BEA WebLogic Integration™

Running the
B2B Integration Samples

Running the B2B Integration Samples

Part Number	Date	Software Version
N/A	October 2001	2.1

Contents

3. Channel Master Sample

4. RosettaNet 2.0 Security Sample

5. Trading Partner Lightweight Client Sample

6. Messaging API Sample

A. JSP Tag Reference

Index

About This Document

This document describes the business-to-business (B2B) integration samples delivered with WebLogic Integration. It provides configuration information and instructions for running and verifying each sample.

This document includes the following topics:

- Chapter 1, "Getting Started," describes the WebLogic Integration B2B samples, and discusses basic installation and configuration.

- Chapter 2, "Hello Partner Sample," demonstrates communication using the default XOCP messaging protocol.

- Chapter 3, "Channel Master Sample," demonstrates both point-to-point and multicast (broadcast) communications using the XOCP business protocol between WebLogic Integration trading partners.

- Chapter 4, "RosettaNet 2.0 Security Sample," shows how WebLogic Integration can be used to implement RosettaNet 2.0 PIP 3A2 and PIP 0A1 using workflows.

- Chapter 5, "Trading Partner Lightweight Client Sample," describes how to configure and use the browser and file-sharing client samples.

- Chapter 6, "Messaging API Sample," shows how the WebLogic Integration Messaging API can be used.

What You Need to Know

This document is intended for independent software vendors (ISVs) who are interested in extending BEA WebLogic Integration. We assume you are familiar with the BEA 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://e-docs.bea.com.

How to Print the Document

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

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

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

Related Information

To learn how to use BEA WebLogic Integration to meet your company's needs, see the following documents:

- WebLogic Integration documentation at the following URL:

  http://edocs.bea.com

- Sun Microsystems, Inc. Java site at the following URL:

  http://java.sun.com/

Contact Us!

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

In your e-mail message, please indicate that you are using the documentation for the BEA WebLogic Integration 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. *Example*s: LPT1 SIGNON OR
{ }	Indicates a set of choices in a syntax line. The braces themselves should never be typed.
[ ]	Indicates optional items in a syntax line. The brackets themselves should never be typed. *Example*: `buildobjclient [-v] [-o name ] [-f file-list]... [-l file-list]...`
|	Separates mutually exclusive choices in a syntax line. The symbol itself should never be typed.
...	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.

1 Getting Started

The business-to-business (B2B) integration samples are provided to help you understand how B2B integration functionality works in WebLogic Integration. In addition, successful installation and setup of the Hello Partner sample provides verification that you have successfully installed WebLogic Integration and configured it properly.

This section includes the following topics:

- Overview of Samples

- Preparing to Run the Samples

Overview of Samples

With the exception of the Hello Partner sample, which verifies the successful use of the XOCP messaging protocol, the samples demonstrate B2B integration solutions to plausible business problems. The following samples are described in this document:

- Hello Partner Sample—Demonstrates the basic handshake necessary for communication. The Hello Partner sample uses XOCP, the default messaging protocol for B2B integration.

- Channel Master Sample—Demonstrates how a large trading partner uses WebLogic Integration to automate its supply chain. The sample shows both point-to-point and multicast (broadcast) communications using the XOCP business protocol between WebLogic Integration trading partners.

- RosettaNet 2.0 Security Sample—Demonstrates how WebLogic Integration can be used to implement RosettaNet 2.0 PIP 3A2 and PIP 0A1 using workflows. This sample uses the WebLogic Integration security features required for RosettaNet 2.0 support: two-way SSL authentication, digital signatures, data encryption, and nonrepudiation.

- Trading Partner Lightweight Client Sample—Illustrates a request-reply scenario in which two trading partners without a B2B integration installation use the B2B integration mailbox interface. Two types of lightweight client communication are simulated for this demonstration:

  - Browser-based client—Uses XML and JMS to prepare, deliver, and collect information

  - File-sharing client—Uses a third-party file-sharing server to exchange messages

- Messaging API Sample—Shows how the WebLogic Integration Messaging API can be used. Specifically, it demonstrates the use of two message-delivery mechanisms available with the Messaging API and the logic plug-in feature of WebLogic Integration B2B integration.

Preparing to Run the Samples

Before you can run the B2B samples, you must install WebLogic Integration and configure the samples. If you have not already done so, complete the following steps:

1. Install WebLogic Integration.

    For instructions, see *Installing BEA WebLogic Integration*. When prompted to choose an install set, select WebLogic Integration Full Installation With Samples (the default selection).

    The WebLogic Integration installer prompts you to specify the databases to configure for the samples and production domains. The samples database information is used when you configure the samples.

2. Configure and start the server in the samples domain.

    For instructions, see "Configuring and Starting the Samples Domain" in "Getting Started" in *Starting, Stopping, and Customizing BEA WebLogic Integration*.

When you execute the RunSamples command as described in "Configuring and Starting the Samples Domain," the following actions are performed:

- The samples database is created.

- The sample repository data is bulk loaded into the database.

- WebLogic Server is started and the workflows are imported. A browser is started and the samples launcher page is displayed.

Note: To display the samples launcher page on a UNIX system, your PATH environment variable must include the directory in which the Netscape executable (netscape) resides. For additional Web browser configuration requirements, see "Web Browser Configuration Requirements" in "WebLogic Integration Administration and Design Tools" in *Starting, Stopping, and Customizing BEA WebLogic Integration*.

Once you have configured the samples and started the server in the samples domain, any additional steps required to configure a particular sample are provided in the documentation for that sample. Instructions are provided for both Windows and UNIX systems.

Switching the Default Database

You can use any supported database with the samples domain. The WebLogic Database Configuration Wizard can be used to update your samples domain to a new database. If you would like to update the samples domain to use a new database, complete the following steps:

1. Use the WebLogic Integration Database Configuration Wizard to switch to the new database.

   For instructions, see the "Using the Database Configuration Wizard" and "Specifying a New Database for a Domain" topics in "Customizing WebLogic Integration" in *Starting, Stopping, and Customizing BEA WebLogic Integration*.

2. Execute the RunSamples command to configure the new database, start the samples domain, and display the samples launcher in your Web browser.

   For instructions, see "Configuring and Starting the Samples Domain" in "Getting Started" in *Starting, Stopping, and Customizing BEA WebLogic Integration*.

WebLogic Integration Domains

As described in "Getting Started" in *Starting, Stopping, and Customizing BEA WebLogic Integration*, a domain is a set of interrelated WebLogic Server resources defined in a single configuration file. A special WebLogic Server domain is set up to run all WebLogic Integration samples that require only one server. This domain, which is fully configured after installation, but is not yet populated, resides in the `WLI_HOME`/`config`/`samples` directory. The Samples domain uses the database chosen during installation, or when the WebLogic Integration Database Configuration Wizard was last run as described in "Switching the Default Database."

For additional information about using the preconfigured domains installed with WebLogic Integration, see "Getting Started" in *Starting, Stopping, and Customizing BEA WebLogic Integration*.

Browser Proxy Settings

If your browser is unable to connect to the URL for the samples launcher, it may be using a proxy server that prevents you from connecting to your local WebLogic Server. If this condition exists, the following error message is displayed:

```
The requested URL could not be retrieved
```

To bypass the proxy server, change the browser proxy settings so that the browser does not use the proxy server to access the servlet. The procedure for making this change is browser-dependent:

- For Internet Explorer, choose Tools→Internet Options→Connections→LAN Settings. The Local Area Network (LAN) Settings Dialog box is displayed. Select the following option: Bypass proxy server for local addresses.

- For Netscape, choose Edit→Preferences→Advanced→Proxies→View. In the Exceptions text field, specify your `localhost:`*`listening_port`*. (The value of *`listening_port`* should be the listening port number specified in the `config.xml` file.) The default is 7001.

2 Hello Partner Sample

The Hello Partner sample demonstrates communication using the default messaging protocol: XOCP. This section discusses the following topics:

- Overview of the Hello Partner Sample

- Before Running the Hello Partner Sample

- Running the Hello Partner Sample

- How the Sample Works

Overview of the Hello Partner Sample

The Hello Partner sample demonstrates business communications between two trading partners using WebLogic Integration.

What the Sample Demonstrates

The Hello Partner sample demonstrates how two trading partners send business messages using the XOCP protocol. For each trading partner, the sample demonstrates the following:

- Public process—A public process that handles communication between the trading partners is shown. The public process workflows in this sample use the XOCP protocol to send messages between trading partners.

■ Private process—A private process for each trading partner that processes message content is shown.

This sample also illustrates the preferred method of handling trading partner message traffic. Public processes are used to manage trading partner message traffic, while private processes are used for message creation, message processing, and links to outside applications.

Hello Partner Sample Scenario Logic

The Hello Partner sample scenario involves two trading partners: one requestor and one replier, as shown in Figure 2-1. The following figure shows the high-level interactions between the different Hello Partner workflows.

Figure 2-1   Interactions Among Hello Partner Workflows
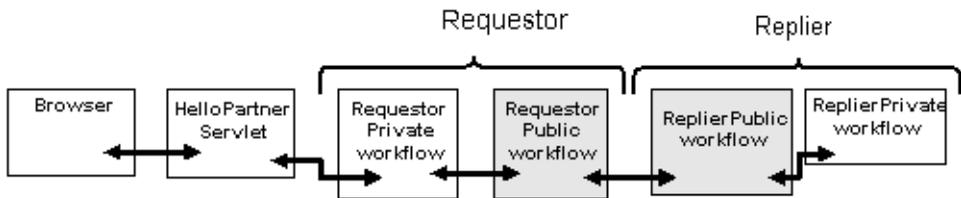
Notes: The public workflows are shaded in gray in the preceding figure.

The following sequence summarizes the main events in this scenario:

1. The RunSamples script is started, with the following results:

    a. The sample instance of the WebLogic Server is started.

    b. A browser opens and displays the samples launcher page.

    c. On the samples launcher page, the Hello Partner link is clicked: the Hello Partner main HTML page is displayed.

2. Values for the integers are selected from the menu. The Click Here to Begin link is selected. Consequently, the Hello Partner main HTML page sends an HTTP request to the HelloPartnerServlet. The HTTP request contains the selected integer values.

3. The HelloPartnerServlet sends an JMS XML event with the integer values.

4. The preceding JMS XML event triggers the RequestorPrivate workflow to start. The RequestorPrivate workflow composes the MultiplyRequestXML workflow variable and starts the RequestorPublic workflow, passing in the MultiplyRequestXML workflow variable. The RequestorPublic workflow then waits for a response from the replier.

5. The ReplierPublic workflow receives the message and extracts integer values from the MultiplyRequestXML workflow variable. It puts these values into the MultiplyInputsXML workflow variable and posts an internal XML event.

6. The ReplierPrivate workflow is triggered by the preceding internal XML event. The ReplierPrivate workflow multiplies the two integers, places the result in the MultiplyOutputsXML workflow variable, and post an internal XML event with the MultiplyOutputsXML workflow variable.

7. The ReplierPublic workflow, which was waiting for the preceding XML event, creates a MultiplyReplyMessage input message workflow variable and sends it with a business message.

8. The RequestPublic workflow, which was waiting for the preceding business message, extracts values from the MultiplyReplyMessage workflow variable, creates the MultiplyReplyXML workflow variable, and posts it with an internal XML event.

9. The RequestPrivate workflow, which was waiting for the preceding XML event, creates the ResultXMLForJSP workflow variable and posts it with an external XML event.

10. The HelloPartnerServlet receives the XML event, extracts the result of the multiplication of the two integers from the XML, and displays it in the browser.

Before Running the Hello Partner Sample

Before running the Hello Partner sample, complete the following steps:

1. Follow the instructions in "Preparing to Run the Samples" on page 1-2.

2. Make sure the proxy settings on your browser do not prevent you from connecting to the sample 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*.

Running the Hello Partner Sample

To run the Hello Partner sample:

1. Run the RunSamples script by completing the procedure appropriate for your platform:

    - Windows:

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

    - UNIX:

        a) Make sure your PATH environment variable includes the directory in which the Netscape (netscape) executable resides.

        b) Go to the WebLogic Integration home directory (the directory in which you installed WebLogic Integration). For example:

        ```
        cd /home/me/bea/wlintegration2.1
        ```

        c) Run the setenv script to set the top-level WebLogic Integration environment variables:

        ```
        . setenv.sh
        ```

        d) Run the RunSamples script:

```
cd samples/bin
RunSamples
```

Warning: For UNIX systems, the directory in which the netscape executable resides must be included in your PATH environment variable. If it is not included, the samples launcher page cannot be displayed.

2. If the RunSamples script detects that the configuration section of the RunSamples script has been run before, the following prompt is displayed:

```
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
```

If you answer N to this question, the RunSamples script skips the steps for creating and populating the repository and runs only the step for booting the sample instance of the WebLogic Server.

If you answer Y to this question, the RunSamples script recreates and repopulates the repository, and then it boots the sample instance of the WebLogic Server. When you answer Y, the RunSamples script destroys all the data currently in the repository and loads an unaltered version of the sample data into the repository. Answer Y only when the current sample data has been altered or removed and you want a fresh or unaltered version of the sample data in the repository.

Now the RunSamples script starts an instance of the WebLogic Server as a background process and the samples launcher page is displayed.

Figure 2-2   Samples Launcher Page
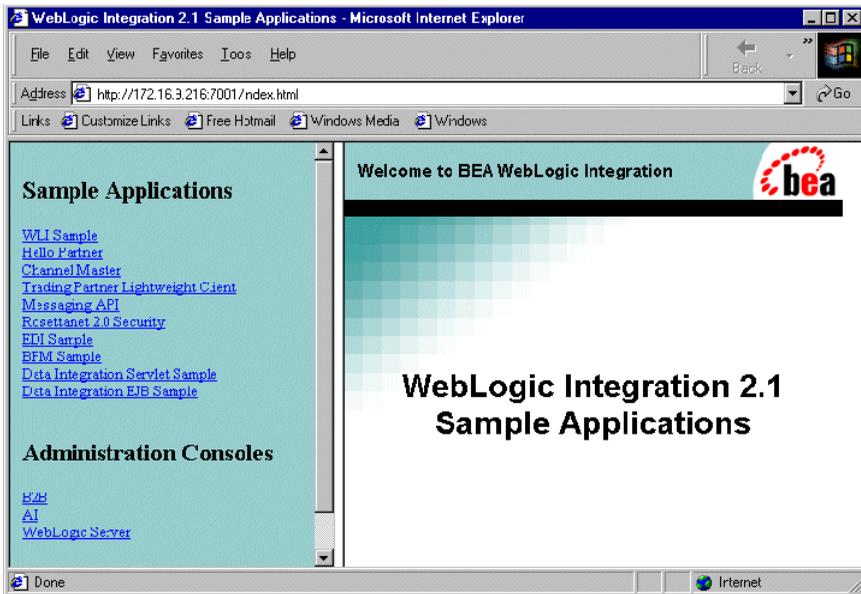
3. Click the link for Hello Partner, listed under Sample Applications in the left pane of the samples launcher page. The Hello Partner sample is displayed in the right pane.

Figure 2-3   Hello Partner Sample Launcher Page
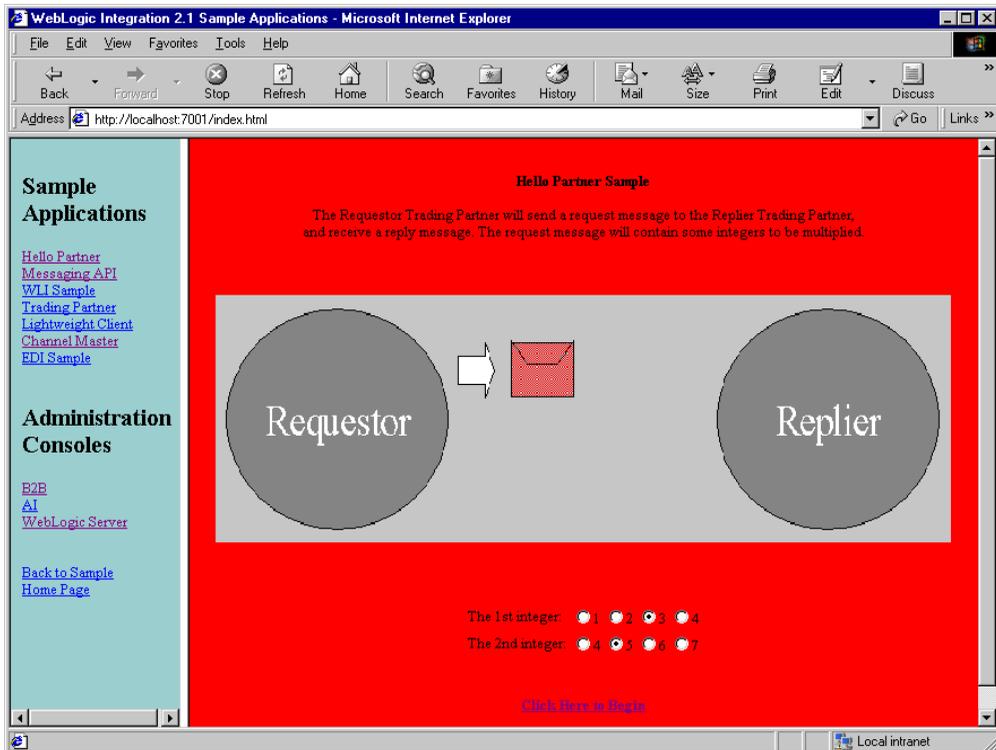
4. Select two numbers using the radio buttons. Click the Click Here to Begin link.

