WLIS_BuyerQPAPublic`):
   a. Extracts the QPA response XML document from the XOCP message it receives.
   b. Aggregates both supplier response documents into a single XML document and posts it, via a JMS queue, to the buyer's private workflow (`WLIS_BuyerQPAPrivate`).
   c. Terminates the QPA conversation and notifies the supplier public workflow (`WLIS_SupplierQPAPublic`).
9. The buyer's private workflow (`WLIS_BuyerQPAPrivate`) receives the aggregated QPA response XML document and writes it to an XML file. A JSP parses the XML and displays the aggregated QPA response in the Web browser.

   This step marks the end of the QPA business process.

Buyer-Side Implementation

To implement this solution, the buyer (GCS) implements a custom client (Web user interface) to drive the sample, process messages, and exchange XML messages with the WebLogic Integration process engine, using JMS queues. GCS also implements a private workflow, to manage the buyer's back-end processes, and a public workflow that choreographs the exchange of messages in the QPA conversation. This section discusses these components:

- Buyer-Side Web User Interface
- Buyer QPA Private Workflow
- Buyer QPA Public Workflow

Buyer-Side Web User Interface

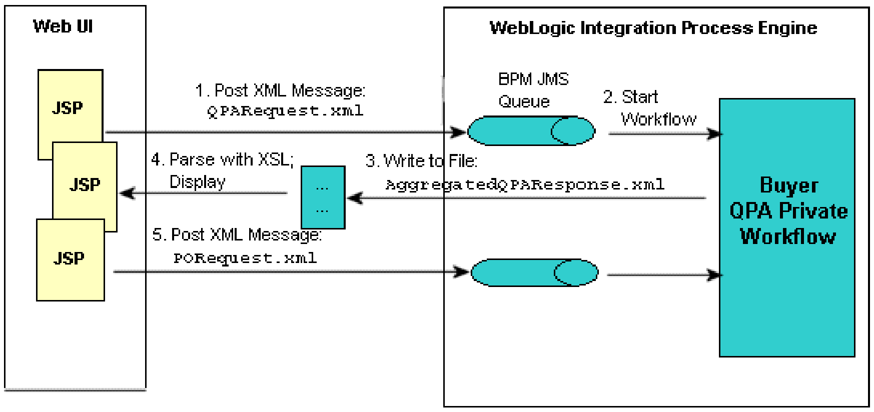
Java Server Pages (JSPs) and JSP tag libraries are used to initiate the QPA process and display request and response data in your Web browser. The source files related to the QPA process can be found in the directories listed in the following table.

Table 3-5 Source Files for the QPA Process

Directory in Your WebLogic Integration Installation	Source File
`\samples\wlis\src\examples\wlis\tags`	■ `SendQPARequestTag.java` ■ `CheckQPAResponseTag.java`
`\samples\wlis\web`	■ `SendQPARequest.jsp` ■ `CheckQPAResponse.jsp` ■ `QPAform.htm` ■ `WaitQPAResponse.htm`
`\samples\wlis\lib\xsl`	`ProcessQPAResponse.xsl`

The Web user interface interacts with the buyer's private workflow (`WLIS_BuyerQPAPrivate`), as illustrated in the following figure.

Figure 3-6 Interaction Between the Web User Interface and the Buyer's Private Workflow


Web UI
JSP
JSP
JSP
WebLogic Integration Process Engine
1. Post XML Message: QPARequest.xml
BPM JMS Queue
2. Start Workflow
4. Parse with XSL; Display
3. Write to File: AggregatedQPAResponse.xml
5. Post XML Message: PORequest.xml
Buyer QPA Private Workflow

The following sequence of events describes the interaction illustrated in the preceding figure:

1. The QPA request is created as XML, based on the input from the HTML Web form (see "Step 4. Send the QPA Request" on page 2-9). The XML is sent, via a JSP (`SendQPARequest.jsp`), to a JMS queue.

   The following listing is code from the `SendQPARequestTag.java` file that defines the name of the JMS queue and the queue connection factory to which the XML message is posted:

Listing 3-4 SendQPARequestTag.java

```
//Defines the JMS connection factory.
final String JMS_FACTORY = "com.bea.wlpi.QueueConnectionFactory";

//Defines the JMS queue.
final String QUEUE = "com.bea.wlpi.EventQueue";
...
```

2. The buyer's private workflow (`WLIS_BuyerQPAPrivate`) is started.

3. The `WLIS_BuyerQPAPrivate` workflow receives responses from both suppliers, aggregates them, and writes the QPA responses to a local file, `AggregatedQPAResponse.xml`.

4. A JSP (`CheckQPAResponse.jsp`) reads the file using a JSP tag, parses the XML data using XSL (see `\samples\wlis\lib\xsl\processQPAResponse.xsl`), and displays the results in the Web browser as shown in Figure 2-5, "Create PO Window" on page 2-11.

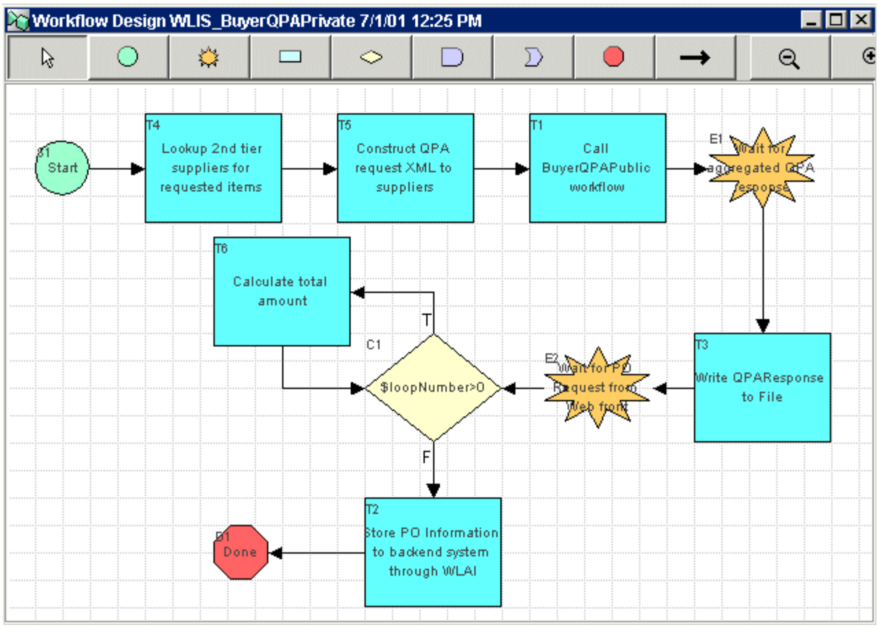
5. When running the sample, you, as the buyer, select the supplier. (See "Step 6. Create the Purchase Order" on page 2-11). The JSP posts an XML message that contains the QPA response data for the selected supplier to the JMS topic. The buyer's private workflow (`WLIS_BuyerQPAPrivate`) receives and processes the message.

Buyer QPA Private Workflow

The `WLIS_BuyerQPAPrivate` workflow is initiated by the JMS message, which contains the QPA request data, sent from a JSP. The interaction of the workflow with the Web user interface is described in "Buyer-Side Web User Interface" on page 3-26.

The `WLIS_BuyerQPAPrivate` workflow template is shown in the following figure.

Figure 3-7 WLIS_BuyerQPAPrivate Workflow Template


Workflow Design WLIS_BuyerQPAPrivate 7/1/01 12:25 PM
S1
Start
T4
Lookup 2nd tier suppliers for requested items
T5
Construct QPA request XML to suppliers
T1
Call BuyerQPAPublic workflow
E1
Wait for aggregated QPA response
T6
Calculate total amount
T
C1
$loopNumber>0
E2
Wait for PO Request from Web front
T3
Write QPAResponse to File
F
T2
Store PO Information to backend system through WLAI
D1
Done

The following sections define the key tasks and events at the nodes in the `WLIS_BuyerQPAPrivate` workflow template, shown in the preceding figure:

- Start
- Lookup 2nd Tier suppliers for requested items
- Construct QPA Request XML to Suppliers
- Call BuyerQPAPublic workflow
- Wait for aggregated QPA Response
- Write QPAResponse to File
- Wait for PO Request from Web Front
- Decision
- Calculate total amount

- Store PO Information to backend system through WLAI

Start

An XML event received from the `SendQPARequest.jsp` JSP triggers the workflow.

The XML is defined by the `QPARequest.dtd` file. DTD files for the sample are available in Appendix A, "DTDs." Note that the first element in the `QPARequest.dtd`:

`QPARequestId`: (`<!ELEMENT QPARequestId (#PCDATA)>`)

For this sample scenario, a unique value for the `QPARequestId` element is created by the JSP every time a QPA request is generated from the buyer's Send QPA Request Web page (see "Step 3. Start the Sample" on page 2-8). You can see the value that is generated for `QPARequestId` every time you run the sample by looking at the form containing the QPA response that is returned to the buyer (see "Step 5. Check for QPA Responses" on page 2-10).

The Start node extracts the XML message using an XPath expression, and stores the message in a workflow variable (`qpaRequestXML`). The Start node also extracts the value of the `QPARequestId` element, and assigns it to a workflow variable, `QPARequestId`. The latter variable is used in other workflows in this sample as the *key value expression* for an *event key*.

What Is an Event Key?

The WebLogic Integration process engine does not start a workflow or trigger an Event node unless the DOCTYPE or root element in the Start or Event Properties dialog box matches that in the incoming XML message. In addition to using the DOCTYPE or root element, you can further qualify the start of a workflow or the triggering of an Event node by using an event key. An event key allows you to specify the contents of incoming XML messages that will trigger a Start or Event node.

An event key consists of two parts: a *key value expression* and an *event key expression*. A key value expression is a workflow expression that specifies the exact data that an incoming document must contain for the node to be triggered. An event key expression is an expression that specifies the exact element in the XML message in which the target data is found.

To configure event key expressions, or to see the configurations in this sample, choose Configuration→Events from the Studio task menu.

The following event key expressions are configured for this sample scenario:

Event Key Expressions

`PORequest.QPARequestId`

`AggregatedQPAResponse.QPARequestId`

`PurchaseOrderAcknowledgement.PONumber`

After you configure event key expressions, they are available for all workflows in all organizations. The event key expression must correspond to the value specified by the key value expression defined in the Start or Event properties dialog box, so that the process engine can compare the two values at run time and determine whether there is a match.

You may notice the `QPARequestId` key value expression used by workflow nodes in other sections of this document. See "Configuring Workflow Resources" in *Using the WebLogic Integration Studio* for details about defining and using event keys in the Studio tool.

Lookup 2nd Tier suppliers for requested items

A set of suppliers, which we refer to as tier 2 (or category 2) suppliers for this sample scenario, are defined in the buyer's enterprise information system (EIS). The EIS for the sample is an RDBMS. The data for these suppliers is loaded in the WebLogic Integration repository when you run the `RunSamples` script to set up the database for the sample (see "Running the Sample" on page 2-2). These suppliers are accessed when the primary supplier of an item cannot meet the demand, as is the case in our sample scenario.

An application view, namely `WLISAppView.sav`, is also deployed for this sample. See "Application Integration" on page 3-77 for details about the application view. An application integration plug-in extends the functionality of the Studio so that application view services and events can be readily integrated with workflow templates. From the Studio, you can select a service to invoke, and specify service request and response variables (the variables must be in XML format).

This task node completes the following actions:

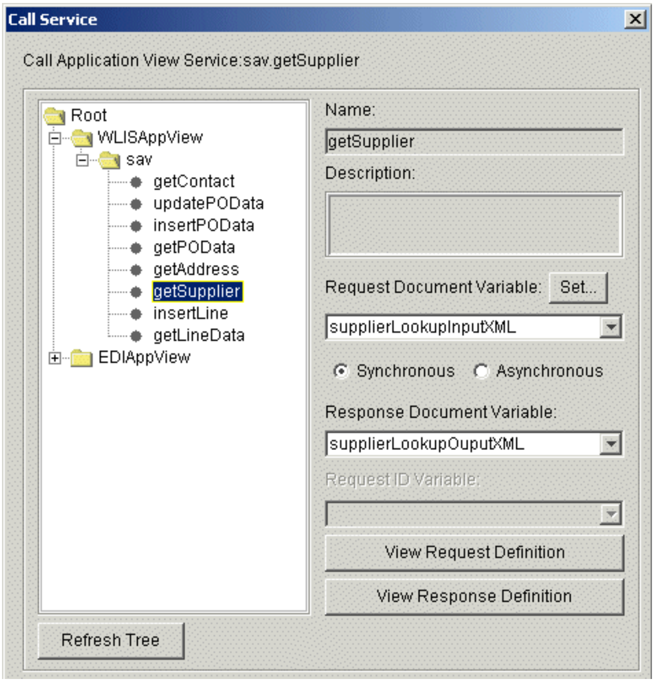
1. Composes a workflow variable (`supplierLookupInputXML`).

2. Calls the `getSupplier` service on the `WLISAppView.sav` application view to retrieve information from the EIS about the tier 2 suppliers.

You can select a service to be invoked from a workflow as follows:

1. Double-click the task node to invoke the Task Properties dialog box.
2. Click Add to open the Add Actions dialog box.
3. Choose AI Actions→Call Application View Service to invoke the Call Service dialog box, which displays a list of the services for the `WLISAppView.sav` application view, as shown in the following figure.

Figure 3-8 Application Integration Call Service Dialog Box



Call Service

Call Application View Service:sav.getSupplier

Name: getSupplier

Description:

Request Document Variable: supplierLookupInputXML

Synchronous [✓ Synchronous] Asynchronous

Response Document Variable: supplierLookupOuputXML

Request ID Variable:

Note that the value in the Request Document Variable field in the preceding figure is the XML variable that was created at this node (`supplierLookupInputXML`).

The application view service is called synchronously at this node. Therefore the workflow waits for the service to return a response document to this node before it continues.

The value in the Response Document Variable field is `supplierLookupOutputXML`. This variable receives the response from the application view service. The `supplierLookupOutputXML` variable is used by the workflow at the next node to construct the request XML messages.

Note: You can examine the XML schema of the input or the response document, by clicking View Request Definition or View Response Definition, respectively. The View Definition dialog box is displayed. Click Close when finished.

See *Using Application Integration* for information about using the application integration plug-in provided by WebLogic Integration.

Construct QPA Request XML to Suppliers

This task node creates workflow variables (`qpaRequestXMLOne` and `qpaRequestXMLTwo`) to hold the QPA Request XML message for the suppliers. These XML messages are sent from the buyer's public workflow (`WLIS_BuyerQPAPublic`) to the suppliers' public workflow (`WLIS_SupplierQPAPublic`) as part of an XOCP business message.

Because the supplier trading partners are collocated on a single instance of WebLogic Integration, and the suppliers share the same public workflow for this sample, the correct recipient for the incoming XOCP message must be identified.