Figure 2-4   Hello Partner Sample Result Page
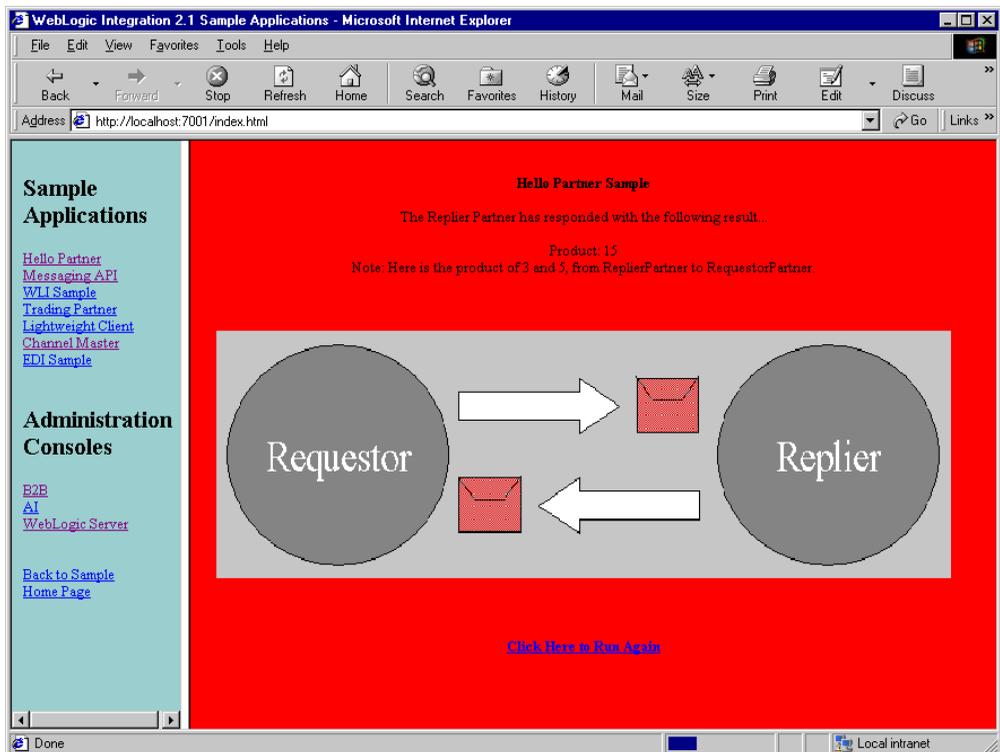
5. If you want to run more B2B samples at this time, keep the samples launcher page open and keep the WebLogic Server running.

    If you do not want to run more B2B samples at this time, exit from your browser and shut down the WebLogic Server by completing the procedure appropriate for your platform:

    ● Windows:

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

    ● UNIX:

      ```
      cd $WLI_HOME/config/samples/bin
      stopWebLogic
      ```

How the Sample Works

A total of four workflows are used in this sample. Two public workflows manage the requestor's and replier's sides of the XOCP message exchange. One private workflow connects to the servlet and the requestor's public workflow; another creates the replier's reply data.

The following sections provide an overview of this process and describe each type of workflow in detail:

- Documents Exchanged

- Requestor Private Workflow

- Requestor Public Workflow

- Replier Public Workflow

- Replier Private Workflow

Documents Exchanged

The following XML documents are used in the Hello Partner sample:

- Request Message from Requestor Role

- Reply Message from Replier Role

- XML Message Over JMS from Servlet to Trigger Private Workflow

- XML Message Over JMS from Private Workflow to Servlet with Result

- XML Event from Replier Public Workflow to Replier Private Workflow

- XML Event from Replier Private Workflows to Replier Public Workflows

The document type definitions (DTDs) for these documents are located in the `%WLI_HOME%\config\samples` directory for Window systems and the `$WLI_HOME/config/samples` directory for UNIX systems.

Request Message from Requestor Role

The following XML message is sent by the requestor. It includes the two numbers to be multiplied:

```
<multiply-request>
        <integer-one>5</integer-one>
        <integer-two>7</integer-two>
</multiply-request>
```

The message conforms to the `multiply-request.dtd`.

Reply Message from Replier Role

The following XML message, sent by the replier, contains the multiplication product, as well as a generated message:

```
<multiply-reply>
        <integer-product>35</integer-product>
        <note>Dear RequestorPartner: Here is the product of 7 and 5,
 from ReplierPartner to RequestorPartner.</note>
</multiply-reply>
```

The message conforms to the `multiply-reply.dtd`.

XML Message Over JMS from Servlet to Trigger Private Workflow

The following message is sent over JMS by the servlet. Its arrival triggers the requestor's private workflow:

```
<from-multiply-request-jsp-to-workflow light-weight="false">
        <integer-one>5</integer-one>
        <integer-two>7</integer-two>
</from-multiply-request-jsp-to-workflow>
```

The message conforms to the `from-multiply-request-jsp-to-workflow.dtd`.

XML Message Over JMS from Private Workflow to Servlet with Result

The following message is sent, over JMS, from the requestor's private workflow to the servlet. It contains the product of the multiplication, as well as a text message:

```
<from-workflow-to-multiply-request-jsp>
        <integer-product>35</integer-product>
```

```
        <note>Dear RequestorPartner: Here is the product of 7 and 5
 from ReplierPartner to RequestorPartner.</note>
</from-workflow-to-multiply-request-jsp>
```

The message conforms to the `from-workflow-to-multiply-request-jsp.dtd`.

XML Event from Replier Public Workflow to Replier Private Workflow

The following XML event contains the request input message with four parameters
(the two multiplication inputs, the name of the requestor, and the name of the replier):

```
<multiply-inputs>
        <integer-one>5</integer-one>
        <integer-two>7</integer-two>
        <requestor-name>PartnerRequestor</requestor-name>
        <replier-name>PartnerReplier</replier-name>
</multiply-inputs>
```

The message conforms to the `multiply-inputs.dtd`.

XML Event from Replier Private Workflows to Replier Public Workflows

The following XML event contains the reply output of the private workflow:

```
<multiply-outputs>
        <integer-product>35</integer-product>
        <note>Dear RequestorPartner: Here is the product of 7 and 5
 from ReplierPartner to RequestorPartner.</note>
</multiply-outputs>
```

The message conforms to the `multiply-outputs.dtd`.

Requestor Private Workflow

The requestor private workflow receives the initial request from the servlet, creates a
message of the appropriate type, and sends it to the public workflow for transmission.
When it receives the reply, it processes the reply message, and sends the results to the
servlet. This process is illustrated by the workflow shown in the following figure.

Figure 2-5   Requestor Private Workflow
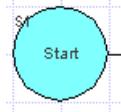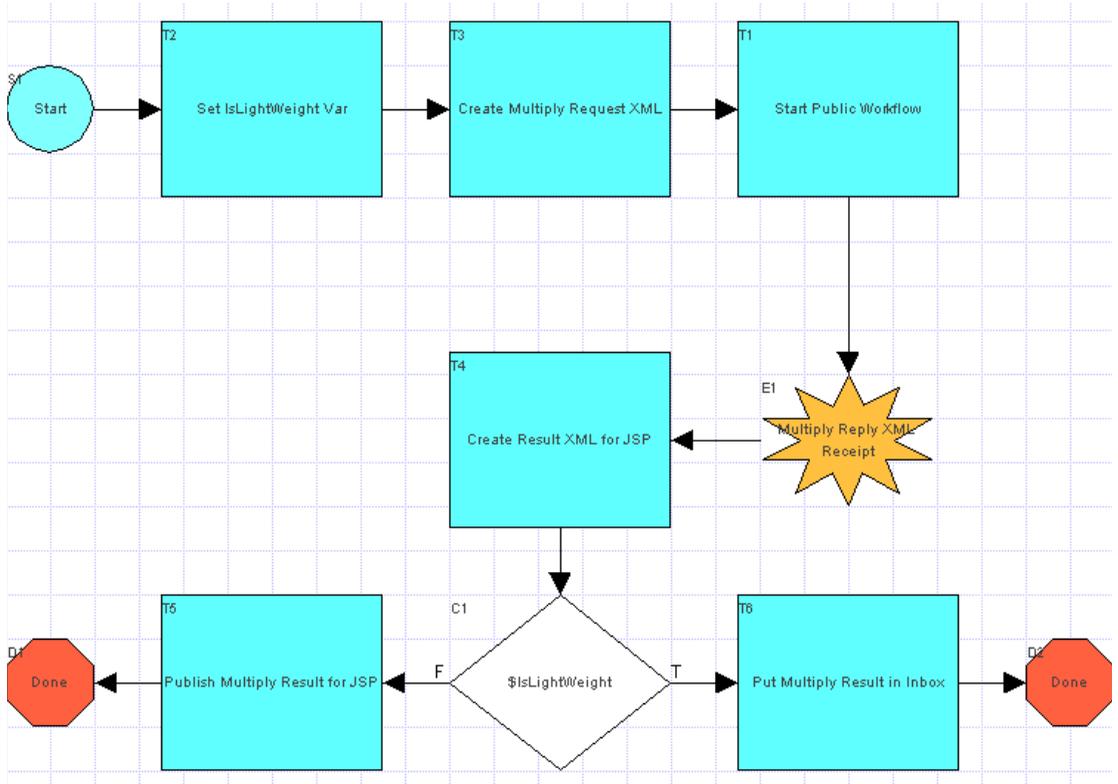
An XML event received from the servlet triggers the workflow. The XML event is of the form `<from-multiply-request-jsp-to-workflow>`, as discussed in "XML Message Over JMS from Servlet to Trigger Private Workflow." The Start node extracts the conversion string from XML, creates a `<multiply-request>` document, and stores it in a workflow variable.
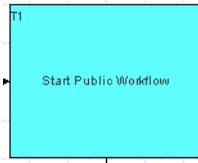
The Action node starts the public workflow and passes it a workflow variable containing the `<multiply-request>` document.

The Event node waits for a `<multiply-reply>` document. When the `<multiply-reply>` document is received, the conversion string is extracted from it. A `<from-workflow-to-multiply-request-jsp>` document is created and sent to the servlet.

A Done node ends the workflow.

Requestor Public Workflow

The Requestor public workflow is initiated by the XML event from the private workflow. It creates a message with the request XML, sends the message to the Replier, waits for a response, extracts the response XML from the response message, and then passes the response to the private workflow for processing. The workflow shown in the following figure illustrates this process.

Figure 2-6 Requestor Public Workflow
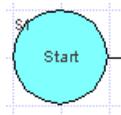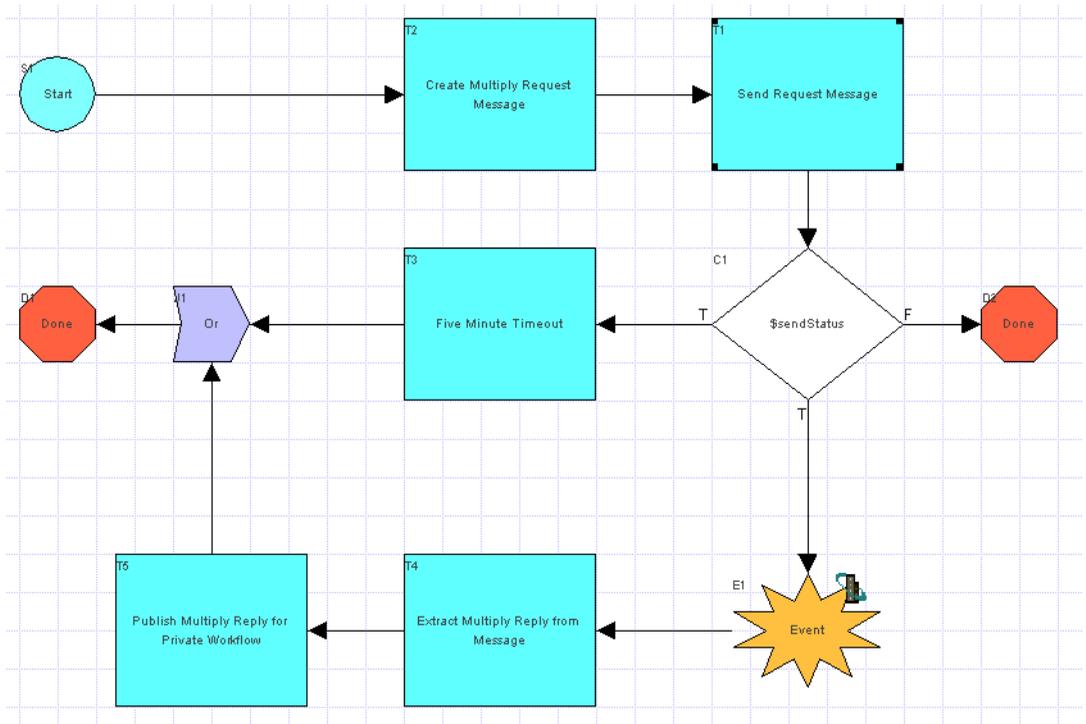
The workflow is called by having a
`<multiply-request>` document passed to it. The Start
node extracts the message string and stores it in a
workflow variable.
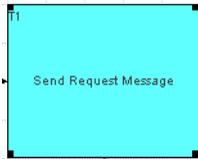
	An Action node sends the `<multiply-request>` document, inside an XOCP message, to the replier role, specifying PartnerReplier as the name of the trading partner.
	An Event node waits for an XOCP reply message from the replier. When the reply arrives, it extracts the `<multiply-reply>` document from the message and stores it in a workflow variable.
	An Action node publishes the `<multiply-reply>` document as an XML Event.
	A Done node ends the workflow.

Replier Public Workflow

The replier public workflow is initiated upon receipt of the requestor's message. It receives the message, extracts the request XML from the message, publishes an XML event containing the request XML to trigger the replier private workflow, receives an XML event back from the replier private workflow, and sends a message containing the response XML back to the requestor as a reply. This process is illustrated by the workflow shown in the following figure.

Figure 2-7   Replier Public Workflow
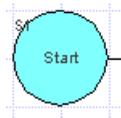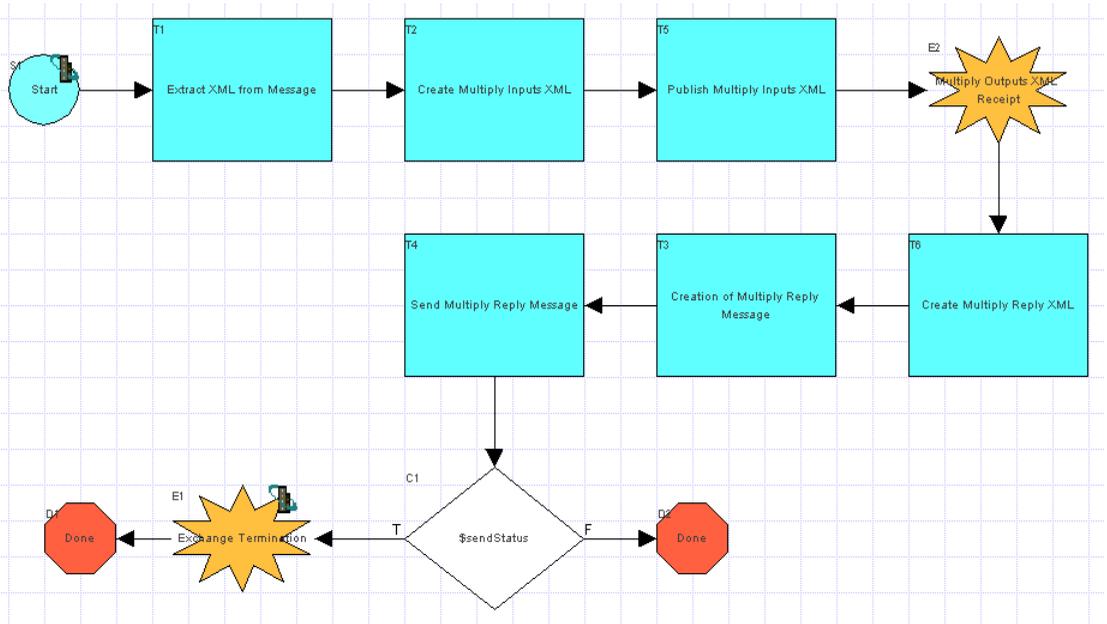
The workflow is started upon receipt of a
<multiply-request> document. The Start node
extracts the content from the message and stores it in a
workflow variable. The format of the workflow variable
is <multiply-input>, using the values from inside the
<multiply-request> document, and the sender and
replier names from within the message.
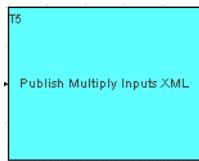
An Action node publishes the <multiply-inputs>
document as an XML Event, initiating the private
workflow.

An Event node waits for a response, which it receives as an XML Event containing a `<multiply-outputs>` document.
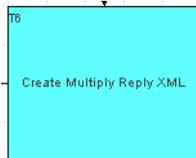
An Action node creates a `<multiply-reply>` document based on the `<multiply-outputs>` document. It stores the result in an XML workflow variable.

An Action node sends the message containing the `<multiply-reply>` document to the requestor, using XOCP.

A Done node ends the workflow.

Replier Private Workflow

The replier private workflow is initiated upon receipt of the XML Event containing the request XML from the replier public workflow. It receives the request, processes the data, generates a reply in an XML document, and sends the reply XML back to the replier public workflow using an XML Event.