To that end, a unique attribute value (*`SupplierName`*) is added to the root element of the QPA Request XML. The value for `SupplierName` is composed from the variable returned as a result of the `CallApplicationViewService:sav.getSupplier` event at the previous node in the workflow: `supplierLookupOutputXML`. For example, the value for the `SupplierName` attribute in the `qpaRequestXMLOne` workflow variable is generated from the following XPath expression:

```
ToString(XPath("//Row[1]/supplierName/text()",
$supplierLookupOuputXML))
```

To see the attributes and elements for `qpaRequestXMLOne` and `qpaRequestXMLTwo`:

1. Double-click this node to display the Task Properties dialog box.

2. Choose Actions→Activated.

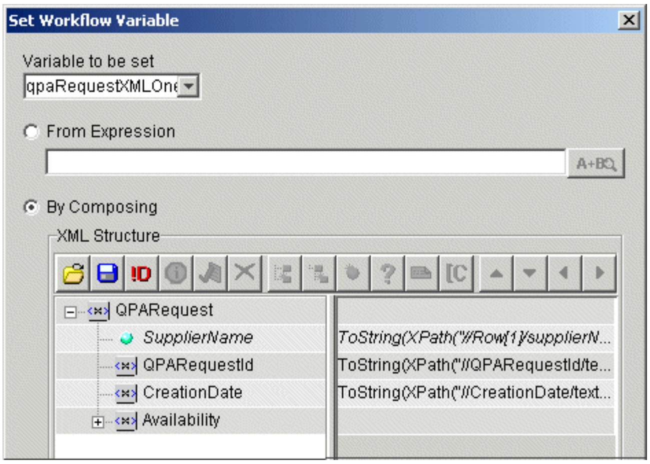
3. Double-click one of the following actions:

`Set workflow variable qpaRequestXMLOne structure`

`Set workflow variable qpaRequestXMLOne structure`

The XML structure dialog box shown in the following figure is displayed.

Figure 3-9 Set Workflow Variable Dialog Box



Set Workflow Variable

Variable to be set: qpaRequestXMLOne

From Expression

[✓ By Composing]

XML Structure

QPARequest	
SupplierName	ToString(XPath("//Row[1]/supplierN...
QPARequestId	ToString(XPath("//QPARequestId/te...
CreationDate	ToString(XPath("//CreationDate/text...
Availability	

Call BuyerQPAPublic workflow

This task node starts the `WLIS_BuyerQPAPublic` workflow and passes workflow variables containing the QPA Request XML documents to the public workflow. Because the `WLIS_BuyerQPAPrivate` workflow is the initiator of a WebLogic Integration B2B conversation (`WLIS_QPAConversation`), this task node uses the special Start Public Workflow action (provided by the B2B integration plug-in), instead of the default Start Workflow action.

The key difference between the Start Public Workflow and the Start Workflow actions is that the workflow template binding is dynamic when you use the former. The `WLIS_BuyerQPAPrivate` workflow determines which workflow template to invoke at run time, based on conversation properties defined in the Start Public Workflow action.

Also, based on the conversation information passed as part of the Start Public Workflow call, the B2B engine uses the collaboration agreement information (stored in the WebLogic Integration repository) to associate the called workflow with other public workflows in the same collaboration agreement.

You can define a Start Public Workflow action in the WebLogic Integration Studio as follows:

1. Double-click the task node to invoke the Task Properties dialog box.

2.ChooseActions→Add→IntegrationActions→B2BIntegration→StartPublic Workflow to display the Start Public Workflow dialog box.

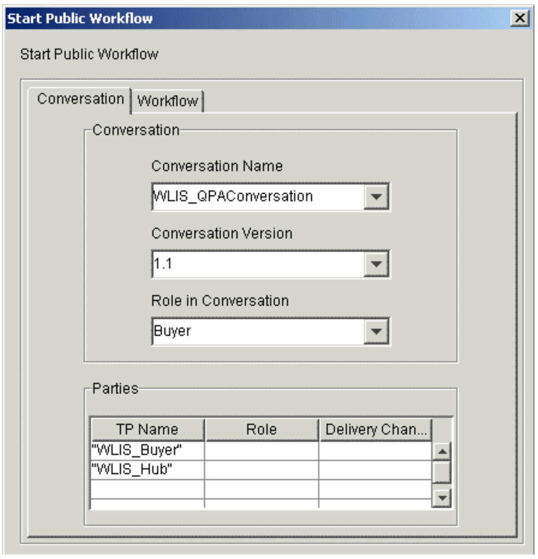
To view the Start Public Workflow properties already specified for this node:

1. Double-click the task node to invoke the Task Properties dialog box.

2. Choose Actions→Activated.

3. Double-click Start Public Workflow to display the Start Public Workflow dialog box shown in the following figure.

Figure 3-10 Start Public Workflow Dialog Box



Conversation Name: WLIS_QPAConversation

Conversation Version: 1.1

Role in Conversation: Buyer

TP Name	Role	Delivery Chan...
"WLIS_Buyer"		
"WLIS_Hub"		

Note the following properties for the WebLogic Integration sample in the preceding figure:

- The Conversation tab—Information specified on this tab is used by WebLogic Integration to locate the active collaboration agreements in the repository that define an agreement between the specified parties, in the specified conversation:

- The conversation name, version, and role are defined as `WLIS_QPAConversation`, `1.1`, and `buyer`, respectively.
- The parties in the conversation are specified by their trading partner names: `WLIS_Buyer` and `WLIS_Hub`. You can optionally add values to the Role and Delivery Channel fields, which can be used, in addition to the values in the Name fields, to locate the collaboration agreement of interest.

See Listing 3-3, "Collaboration Agreement in the BulkLoaderData.xml file," for an excerpt from the `BulkLoaderData.xml` file. It describes the collaboration agreement that the process engine locates in the repository to meet the specifications defined on this Conversation tab.

- The Workflow tab—Variables that are passed to the called workflow are defined on the Workflow tab. The variable names specified here are also specified as input variables to the called workflow (`WLIS_BuyerQPAPublic`). They are:
  - `QPARequestXMLOne`—Contains the value of the `qpaRequestXMLOne` variable defined at the previous workflow node.
  - `QPARequestXMLTwo`—Contains the value of the `qpaRequestXMLTwo` variable defined at the previous workflow node.
  - `QPAPrivateFlowId`—Workflow instance ID used to identify the instance of the `WLIS_BuyerQPAPrivate` workflow that calls the public workflow. (Several instances of the workflow template may be running.) See "Publish Aggregated QPA Response" on page 3-46 to understand where this instance ID is used by the called public workflow.

Note also that the subworkflow (`WLIS_BuyerQPAPublic`) is invoked asynchronously at this node. If you open the Task Properties dialog box and choose Actions→Activated, the following actions are specified in the order shown:

1. `Start Public Workflow`
2. `Mark task` "`CallBuyerPublic workflow`" `done`

See *Creating Workflows for B2B Integration* for more details about workflows in B2B conversations.

Wait for aggregated QPA Response

The workflow waits, at this event node, for a specific XML event from the `WLIS_BuyerQPAPublic` workflow. When it receives the XML event, it extracts the XML using an XPath expression, and stores it in a workflow variable. The XML is defined by the `AggregatedQPAResponse.dtd` file. DTDs for the sample application are available in Appendix A, “DTDs.”

This node uses a key value expression defined for it: `QPARequestId`. This expression is used to specify the exact data that the incoming document must contain for the node to be triggered.

If there are a number of instances of the private workflow (`WLIS_BuyerQPAPrivate`), the appropriate Wait for aggregated QPA Response node, in the appropriate workflow template instance, needs to be triggered at this point in the QPA process. The workflow instance that contains the node to be triggered is specified by the value of `QPARequestId`. (See the discussion about event keys in “Start” on page 3-30.)

Write QPAResponse to File

This action node uses the `xmlToFile` business operation to write the QPA response from both suppliers to the following local file in your WebLogic Integration installation:
`\config\samples\data\AggregatedQPAResponse.xml`.

The local file is consumed by a JSP.

To see the business operations defined for this sample, choose Configuration→Business Operations from the Studio task menu. The Business Operations dialog box containing a list of business operations is displayed. Double-click any business operation to see more information about it. You can find the Java class relevant to the `xmlToFile` business operation in the following location in your WebLogic Integration installation:
`\samples\wlis\src\examples\wlis\common\util\Utils.java`.

Wait for PO Request from Web Front

An event node that causes the workflow to wait until it receives the PO Request from the PO Request Web page. The PO Request XML message is assigned to the `poRequestXML` variable at this node. Note also that three other variables are assigned at this node. They are `loopNumber`, `counter`, and `totalAmount`, and are used in the next node in the workflow.

Decision

This node specifies a condition that must be evaluated. The result of the evaluation determines the path the workflow follows. When the condition evaluates to true, that is the value of `loopNumber` is greater than zero, the workflow proceeds to the Calculate total amount node. When the condition evaluates to false, the workflow proceeds to the following task node: Store PO information to the back-end system through WLAI.

Calculate total amount

This task node assigns values to three workflow variables, based on the results of expressions defined in the Set Workflow Variable dialog box. Values are first calculated for the quantity and the unit price of the requested items. The total amount for the order is the product of the quantity and unit price values.

Store PO Information to backend system through WLAI

This task node inputs the PO information into the Enterprise Information System (in this case, an RDBMS) using the application integration functionality of WebLogic Integration. From the Studio, you can select a service to invoke on the `WLISAppView.sav` application view that is also deployed for this sample, and specify the service request and response variables, which must be in XML format.

The application view services called from this node**,** `insertPOData` and `insertLine`, are called synchronously. Therefore the workflow waits for the service to return a response document to this node before it proceeds to the Done node.

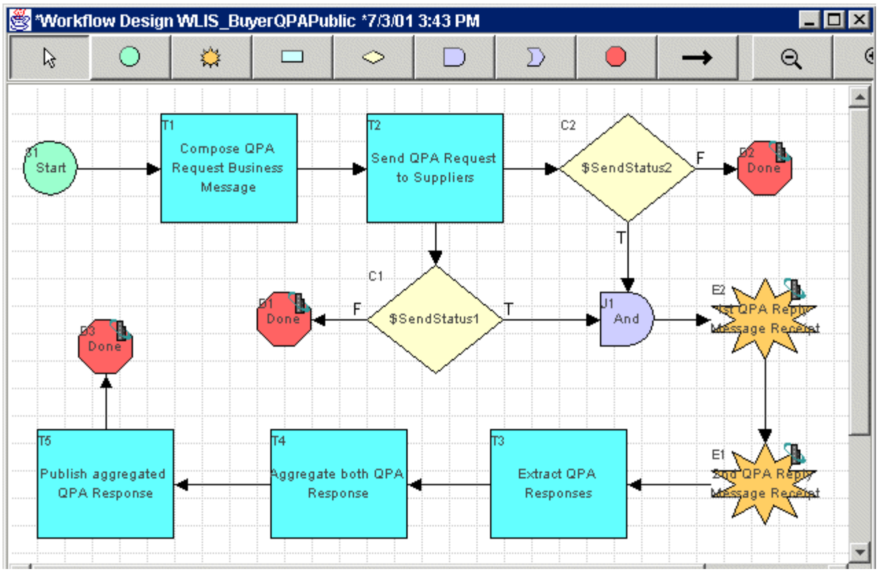
See “Buyer PO Private Workflow” on page 3-61 for more details.

Buyer QPA Public Workflow

The buyer’s public workflow (`WLIS_BuyerQPAPublic`) is initiated by the buyer’s private workflow (`WLIS_BuyerQPAPrivate`) for the QPA business process.

The following figure shows the `WLIS_BuyerQPAPublic` workflow template.

Figure 3-11 WLIS_BuyerQPAPublic Workflow Template


*Workflow Design WLIS_BuyerQPAPublic *7/3/01 3:43 PM
S1 Start
T1 Compose QPA Request Business Message
T2 Send QPA Request to Suppliers
C2 $SendStatus2
F
D2 Done
T
C1 $SendStatus1
F
D1 Done
T
J1 And
E2 1st QPA Reply Message Receipt
E1 2nd QPA Reply Message Receipt
D3 Done
T5 Publish aggregated QPA Response
T4 Aggregate both QPA Response
T3 Extract QPA Responses

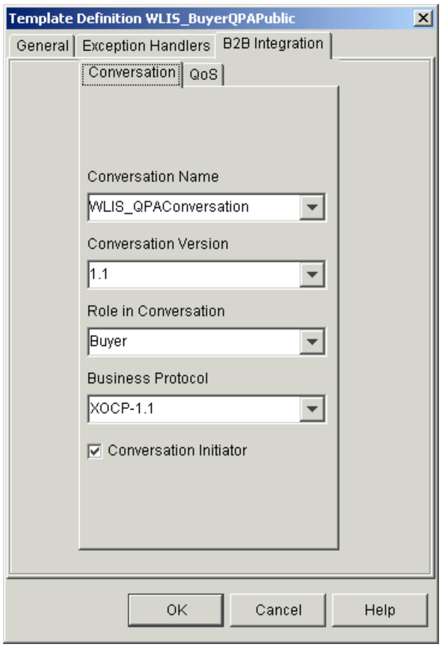
The following sections define the key tasks and events at the nodes in the `WLIS_BuyerQPAPublic` workflow template, shown in the preceding figure:

- Start
- Compose QPA Request Business Message
- Send QPA Request to Suppliers
- 1st/2nd QPA Reply Message Receipt
- Extract QPA Responses
- Aggregate both QPA Responses
- Publish Aggregated QPA Response
- Done

Note: You must define conversation properties for workflows involved in B2B conversations, that is, for the *public* workflows, which are also called *collaborative* workflows in the WebLogic Integration environment. The `WLIS_BuyerQPAPublic` workflow is an example of a collaborative workflow.

To see the conversation properties for the `WLIS_BuyerQPAPublic` workflow template, right click the template name in the left pane of the Studio, and select Properties from the drop-down menu. The template definition dialog box shown in the following figure is displayed.

Figure 3-12 Template Definition for the WLIS_BuyerQPAPublic Workflow


Template Definition WLIS_BuyerQPAPublic

Conversation Name: WLIS_QPAConversation

Conversation Version: 1.1

Role in Conversation: Buyer

Business Protocol: XOCP-1.1

☑ Conversation Initiator

The name, version, role, and business protocol binding for the conversation are defined on the B2B Integration tab. Note also that the `WLIS_BuyerQPAPublic` workflow is specified as the Conversation Initiator on this tab.

Start

Starts the QPA conversation. Note that the `WLIS_BuyerQPAPublic` workflow is specified as the Conversation Initiator in the Template Definition dialog box shown in the previous figure. This workflow is started by the buyer's private workflow (`WLIS_BuyerQPAPrivate`), and is therefore defined as a called workflow.

The workflow variables assigned at this node include those that identify the suppliers to which the QPA business message is to be sent: `FirstSupplierName` and `SecondSupplierName`. (See the Send QPA Request to Suppliers node in this workflow.)