Figure 2-8 Replier Private Workflow
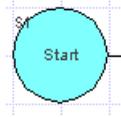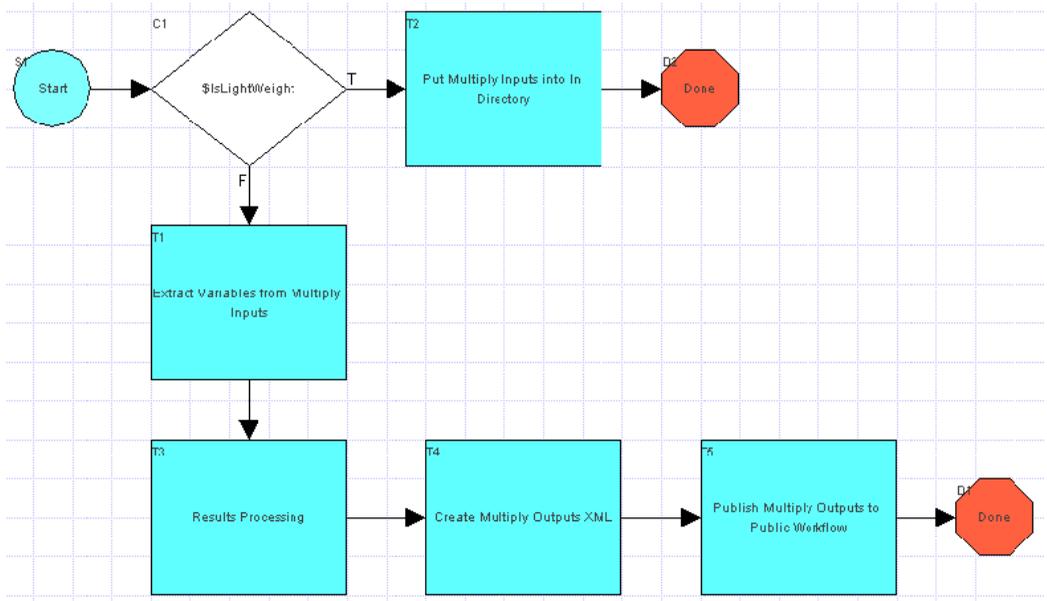
	The workflow is started when an XML Event containing a document of type `<multiply-inputs>` is received. The Start node stores the document in a workflow variable.
	The Action node creates an integer workflow variable containing the product obtained by multiplying the input integers from the `<multiply-inputs>` document.
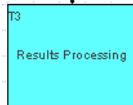
The Action node creates a document of type
`<multiply-outputs>` as a workflow XML variable.
The values from the integer and note workflow variables
in steps 2 and 3 are stored in this document.
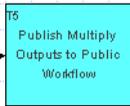
The Action node publishes the `<multiply-outputs>`
document as an XML Event.

A Done node ends the workflow.

3 Channel Master Sample

The Channel Master sample shows how a large trading partner uses WebLogic Integration to automate its supply chain. The sample demonstrates both point-to-point and multicast (broadcast) communications using the XOCP business protocol between WebLogic Integration trading partners.

This section includes the following topics:

- Channel Master Sample Overview

- Before Running the Channel Master Sample

- Running the Channel Master Sample

- Workflows Behind the Channel Master Sample

Channel Master Sample Overview

The following sequence summarizes the communication between the trading partners for this sample:

1. One trading partner, the channel master buyer, broadcasts a query for pricing and availability (QPA) of a particular item. In this sample, two supplier trading partners are listening for queries, so two supplier trading partners receive the query. (A broadcast communication can be received by many trading partners. In this sample

only two suppliers are listening, so only two receive the query.) This action demonstrates broadcast or multicast communication. In the following figure, the envelopes labeled QPA represent XML messages.
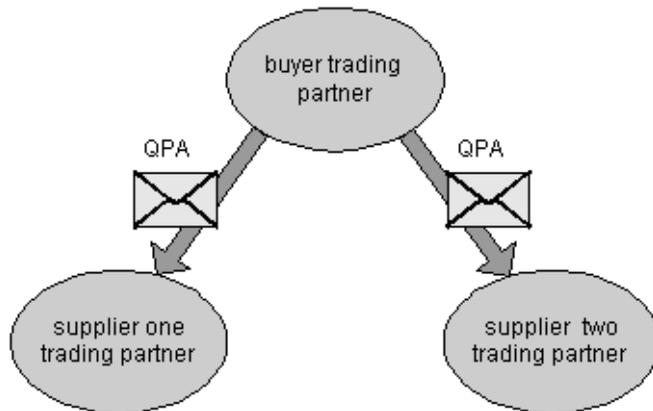
In this case, an XML message containing a query for pricing and availability (QPA) is passed from one trading partner to another. This illustration is a simple representation of trading partners. For a more detailed representation, see "Multicast or Broadcast Messages" on page 3-23.

2. The two suppliers send the buyer reply quotes, each of which contains the price and availability of the requested item. In this step, the two suppliers have point-to-point communications with the buyer.
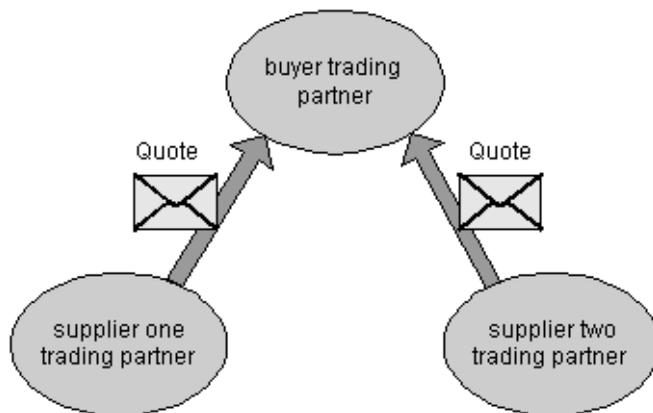
3. The buyer selects a supplier and sends a purchase order (PO) to that supplier. This action provides another example of point-to-point communication.
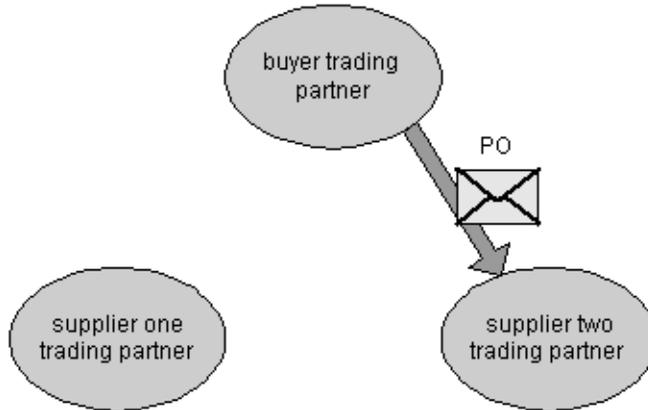
4. The supplier replies with a purchase order acknowledgment message.
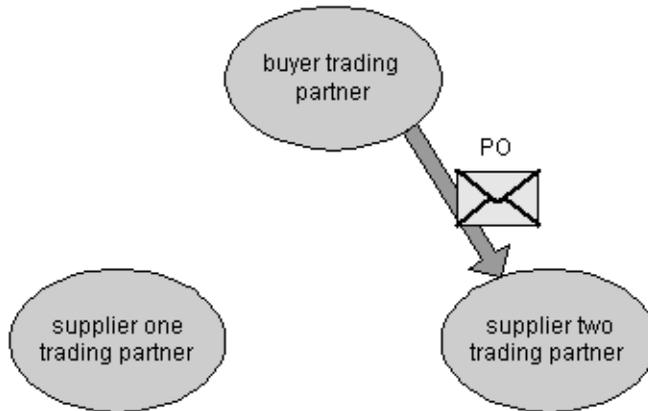
Before Running the Channel Master Sample

Before running the Channel Master sample, complete the following steps:

1. Follow the instructions in "Preparing to Run the Samples" on page 1-2.

2. Make sure the proxy settings on your browser do not prevent you from connecting to the sample 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.*

Running the Channel Master Sample

To run the Channel Master sample, complete the following steps:

Note: If the instance of the WebLogic Server started by the RunSamples script is running, skip to step 3.

1. Run the RunSamples script by completing the procedure appropriate for your platform:

   - Windows:

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

   - UNIX:

     a) Make sure your PATH environment variable includes the directory in which the Netscape executable (netscape) resides.

     b) Go to the WebLogic Integration home directory (the directory in which you installed WebLogic Integration). For example:

     ```
     cd /home/me/bea/wlintegration2.1
     ```

     c) Run the setenv script to set the top-level WebLogic Integration environment variables:

     ```
     . setenv.sh
     ```

     d) Run the RunSamples script:

     ```
     cd samples/bin
     RunSamples
     ```

Note: For UNIX systems, the directory in which the `netscape` executable resides must be included in your `PATH` environment variable. If it is not included, the samples launcher page cannot be displayed.

2. If the `RunSamples` script detects that the configuration section of the `RunSamples` script has been run before, the following prompt is displayed:

```
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
```

If you answer `N` to this question, the `RunSamples` script skips the steps for creating and populating the repository and runs only the step for booting the sample instance of the WebLogic Server.

If you answer `Y` to this question, the `RunSamples` script recreates and repopulates the repository, and then it boots the sample instance of the WebLogic Server. When you answer `Y` the `RunSamples` script destroys all the data currently in the repository and loads an unaltered version of the sample data into the repository. Answer `Y` only when the current sample data has been altered or removed and you want a fresh or unaltered version of the sample data in the repository.

Now the `RunSamples` script starts an instance of the WebLogic Server (as a background process) and the samples launcher page is displayed.

Unlike most of the WebLogic Integration B2B samples, the Channel Master sample is not started from the samples launcher page. It does, however, require the sample instance of the WebLogic Server to be running. Until you complete this procedure, make sure that this instance of WebLogic Server and the browser with the sample launcher page continue to run.

3. Log in to the WebLogic Integration Worklist using the following information:

- Login: `joe`
- Password: `password`
- URL: `t3//localhost:7001`

The main WebLogic Integration Worklist window is displayed.

4. Run the sample using the WebLogic Integration Worklist.

a. Expand the drop-down list in the upper-right corner of the Worklist window. Select ORG.
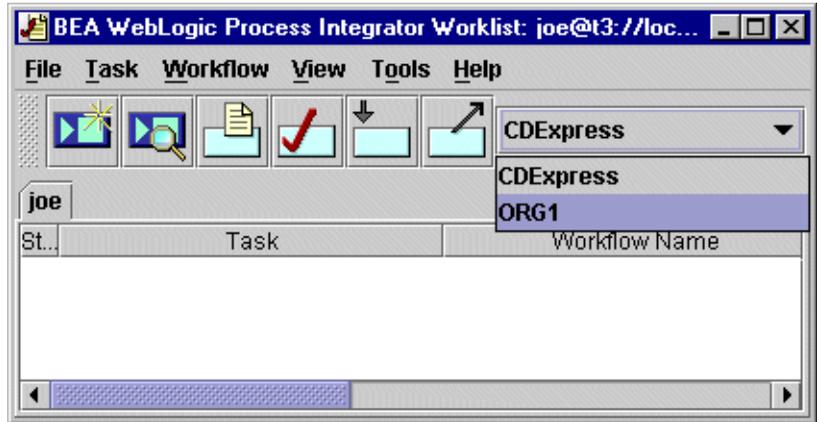
b. Select the joe tab.
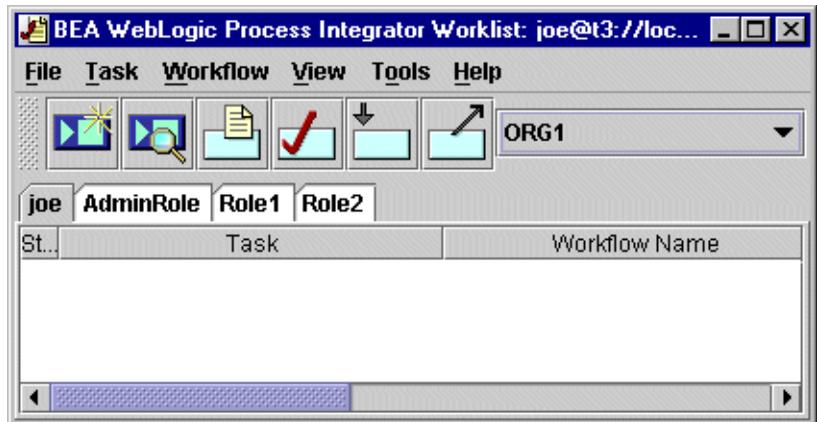
c. From the Worklist menu bar, choose Workflow→Start a Workflow. The Start Workflow dialog box is displayed.

d. Select the BuyerPrivate workflow. Click OK.
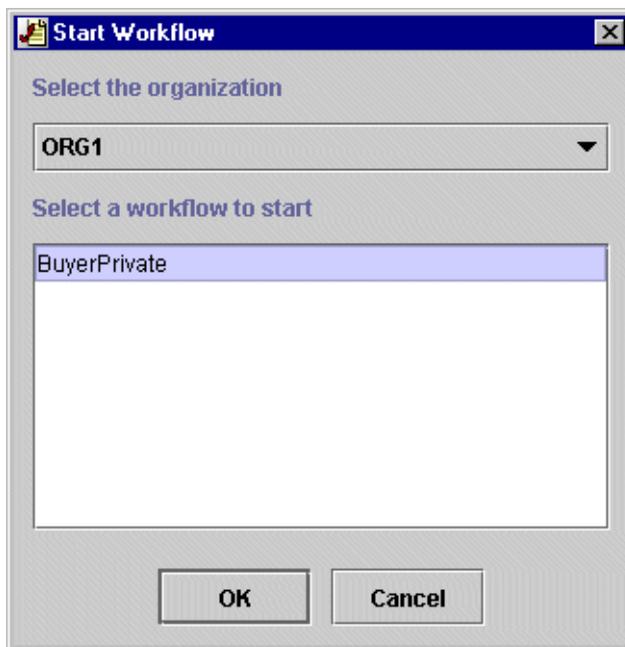
e. To verify that the sample has run successfully, look for the following message in the `myserver.log` file:

```
CHANNEL MASTER SAMPLE RAN SUCCESSFULLY!!!
```

The `myserver.log` file is located in the `%WLI_HOME%\config\samples\logs` directory on a Windows platform and in the `$WLI_HOME/config/samples/logs` directory on a UNIX platform.

5. Exit the WebLogic Integration Worklist: From the Worklist menu bar, choose File→Exit.

6. If you want to run more B2B samples or complete the steps described in "Viewing the SupplierOnePrivate Workflow" on page 3-15 at this time, keep the samples launcher page open and keep the WebLogic Server running.

If you do not want to run more B2B samples or complete the steps described in "Viewing the SupplierOnePrivate Workflow" on page 3-15 at this time, exit from your browser and shut down the WebLogic Server by completing the procedure appropriate for your platform:

- Windows:

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

- UNIX:

  ```
  cd $WLI_HOME/config/samples
  stopWebLogic
  ```

Workflows Behind the Channel Master Sample

The following figure shows the high-level interactions between the different Channel Master workflows.

Figure 3-1   Interactions Among Channel Master Workflows
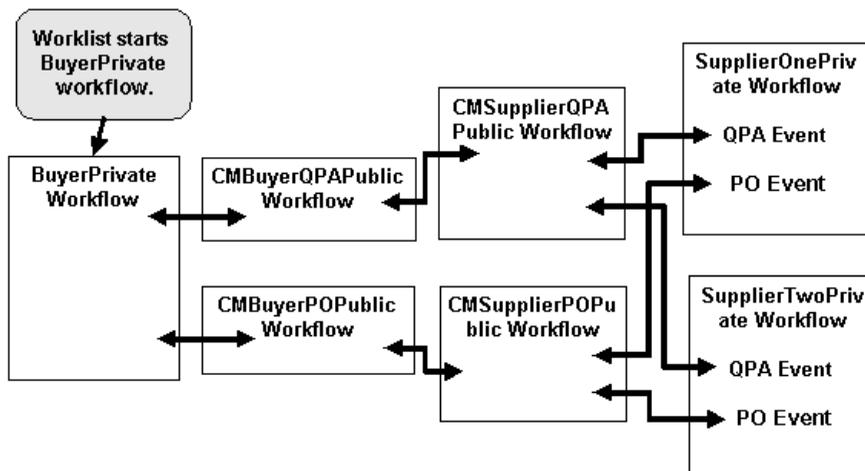
The following figure is the first of several that trace the execution flow of the Channel Master sample.

Figure 3-2   Tracing the Workflows: Steps 1-4
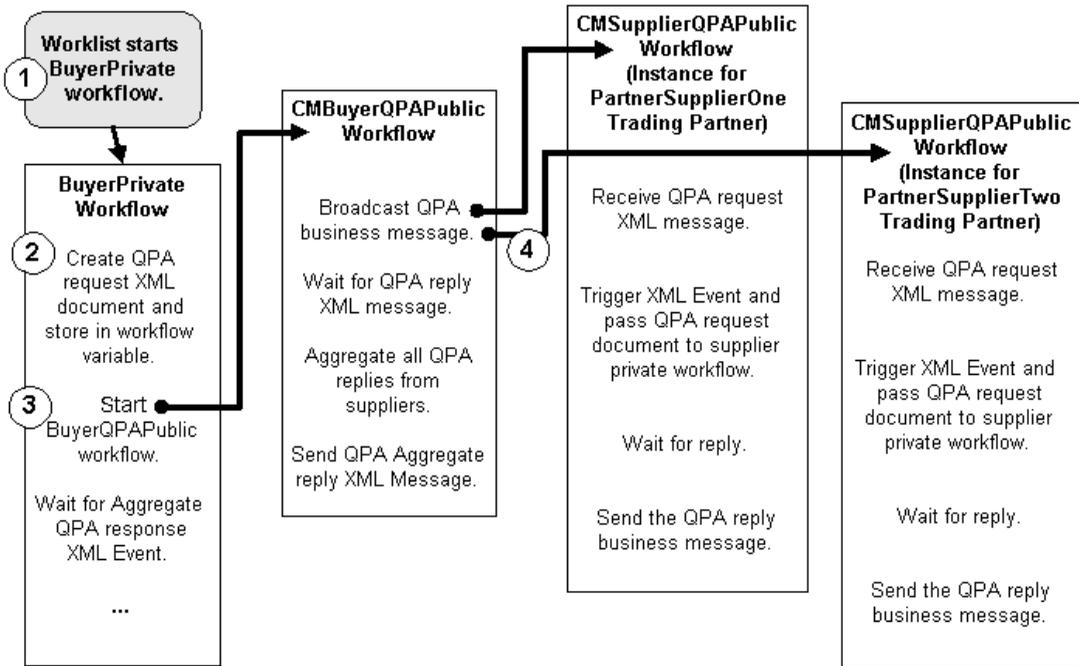
The following sequence provides details about each of the steps indicated by a corresponding number in Figure 3-2:

1. The user invokes the WebLogic Integration Worklist utility, which starts the BuyerPrivate workflow.

2. The BuyerPrivate workflow creates the QPA request XML document and stores it in a workflow variable.

3. The BuyerPrivate workflow starts the CMBuyerQPAPublic workflow.

4. The CMBuyerQPAPublic workflow broadcasts a QPA business message based on the QPA request document to all the suppliers. This broadcast causes two instances of the CMSupplierQPAPublic workflows to start: one for the PartnerSupplierOne trading partner and one for the PartnerSupplierTwo trading partner.

Two CMSupplierQPAPublic workflow instances are started because each trading partner has a collaboration agreement with the ChannelMasterHub trading partner that specifies the following: when a workflow receives a business message with the conversation definition name of CMQPAConversation and the conversation definition of 1.1, it must use the role name of CMSupplier.

The following listing shows the collaboration agreement between the ChannelMasterHub and the PartnerSupplierOne trading partner. (For more information about why the ChannelMasterHub trading partner is necessary, see "Multicast or Broadcast Messages" on page 3-23.)

Listing 3-1   Collaboration Agreement Section of Import Repository Data File

```
<collaboration-agreement
  name="CMQPAConversation|1.1|PartnerSupplierOne|ChannelMasterHub"
  global-identifier="CMQPAConversation|1.1|PartnerSupplierOne|ChannelMasterHub"
  version="1.1"
  status="ENABLED"
  conversation-definition-name="CMQPAConversation"
  conversation-definition-version="1.1">
  <party
    trading-partner-name="PartnerSupplierOne"
    party-identifier-name="PartnerSupplierOnePartyId"
    delivery-channel-name="PartnerSupplierOneDeliveryChannel"
    role-name="CMSupplier"/>
  <party
    trading-partner-name="ChannelMasterHub"
  party-identifier-name="ChannelMasterHubPartyId"
    delivery-channel-name="ChannelMasterHubDeliveryChannel"
    role-name="CMBuyer"/>
</collaboration-agreement>
```

This listing is an excerpt from the `BulkLoaderData.xml` file for the Channel Master sample. This file is used to import the data needed for the sample into the WebLogic Integration repository. It is located in the `%WLI_HOME%\samples\ChannelMaster\lib` directory on a Windows system and in `$WLI_HOME/samples/ChannelMaster/lib` on a UNIX system. Repository data can also be entered via the WebLogic Integration B2B Console.

A similar collaboration agreement for the PartnerSupplierTwo trading partner, in the import repository data file, specifies the following: when a workflow receives a business message with the conversation definition name of

CMQPAConversation and the conversation definition of 1.1, it must use the role name of CMSupplier.

The conversation definition for CMQPAConverstion specifies that for the CMSupplier role, an instance of the CMSupplierQPAPublic workflow should be started, as shown in the following listing.

Listing 3-2   Conversation Definition in the Import Repository Data File

```
<conversation-definition
      name="CMQPAConversation"
      version="1.1"
      business-protocol-name="XOCP"
      protocol-version="1.1">
      <role
            name="CMBuyer"
            wlpi-template="CMBuyerQPAPublic">
            <process-implementation wlpi-org="ORG1" />
      </role>
      <role
            name="CMSupplier"
            wlpi-template="CMSupplierQPAPublic">
            <process-implementation wlpi-org="ORG1" />
      </role>
</conversation-definition>
```

The following figure shows the execution flow for steps 5-7. Each step is described in text after the figure.

Figure 3-3   Tracing the Workflows: Steps 5-7
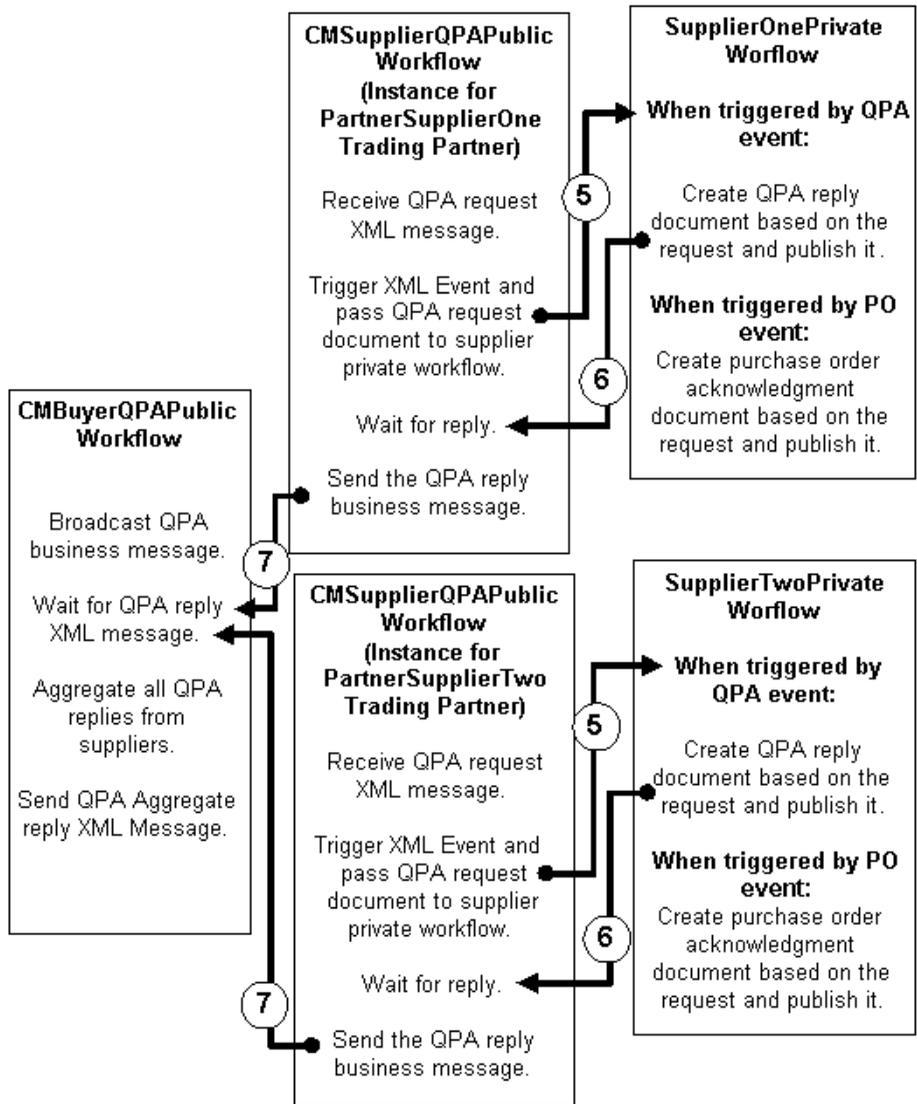
5. Each instance of the CMBuyerQPAPublic workflow posts an internal XML event. When the CMSupplierQPAPublic workflow instance for PartnerSupplierOne posts an internal XML event, this event triggers the

SupplierOnePrivate workflow to start. When the CMSupplierQPAPublic workflow instance for PartnerSupplierTwo posts an internal XML event, this event triggers the SupplierTwoPrivate workflow to start.

The PartnerSupplierOne trading partner triggers only the SupplierOnePrivate workflow. It does not trigger the SupplierTwoPrivate workflow because the start node for the SupplierOnePrivate workflow specifies that the workflow should start only if the triggering event comes from PartnerSupplierOne. For detailed directions on opening the SupplierOnePrivate workflow and viewing the conditions in which the SupplierOnePrivate workflow can be started, see "Viewing the SupplierOnePrivate Workflow" on page 3-15.

6. Both the SupplierOnePrivate and SupplierTwo Private workflows extract variable information from the QPA request, create QPA XML replies, and then post QPA XML reply messages as internal XML events.

7. Each instance of the CMSupplierQPAPublic workflow waits for a QPA XML reply message. Once a reply message is received, the CMSupplierQPAPublic workflow constructs a business message from the QPA XML reply message and sends it.

The following figure shows the execution flow for steps 8-16. Each step is described in text after the figure.

Figure 3-4   Tracing the Workflows: Steps 8-16
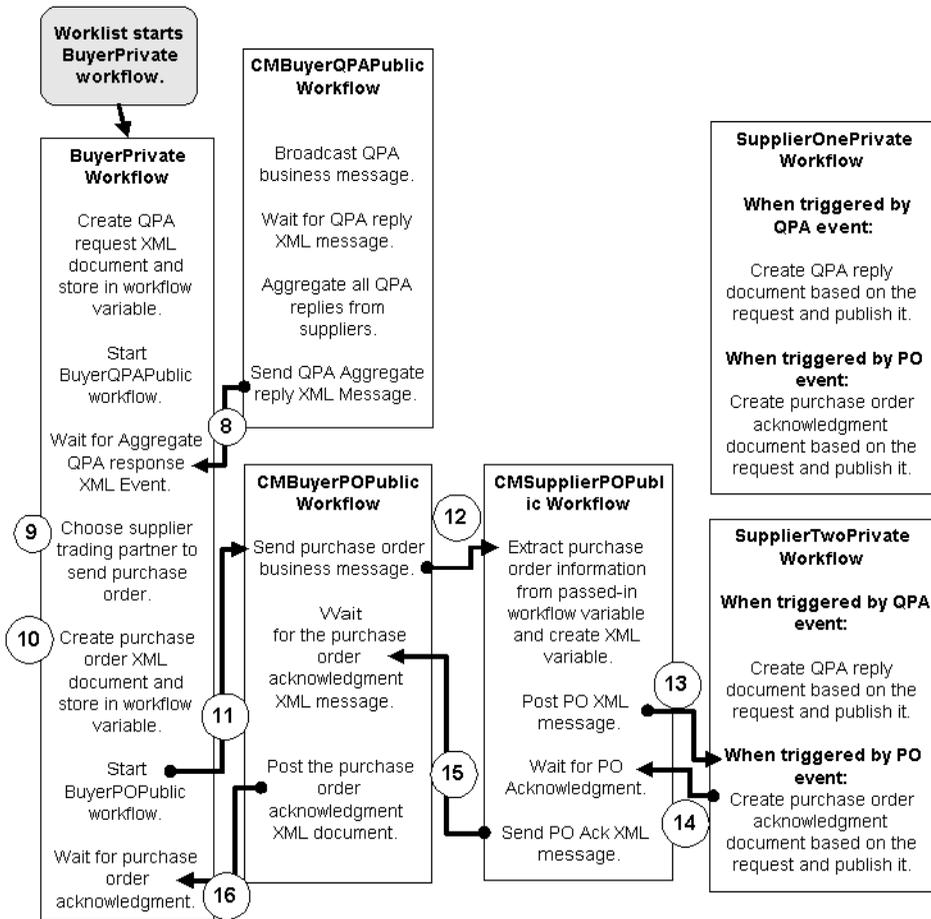
8. The CMBuyerQPAPublic workflow sends an aggregate QPA XML reply message. The message contains price and availability information from each supplier.

9. The BuyerPrivate workflow that was waiting for the QPA XML reply begins execution again and chooses a supplier based on the QPA XML reply message.

10. The BuyerPrivate workflow creates a purchase order XML document and stores it in the POXml XML workflow variable.

11. The BuyerPrivate workflow starts the CMBuyerPOPublic workflow.

12. The CMBuyerPOPublic workflow creates and sends the purchase order business message. Then it waits for the purchase order acknowledgment receipt.

13. The CMSupplierPOPublic workflow extracts the purchase order information from the passed-in POMessage workflow variable, constructs the POXml XML workflow variable, posts the XML event, and then waits for the purchase order acknowledgment. PartnerSupplierTwo offers the best price and availability, so it is chosen as the supplier.

14. The SupplierTwo Private workflow extracts the variable information from the PO request, creates a PO XML reply, and then posts the PO XML reply message as an internal XML event.

15. The CMSupplierPOPublic workflow waits for a purchase order acknowledgment. Once such an acknowledgment is received, the CMSupplierPOPublic workflow sends an XML acknowledgment message back to the CMBuyerPOPublic workflow.

16. The CMBuyerPOPublic workflow waits for a purchase order acknowledgment. Once such an acknowledgment is received, the CMBuyerPOPublic workflow extracts information from the incoming XML document and then posts the purchase order XML acknowledgment document.

17. The BuyerPrivate workflow ends. (This step is not shown in Figure 3-4.)

Viewing the SupplierOnePrivate Workflow

The SupplierOnePrivate workflow accepts XML events only from the PartnerSupplierOne trading partner. To view the SupplierOnePrivate workflow and to see how the workflow limits the XML events it accepts, complete the following steps:

Note: If the instance of the WebLogic Server started by the `RunSamples` script is running, skip to step 3.

1. Run the `RunSamples` script by completing the procedure appropriate for your platform:

- Windows:

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

- UNIX:

  a) Make sure your PATH environment variable includes the directory in which the Netscape executable (netscape) resides.

  b) Go to the WebLogic Integration home directory (the directory in which you installed WebLogic Integration). For example:

  ```
  cd /home/me/bea/wlintegration2.1
  ```

  c) Run the setenv script to set the top-level WebLogic Integration environment variables:

  ```
  . setenv.sh
  ```

  d) Run the RunSamples script:

  ```
  cd samples/bin
  RunSamples
  ```

Note: For UNIX systems, the directory in which the netscape executable resides must be included in your PATH environment variable. If it is not included, the samples launcher page cannot be displayed.

2. If the RunSamples script detects that the configuration section of the RunSamples script has been run before, the following prompt is displayed:

   ```
   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
   ```

   If you answer N to this question, the RunSamples script skips the steps for creating and populating the repository and runs only the step for booting the sample instance of the WebLogic Server.

   If you answer Y to this question, the RunSamples script recreates and repopulates the repository, and then it boots the sample instance of the WebLogic Server. When you answer Y, the RunSamples script destroys all the data currently in the repository and loads an unaltered version of the sample data into the repository. Answer Y only when the current sample data has been altered

or removed and you want a fresh or unaltered version of the sample data in the repository.

Now the `RunSamples` script starts an instance of the WebLogic Server as a background process and the samples launcher page is displayed.

3. Start the WebLogic Integration Studio by completing the procedure appropriate for your platform:

   - Windows:

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

   - UNIX:

     ```
     cd $WLINT_HOME/bin
     studio
     ```

4. Log in to the WebLogic Integration Studio using the following information:

   - Login: `joe`

   - Password: `password`

   - URL: `t3//localhost:7001`

   The main WebLogic Integration Studio window is displayed.

5. Expand the drop-down list under Organization (in the left pane) and select ORG1.
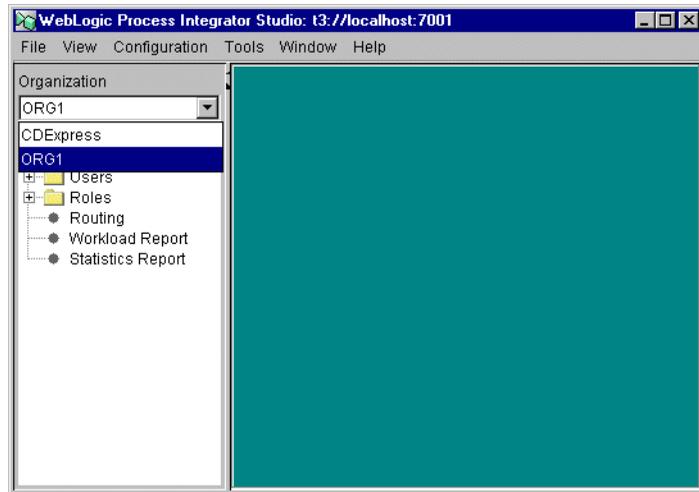
6. Expand the Templates folder in the left pane. A list of all the templates for the samples are displayed.
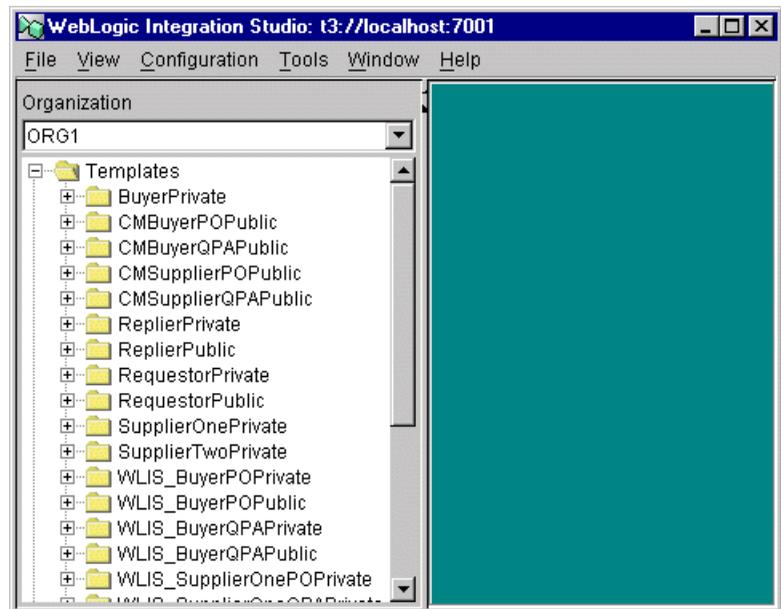
7. Expand the SupplierOnePrivate folder in the left pane.