Values are assigned to these variables using an XPath expression that gets information from the XML messages passed to this workflow: `QPARequestXMLOne` and `QPARequestXMLTwo`. For example, the following XPath expression assigns a value to `QPARequestXMLOne`:

```
XPath("/QPARequest/@SupplierName/text()",
$QPARequestXMLOne)
```

Compose QPA Request Business Message

This task node composes XOCP business messages by packing XML documents into the XOCP message payload. The XML is created by the buyer's private workflow (`WLIS_BuyerQPAPrivate`) and passed to this workflow in the following variables: `QPARequestXMLOne` and `QPARequestXMLTwo`.

To compose such a business message in the Studio, you must choose the Compose Business Message action, a B2B action provided by the B2B integration plug-in. Access the Compose Business Message dialog box as follows:

1. Double-click this node to display the Task Properties dialog box.

2. Choose Actions→Activated.

3. Choose Add→Integration Actions→B2B Integration→Compose Business Message to display the Compose Business Message dialog box.

Two business messages are composed at this node: `QPARequestMessageOne` and `QPARequestMessageTwo`. They are stored in variables of type Java Object.

Send QPA Request to Suppliers

Actions defined at this task node send an XOCP message to each of two suppliers through the WLIS_Hub trading partner.

The sample application uses a router expression on this task node. The WLIS_Hub routes the QPA Request XOCP message to a supplier, based on the Router Expression Type (in this case, Trading Partner Name).

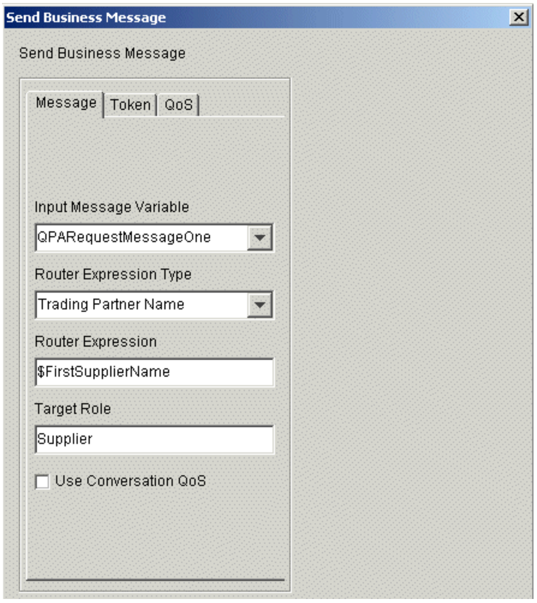
To see the Router Expression defined for this node, invoke the Send Business Message dialog box as follows:

1. Double-click this node to display the Task Properties dialog box.

2. Choose Actions→Activated, and double-click Send Business Message.

The dialog box shown in the following figure is displayed.

Figure 3-13 Send Business Message Dialog Box



Send Business Message

Send Business Message

Message | Token | QoS

Input Message Variable: QPARequestMessageOne

Router Expression Type: Trading Partner Name

Router Expression: $FirstSupplierName

Target Role: Supplier

☐ Use Conversation QoS

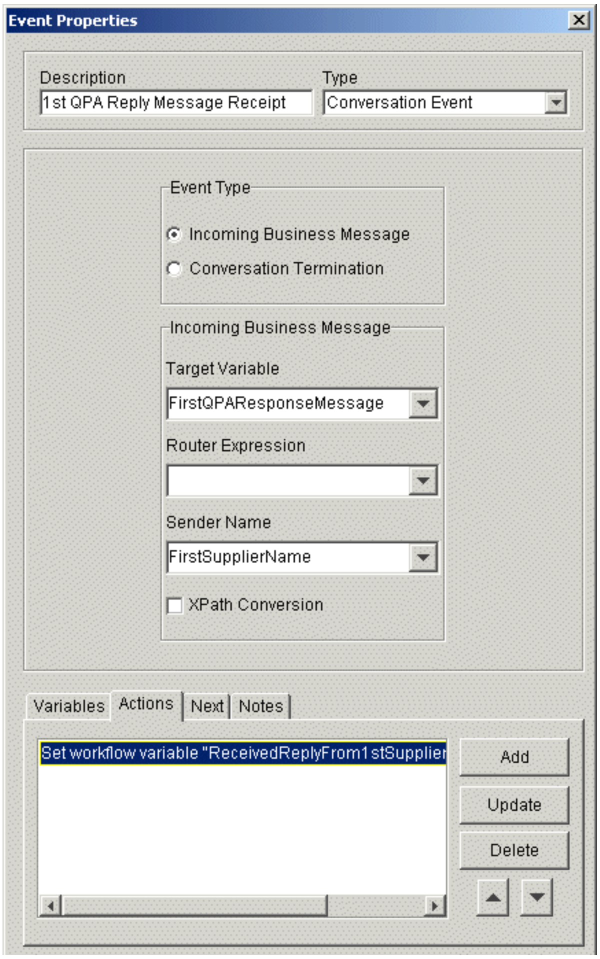
In the preceding figure, note that the value selected for Router Expression Type is Trading Partner Name, and that the value for the Router Expression is contained in the `FirstSupplierName` variable. The `FirstSupplierName` and `SecondSupplierName` variables are assigned at the Start node for this workflow.

1st/2nd QPA Reply Message Receipt

These are event nodes, at which the workflow waits for replies from the suppliers. Because there are two suppliers, two event nodes are defined. The order of receipt of reply messages is not defined; that is, an incoming business message is assigned to a target variable. The supplier’s name is subsequently derived from the business message XML.

Note that both of these event nodes are of type *Conversation Event*. Double-click one of these nodes to see the Event Properties dialog box displayed in the following figure.

Figure 3-14 B2B Conversation: Event Node Properties


Event Properties

Description: 1st QPA Reply Message Receipt

Type: Conversation Event

Event Type

[✓ Incoming Business Message]
Conversation Termination

Incoming Business Message

Target Variable: FirstQPAResponseMessage

Router Expression:

Sender Name: FirstSupplierName

☐ XPath Conversion

Set workflow variable "ReceivedReplyFrom1stSupplier

To distinguish between event nodes that wait for business messages from other trading partners in a B2B conversation and other event nodes (for example, those waiting for an internal XML event), specify the former event nodes as Conversation Event type. The Event Properties dialog box in the preceding figure defines the properties for a node that waits to receive the first QPA Reply Message.

Extract QPA Responses

After receiving replies from both suppliers, this task node unpacks the XOCP messages (in the `FirstQPAResponseMessage` and `FirstQPAResponseMessage` variables). It then extracts the QPA response XML documents into the following variables: `FirstQPAResponseXML` and `SecondQPAResponseXML`.

Aggregate both QPA Responses

This task node defines a Set Workflow Variable action by composing an XML Structure. The action aggregates the XML response documents into a single QPA response XML document. The XML document is stored in the `AggregatedQPAReply` variable.

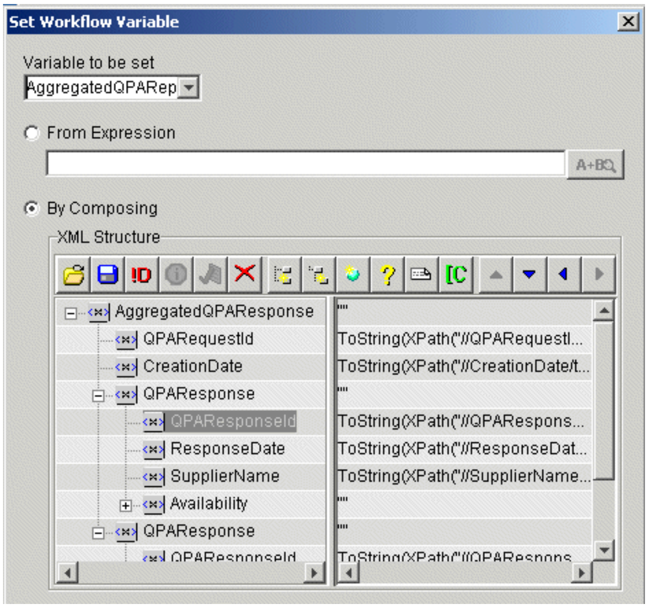
To view the XML workflow variable created for this sample in the Set Workflow Variable dialog box:

1. Double-click the node.

2. Choose Actions→Activated.

3. Double-click the Set Workflow Variable action defined for this node (`Set workflow variable` “`AggregatedQPAReply`” `XML structure`) to invoke the Set Workflow Variable dialog box shown in the following figure.

Figure 3-15 Set Workflow Variable Dialog Box


Set Workflow Variable

Variable to be set: AggregatedQPARep

From Expression

[✓ By Composing]

XML Structure

AggregatedQPAResponse	""
QPARequestId	ToString(XPath("//QPARequestI...
CreationDate	ToString(XPath("//CreationDate/t...
QPAResponse	""
QPAResponseId	ToString(XPath("//QPARespons...
ResponseDate	ToString(XPath("//ResponseDat...
SupplierName	ToString(XPath("//SupplierName...
Availability	""
QPAResponse	""
QPAResponseId	ToString(XPath("//QPARespons

In the preceding figure, note that the `AggregatedQPAResponse` XML is composed of the following elements: `QPARequestId`, `Creation Date`, and the `QPA Response` from each supplier.

Publish Aggregated QPA Response

This task node defines a Post XML Event action. (You can define such an event for a node by double-clicking the node to display the Task Properties dialog box, then choosing Add→Integration Actions→Post XML Event.) This action posts the aggregated QPA response (the `AggregatedQPAReply` variable defined at the previous node) as an *internal* XML event.

An internal XML event is an event internal to BPM: it triggers an event in the current workflow or another one, or it starts another workflow for which the Start node is triggered by an event. Because the destination is internal, the XML message is sent to an internal JMS queue maintained by BPM. The JNDI name of the internal JMS queue is `com.bea.wlpi.EventQueue`.

This aggregated QPA response message is called an *Addressed Message,* indicating that the message should persist for a particular workflow instance. In this case, the addressing is achieved through the instance ID, `QPAPrivateFlowId`. This instance ID specifies the instance of the private workflow that calls this public workflow. In this case, the `QPAPrivateFlowId` instance ID is first specified by the private workflow that calls this public workflow (`WLIS_BuyerQPAPrivate`), as described in "Call BuyerQPAPublic workflow" on page 3-34. In this way, various instances of caller and called workflows are synchronized.

Done

Terminates the QPA conversation successfully. Other Done nodes in the workflow terminate the conversation unsuccessfully if the business messages are not sent to both suppliers from the Send QPA Request to Suppliers node.

Supplier-Side Implementation

To implement this solution, the suppliers implement private workflows to manage the back-end processes, and a public workflow that choreographs the exchange of messages in the QPA conversation. This section describes the following workflows:

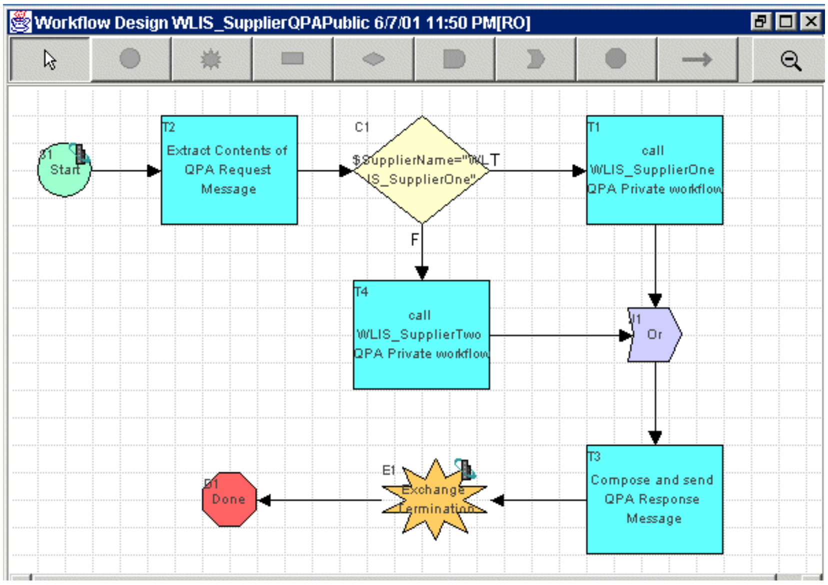
- Supplier QPA Public Workflow
- Supplier QPA Private Workflow

Supplier QPA Public Workflow

The suppliers' QPA public workflow (`WLIS_SupplierQPAPublic`) is initiated by the incoming message from the WLIS_Hub trading partner.

The following figure shows the `WLIS_SupplierQPAPublic` workflow template.

Figure 3-16 WLIS_SupplierQPAPublic Workflow Template


Workflow Design WLIS_SupplierQPAPublic 6/7/01 11:50 PM[RO]
S1 Start
T2 Extract Contents of QPA Request Message
C1 $SupplierName="WLIS_SupplierOne"
T
F
T1 call WLIS_SupplierOne QPA Private workflow
T4 call WLIS_SupplierTwo QPA Private workflow
J1 Or
T3 Compose and send QPA Response Message
E1 Exchange Termination
D1 Done

The following sections define the key tasks and events at the nodes in the `WLIS_SupplierQPAPublic` workflow template, shown in the preceding figure:

- Start
- Extract Contents of QPA Request Message
- Call WLIS_SupplierOneQPA Private Workflow
- Or
- Call WLIS_SupplierTwoQPA Private Workflow
- Compose and Send QPA Response Message
- Exchange Termination

Start

The `WLIS_SupplierQPAPublic` workflow is started by an event, that is, the receipt of the QPA Request business message.

Extract Contents of QPA Request Message

The first Extract Business Message Parts action extracts the XML message from the incoming business message (from the `QPARequestMessage` variable) and assigns the XML to the `QPARequestXML` variable.

Another action defined at this node sets a workflow variable that, in turn, defines the supplier's name for the QPA request message. The suppliers in this sample are collocated on the same instance of WebLogic Integration, they have identical roles in the QPA conversation, and they share the same public workflow template. The WLIS_Buyer trading partner sends two XOCP messages (one for each supplier) via the WLIS_Hub trading partner. This action sets a variable (`SupplierName`) by extracting the unique attribute value in the incoming QPA request XML document, using the following XPath expression:

```
ToString(XPath("/QPARequest/@SupplierName",
$QPARequestXML))
```

Call WLIS_SupplierOneQPA Private Workflow

Based on the value of the `SupplierName` variable, which was set at the previous node in this workflow, the instance of this public workflow starts the appropriate supplier private workflow (in this case, `WLIS_SupplierOneQPAPrivate`). Note that this action starts a private workflow; it is different from the Start Public Workflow action described (in "Call BuyerQPAPublic workflow" on page 3-34) as part of the `WLIS_BuyerQPAPrivate` workflow.

You can define the Start Workflow action as follows:

1. Double-click the task node to display the Task Properties dialog box.

2.ChooseActions→Activated→Add→WorkflowActions→StartWorkflowto display the Start Workflow dialog box.

3. Select the name of the workflow to call and define other parameters.

To view the Start Workflow action defined for this node in this workflow:

1. Double-click the node.

2. Choose Actions→Activated.

3. Double-click the Start Workflow action defined for this node (`Start workflow` "`WLIS_SupplierOneQPAPrivate`") to invoke the Set Workflow Variable dialog box.

For this node, the Start Workflow dialog box allows you to define the `WLIS_SupplierOneQPAPrivate` workflow in the Workflow to Start field. The Start Workflow dialog box also contains a Results tab, which, in turn, contains a Variable field and a Result field. In this case, `QPAResponseXML` is specified in the Variable field; `QPAReplyXML`, in the Result field. The Results tab displays variables that are defined as output parameters in the called workflow. An output parameter is used by a calling workflow to receive values from a called workflow.

In this case, the Results tab assigns the `QPAResponseXML` variable to the `QPAReplyXML` Result. The Result value is assigned to the `QPAReplyXML` output parameter at the Create QPA Reply XML node in the `WLIS_SupplierOneQPAPrivate` workflow (see “Supplier QPA Private Workflow” on page 3-52).

After this assignment is made, the caller workflow, `WLIS_SupplierQPAPublic` in this case, proceeds to the next task. In short, the call from this node to the private workflow is a synchronous call.

Or

A join node. The called workflows (`WLIS_SupplierOneQPAPrivate` and `WLIS_SupplierTwoQPAPrivate`) are invoked synchronously. As a result, the caller workflow (`WLIS_SupplierQPAPublic`) does not proceed until it gets results from one of the private workflows.

Call WLIS_SupplierTwoQPA Private Workflow

Based on the value of the `SupplierName` variable, which is set in the Extract Contents of QPA Request Message node in this workflow, this instance of this public workflow starts the appropriate supplier private workflow (in this case, `WLIS_SupplierTwoQPAPrivate`).

The tasks performed at this node are the same as those performed at the Call WLIS_SupplierOneQPA Private Workflow (described earlier), except that the `WLIS_SupplierTwoQPAPrivate` workflow is called at this node.

Compose and Send QPA Response Message

The first action at this node composes the response message from the supplier. It takes the `QPAResponseXML` variable from the private workflow and builds the response message into the `QPAReplyMessage` output variable.

The Send Business Message action sends the business message, which is assigned to the `QPAReplyMessage` variable, to the WLIS_Buyer trading partner via the WLIS_Hub. WLIS_Hub routes this QPA Response XOCP message to the WLIS_Buyer trading partner, based on the Router Expression Type (in this case, Trading Partner Name). To see the router expression as defined for this node, you can invoke the Send Business Message dialog box as follows:

1. Double-click this node to display the Task Properties dialog box.

2. Choose Actions→Activated, and double-click Send Business Message to display the Send Business Message dialog box.

The dialog box contains three tabs: Message, Token, and QoS. Note the following parameters on the Message tab.

Field	**Value**	**Description**
Input Message Variable	`QPAReplyMessage`	A Java object variable that contains the XOCP message to be sent to the WLIS_Buyer trading partner.
Router Expression Type	`Trading Partner Name`	WLIS_Hub routes this QPA Response message to the WLIS_Buyer trading partner, based on the trading partner name.
Router Expression	`$QPASenderName`	The trading partner that should receive this QPA Response message is identified by the value of the `QPASenderName` variable.
Target Role	`Buyer`	Defines the role in the conversation for the trading partner who receives the message.
Use Conversation QoS	Is not selected.	This option is available because the workflow template is configured with the XOCP protocol. It allows you to specify whether to use the Quality of Service defined at the conversation level or at this Send Business Message action level. In the sample, the process engine uses the QoS information defined at this Send Business Message action level.

Exchange Termination

An event node of type Conversation Event. This node terminates the QPA conversation.

Supplier QPA Private Workflow

There are two workflow templates in this category, one for each supplier:

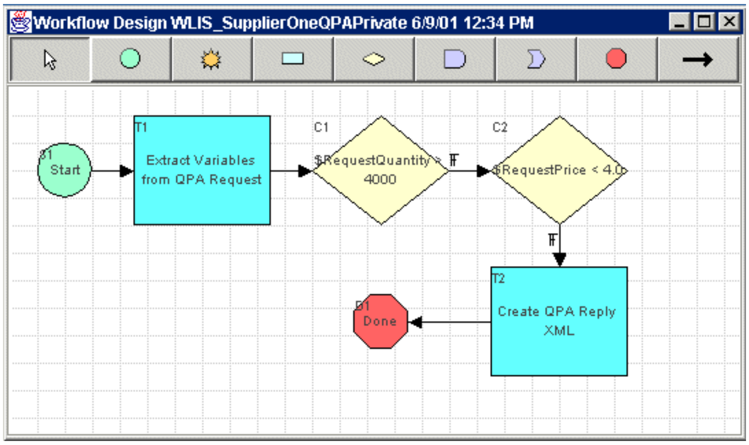
- `WLIS_SupplierOneQPAPrivate`
- `WLIS_SupplierTwoQPAPrivate`

The templates differ only slightly from each other. We leave it as an exercise for readers to determine the difference.

The suppliers' QPA private workflows are invoked from the suppliers' QPA public workflow (`WLIS_SupplierQPAPublic`).

The following figure shows the `WLIS_SupplierOneQPAPrivate` workflow template.

Figure 3-17 WLIS_SupplierOneQPAPrivate Workflow Template


Workflow Design WLIS_SupplierOneQPAPrivate 6/9/01 12:34 PM
S1
Start
T1
Extract Variables from QPA Request
C1
$RequestQuantity < 4000
F
C2
$RequestPrice < 4.0
F
T2
Create QPA Reply XML
D1
Done

The following sections define the key tasks and events at the nodes in the `WLIS_SupplierOneQPAPrivate` workflow template shown in the preceding figure:

- Start
- Extract Variables from QPA Request
- Decision Nodes
- Create QPA Reply XML

Start

The workflow is a called workflow, that is, it is started from the suppliers' public workflow (`WLIS_SupplierQPAPublic`). A variable called `selfName` is set at this node. In this case, the value of `selfName` is `WLIS_SupplierOne`.

Extract Variables from QPA Request

Several Set Workflow Variable actions are defined at this node. These actions use XPath expressions to extract the quantity, unit price, and required date of delivery from the incoming QPA Request XML document, and to assign the data to workflow variables.

For example, the unit price is extracted from the incoming XML document, and assigned to the `RequestPrice` variable, using the following XPath expression:

```
ToString(XPath("//Availability/UnitPrice/text()",
$QPARequestXML))
```

Decision Nodes

The workflow contains two decision nodes, which evaluate quantity and price conditions:

RequestQuantity

The content of the `RequestQuantity` variable is evaluated and a return variable (`ReplyQuantity`) is set, depending on the result of the evaluation.

RequestPrice

The content of the `RequestPrice` variable is evaluated and a return variable (`ReplyPrice`) is set, depending on the result of the evaluation.

Examine Figure 2-2, “Send QPA Request Window,” and Figure 2-5, “Create PO Window,” to compare the quantity and price per unit that were requested by the buyer in this sample scenario, with the quotes the buyer subsequently received from the suppliers.

Create QPA Reply XML

This task node creates the QPA Response XML document using the XML editor. The document is assigned to the return workflow variable (`QPAReplyXML`) of the public workflow (`WLIS_SupplierQPAPublic`) that invoked this private workflow.

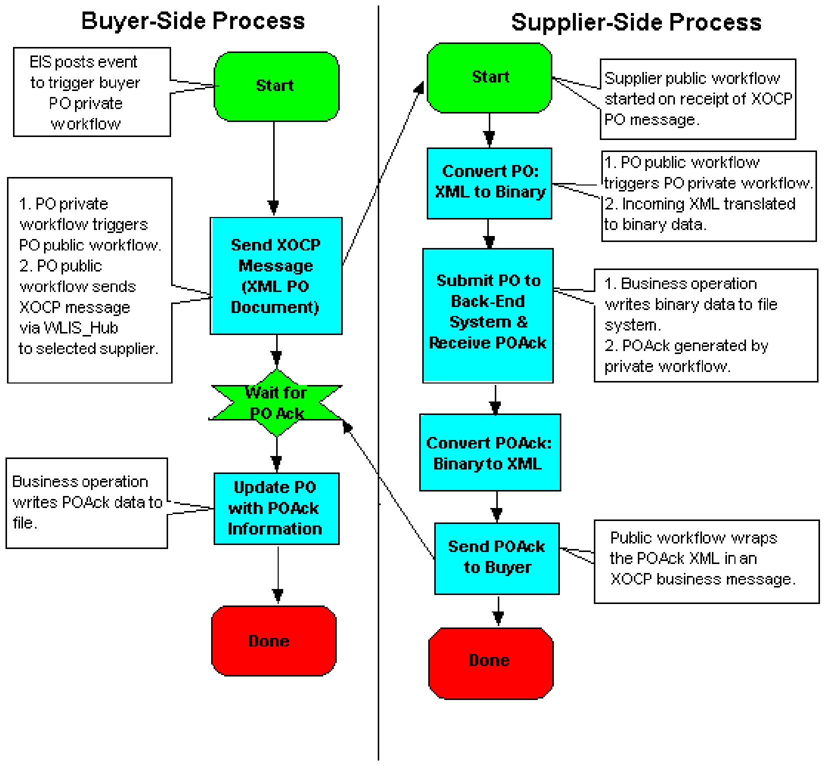
The QPA Response XML document is defined by the `QPAResponse.dtd` DTD. See Appendix A, "DTDs," for the QPA Response DTD.

After the `QPAReplyXML` variable is set by this workflow, the public workflow that called it can proceed to its next task. This sequence of `WLIS_SupplierQPAPublic` public workflow tasks is described in "Call WLIS_SupplierOneQPA Private Workflow" on page 3-49.

PO Business Process

Assume that you are the buyer in this sample scenario. After reviewing the QPA responses from the suppliers, select one of the suppliers and start the PO business process, as described in "Step 6. Create the Purchase Order" on page 2-11. The following diagram illustrates the flow of events for the PO business process.

Figure 3-18 Process Flow in the PO Business Process


Buyer-Side Process
Supplier-Side Process
EIS posts event to trigger buyer PO private workflow
Start
1. PO private workflow triggers PO public workflow.
2. PO public workflow sends XOCP message via WLIS_Hub to selected supplier.
Send XOCP Message (XML PO Document)
Wait for PO Ack
Business operation writes POAck data to file.
Update PO with POAck Information
Done
Start
Supplier public workflow started on receipt of XOCP PO message.
Convert PO: XML to Binary
1. PO public workflow triggers PO private workflow.
2. Incoming XML translated to binary data.
Submit PO to Back-End System & Receive POAck
1. Business operation writes binary data to file system.
2. POAck generated by private workflow.
Convert POAck: Binary to XML
Send POAck to Buyer
Public workflow wraps the POAck XML in an XOCP business message.
Done

The following sequence summarizes the events illustrated in the preceding figure:

1. The buyer composes a purchase order (PO) document.
2. The buyer sends the PO document to the selected supplier.
3. The supplier receives the PO and converts the XML-based PO document into a binary data file. (In this sample we assume the supplier's order management system expects to receive a binary file to process the order).
4. The supplier generates an XML-based PO acknowledgment document, based on another binary data file.
5. The supplier sends the PO acknowledgment document to the buyer.
6. The buyer updates the PO information, based on the PO acknowledgment.

Note: The preceding figure shows a high-level view of the PO business process. Each side of the process is implemented by a public and a private workflow. The following sections describe these workflows:

- Overview of the PO Implementation
- Buyer-Side Implementation
- Supplier-Side Implementation

Overview of the PO Implementation

In this scenario, each trading partner implements a private workflow and a public workflow for the PO process. The following workflow templates are used in the sample.

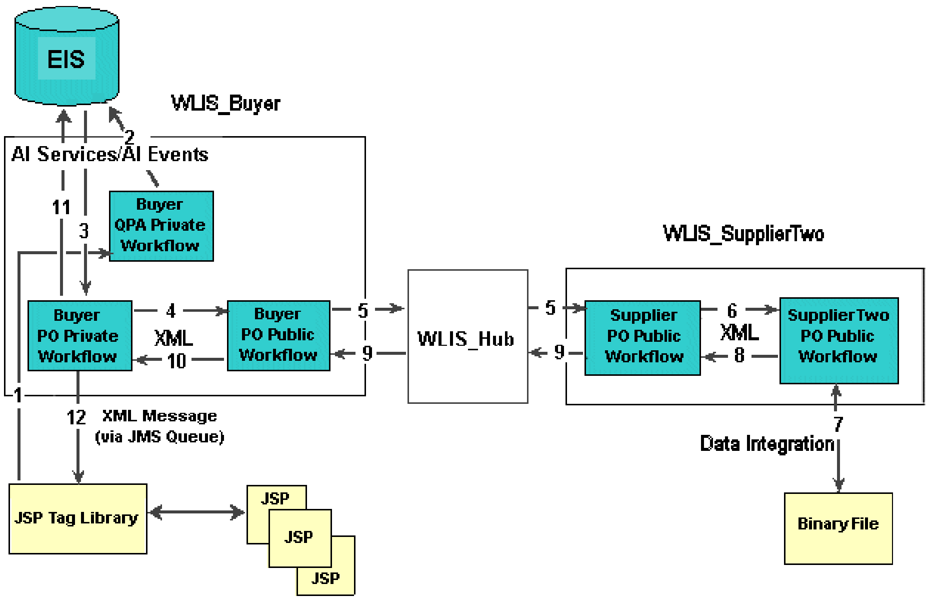
Table 3-6 Workflows for the Sample PO Process

Role	Public/Private	Workflow Name
Buyer	Private	`WLIS_BuyerQPAPrivate`
Buyer	Private	`WLIS_BuyerPOPrivate`
Buyer	Public	`WLIS_BuyerPOPublic`
Supplier	Public	`WLIS_SupplierPOPublic` **Note:** Both suppliers in the scenario use the same public workflow.
Supplier	Private	`WLIS_SupplierOnePOPrivate`
Supplier	Private	`WLIS_SupplierTwoPOPrivate`

The PO implementation for this sample requires WebLogic Integration support for application integration, data integration, and management of business processes. This section describes the PO workflows, including their integration with back-end applications and heterogeneous data formats. See “Application and Data Integration” on page 3-77 for more information about application integration and data integration functionality as it applies to the PO business process in this scenario.

The following figure illustrates the data flow among the trading partners involved in the PO business process.

Figure 3-19 Data Flow in the PO Business Process



For details about the buyer-side and supplier-side implementations, see “Buyer-Side Implementation” on page 3-59 and “Supplier-Side Implementation” on page 3-47.

The following sequence of events summarizes the data flow among trading partners, workflows, and back-end systems:

1. The PO business process is started when you, as the buyer in this scenario, select a supplier and create a purchase order, as described in “Step 6. Create the Purchase Order” on page 2-11.