8. Complete the following steps to open and view an instance of the SupplierOnePrivate workflow:

   a. Right-click the folder, named with a date, that is listed under the SupplierOnePrivate.
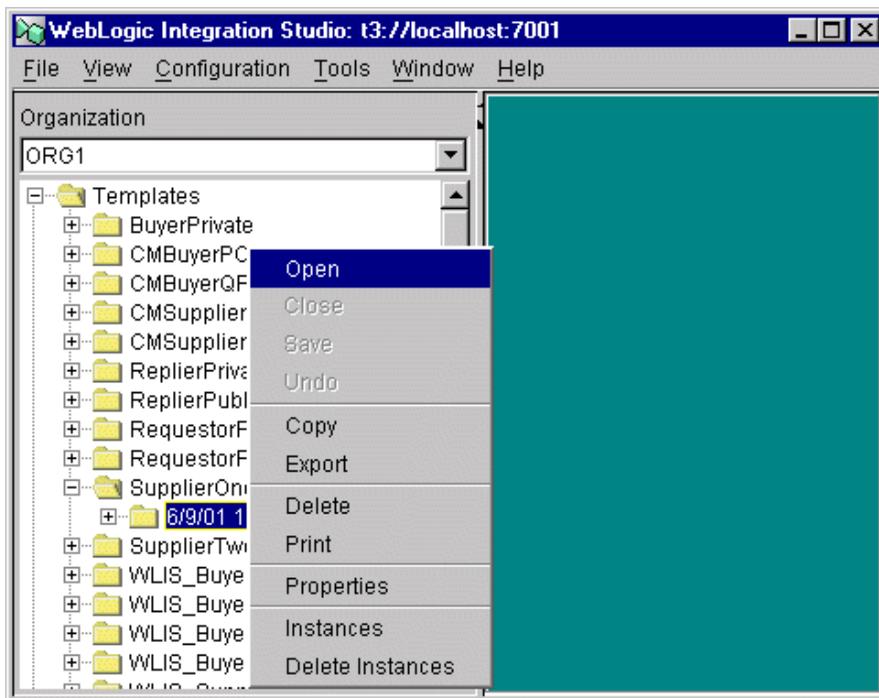
   b. Select Open.

      A graphical depiction of the workflow is displayed. It shows the start, task, decision, and event nodes that make up the SupplierOnePrivate workflow.
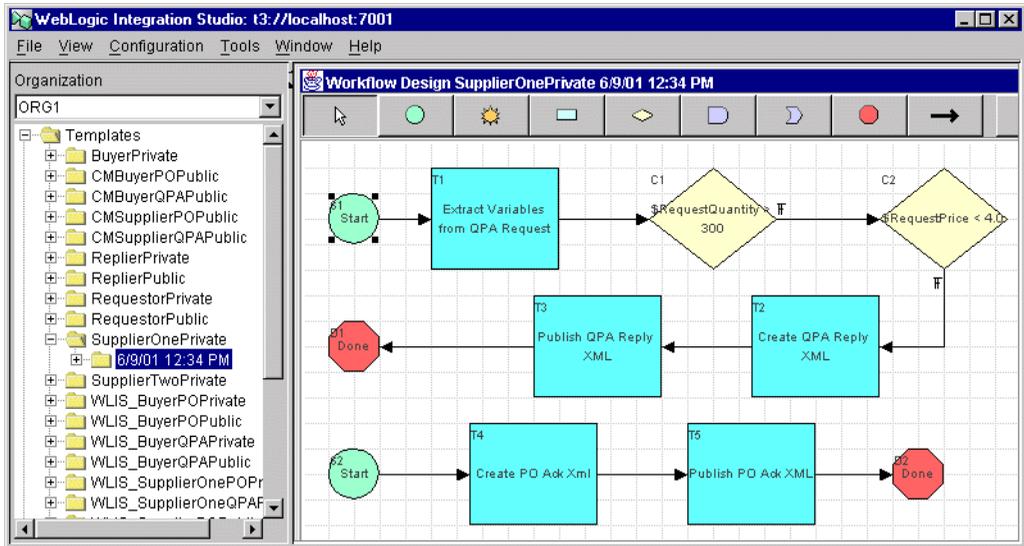
9. Double-click the Start node in the top left corner of the right pane.

The Start Properties dialog box is displayed.

Before this workflow can be started, the expression defined in the Condition field of the preceding Start Properties dialog box must be equal to true.

10. Double-click the A+B option to the right of the Condition Field. The Expression Builder dialog box is displayed.

Expression Builder

Expression

```
ToString(XPath("//CMQPARequest/@TPName"))=
"PartnerSupplierOne"
```

- ⊙ Functions
- ○ Operators
- ○ Literals

AIGetErrorMessage
AIGetResponseDocument
AIHasError
Abs
CurrentUser

XPath Wizard...

OK   Cancel   Help
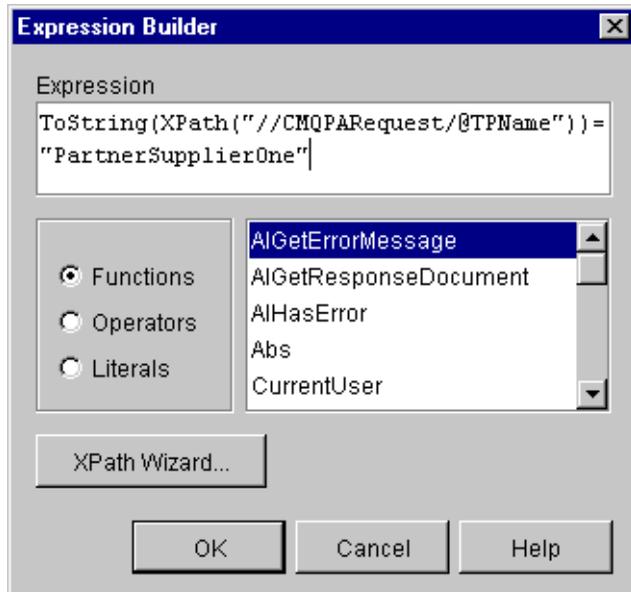
The specified expression evaluates the XML message received by the workflow to determine whether the value of the CMQPARequest TPName node is PartnerSupplierOne. If it is, the expression is true and the workflow is started.

11. If you want to continue running B2B samples or complete the steps in "Viewing the SupplierOnePrivate Workflow" on page 3-15, keep the samples launcher page open and the sample instance of the WebLogic Server running.

If you do not want to continue running B2B samples, shut down the sample instance of the WebLogic Server and exit from the browser in which you are running the samples launcher page. To shut down the WebLogic Server, complete the procedure appropriate for your platform:

- Windows:

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

- UNIX:

  ```
  cd $WLI_HOME/config/samples
  stopWebLogic
  ```

Multicast or Broadcast Messages

Multicasting, or broadcasting of messages is supported only for the XOCP business protocol and only when messages are routed through a routing proxy delivery channel (hub). It is not supported for direct point-to-point communication. Consequently, the PartnerBuyer trading partner in the sample does not send a message directly to the PartnerSupplierOne and PartnerSupplierTwo trading partners. Instead, it sends the messages through the ChannelMasterHub trading partner via the routing proxy delivery channel. The following figure shows how multicast messages are routed among trading partners.
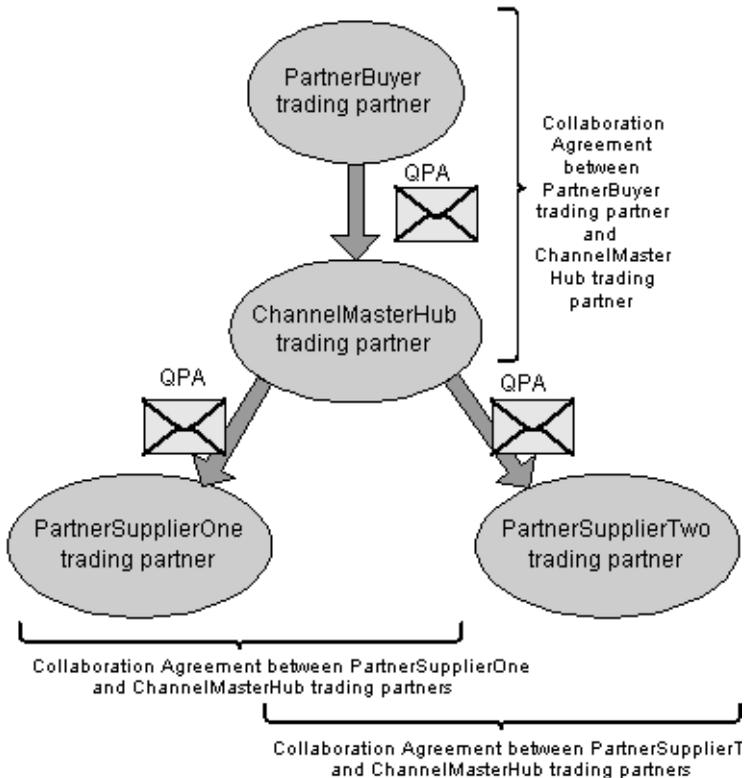
Caution: For simplicity, all the trading partners in this sample use the same WebLogic Server instance. In a production environment, the buyer and each supplier trading partner runs a separate WebLogic Server instance.

4 RosettaNet 2.0 Security Sample

The RosettaNet 2.0 Security sample shows how WebLogic Integration can be used to implement RosettaNet 2.0 PIP 3A2 and PIP 0A1 using workflows. Specifically, it shows two trading partners exchanging business messages that conform to the RosettaNet 2.0 PIP 3A2 standard.

This section includes the following topics:

- Introduction to the RosettaNet 2.0 Security Sample

- RosettaNet 2.0 Security Sample Overview

- Before Running the RosettaNet 2.0 Security Sample

- Running the RosettaNet 2.0 Security Sample

- Workflows Behind the RosettaNet 2.0 Security Sample

Introduction to the RosettaNet 2.0 Security Sample

RosettaNet is a non-profit consortium of companies that creates, implements, and promotes open e-business process standards. A RosettaNet Partner Interface Process (PIP) defines business processes between trading partners. PIP 3A2 provides an automated process that can be used by trading partners to request and provide product price and availability information.

The RosettaNet 2.0 Security sample demonstrates the implementation of PIP 3A2. It also implements PIP 0A1, which provides a mechanism for sending failure notifications. Finally, the sample demonstrates the WebLogic Integration security features required for RosettaNet 2.0 support: two-way SSL authentication, digital signatures, data encryption, and nonrepudiation.

For more information about implementing RosettaNet with WebLogic Integration, see *Implementing RosettaNet for B2B Integration*. For more information about using security with the B2B integration functionality of WebLogic Integration, see *Implementing Security with B2B Integration*.

RosettaNet publishes several documents that are helpful in understanding the RosettaNet 2.0 PIPs used in this sample. The following documents and zip files are available in the Standards section at the RosettaNet Web site (`http://www.rosettanet.org`):

■ Understanding a PIP Blueprint—Explains how to read a PIP blueprint. Available under Supporting Documents in the Standards section.

■ PIP 3A2 - Request Quote zip file—Contains PIP 3A2 specification information and DTDs. Available under Standards, PIPs, Cluster 3: Order Management, Segment 3A: Quote and Order Entry, PIP 3A2: Request Price and Availability, Version R1.3.

■ PIP 0A1 - Notification of Failure zip file—Contains PIP 0A1 specification information and DTDs. Available under Standards, PIPs, Cluster 0: RosettaNet Support, Segment 0A: Administrative, PIP 0A1: Nofication of Failure, Version R1.3.

RosettaNet 2.0 Security Sample Overview

The following sequence provides a high-level overview of the communications between PIP 3A2 trading partners in this sample:

1. A customer trading partner sends a price and availability request to a supplier trading partner. Such a request might be sent, for example, by a computer manufacturer (customer trading partner) who wants to know whether a supplier can provide a certain quantity of memory chips at a particular price. The following figure shows the request being sent. The envelope in the following figure represents an XML message that contains the price and availability request.

2. The supplier sends the customer an acknowledgment that it has received the request for price and availability.
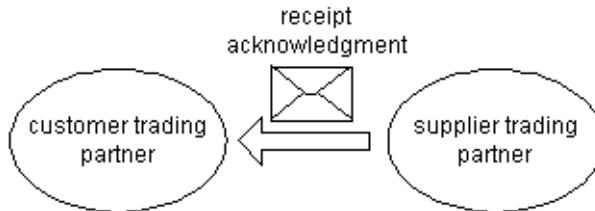
3. The supplier sends the customer a response containing the quantity and the price at which the supplier will sell the requested item.

price and availability
response

customer trading
partner

supplier trading
partner
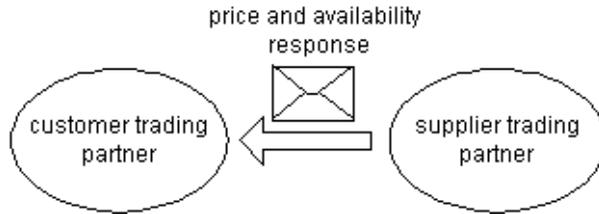
4. The customer sends a receipt acknowledgment to the supplier, indicating that it has received the price and availability response.

receipt
acknowledgment

customer trading
partner

supplier trading
partner
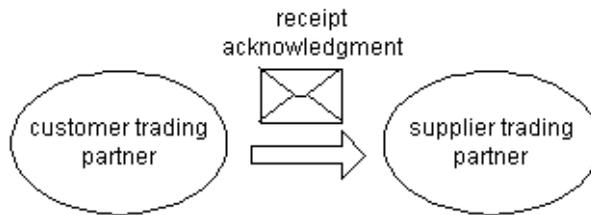
Before Running the RosettaNet 2.0 Security Sample

Before running the RosettaNet 2.0 Security sample, complete the following steps:

1. Follow the instructions in "Preparing to Run the Samples" on page 1-2.

2. Make sure the proxy settings on your browser do not prevent you from connecting to the sample 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*.

Running the RosettaNet 2.0 Security Sample

Unlike the other B2B samples provided with WebLogic Integration, the RosettaNet 2.0 Security sample does not run in the samples domain and is not started from the samples launcher page.

To run the RosettaNet 2.0 Security sample, complete the following steps:

1. Go to the WebLogic Integration home directory (the directory in which you installed WebLogic Integration) by entering the appropriate command, as shown in the following examples:

   - Windows:

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

   - UNIX:

     ```
     cd /home/me/bea/wlintegration2.1
     ```

2. Run the `setenv` script to set the top-level WebLogic Integration environment variables by entering the appropriate command for your platform:

   - Windows:

     ```
     setenv
     ```

   - UNIX:

     ```
     ./setenv.sh
     ```

3. Go to the RosettaNet 2.0 Security sample `bin` directory by entering the appropriate command for your platform:

   - Windows:

     ```
     cd samples\RN2Security\bin
     ```

   - UNIX:

     ```
     cd samples/RN2Seurity/bin
     ```

4. Run the `RunRN2Security` script with one of the following options: cloudscape, mssql, or oracle.

   The database specified on the command line must match the database that you entered using the installer for the samples domain.

For example, if oracle was specified for the samples domain, enter:

```
RunRN2Security oracle
```

Wait until both instances of the WebLogic Server finish booting before starting the next step. (The `RunRN2Security` script starts two instances of WebLogic Server as background processes.) When the servers finish booting, the following log message is displayed in your WebLogic Server console window:

```
RunRN2Security execution successful
```

Warning: Unlike the other B2B samples, the RosettaNet 2.0 Security sample does not support the use of the WebLogic Integration Database Configuration Wizard to switch the database to be used with the samples domain.

Note: The Cloudscape database is not supported on UNIX systems.

5. Start a browser. Load the QPASubmit JSP page by entering the following URL:

```
http://localhost:7501/QPASubmit.jsp
```

The QPA (RosettaNet 2.0 PIP 3A2) Submission page is displayed.
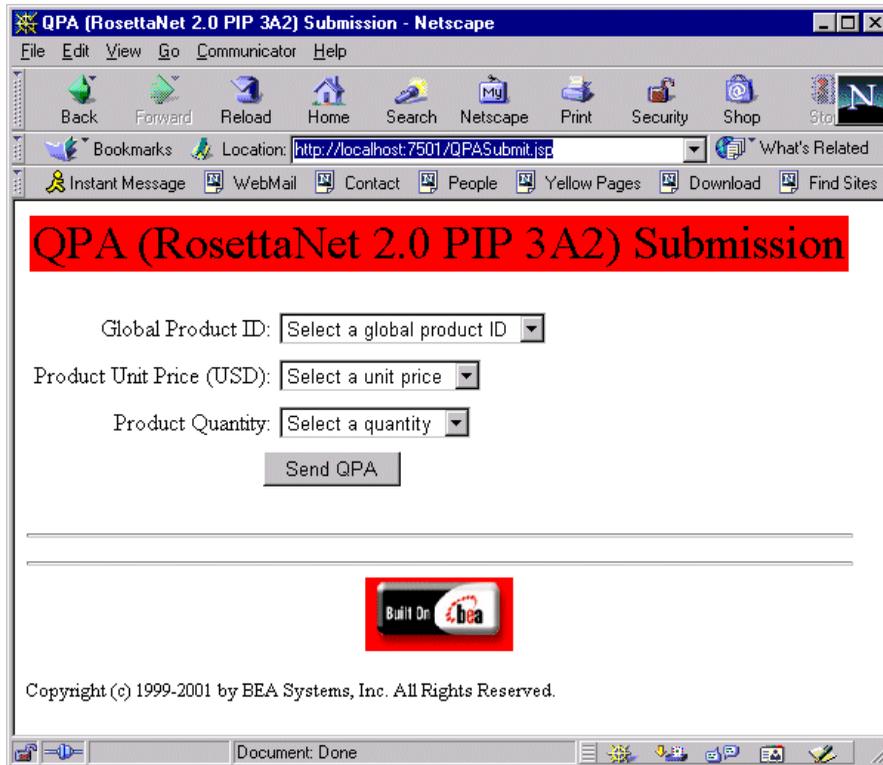
6. Select a Global Product ID, a Product Unit Price, and a Product Quantity. Click Send QPA.