   A JSP extracts the PO request information, including the party ID, unit price, quantity, and delivery date, and puts it in an XML message. The XML message is posted to the BPM JMS queue.

2. The buyer’s private workflow (`WLIS_BuyerQPAPrivate`) that subscribes to the XML event is invoked by the event posting.

The `WLIS_BuyerQPAPrivate` workflow invokes the `insertPO` service on the `WLISAppView.sav` application view to insert the PO information into the Enterprise Information System (EIS). The EIS is simulated by an RDBMS in this sample. (The `WLISAppView.sav` application view, together with its services and events, is configured and deployed in WebLogic Integration when you set up the sample.)

3. The EIS sends an event containing the PO information to the WebLogic Integration process engine.

   The application view event triggers the start of the buyer's private workflow for the PO process (`WLIS_BuyerPOPrivate`).

4. The `WLIS_BuyerPOPrivate` workflow starts the buyer's public workflow (`WLIS_BuyerPOPublic`).

5. `WLIS_BuyerPOPublic` sends an XOCP message to the suppliers' public workflow (`WLIS_SupplierPOPublic`). This step initiates the PO conversation.

   The WLIS_Hub trading partner routes the message to the destination trading partner, WLIS_Buyer, based on a registered collaboration agreement between itself and WLIS_Buyer.

   Note: In this step, the WLIS_Hub trading partner changes roles and becomes the proxy supplier in the collaboration agreement between itself and the buyer.

6. On the supplier side, an instance of the suppliers' public workflow (`WLIS_SupplierPOPublic`) is started when the supplier receives the XOCP message.

   The `WLIS_SupplierPOPublic` workflow starts the selected supplier's private workflow (`WLIS_SupplierOnePOPrivate` or `WLIS_SupplierTwoPOPrivate`).

7. The selected supplier's private workflow:

   a. Translates the XML PO data it receives to binary data, using the data integration functionality of WebLogic Integration.

   b. Generates a PO acknowledgment in binary format.

   c. Translates the PO acknowledgment from binary format to XML.

8. The supplier's private workflow returns the PO acknowledgment XML data to the `WLIS_SupplierPOPublic` workflow.

9. The `WLIS_SupplierPOPublic` workflow wraps the PO acknowledgment information in an XOCP message and sends it to the buyer's public workflow (`WLIS_BuyerPOPublic`).

10. The `WLIS_BuyerPOPublic` workflow receives the XOCP message, extracts the XML content, and sends an XML event to the buyer's private workflow (`WLIS_BuyerPOPrivate`).

    This step marks the end of the PO conversation.

11. The `WLIS_BuyerPOPrivate` workflow uses the application integration framework provided by WebLogic Integration to update the PO information in the EIS, based on the data provided in the PO acknowledgment. The workflow also writes PO acknowledgment information to the `POAcknowledgement.xml` file.

12. A JSP reads the `POAcknowledgement.xml` file and displays the contents in the PO Confirmation window described in "Step 8. Check for Purchase Order Acknowledgment" on page 2-14.

    This step marks the end of the PO business process.

Buyer-Side Implementation

The buyer (GCS) in our sample scenario implements a Web user interface to drive the business transaction, a private workflow to manage the buyer's back-end processes, and a public workflow that choreographs the message exchange in the PO conversation. The WLIS_Buyer trading partner stores its PO information in an RDBMS.

Through the use of the application integration framework provided by WebLogic Integration, the workflows in the PO business process can integrate with the RDBMS. To support application integration, an application view (`WLISAppView.sav`) is deployed for this sample when you set up and configure the samples domain. (The sample domain setup is described in "Step 1A: Invoke the RunSamples Script" on page 2-4.)

This section discusses the following components of the buyer-side PO business process:

- Buyer-Side Web User Interface
- Buyer PO Private Workflow
- Buyer PO Public Workflow

Buyer-Side Web User Interface

The interaction between the Web browser and the buyer-side workflows for the buyer-side implementation is similar to that for the QPA business process—Java Server Pages (JSPs) and JSP tag libraries are used to initiate the PO process and display request and response data in your Web browser. The source files related to the PO business process can be found in the directories listed in the following table.

Table 3-7 Source Files for the PO Process

Directory in your WebLogic Integration Installation	Source Files
`\samples\wlis\src\examples\wlis\tags`	■ `CreatePOTag.java` ■ `CheckPOTag.java` ■ `CheckPOAckTag.java`
`\samples\wlis\web`	■ `CreatePO.jsp` ■ `CheckPO.jsp` ■ `CheckPOAck.jsp` ■ `WaitPOCreated.htm` ■ `WaitPOAck.htm`
`\samples\wlis\lib\xsl`	■ `ProcessPO.xsl` ■ `ProcessPOAck.xsl`

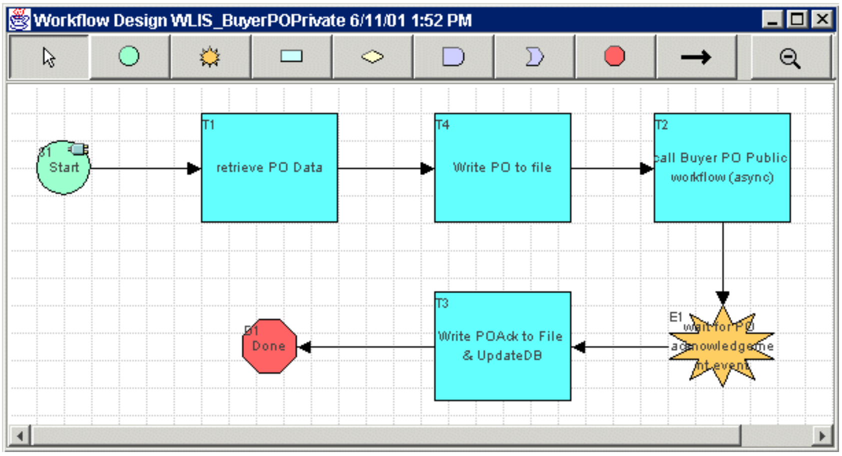
Buyer PO Private Workflow

The `WLIS_BuyerPOPrivate` workflow performs the following key tasks:

- Receives, from the process engine, the `insertPO` event containing PO information from the buyer's EIS. The `insertPO` event is an event defined for the `WLISAppView.sav` application view.
- Retrieves the PO information from the EIS.
- Wraps the PO information in an XOCP business message.
- Calls the buyer's public workflow (`WLIS_BuyerPOPublic`) which, in turn, sends PO information to the supplier's public workflow (`WLIS_SupplierPOPublic`), thus starting the PO conversation.
- Waits for the PO acknowledgment information from the supplier.
- Uses the application integration framework provided by WebLogic Integration to update the PO information in the EIS, based on the PO acknowledgment.
- Writes PO acknowledgment information to the `POAcknowledgement.xml` file.

The following figure shows the `WLIS_BuyerPOPrivate` workflow template.

Figure 3-20 WLIS_BuyerPOPrivate Workflow Template


Workflow Design WLIS_BuyerPOPrivate 6/11/01 1:52 PM
S1
Start
T1
retrieve PO Data
T4
Write PO to file
T2
call Buyer PO Public workflow (async)
E1
wait for PO acknowledgement event
T3
Write POAck to File & UpdateDB
D1
Done

The following sections define the key tasks and events at the nodes in the `WLIS_BuyerPOPrivate` workflow template, shown in the preceding figure:

- Start
- Retrieve PO Data
- Write PO to file
- Call Buyer PO Public workflow
- Wait for PO acknowledgement
- Write POAcktoFile & UpdateDB

Start

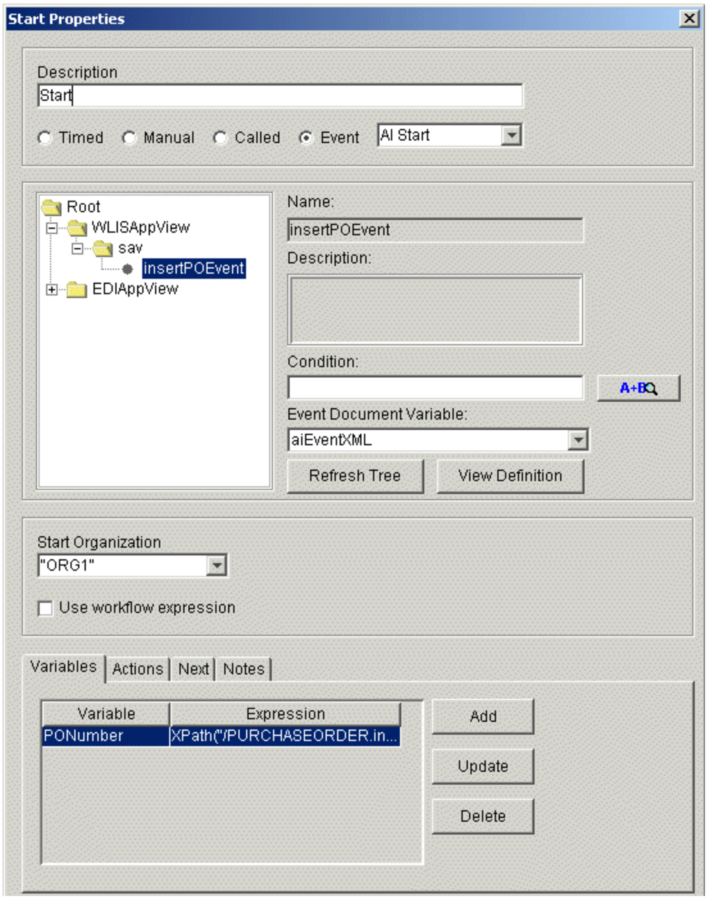
The final task in the buyer's QPA private workflow (`WLIS_BuyerQPAPrivate`), described in "Buyer QPA Private Workflow" on page 3-28, is to input PO information into the EIS system, using the `insertPO` service defined for the `WLISAppView.sav` application view, which is deployed for this sample application. (See "Application Integration" on page 3-77 for details about the application view.)

The EIS subsequently uses an application view event to post an event to the WebLogic Integration process engine. The event (`WLISAppView.sav.insertPOEvent`) triggers the start of the `WLIS_BuyerPOPrivate` workflow at this Start node.

The Start Properties dialog box for this node specifies that this workflow is started by an application integration event, namely the `insertPO` event defined for the `WLISAppView.sav` application view.

Double-click this Start node to view the Start Properties dialog box shown in the following figure.

Figure 3-21 Start Properties Dialog Box



Start Properties

Description: Start

( ) Timed ( ) Manual ( ) Called [✓ Event] AI Start

Root / WLISAppView / sav / insertPOEvent; EDIAppView

Name: insertPOEvent

Description:

Condition:

Event Document Variable: aiEventXML

Start Organization: "ORG1"

[ ] Use workflow expression

Variable	Expression
PONumber	XPath("/PURCHASEORDER.in...

In the preceding figure, note that the Event Document Variable field in the Start Properties dialog box is populated with the `aiEventXML` variable. The data format for the event is XML and the XML Schema language defines the event schema. When the start node receives data from the application view event, the data is stored in the `aiEventXML` variable.

Note: You can examine the XML schema of the event document by clicking View Definition in the Start Properties dialog box.

A workflow variable (`PONumber`) is set at this node, using the following XPath expression to get the PO number from the incoming PO data:

```
XPath("/PURCHASEORDER.insert/PONUMBER/text()",$aiEventXML)
```

Retrieve PO Data

This task node completes the following actions:

1. Composes workflow variables.

2. Calls application view services to retrieve PO information from the EIS.

This task node is set up to call application view services through the same procedure provided for the node called Lookup 2nd Tier suppliers for requested items, which is described in "Buyer QPA Private Workflow" on page 3-28.

See *Using Application Integration* for more information about using the application integration plug-in provided by WebLogic Integration.

Write PO to file

This action node uses the `xmlToFile` business operation to write the PO to a local file (*`BEA_HOME`*`/config/samples/data/PO.xml`). The local file is consumed by a JSP.

To see business operations defined for this sample, choose Configuration→Business Operations from the Studio task menu. The Business Operations dialog box is displayed. Double-click any business operation to see more information about it. You can find the Java class relevant to the `xmlToFile` business operation in the following location in your WebLogic Integration installation: `\samples\wlis\src\examples\wlis\common\util\Utils.java`.

Call Buyer PO Public workflow

This task node starts the buyer's public workflow (`WLIS_BuyerPOPublic`) asynchronously, and passes workflow variables containing the QPA Request XML documents to the public workflow. Because the `WLIS_BuyerPOPrivate` workflow is the initiator of a WebLogic Integration B2B conversation (`WLIS_POConversation`), this task node uses the special Start Public Workflow action (provided by the B2B integration plug-in), instead of the default Start Workflow action.

The key difference between the Start Public Workflow and the Start Workflow actions is that the workflow template binding is dynamic when you use the former. The `WLIS_BuyerPOPrivate` workflow determines which workflow template to invoke at run time, based on conversation properties defined in the Start Public Workflow action.

Also, based on the conversation information passed as part of the Start Public Workflow call, the B2B engine uses collaboration agreement information (stored in the WebLogic Integration repository) to associate the called workflow with other public workflows in the same collaboration agreement.

You can define a Start Public Workflow action in the WebLogic Integration Studio as follows:

1. Double-click the task node to invoke the Task Properties dialog box.

2. Choose Actions→Add→Integration Actions→B2B Integration→Start Public Workflow to display the Start Public Workflow dialog box.

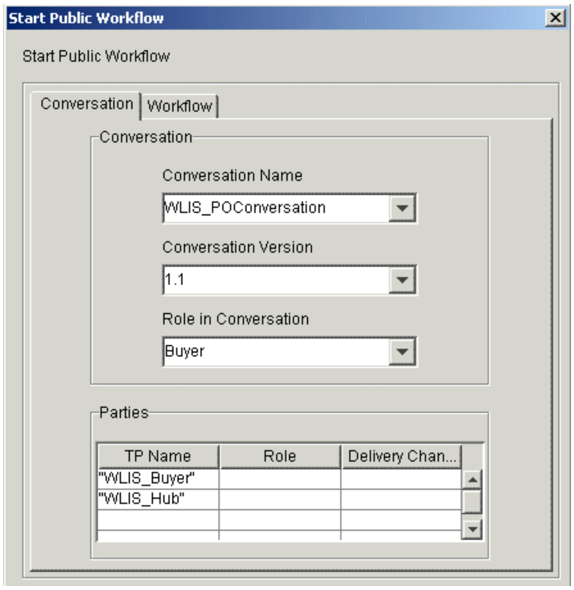
To view the Start Public Workflow properties already specified for this node:

1. Double-click the task node to invoke the Task Properties dialog box.

2. Choose Actions→Activated.

3. Double-click Start Public Workflow to display the Start Public Workflow dialog box shown in the following figure.

Figure 3-22 Start Public Workflow Dialog Box



Start Public Workflow

Start Public Workflow

Conversation | Workflow

Conversation

Conversation Name: WLIS_POConversation

Conversation Version: 1.1

Role in Conversation: Buyer

Parties

TP Name	Role	Delivery Chan...
"WLIS_Buyer"		
"WLIS_Hub"		

Note the following parameters for the Start Public Workflow action shown in the preceding figure:

- The Conversation tab—Information specified on this tab is used by WebLogic Integration to locate the active collaboration agreements in the repository that define an agreement between the specified parties, in the specified conversation:
  - The conversation name, version, and role are defined as `WLIS_POConversation`, `1.1`, and `buyer`, respectively.
  - The parties in the conversation are specified by the trading partner names: `WLIS_Buyer` and `WLIS_Hub`. You can optionally add values to the Role and Delivery Channel fields, which can then be used, in addition to the values in the Name fields, to locate the collaboration agreement of interest.
- The Workflow tab—Variables passed to the called workflow are defined on the Workflow tab. The variable names specified here are also specified as input variables to the called workflow (`WLIS_BuyerPOPublic`). They are:
  - `POxml`—Contains the value of the `purchaseOrderXML` variable that was defined at the Retrieve PO Data node (see "Retrieve PO Data" on page 3-64).