   The following information about the status of the submission and responses to it are displayed in the browser.

Submit Status

The following QPA entries have been sent successfully:
- Global Product ID = *12345678901234*
- Unit Price = *249.95*
- Quantity = *1000*

Response Status

The following replies have been received:
- Quote from: *Manufacturer(DUNS Number): 987654321*
- Product ID: *12345678901234*
- Quantity Available: *1000*
- Unit Price: *249.95*

7. If you want to complete the steps described in "A Peek at the Workflows" on page 4-11 at this time, keep both instances of the WebLogic Server running.

   If you do not want to complete the steps described in "A Peek at the Workflows" on page 4-11 at this time, exit your browser and shut down both instances of the WebLogic Server by completing the following procedure:

   ```
   StopRN2Security
   ```

Workflows Behind the RosettaNet 2.0 Security Sample

The RosettaNet 2.0 Security sample shows how WebLogic Integration can be used to implement RosettaNet 2.0 PIP 3A2 and PIP 0A1 using workflows. A public process is part of a formal contract between trading partners that specifies the content and semantics of the messages they exchange. The public process between the two public workflows (PIP3A2_Customer_RN2 and PIP3A2_Supplier_ RN2) is the WebLogic

Integration implementation of the PIP 3A2 standard. A private process is specific to a particular business organization and is not visible outside that organization. The public workflows that implement PIP 3A2 are represented by the shaded rectangles in Figure 4-1. The private process is represented by the unshaded rectangles in Figure 4-1.

The following figure shows different interactions between various RosettaNet security sample workflows.

Figure 4-1   Interactions Among RosettaNet 2.0 Security Workflows
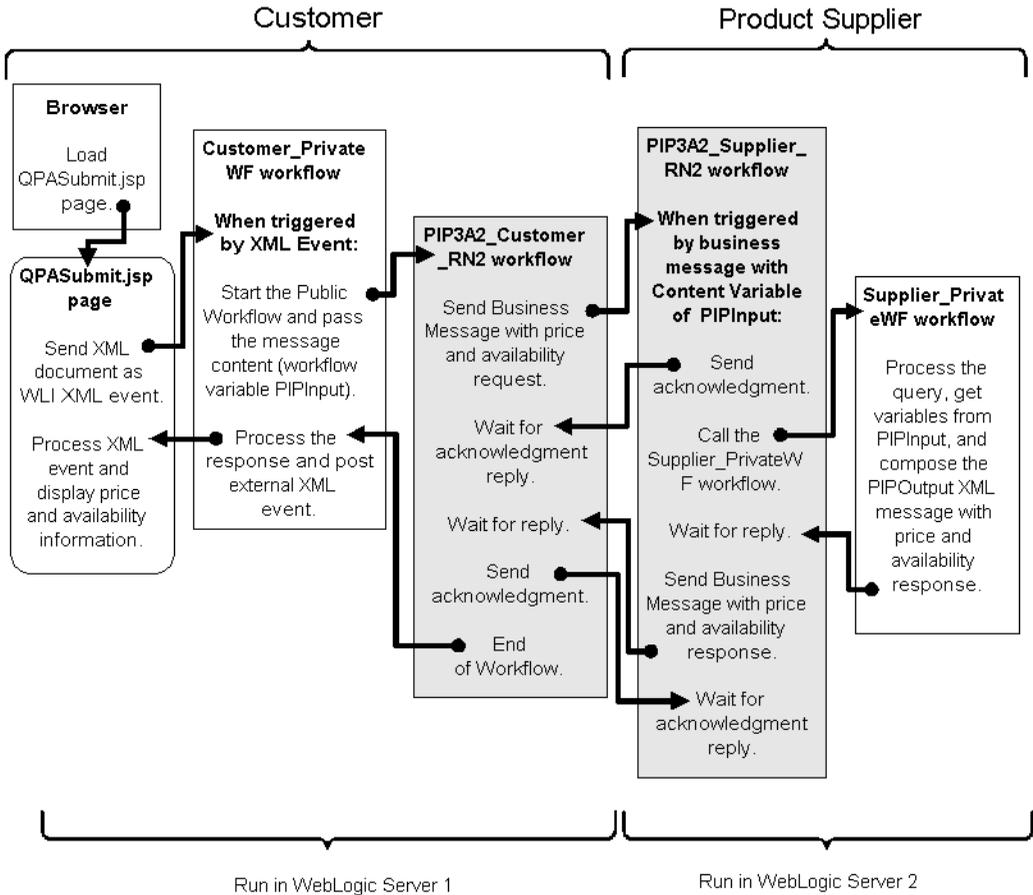
Note: The flow of logic shown in the figure illustrates only the sequence of workflows that is executed if no errors occur. This sequence does not contain any error-handling logic. For example, PIP 0A1, the workflow for issuing failure notifications, is not shown.

Before the flow of execution can start, two instances of the WebLogic Server must be booted. (One instance for the customer and the other for the supplier.) The flow of execution for this sample starts at the Browser square in the upper left corner of Figure 4-1. The following steps trace a portion of the flow of execution:

1. To start the flow of execution, the user launches a browser and loads the QPASubmit.jsp page.

2. The user then enters values on the JSP page and selects Submit.

3. The selection of Submit triggers an XML document to be sent as a WebLogic Integration XML event.

4. This event, in turn, triggers the Customer_PrivateWF workflow.

5. The first item in the Customer_PrivateWF workflow is executed.

6. As a result, the first action in the PIP3A2_Cusotmer_RN2 workflow (Send Business Message with price and availability request) is executed.

To trace the remainder of the execution flow, see Figure 4-1.

A Peek at the Workflows

To peruse the sample private and public workflows, complete the following steps:

1. Go to the WebLogic Integration home directory (the directory in which you installed WebLogic Integration) by entering the command appropriate for your platform, as shown in the following examples:

   - Windows:

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

   - UNIX:

     ```
     cd /home/me/bea/wlintegration2.1
     ```

2. Run the setenv script to set the top-level WebLogic Integration environment variables by entering the command appropriate for your platform:

   - Windows:

     ```
     setenv
     ```

- UNIX:

  ```
  ./setenv.sh
  ```

3. Complete this step only if you have not booted two instances of the WebLogic Server by running the `RunRN2Security` script as described in "Running the RosettaNet 2.0 Security Sample" on page 4-5. Boot two instances of the WebLogic Server by performing the procedure appropriate for your platform and database, as shown in the following examples:

   - Windows:

     ```
     cd %WLI_HOME%\samples\RN2Security\bin
     RunRN2Security oracle
     ```

   - UNIX:

     ```
     cd $WLI_HOME/samples/RN2Security/bin
     RunRN2Secuirty oracle
     ```

     Wait until both instances of the WebLogic Server finishes booting before starting the next step. When the servers finish booting, the following log message is displayed in your WebLogic Server console window.

     ```
     RunRN2Security execution successful
     ```

4. Start the WebLogic Integration Studio by performing the procedure appropriate for your platform:

   - Windows:

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

   - UNIX:

     ```
     cd $WLINT_HOME/bin
     studio
     ```

5. Log in to the WebLogic Integration Studio using the following information:

   - Login: `joe`
   - Password: `password`
   - URL: `t3//localhost:7501`

   The main WebLogic Integration Studio window is displayed.

6. Expand the drop-down list under Organization (in the left pane) and select ORG1.
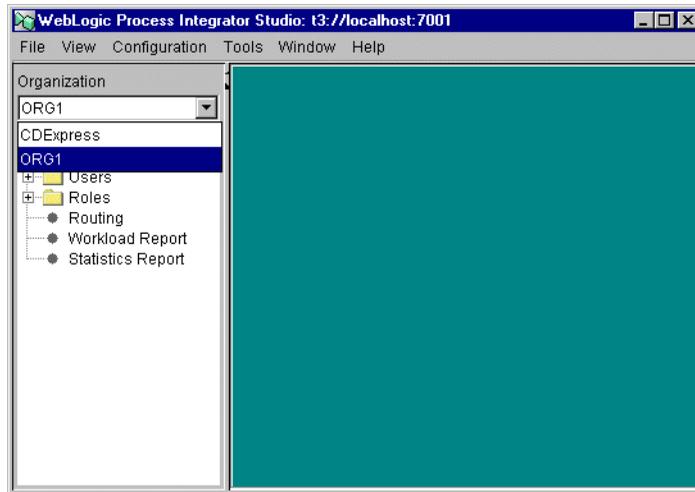
7. Expand the Templates folder in the left pane. A list of all the templates for the sample is displayed.
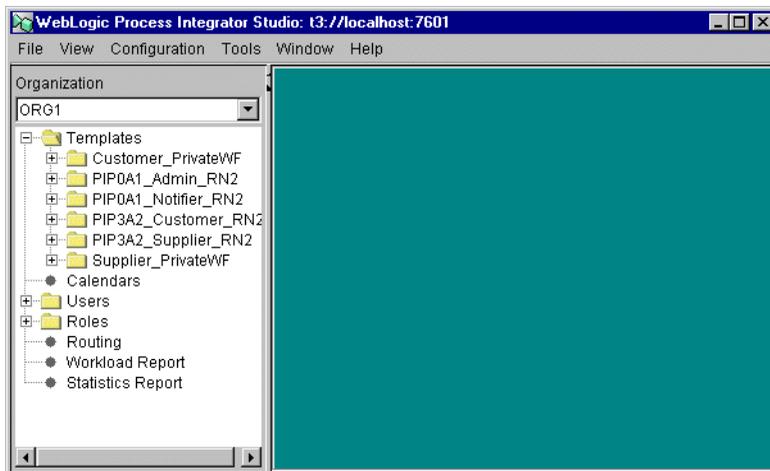
8. Expand the Customer_PrivateWF folder in the left pane.

9. Complete the following steps to open and view an instance of the Customer_PrivateWF workflow:

a. Right-click the folder, named with a date and time, in the Customer_PrivateWF folder. A menu is displayed.
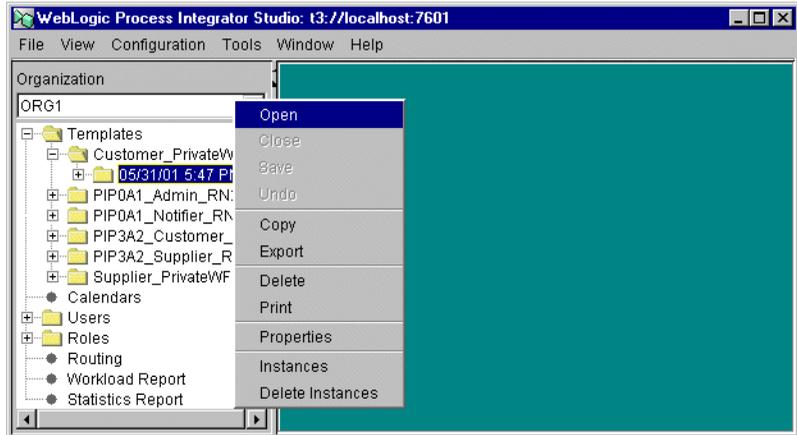
b. Select Open.

The start, task, decision, and event nodes that make up the Customer_PrivateWF workflow are displayed.

10. Double-click the Start the Public Workflow task. The Task Properties window is displayed.
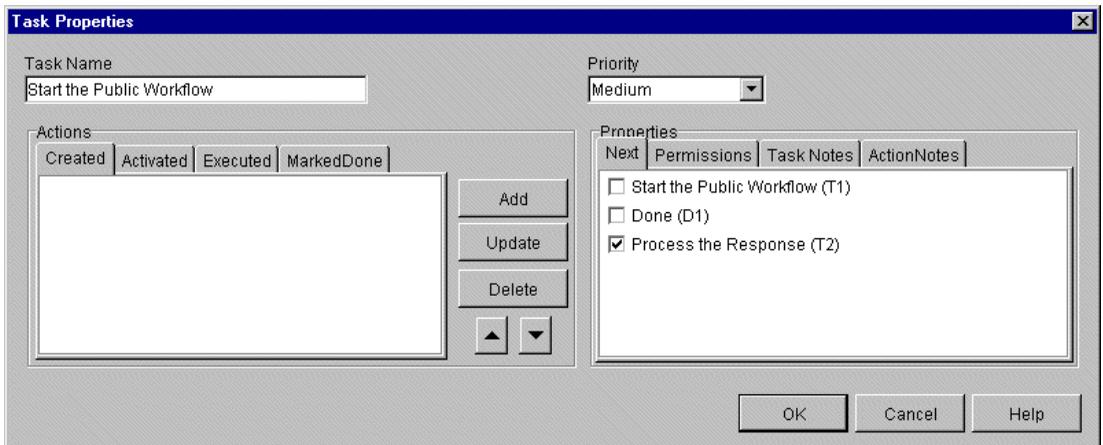
11. Select the Activated tab.
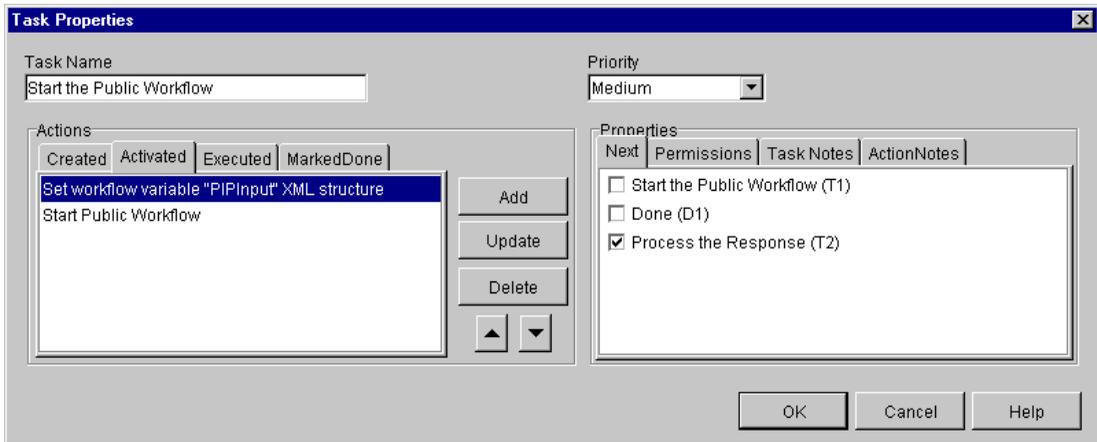
The actions that make up the Start the Public Workflow task are displayed.

12. Double-click Set workflow variable "PIPInput" XML structure to display the Set Workflow Variable window. This window shows how the PIPInput XML workflow variable is composed. (In this sample, PIPInput has been composed already; these steps show how to view it.) PIPInput is a mandatory WebLogic Integration RosettaNet Input workflow variable. It must be set in the private workflow before the public workflow is invoked. The format of this variable is XML that conforms to the RosettaNet PIP DTD for the PIP message being implemented. In this sample, the XML must conform to the `3A2PriceAndAvailabilityQueryMessageGuidline.dtd`. This DTD, supplied by RosettaNet, defines the content of the first message that the Customer trading partner passes to the Supplier trading partner. (See step 1 in the "RosettaNet 2.0 Security Sample Overview" on page 4-3.)

In this sample, the private Customer_PrivateWF workflow sets the contents of the PIPInput workflow variable and calls the PIP3A2_Customer_RN2 workflow. The PIP3A2_Customer_RN2 workflow uses the contents of the PIPInput variable to construct an XML business message. This message is sent to the PIP3A2_Supplier_RN2 workflow when the Send Business Message Action is invoked. For a complete list of RosettaNet Template variables, see "RosettaNet Template Variables" in "Using Workflows with RosettaNet" in *Implementing RosettaNet for B2B Integration*.

13. Expand the following nested nodes in the XML tree:

- Pip3A2PriceAndAvailabilityQuery

- ProductPriceAndAvailabiltyQuery

- ProductPriceAndAvailability

- ProductLineItem
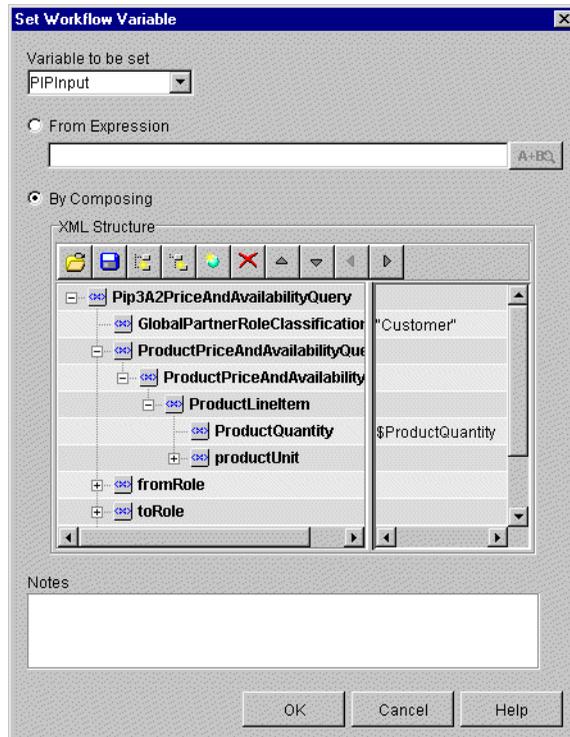
The values of two important nodes are set in the right pane:

- The GlobalPartnerRoleClassificationCode XML node is set to the string Customer.

- The ProductQuanity node is set to the value of the ProductQuantity workflow variable. The product quantity (the number of items requested by the customer) is set on the QPASubmit JSP page and passed to the workflow through the ProductQuantity workflow variable.

Note: This procedure is just one way of populating the PIPInput variable. The PIPInput variable can also be populated by reading in values from a file or by passing in the entire contents of an XML document.

14. Click Cancel in the Set Workflow Variable window.

15. In the Task Properties window, double-click Start Public Workflow on the Activated tab.
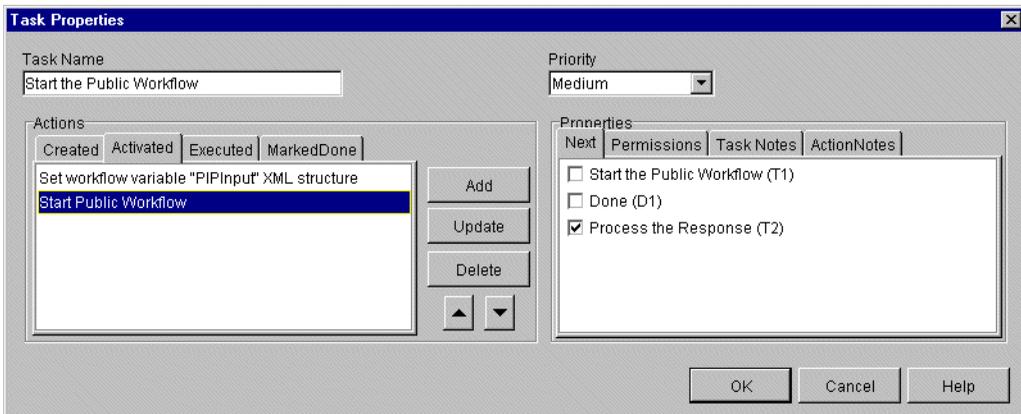
The Start Public Workflow window is displayed.

In this window, parameters are defined for the Start the Public Workflow action as follows:

- Conversation Name is set to 3A2.

- Conversation Version is set to 1.3.

- Role in Conversation is set to Customer.

In addition, two TP (trading partner) Names, RNBuyer and RNSeller, are specified in the Parties field. WebLogic Integration uses the conversation name,

conversation version, conversation role, and the trading partners specified as parties to locate the appropriate collaboration agreement to use for an action.

For the preceding action, WebLogic Integration searches the active collaboration agreements in the repository for a collaboration agreement between two trading partners named RNBuyer and RNSeller that specifies a conversation named 3A2, a conversation version of 1.3, with the role of customer. The following excerpt from the rn2_peer1_sec.xml file defines the collaboration agreement that fits the criteria for the preceding Start Public Workflow action.

Listing 4-1   Collaboration Agreement in the Import Repository Data File

```
<collaboration-agreement
    name="RN2|9.9|RosettaNet2|100"
    global-identifier="RN2|9.9|RosettaNet2|RNBuyer|RNSeller|102"
    version="1.0"
    status="ENABLED"
    conversation-definition-name="3A2"
    conversation-definition-version="1.3">
    <party
        trading-partner-name="RNBuyer"
        party-identifier-name="RNBuyerPID"
        delivery-channel-name="RNBuyerChannel"
        role-name="Customer"/>
    <party
        trading-partner-name="RNSeller"
        party-identifier-name="RNSellerPID"
        delivery-channel-name="RNSellerChannel"
        role-name="Product Supplier"/>
</collaboration-agreement>
```

The collaboration agreement defines the name and version of the conversation definition to be used between the specified parties. The collaboration agreement in Listing 4-1 specifies use of the conversation definition name of 3A2 and the conversation definition version of 1.3 between the RNBuyer trading partner in the role of Customer and the RNSeller trading partner in the role of Product Supplier.

Using the name and version of the conversation definition, as well as the assigned roles, WebLogic Integration can determine which workflow template to start. The conversation definition in Listing 4-2 specifies that for conversations named 3A2, with a version number of 1.3, and with a trading partner in the

customer role, an instance of the PIP3A2_Customer_RN2 workflow template will be started.

Listing 4-2   Conversation Definition in the Import Repository Data File

```
<conversation-definition
    name="3A2"
    version="1.3"
    business-protocol-name="RosettaNet"
    protocol-version="2.0">
    <role
        name="Customer"
        wlpi-template="PIP3A2_Customer_RN2">
        <process-implementation wlpi-org="ORG1"/>
    </role>
    <role
        name="Product Supplier"
        wlpi-template="PIP3A2_Supplier_RN2">
        <process-implementation wlpi-org="ORG1"/>
    </role>
</conversation-definition>
```

Thus the Start Public Workflow action triggers the PIP3A2_Customer_RN2 workflow, in this sample.

The conversation name and version number defined in the Start Public Workflow are the PIP name and version defined by RosettaNet for PIP 3A2. These parameters, along with values specified for the roles and trading partners, correspond to the conversation name, conversation version, role, and trading partners that are registered in the repository.

16. Select the Workflow tab.
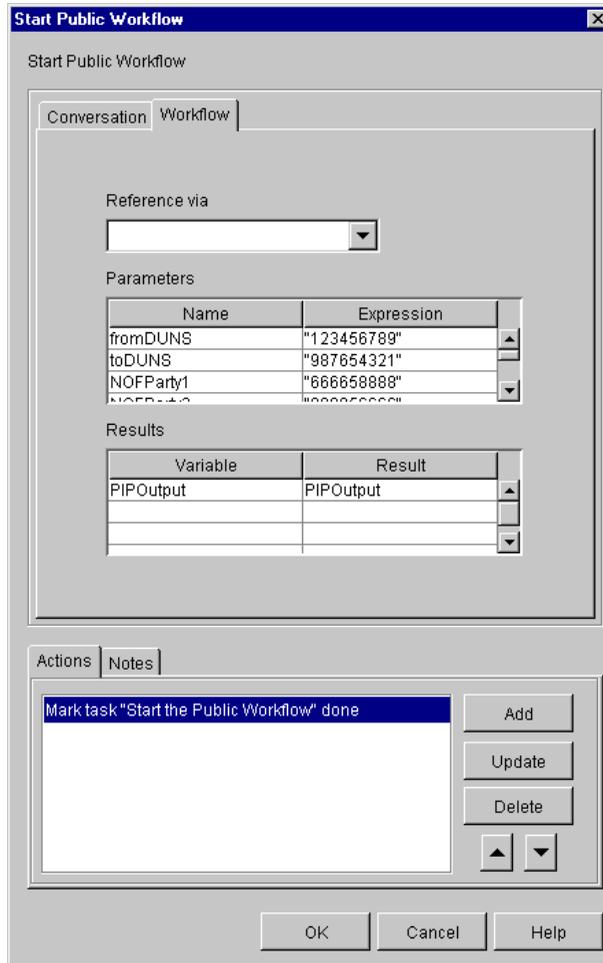
The template variables to be passed to and received from the public PIP workflow are defined on the Workflow tab:

- The template variables passed to the public PIP workflow are listed under Parameters.

- The template variables returned by the public PIP workflow are listed under Results.

In this sample, the template variable `fromDUNS` is set by the
Customer_PrivateWF in the Start Public Workflow action. It is passed to the
PIP3A2_Customer_RN2 workflow when the PIP3A2_Customer_RN2 is invoked
by the Customer_PrivateWF workflow. Some template variables for RosettaNet
2.0 are mandatory; others are optional.

The `fromDUNS` variable, which defines the DUNS number of the sender, is
mandatory. (A DUNS number is a unique nine-digit identifier assigned to a
business entity by Dun & Bradstreet.) The DUNS number specified in the
`fromDUNS` variable must match the business ID defined in the repository for that
trading partner. For a complete list of RosettaNet template variables, see
"RosettaNet Template Variables" in "Using Workflows with RosettaNet" in
Implementing RosettaNet for B2B Integration.

`PIPOutput` is a mandatory template variable that contains the service content of
the received message. It is set by the PIP3A2_Customer_RN2 workflow and is
passed to the Customer_PrivateWF workflow when the PIP3A2_Customer_RN2
workflow returns control back to the workflow that called it:
Customer_PrivateWF.

17. Click Cancel in the Start Public Workflow window.

18. Click Cancel in the Task Properties window.

19. In the left pane of the main Studio window, expand the PIP3A2_Customer_RN2
    folder. (The PIP3A2_Customer_RN2 workflow is called from the
    Customer_PrivateWF workflow.)
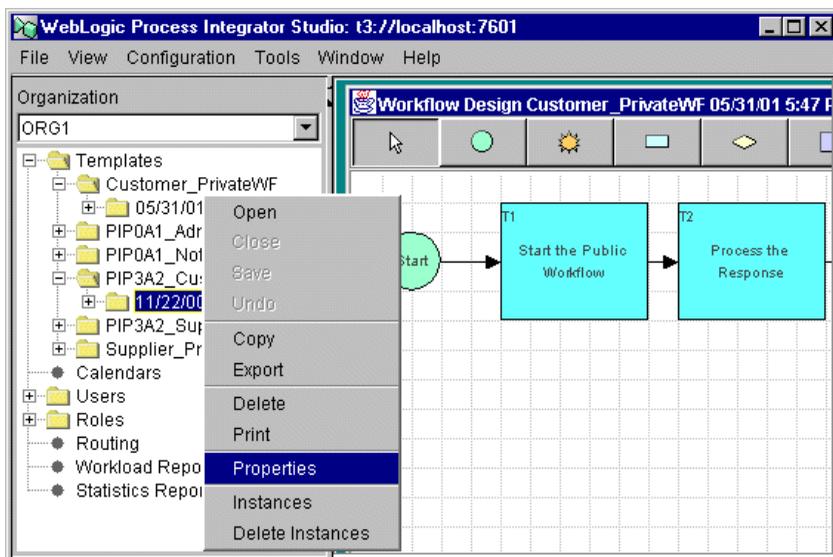
20. To view the properties of the PIP3A2_Customer_RN2 workflow, right-click the folder, named with a date and time, in the PIP3A2_Customer_RN2 folder. A menu is displayed. Select Properties.

21. The Template Definition PIP3A2_Customer_RN2 dialog box is displayed.

22. Select the B2B Integration tab.

    The Conversation tab (nested on the B2B Integration tab) is displayed.

The data entered on this tab specifies that the PIP3A2_Customer_RN2 workflow should be started if a Start Public Workflow action is invoked with the specified conversation and role.

The conversation name, version, and role defined in this window:

- Match the PIP name, version, and role defined by RosettaNet for PIP 3A2.

- Correspond to the conversation name, version, and role that are registered in the repository.

23. Click OK.

24. Complete the following steps to open and view a PIP3A2_Customer_RN2
    workflow instance:

    a. Right-click the folder, named with a date and time, in the
       PIP3A2_Customer_RN2 folder. A menu is displayed.
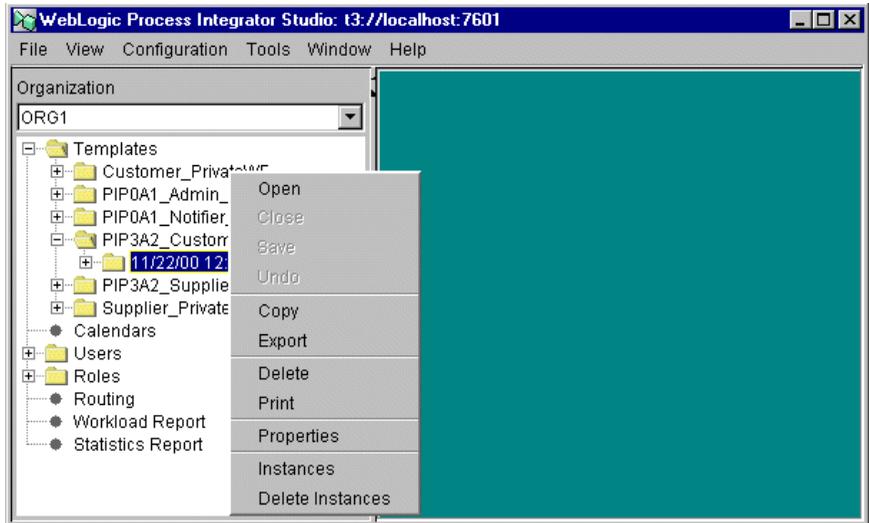
    b. Select Open.

    The start, task, decision, and event nodes that make up the
    PIP3A2_Customer_RN2 workflow are displayed.

    The Start node is the first task that is executed in this workflow.
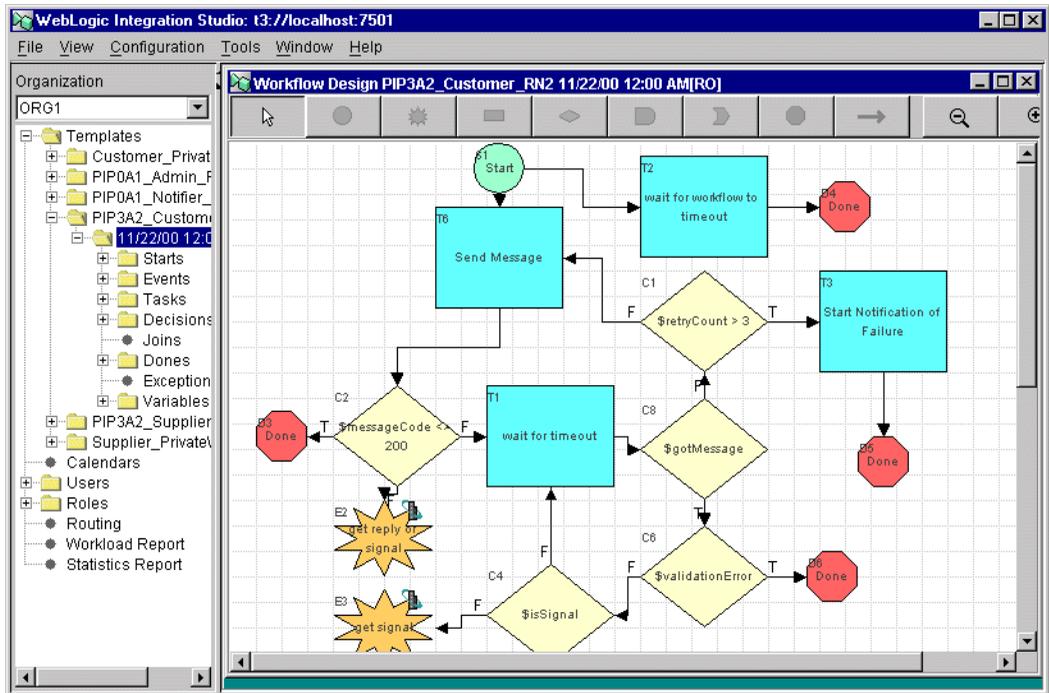
25. Double-click the Send Message task. The Task Properties window is displayed.

26. Select the Activated tab.
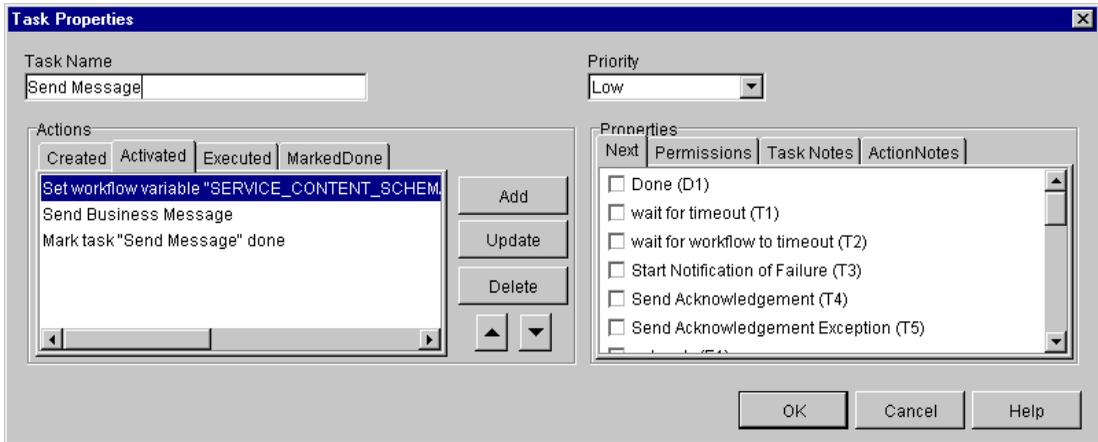
The actions that make up the Send Message task are listed on the Activated tab.

27. In the left pane of the Task Properties window, double-click the Send Business Message action. This action sends an XML business message, which is based on the contents of the PIPInput XML workflow variable. (This variable was defined earlier in the private Customer_PrivateWF workflow.)

 Note: This step corresponds to step 1 in "RosettaNet 2.0 Security Sample Overview" on page 4-3.

28. Exit the WebLogic Integration Studio: From the Studio menu bar, choose File→Exit.

5 Trading Partner Lightweight Client Sample

The Trading Partner Lightweight Client sample demonstrates two communication methods used when one or more trading partner hosts no BEA software: browsers and file sharing. The sample is based on business practice functions and business processes for a requestor and replier communicating via either a browser or a file-sharing client.

This section includes the following topics:

Overview of the Lightweight Client Sample

Most B2B integration conversations involve two trading partners that have both installed BEA WebLogic Integration. Browser and file-sharing clients provide a way to communicate with trading partners who have not installed BEA software. The Lightweight Client sample demonstrates communication between a requestor trading partner using a browser client, and a replier trading partner using a file-sharing client.