- `POPrivateFlowId`—Workflow instance ID, specifying the instance of the `WLIS_BuyerPOPrivate` workflow that calls the public workflow. (Several instances of the workflow template may be running.) See “Send PO Acknowledgement to PO Private Workflow” on page 3-70 to understand where this instance ID is used by the called public workflow.

Wait for PO acknowledgement

The workflow waits, at this event node, for a specific XML event from the `WLIS_BuyerPOPublic` workflow. This node uses a key value expression to specify the exact data that the incoming document must contain for the node to be triggered. The key value expression defined for this node is `PONumber`. The `PONumber` key value is set in the Start node for this `WLIS_BuyerPOPrivate` workflow. See “Start” on page 3-62.

If a number of instances of the buyer’s private workflow (`WLIS_BuyerPOPrivate`) are active, the appropriate Wait for PO Acknowledgement node, in the appropriate workflow template instance, needs to be triggered at this point in the PO process. The workflow instance that contains the node to be triggered is specified by the value of `PONumber`. (See the discussion about event keys in “Start” on page 3-30.)

Write POAcktoFile & UpdateDB

The actions performed at this task node include the following:

Perform business operation “xmlToFile”

This business operation writes the PO acknowledgment XML data to the following location in your WebLogic Integration installation:

```
/config/samples/data/POAcknowledgement.xml
```

Set Workflow variable “updatePOXML” XML structure

Composes an XML document and stores the content in an XML-type `updatePOXML` variable.

Call Application View Service

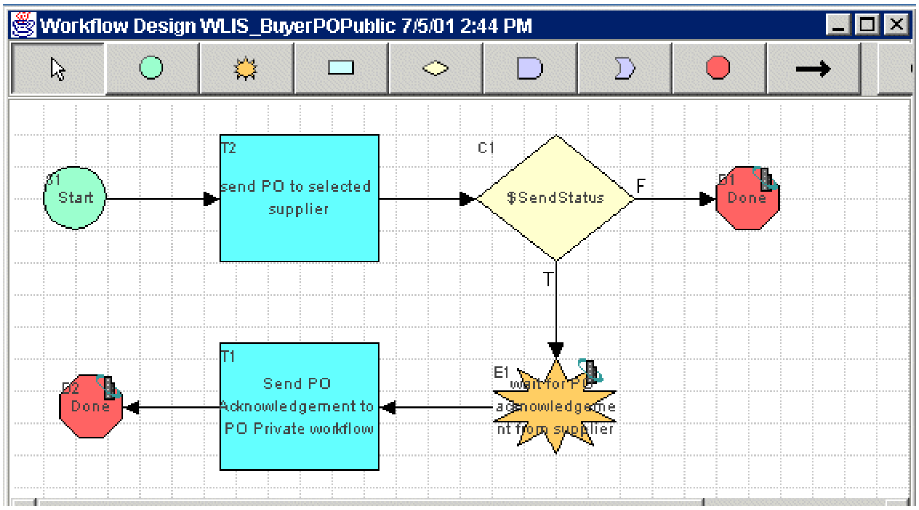
Calls the `updatePOData` service on the `WLISAppView.sav` application view to update the RDBMS with the PO data.

Buyer PO Public Workflow

The primary task of the `WLIS_BuyerPOPublic` workflow is to send and receive the XOCP business messages in the purchase order conversation (`WLIS_POConversation`).

The following figure shows the `WLIS_BuyerPOPublic` workflow template.

Figure 3-23 WLIS_BuyerPOPublic Workflow Template


Workflow Design WLIS_BuyerPOPublic 7/5/01 2:44 PM
S1 Start
T2 send PO to selected supplier
C1 $SendStatus
F
D1 Done
T
E1 wait for PO acknowledgement from supplier
T1 Send PO Acknowledgement to PO Private workflow
D2 Done

The following sections define the key tasks and events at the nodes in the `WLIS_BuyerPOPublic` workflow template, shown in the preceding figure:

- Start
- Send PO to Selected Supplier
- Wait for PO Acknowledgement from Supplier
- Send PO Acknowledgement to PO Private Workflow
- Done

Start

This workflow is started by the buyer's private workflow, `WLIS_BuyerPOPrivate`, as described in "Call Buyer PO Public workflow" on page 3-64. The `POxml` variable, specified in the Workflow tab of the Call Buyer PO Public workflow node, is an input variable for this workflow. A workflow variable (`SupplierName`) that contains the supplier's name is set at this node, using the following XPath expression:

```
ToString(XPath("//SupplierName/text()", $POxml))
```

Send PO to Selected Supplier

The first action defined at this node is Compose Business Message. This action composes the PO message to be sent from the buyer to the selected supplier. It takes the `POxml` variable from the private workflow and builds the response message into the `POXOCPMessage` output variable.

A Send Business Message action is also defined at this node. The PO message (`POXOCPMessage`) is routed through the WLIS_Hub trading partner. The WLIS_Hub trading partner routes the message based on the Router Expression Type (in this case, Trading Partner Name) defined at this node. To see the router expression defined for this node, you can invoke the Send Business Message dialog box as follows:

1. Double-click this node to display the Task Properties dialog box.

2. Choose Actions→Activated, and double-click Send Business Message to display the Send Business Message dialog box.

The dialog box contains three tabs: Message, Token, and QoS. Note the following parameters on the Message tab.

Field	**Value**	**Description**
Input Message Variable	`POXOCPMessage`	Java object variable that contains the XOCP message to be sent to the selected supplier.
Router Expression Type	`Trading Partner Name`	WLIS_Hub routes this PO message to the selected supplier trading partner, based on the trading partner name.
Router Expression	`$SupplierName`	The trading partner that should receive the PO message is identified by the value of the `SupplierName` variable. The `SupplierName` variable is set at the Start node for this workflow.
Target Role	`Supplier`	Defines the role in the conversation for the trading partner who receives the message.

Use Conversation QoS	Is not selected.	This option is available because the workflow template is configured with the XOCP protocol. It allows you to specify whether to use the Quality of Service defined at the conversation level or at this (Send Business Message action) level. In this case, the process engine uses the QoS information defined at the Send Business Message action level.

Wait for PO Acknowledgement from Supplier

This node is a conversation event node at which the public workflow waits for the returned PO acknowledgment XOCP message from the supplier. The acknowledgment message is assigned to a Java object variable called `POXOCPMsg`.

Send PO Acknowledgement to PO Private Workflow

The first action defined at this node (Extract Business Message Part) extracts the business message parts from the incoming PO Reply XOCP message from the selected supplier. It then assigns the XML content to the `POReplyXML` variable.

Another action at this node defines a Post XML Event action to post the PO Acknowledgment (the `POReplyXML` variable) to the PO private workflow as an *internal* XML event. In the sample scenario, this internal XML event triggers an event in another workflow. Because the destination is internal to BPM, the XML message is sent to an internal JMS queue maintained by BPM.

You can post an XML event action for a workflow node as follows:

1. Double-click the node to display the Task Properties dialog box.

2. Choose Add→Integration Actions→Post XML Event to display the Post XML Event dialog box.

To view the parameters already defined in the Post XML Event dialog box for this node:

1. Double-click the node to display the Task Properties dialog box.

2. Choose Actions→Activated, and double-click Post Internal XML Event to display the Post XML Event dialog box.

In this case, the XML message is contained in the `POReplyXML` variable. This PO acknowledgment message is referred to as an *Addressed Message,* indicating that the message should persist for a particular workflow instance. In the sample scenario, addressing is achieved through the instance ID, `POPrivateFlowId`. This instance ID specifies the instance of the private workflow that called this public workflow. The `POPrivateFlowId` instance ID is specified by the WLIS_BuyerPOPrivate workflow, that is, by the private workflow that called this public workflow (see "Call Buyer PO Public workflow" on page 3-64). In this way, various instances of caller and called workflows are synchronized.

Done

Terminates the PO conversation successfully. Another Done node in the workflow terminates the conversation unsuccessfully if the workflow fails to send the purchase order to any supplier.

Supplier-Side Implementation

In this scenario, each supplier implements a private workflow to integrate its back-end processes, and a public workflow to choreograph the exchange of messages in the PO conversation. This section describes the following workflows:

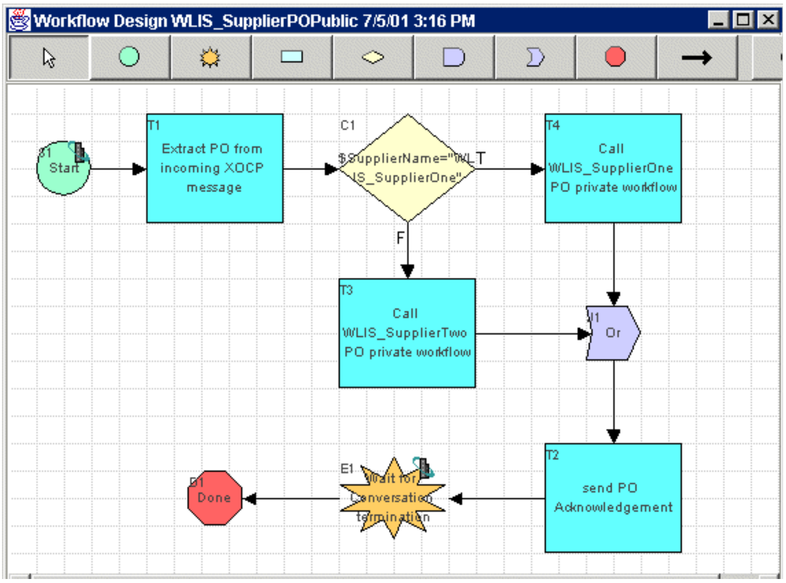
- Supplier PO Public Workflow
- Supplier PO Private Workflow

Supplier PO Public Workflow

The PO public workflow (`WLIS_SupplierPOPublic`) is started on receipt of an XOCP business message from the buyer's PO public workflow (`WLIS_BuyerPOPrivate`).

The following figure shows the `WLIS_SupplierPOPublic` workflow template.

Figure 3-24 WLIS_SupplierPOPublic Workflow Template


Workflow Design WLIS_SupplierPOPublic 7/5/01 3:16 PM
S1 Start
T1 Extract PO from incoming XOCP message
C1 $SupplierName="WLIS_SupplierOne"
T
F
T4 Call WLIS_SupplierOne PO private workflow
T3 Call WLIS_SupplierTwo PO private workflow
J1 Or
T2 send PO Acknowledgement
E1 Wait for Conversation termination
D1 Done

The following sections define the key tasks and events at the nodes in the `WLIS_SupplierPOPublic` workflow template, as shown in the preceding figure:

- Start
- Extract PO from incoming XOCP message
- Call WLIS_SupplierOne PO private workflow
- Send PO Acknowledgement
- Wait for Conversation Termination
- Done

Start

This workflow is started when the supplier receives an XOCP business message from the buyer's PO public workflow (`WLIS_BuyerPOPublic`). This message is one of several messages exchanged during the PO conversation (`WLIS_POConversation`).

Extract PO from incoming XOCP message

In the first action at this node, message parts are extracted from the XOCP message received from the `WLIS_BuyerPOPublic` workflow. The XML is assigned to the `PODataXML` variable.

Next, a workflow variable expression for `SupplierName` is set. The name of the supplier is derived from the incoming PO XML and is assigned to `SupplierName`, using the following XPath expression:

```
ToString(XPath("/PurchaseOrder/SupplierInformation/Supplier
Name/text()", $PODataXML))
```

Call WLIS_SupplierOne PO private workflow

Based on the unique attribute value (`SupplierName`) extracted from the incoming PO XML document, which specifies the supplier's name, the workflow proceeds to the appropriate task node and starts the PO private workflow for either WLIS_SupplierOne or WLIS_SupplierTwo.

Note that the action in the Task Properties dialog box for both nodes is the Start Workflow action. (From the Task Properties dialog box, choose Add→Workflow Actions→Start Workflow.) When you add a Start Workflow action, you select the name of the workflow to call.

At this node, for example, the `WLIS_SupplierOnePOPrivate` workflow is selected in the Start Workflow dialog box. Additional information in the Start Workflow dialog box for this action includes variables specified on the Results tab—this tab displays variables that are defined as output parameters in the called workflow. An output parameter is used by a calling workflow to receive values from a called workflow.

In this case, a result is assigned to the `POAck_XML` output parameter at the penultimate node in the `WLIS_SupplierOnePOPrivate` workflow (described in "Get PO Ack and transform into XML" on page 3-76). Once this assignment is made, the caller workflow proceeds to the next task. This call from the `WLIS_SupplierPOPublic` workflow to the private workflow (`WLIS_SupplierOnePOPrivate` or `WLIS_SupplierTwoPOPrivate`) is therefore a synchronous call.

Send PO Acknowledgement

This node gets the return PO acknowledgment from the supplier's private workflow, wraps it in an XOCP message, and sends a PO acknowledgment business message to the WLIS_Buyer trading partner. The business message is routed through the WLIS_Hub, based on the value of the router expression input (in this case, the name of the trading partner) defined at this node. To see the router expression defined for the Send PO Acknowledgement node, invoke the Send Business Message dialog box as follows:

1. Double-click this node to display the Task Properties dialog box.

2. Choose Actions→Activated, and double-click Send Business Message to display the Send Business Message dialog box.

The dialog box contains three tabs: Message, Token, and QoS. Note the following parameters on the Message tab.

Field	**Value**	**Description**
Input Message Variable	`POreplyXOCPMsg`	Java object variable that contains the XOCP message to be sent to the buyer trading partner.
Router Expression Type	`Trading Partner Name`	WLIS_Hub routes this PO message to the buyer trading partner, based on the trading partner name.
Router Expression	`$POBuyerName`	The trading partner that should receive the PO reply message is identified by the value of the `POBuyerName` variable. The `POBuyerName` variable is set at the Start node for this workflow.
Target Role	`Buyer`	Defines the role in the conversation for the trading partner who receives the message.

Wait for Conversation Termination

The system waits for the conversation to end. The workflow that initiates the PO conversation (`WLIS_BuyerPOPublic`) also terminates it. See “Buyer PO Public Workflow” on page 3-67.

Done

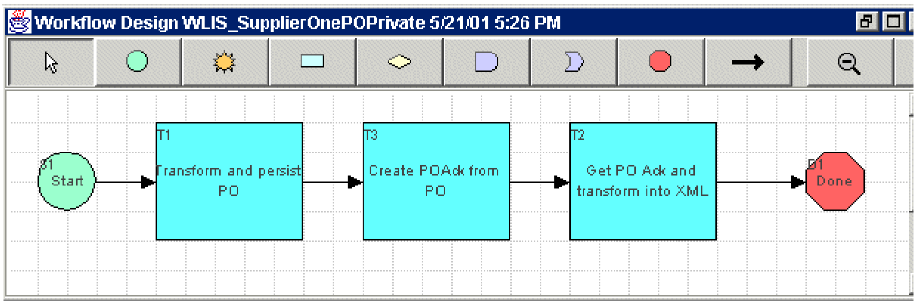
At this node, the workflow finishes and exits the conversation.

Supplier PO Private Workflow

The sample application defines the following private workflow templates for the suppliers in the PO process: `WLIS_SupplierOnePOPrivate` and `WLIS_SupplierTwoPOPrivate`. The templates are similar.

The following figure shows the `WLIS_SupplierOnePOPrivate` workflow template.

Figure 3-25 WLIS_SupplierOnePOPrivate Workflow Template


Workflow Design WLIS_SupplierOnePOPrivate 5/21/01 5:26 PM
S1
Start
T1
Transform and persist PO
T3
Create POAck from PO
T2
Get PO Ack and transform into XML
D1
Done

The following sections define the key tasks and events at the nodes in the `WLIS_SupplierOnePOPrivate` workflow template, shown in the preceding figure:

- Start
- Transform and persist PO
- Create POAck from PO
- Get PO Ack and transform into XML

Start

This private workflow is started at this node when it is called from the suppliers' PO public workflow (`WLIS_SupplierPOPublic`).

Transform and persist PO

WLIS_SupplierOne maintains its purchase order information as binary data, but the PO is sent from the buyer in XML format. At this node the workflow uses the data integration functionality provided by WebLogic Integration to translate the incoming XML data to binary data, which can be understood by the supplier's systems. This translation is accomplished through an action in the Studio which, in turn, is provided by the data integration plug-in.

You can define a data integration action in the Studio as follows:

1. Double-click the task node to display the Task Properties dialog box.

2. Choose Actions→Add→Integration Actions→Data integration.

3. Select one of the following options: Translate Binary to XML or Translate XML to Binary.

In this case, the data integration action is Translate XML to Binary.

A business operation (`binary to file`) is also defined at this node to write the translated binary data to the file system, simulating a back-end ERP system for this scenario. The Java class for the business operation is `examples.wlis.common.util.Utils`.

Create POAck from PO

A business operation (`create POAck from PO`) is defined at this node to create a PO acknowledgment message in binary format, based on the binary PO data. The Java class for the business operation is `examples.wlis.common.util.Utils`.

Get PO Ack and transform into XML

The first action at this node is Perform business operation. A business operation (`file to binary`) reads the PO acknowledgment binary data from the file system. The resulting binary data is assigned to the `POAckbinary` variable.

Subsequently, this node uses the data integration functionality provided by WebLogic Integration to translate the PO acknowledgment binary data to XML. This translation is accomplished through the Translate Binary to XML action, which is provided by the data integration plug-in.

The translation action takes the `POAckbinary` variable as input, and assigns the result of the translation to the `POAck_XML` variable. After the `POAck_XML` variable is set in this way, the public workflow (`WLIS_SupplierPOPublic`) that calls this private workflow can proceed to its next task, as described in "Call WLIS_SupplierOne PO private workflow" on page 3-73.

For more information about data integration actions that can be executed through the Studio, see "Data Integration" on page 3-81.

Application and Data Integration

This section contains the following topics:

- Introduction
- Application Integration
- Data Integration

Introduction

The workflows for the PO business process and the `WLIS_BuyerQPAPrivate` workflow integrate the BPM functionality of WebLogic Integration with the application integration and data integration functionality that are also provided by WebLogic Integration.

A buyer-side process uses the application integration framework to update the PO information in the enterprise information system (EIS), based on the PO acknowledgment. It then writes PO acknowledgment information to the `POAcknowledgement.xml` file.

A supplier-side process uses the data integration framework to translate XML data to binary data, and vice versa.

Application Integration

The `WLIS_BuyerQPAPrivate` and `WLIS_BuyerPOPrivate` workflows highlight the application integration functionality provided by WebLogic Integration, and the interaction between workflows and application integration services. (See “Buyer QPA Private Workflow” on page 3-28 and “Buyer PO Private Workflow” on page 3-61.)

Businesses need enterprise application integration (EAI) solutions that make it possible for applications to share data and business processes without first making changes to their original application code or data structures. WebLogic Integration provides an application integration framework that supports inter-enterprise integration through the use of *adapter*s.

Adapters provide an interface that applications can use to access enterprise data programmatically. For example, an adapter can use a Java class to represent enterprise data, and it can provide methods that applications can invoke to access the data. When an application invokes an access method, the adapter executes the method to retrieve the enterprise data. The application integration functionality provided by WebLogic Integration is based on J2EE Connector Architecture (JCA). In addition to complete compliance with the J2EE standard, the application integration tools available with WebLogic Integration offer other important functionality:

- WebLogic Integration supports the bidirectional communication that enables a Java program to invoke services provided in the EIS, and to respond to an event generated by the EIS.
- WebLogic Integration provides high-level, XML-based *application views.* Application views are a business service abstraction over JCA-compliant EIS adapters. They are used to define the event/service interface between a Java program and an EIS. In the WebLogic Integration Studio they are used to enlist an EIS into workflows.

See *Introducing Application Integration* for details about application integration in the WebLogic Integration environment.

When you define an application view, you create an XML-based interface between WebLogic Integration and an EIS application. In this sample, the EIS system is an RDBMS. The application integration adapter used in the sample is the DBMS adapter that is packaged with the WebLogic Integration product. Based on the DBMS adapter, an application view (`WLISAppView.sav`) is defined and deployed in WebLogic Integration for this sample. See `\samples\wlis\src\examples\wlis\wlai\WLISAppViewDeployer.java` in your WebLogic Integration installation.

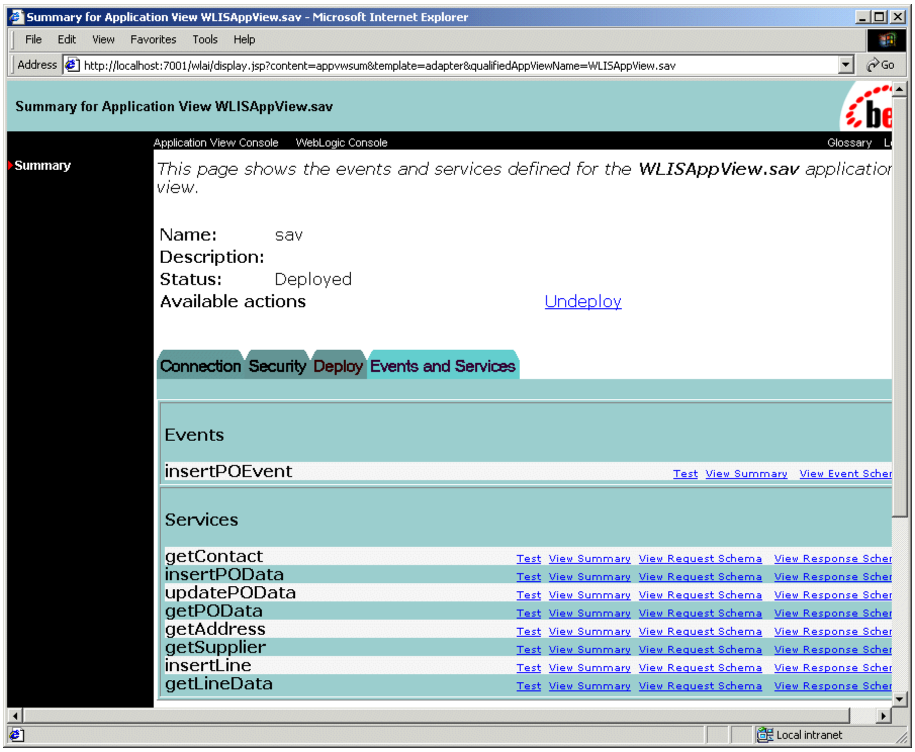
You can define, test, and deploy an application view using the WebLogic Integration Application View Console, a Web-based user interface. For details about defining application views, see *Using Application Integration.*

You can also use the Application View Console to view the details of the application view created by the `WLISAppviewDeployer.java` Java program for this sample, as follows:

1. If you have not already done so, start the WebLogic Integration sample as described in "Setting Up and Running the Sample" on page 2-1.
2. Do one of the following:
   - Click the AI link on the Web page from which you started the sample: `http://localhost:7001`.
   - Choose Start→Programs→BEA WebLogic E-Business Platform→WebLogic Integration 2.1→Samples→Application View Console.

   The Application View Console is displayed. It displays the Root folder, which contains a list of folders that organize the application views for your enterprise. One of the folders in this window is `WLISAppView`, the folder that contains the application view for this sample.
3. Click `WLISAppView` to open a window displaying a list of application views. The list includes an application view defined for the sample application: `sav`. Note that its status is Deployed.
4. Double-click `sav` to display details about the `WLISAppView.sav` application view, as shown in the following figure.

Figure 3-26 WebLogic Integration Application View Console



Summary for Application View WLISAppView.sav

This page shows the events and services defined for the *WLISAppView.sav* application view.

Name: sav
Description:
Status: Deployed
Available actions Undeploy

Events

insertPOEvent

Services

getContact
insertPOData
updatePOData
getPOData
getAddress
getSupplier
insertLine
getLineData

Summary for Application View WLISAppView.sav - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Address http://localhost:7001/wlai/display.jsp?content=appvwsum&template=adapter&qualifiedAppViewName=WLISAppView.sav
Go
Application View Console
WebLogic Console
Glossary
Summary
Connection
Security
Deploy
Events and Services
Test
View Summary
View Event Scher
View Request Schema
View Response Scher
Local intranet

The `WLISAppView.sav` application view includes the following services and events.

Services	`getContact`
	`updatePOData`
	`insertPOData`
	`getPOData`
	`getAddress`
	`getSupplier`
	`insertLine`
	`getLineData`

Events	`insertPOEvent`

Click on the following links in the Application View Console to view more information about any service or event:

- View Summary—Provides the SQL expression
- View Request Schema—Provides XML schema for request data
- View Response Schema—Provides XML schema for response data

Data Integration

The `WLIS_SupplierOnePOPrivate` and `WLIS_SupplierTwoPOPrivate` workflows highlight the data integration functionality provided by WebLogic Integration. This feature is used to translate data from XML to binary format, and vice versa. For this sample application, the following translations are performed:

- PO XML data is converted to binary data.
- PO acknowledgment binary data is converted to XML.

Both translations are described in “Supplier PO Private Workflow” on page 3-75.

The following data integration plug-in actions are available in the Studio to support data integration:

- Translate Binary to XML
- Translate XML to Binary

To perform a data translation at a task node in the Studio:

1. Double-click the node to invoke the Task Properties dialog box.
2. In the Actions pane, click Add to display the Add Action dialog box.
3. Choose Integrated Actions→Data Integration.
4. Select one of the following actions:
   - Translate Binary to XML

     The Binary to XML dialog box is displayed.

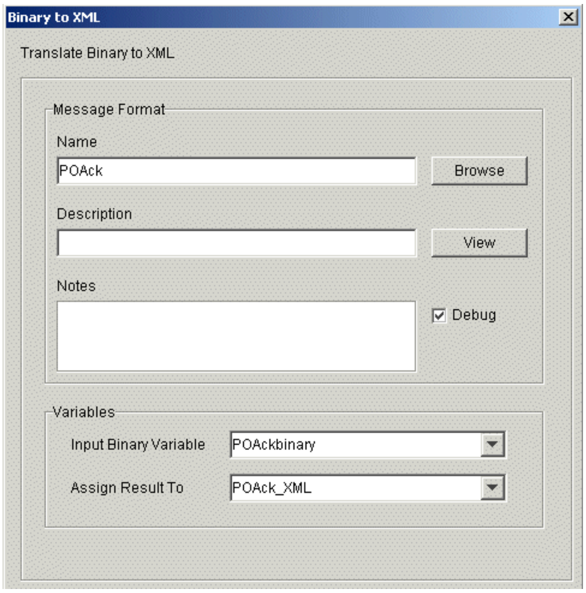
- Translate XML to Binary

  The XML to Binary dialog box is displayed.

5. In the Binary to XML (or XML to Binary) dialog box, browse the WebLogic Integration repository and select an available map.

   The following figure shows the Binary to XML dialog box for the final task node (Get PO Ack and transform into XML) in the `WLIS_SupplierTwoPOPrivate` workflow template, described in "Supplier PO Private Workflow" on page 3-75.

Figure 3-27 Binary to XML Dialog Box



Name: POAck
Description:
Notes:
☑ Debug
Input Binary Variable: POAckbinary
Assign Result To: POAck_XML

In the preceding figure note the following:

- `POAck`, the value in the Name field in the Message Format area, represents the `POAck.mfl` translation map.
- The settings for the input and output variables specify that the binary data will come from the `POAckBinary` variable and the translated data will be written to the `POAck_XML` variable.

For this sample application, two translation maps, `PO.mfl` and `POAck.mfl`, were built in the Format Builder design tool and saved in the WebLogic Integration repository. The maps, stored in the `workflow.jar` package, are imported during the setup and configuration of the sample.

You can view the translation maps for this sample by invoking the Format Builder tool, as described in the following section.

Invoke Format Builder

To invoke Format Builder, complete the procedure appropriate for your platform:

- Windows:
  - Using menus, invoke the Format Builder as follows:

    Choose Start→Programs→BEA WebLogic E-Business Platform→WebLogic Integration 2.1→Format Builder.
  - From the command line, invoke the Format Builder as follows:

    a. Open a command window.

    b. Go to the `bin` directory where you installed WebLogic Integration, and execute the `fb` command. For example, if WebLogic Integration is installed in the default location, enter the following sequence of commands:

    ```
    cd \bea\wlintegration2.1\bin
    fb.cmd
    ```
- UNIX:

  a. Go to the `bin` directory in the samples domain.

  For example, if WebLogic Integration is installed in the default location, enter the following:

  ```
  cd BEA_Home/wlintegration2.1/bin
  ```

  Here `BEA_Home` represents the default directory in which you install BEA products.

  b. Start the Format Builder by entering:

  ```
  . ./fb
  ```

View the Sample Translation Maps

You can view the translation maps for this sample as follows:

- From the Format Builder menu options, choose File→Open.

  A dialog box that allows you to browse for files on your system is displayed.

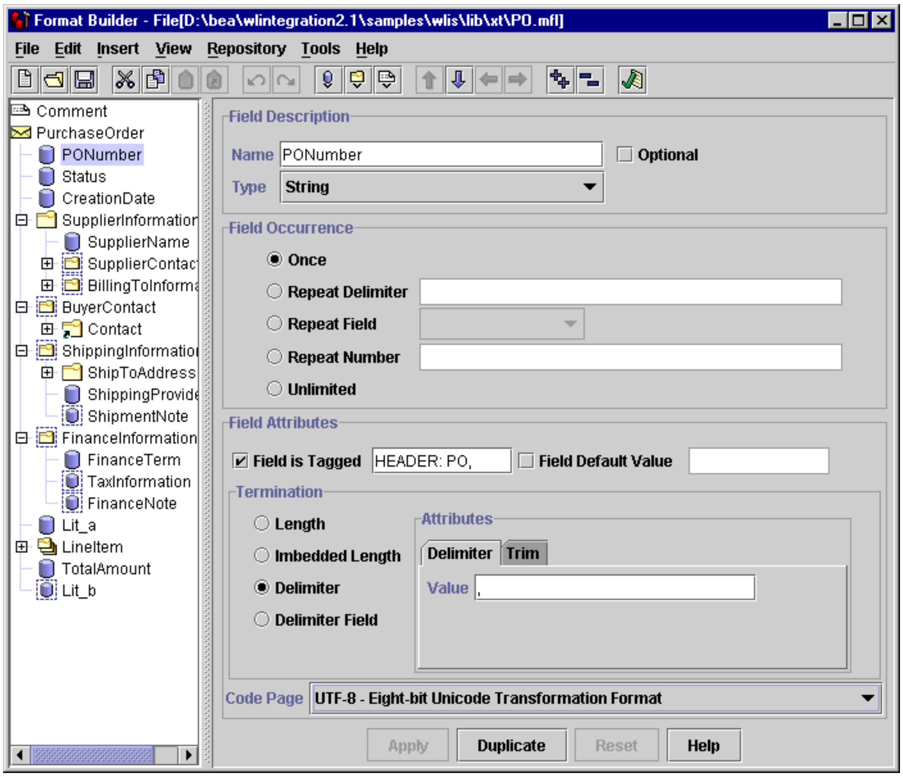
- Select the following directory in your WebLogic Integration installation:

  `/samples/wlis/lib/xt`

- Select the following map files:
  - `PO.mfl`
  - `POAck.mfl`

See *Translating Data with WebLogic Integration* for information about using the Format Builder design tool to build and test your translation maps. The following figure shows the `PO.mfl` translation map being defined in the Format Builder.

Figure 3-28 PO Map in Format Builder



Name: PONumber
Optional: ☐
Type: String

Field Occurrence: [✓ Once] Repeat Delimiter / Repeat Field / Repeat Number / Unlimited

Field is Tagged: ☑ HEADER: PO,
Field Default Value: ☐

Termination: Length / Imbedded Length / [✓ Delimiter] / Delimiter Field

Value: ,

Code Page: UTF-8 - Eight-bit Unicode Transformation Format

A DTDs

The DTDs for XML data that are used by this WebLogic Integration sample can be found in the following directory in your WebLogic Integration installation:

`\samples\wlis\lib\dtds`

For your convenience, they are provided in this appendix. They include:

- General DTDs
- QPA DTDs
- PO DTDs

General DTDs

Listing A-1 common.dtd

```
<!-- Some common definition used by the DTD -->

<!ENTITY % ADDRESS "Address">
<!ELEMENT Address (Street1, Street2?, Street3?, City, State,
ZipCode)>
<!ELEMENT Street1 (#PCDATA)>    <!-- String -->
<!ELEMENT Street2 (#PCDATA)>    <!-- String -->
<!ELEMENT Street3 (#PCDATA)>    <!-- String -->
<!ELEMENT City    (#PCDATA)>    <!-- String -->
<!ELEMENT State   (#PCDATA)>    <!-- String -->
<!ELEMENT ZipCode (#PCDATA)>    <!-- String -->

<!ENTITY % CONTACT "Contact">
<!ELEMENT Contact (ContactName, ContactPhone, ContactEmail?,
                    ContactFax?, ContactAddress)>
<!ELEMENT ContactName    (#PCDATA)>    <!-- String -->
<!ELEMENT ContactPhone   (#PCDATA)>    <!-- String -->
<!ELEMENT ContactPhone   (#PCDATA)>    <!-- String -->
<!ELEMENT ContactEmail   (#PCDATA)>    <!-- String -->
<!ELEMENT ContactFax     (#PCDATA)>    <!-- String -->
<!ELEMENT ContactAddress (%ADDRESS;)>
```

QPA DTDs

Listing A-2 QPARequest.dtd

```
<!ELEMENT QPARequest (QPARequestId, CreationDate, Availability+)>
    <!ELEMENT QPARequestId (#PCDATA)>  <!-- String -->
   <!ELEMENT CreationDate (#PCDATA)>  <!-- DateStamp (CCYYMMDD) -->
  <!ELEMENT Availability (PartId, Quantity, Price, RequiredDate,
                                               Location*, Note?)>

        <!ELEMENT PartId       (#PCDATA)>    <!-- String -->
        <!ELEMENT Quantity     (#PCDATA)>    <!-- Long  -->
        <!ELEMENT UnitPrice        (#PCDATA)>    <!-- Float  -->
        <!ELEMENT RequiredDate (#PCDATA)>    <!-- DateStamp
                                                  (CCYYMMDD) -->
        <!ELEMENT Location     (#PCDATA)>    <!-- String -->
        <!ELEMENT Note         (#PCDATA)>    <!-- String -->
```

Listing A-3 QPAResponse.dtd

```
<!ELEMENT QPAResponse (QPAResponseId, ResponseDate, QPARequestId,
CreationDate, SupplierName, Availability+)>
<!ELEMENT QPAResponseId (#PCDATA)>    <!-- String -->
<!ELEMENT ResponseDate (#PCDATA)>    <!-- DateStamp (CCYYMMDD) -->
<!ELEMENT QPARequestId (#PCDATA)>    <!-- String -->
<!ELEMENT CreationDate (#PCDATA)>    <!-- DateStamp (CCYYMMDD) -->
<!ELEMENT SupplierName (#PCDATA)>    <!-- String -->
<!ELEMENT Availability (PartId, Quantity, Price, AvailableDate,
                            Location*, Note?)>
```

```
        <!ELEMENT PartId        (#PCDATA)>    <!-- String -->
        <!ELEMENT Quantity      (#PCDATA)>    <!-- Long  -->
        <!ELEMENT UnitPrice        (#PCDATA)>    <!-- Float  -->
        <!ELEMENT AvailableDate (#PCDATA)>    <!-- DateStamp
                                                   (CCYYMMDD) -->
        <!ELEMENT Location      (#PCDATA)>    <!-- String -->
        <!ELEMENT Note          (#PCDATA)>    <!-- String -->
```

Listing A-4 Aggregated QPAResponse.dtd

```
<!ELEMENT AggregatedQPAResponse (QPARequestId, CreationDate,
QPAResponse+>
<!ELEMENT QPAResponse (QPAResponseId, ResponseDate, SupplierName,
Availability+)>
<!ELEMENT QPARequestId (#PCDATA)>    <!-- String -->
<!ELEMENT CreationDate (#PCDATA)>    <!-- DateStamp (CCYYMMDD) -->
<!ELEMENT QPAResponseId (#PCDATA)>    <!-- String -->
<!ELEMENT ResponseDate (#PCDATA)>    <!-- DateStamp (CCYYMMDD) -->
<!ELEMENT SupplierName (#PCDATA)>    <!-- String -->

<!ELEMENT Availability (PartId, Quantity, UnitPrice, AvailableDate,
Location*, Note?)>
<!ELEMENT PartId        (#PCDATA)>    <!-- String -->
<!ELEMENT Quantity      (#PCDATA)>    <!-- Long  -->
<!ELEMENT UnitPrice        (#PCDATA)>    <!-- Float  -->
<!ELEMENT AvailableDate (#PCDATA)>    <!-- DateStamp (CCYYMMDD) -->
<!ELEMENT Location      (#PCDATA)>    <!-- String -->
<!ELEMENT Note          (#PCDATA)>    <!-- String -->
```

PO DTDs

Listing A-5 PORequest.dtd

```
<!ELEMENT PORequest (SupplierName, QPAResponseId, QPARequestId,
POItems+)>
<!ELEMENT PORequest    (#PCDATA)>    <!-- String -->
<!ELEMENT SupplierName (#PCDATA)>    <!-- String -->
<!ELEMENT QPAResponseId (#PCDATA)>   <!-- String -->
<!ELEMENT QPARequestId (#PCDATA)>    <!-- String -->
<!ELEMENT POItems (PartId, Quantity, UnitPrice, DeliveryDate,
Location*, Note?)>
<!ELEMENT PartId       (#PCDATA)>    <!-- String -->
<!ELEMENT Quantity     (#PCDATA)>    <!-- Long  -->
<!ELEMENT UnitPrice    (#PCDATA)>    <!-- Float  -->
<!ELEMENT DeliveryDate (#PCDATA)>    <!-- DateStamp (CCYYMMDD) -->
<!ELEMENT Location     (#PCDATA)>    <!-- String -->
<!ELEMENT Note         (#PCDATA)>    <!-- String -->
```

Listing A-6 PO.dtd

```
<!ELEMENT PurchaseOrder (PONumber, Status, CreationDate,
SupplierInformation, BuyerContact?, ShippingInformation?,
FinanceInformation?, LineItem+, TotalAmount?)>

    <!ELEMENT PONumber     (#PCDATA)>       <!-- String -->
    <!ELEMENT Status       (#PCDATA)>       <!-- String -->
   <!ELEMENT CreationDate (#PCDATA)>  <!-- DateStamp (CCYYMMDD) -->
   <!ELEMENT BuyerContact (%CONTACT;)>    <!-- Entity: CONTACT -->

<!ELEMENT ShippingInformation (ShipToAddress?,
                            ShippingProvider, ShipmentNote?)>
    <!ELEMENT ShipToAddress   (%ADDRESS;)> <!-- Entity:ADDRESS -->
    <!ELEMENT ShippingProvider (#PCDATA)>       <!-- String -->
    <!ELEMENT ShipmentNote     (#PCDATA)>       <!-- String -->

<!ELEMENT FinanceInformation (FinanceTerm, TaxInformation?,
FinanceNote?)>

     <!ELEMENT FinanceTerm    (#PCDATA)>    <!-- String -->
     <!ELEMENT TaxInformation (#PCDATA)>    <!-- String -->
     <!ELEMENT FinanceNote    (#PCDATA)>    <!-- String -->
```

```
<!ELEMENT SupplierInformation(SupplierName, SupplierContact?,
                                    BillingToInformation?)>

     <!ELEMENT SupplierName    (#PCDATA)>
     <!ELEMENT SupplierContact (%CONTACT)>
     <!ELEMENT BillingToInformation (BillToAddress, AccountId)>
      <!ELEMENT BillToAddress (%ADDRESS;)> <!-- Entity:ADDRESS -->
      <!ELEMENT AccountID     (#PCDATA)>      <!-- String -->

<!ELEMENT LineItem (LineNumber, PartId, PartDescription?, Quantity,
UnitPrice, DeliveryDate,  Note?)>

        <!ELEMENT LineNumber      (#PCDATA)>    <!-- String -->
        <!ELEMENT PartId          (#PCDATA)>    <!-- String -->
        <!ELEMENT PartDescription (#PCDATA)>    <!-- String -->
        <!ELEMENT Quantity        (#PCDATA)>    <!-- Long  -->
        <!ELEMENT UnitPrice          (#PCDATA)>    <!-- Float  -->
        <!ELEMENT DeliveryDate    (#PCDATA)>    <!-- DateStamp
(CCYYMMDD) -->
        <!ELEMENT Note            (#PCDATA)>    <!-- String -->
<!ELEMENT TotalAmount            (#PCDATA)>    <!-- Float -->
```

Listing A-7 POAcknowledgement.dtd

```
<!ELEMENT PurchaseOrderAcknowledgement (PONumber, POCreationDate,
SONumber, SOCreationDate, SupplierInformation,
ShippingInformation?, FinanceInformation?, LineItem+,
TotalAmount?)>

  <!ELEMENT PONumber     (#PCDATA)>    <!-- String -->
  <!ELEMENT POCreationDate (#PCDATA)>    <!-- DateStamp (CCYYMMDD)
-->
  <!ELEMENT SONumber     (#PCDATA)>    <!-- String -->
  <!ELEMENT SOCreationDate (#PCDATA)>    <!-- DateStamp (CCYYMMDD)
-->
<!ELEMENT SupplierInformation(SupplierName, SupplierContact?,
BillingToInformation?)>
    <!ELEMENT SupplierName    (#PCDATA)>
    <!ELEMENT SupplierContact (%CONTACT)>
    <!ELEMENT BillingToInformation (BillToAddress, AccountId)>
   <!ELEMENT BillToAddress (%ADDRESS;)>   <!-- Entity: ADDRESS -->
    <!ELEMENT AccountID     (#PCDATA)>      <!-- String -->

<!ELEMENT ShippingInformation (ShipToAddress?, ShippingProvider,
TrackingId, ShipmentNote?)>
   <!ELEMENT ShipToAddress   (%ADDRESS;)> <!-- Entity: ADDRESS -->
```

```
    <!ELEMENT ShippingProvider (#PCDATA)>       <!-- String -->
    <!ELEMENT TrackingId       (#PCDATA)>       <!-- String -->
    <!ELEMENT ShipmentNote     (#PCDATA)>       <!-- String -->
<!ELEMENT FinanceInformation (FinanceTerm,
TaxInformation?,FinanceNote?)>
    <!ELEMENT FinanceTerm    (#PCDATA)>    <!-- String -->
    <!ELEMENT TaxInformation (#PCDATA)>    <!-- String -->
    <!ELEMENT FinanceNote    (#PCDATA)>    <!-- String -->
<!ELEMENT LineItem (LineNumber, PartId, PartDescription?, Quantity,
UnitPrice, DeliveryDate,  Note?)>
    <!ELEMENT LineNumber      (#PCDATA)>    <!-- String -->
    <!ELEMENT PartId          (#PCDATA)>    <!-- String -->
    <!ELEMENT PartDescription (#PCDATA)>    <!-- String -->
    <!ELEMENT Quantity        (#PCDATA)>    <!-- Long  -->
    <!ELEMENT UnitPrice           (#PCDATA)>    <!-- Float  -->
   <!ELEMENT DeliveryDate  (#PCDATA)> <!-- DateStamp (CCYYMMDD) -->
    <!ELEMENT Note            (#PCDATA)>    <!-- String -->
<!ELEMENT TotalAmount            (#PCDATA)>    <!-- Float -->
```