Purpose of the Sample

The Lightweight Client sample demonstrates how business communication can take place between a requestor and one or more repliers that do not have WebLogic Integration installations. This communication is accomplished using two types of lightweight clients that are configured in a remote or host B2B integration installation:

- Browser client—Uses JSP served up by predefined WebLogic Integration business processes to initiate a conversation, and to send and receive messages. The Web host facilitates communication via JSP delivered, on demand, to the browser client. The browser client uses the WebLogic Integration JSP tag libraries to send and check messages through the B2B integration mailbox interface.

- File-sharing client—The requestor puts a message in a preconfigured mailbox located on the Web host. The replier uses a file-sharing client provided by WebLogic Integration to transfer messages between its mailboxes and file-sharing directories. The party using a file-sharing client in the conversation must have a pre-existing FTP installation.

In this sample, a business operation called by the replier's private workflow gets a request from a preconfigured directory, creates a reply based on the request, and puts the reply into a preconfigured output directory. An actual FTP server is not used; the sample uses the business operation to simulate an FTP server.

Lightweight Client Sample Scenario and Diagrams

The Lightweight Client sample scenario involves two trading partners (one requestor and one replier) who communicate via a remote WebLogic Integration installation that is preconfigured to handle lightweight clients. The requestor sends a request for multiplication of two integer numbers. The replier performs the multiplication and returns the product to the requestor.

Use the browser of your choice on any machine other than the one on which you are running the samples instance of WebLogic Integration. As a result, the requestor does not have a local WebLogic Integration installation. Instead, it uses a browser to access the JSPs that reside on a remote installation of WebLogic Integration.

The JSPs create mailboxes and send XML requests using the JSP tag libraries provided by WebLogic Integration. The same JSPs are also used to check messages in the requestor's mailbox, and to delete messages from the requestor's and replier's mailboxes.

Note: The default configuration of the Lightweight Client sample specifies that the samples instance of WebLogic Integration, the browser client, and the FTP Client run on the same machine. In a production deployment of the Lightweight Client sample, however, WebLogic Integration, the browser client, and the FTP client run on different machines.

The replier also has no WebLogic Integration installation. It uses a third-party FTP server application to communicate with some trading partners. It may also use a proprietary mechanism for handling messages. The replier communicates with the requestor via a file-sharing client.

The Lightweight Client sample demonstrates the following events:

1. Requestor's private workflow is triggered by an XML event sent from a JSP tag in the browser client.

2. Replier's public workflow is triggered by a reply placed in the replier's out mailbox by the file-sharing client.

3. Requestor's message and replier's message are put in the appropriate mailboxes.

In the Lightweight Client sample, four preconfigured mailboxes are used. Each trading partner uses one inbox and one outbox, as follows:

- Requestor using a browser client uses `BrowserTP1_Inbox` and `BrowserTP1_Outbox`.

- Replier using a file-sharing client uses `FtpTP1_Inbox`, and `FtpTP1_Outbox`.

Figure 5-1   Lightweight Client Deployment
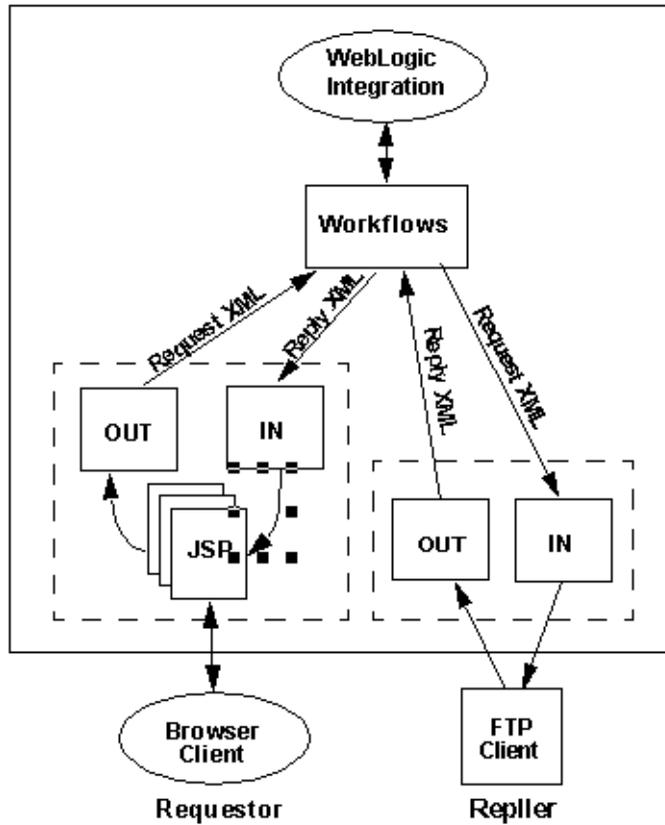
Before Running the Lightweight Client Sample

Before running the Lightweight Client sample, complete the following steps:

1. Follow the instructions in "Preparing to Run the Samples" on page 1-2.

2. Make sure the proxy settings on your browser do not prevent you from connecting to the sample 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*.

Running the Lightweight Client Sample

To run the Lightweight Client sample, complete the following steps:

1. Run the `RunSamples` script by completing the procedure appropriate for your platform:

   - Windows:

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

   - UNIX:

     a) Make sure your `PATH` environment variable includes the directory in which the Netscape executable (`netscape`) resides.

     b) Go to the WebLogic Integration home directory (the directory in which you installed WebLogic Integration). For example:

     `cd /home/me/bea/wlintegration2.1`

     c) Run the `setenv` script to set the top-level WebLogic Integration environment variables:

```
. setenv.sh
```

d) Run the `RunSamples` script:

```
cd samples/bin
RunSamples
```

Warning: For UNIX systems, the directory in which the `netscape` executable resides must be included in your `PATH` environment variable. If it is not included, the samples launcher page cannot be displayed.

2. If the `RunSamples` script detects that the configuration section of the `RunSamples` script has been run before, the following prompt is displayed:

```
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
```

If you answer `N` to this question, the `RunSamples` script skips the steps for creating and populating the repository and runs only the step for booting the sample instance of WebLogic Server.

If you answer `Y` to this question, the `RunSamples` script recreates and repopulates the repository, and then it boots the sample instance of WebLogic Server. When you answer `Y`, the `RunSamples` script destroys all the data currently in the repository and loads an unaltered version of the sample data into the repository. Answer `Y` only when the current sample data has been altered or removed and you want a fresh or unaltered version of the sample data in the repository.

Now the `RunSamples` script starts an instance of WebLogic Server as a background process and the samples launcher page is displayed.
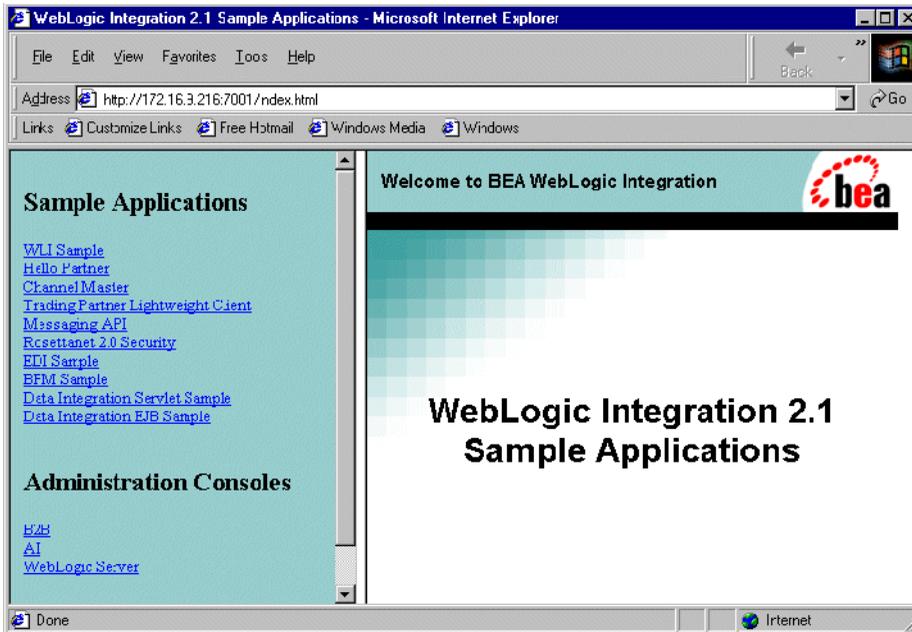
3. In the left pane, click the link for the Trading Partner Lightweight Client application under Sample Applications. The Lightweight Client Main Page is displayed in the right pane.

4. Launch the file-sharing client by completing the steps appropriate for your platform:

- Windows:

```
cd %WLI_HOME%\samples\bin
startFSClient.cmd
```

- UNIX:

```
cd $WLI_HOME/samples/bin
startFSClient
```

Note: If you start the file-sharing client before the first JSP is loaded, you get a repeated `Mailbox not found` exception. These exceptions can be ignored because mailboxes are created when the main JSP is loaded for the first time. Once the mailboxes are created, these exceptions no longer appear.

5. Select the following option: Go to the request input page. The Request for Multiplication page is displayed.

6. Now send a message to `BrowserTP1_Outbox`, manually, by completing the following steps:

   a. Select two integers.

   b. Click Send Multiplication Request.

   Note: As a side effect of this step, `SendmsgTag` and the wrapper Mailbox API are tested.

   Your display is updated, as shown in the following figure.
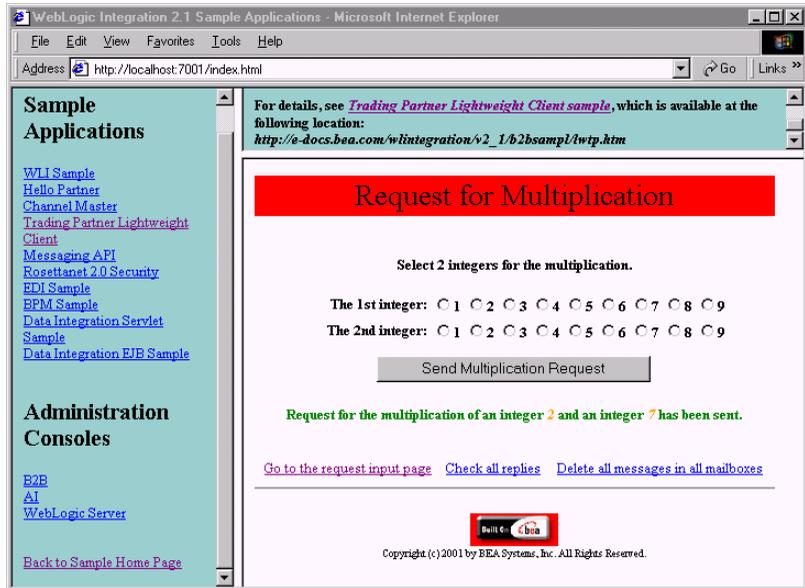
7. To check the replies that have arrived in your incoming mailbox, click Check all replies. A list of replies is displayed.
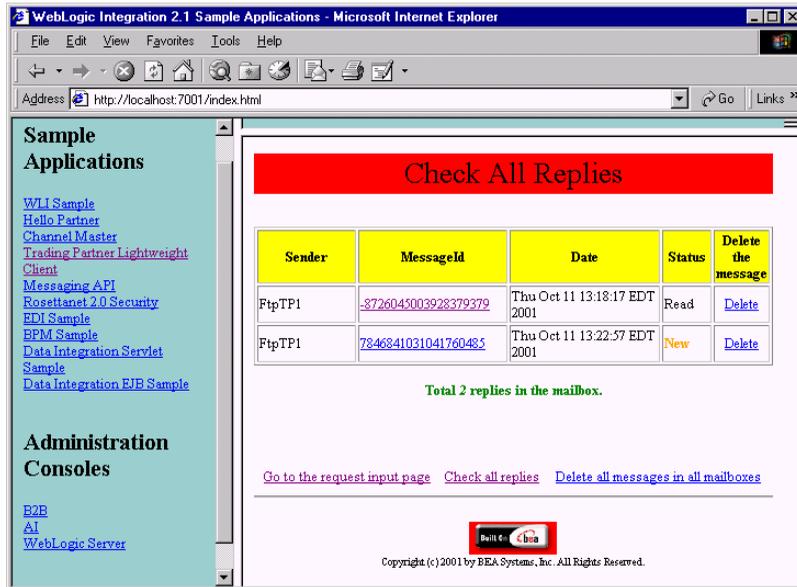
Note: When the list of replies is checked, `CheckallmsgTag` is checked, too.

If no replies are found, the following message is displayed.
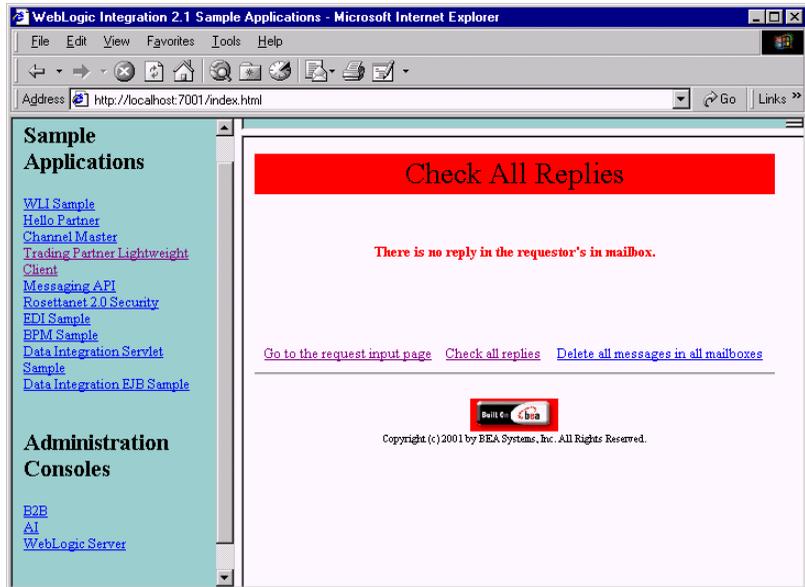
8. To delete all messages from the incoming and outgoing mailboxes of all participating trading partners (which, in the sample, are the mailboxes for BrowserTP1and FtpTP1), complete the following steps:

   a. Click Delete all messages in all mailboxes. A list of all the messages in all the B2B integration mailboxes is displayed.
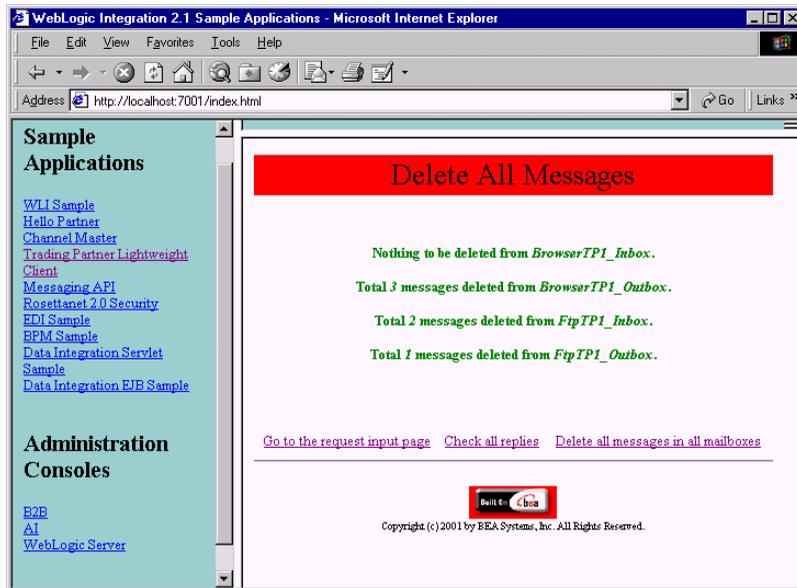
   b. Click Delete all messages in all mailboxes again. The total number of messages deleted is displayed.

   Note: As a side effect of this procedure you also test `DeleteallmsgTag`.

9. To delete a particular message from a mailbox, complete the following steps:

   a. In the Check All Replies window (shown in step 6), select a message.

   b. Click Delete.

   The Delete One Reply window confirms that the message has been deleted successfully.
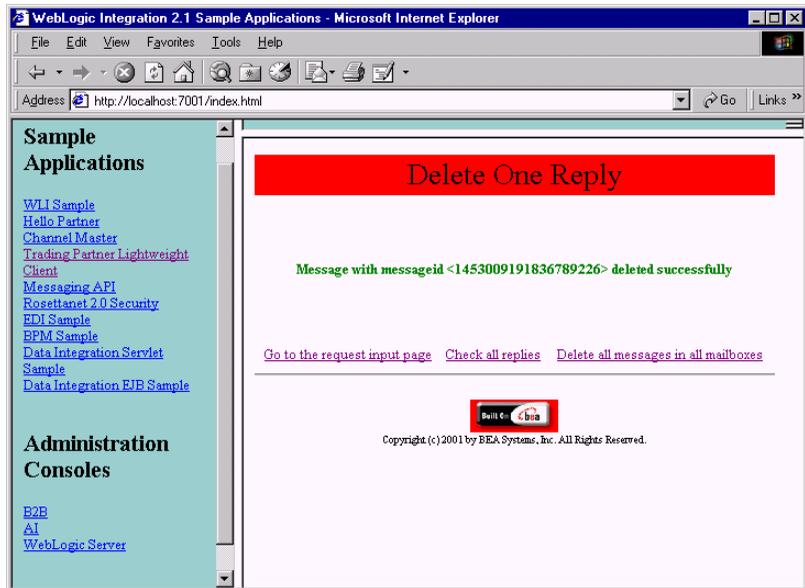
Note: As a side effect of this procedure, you also test `DeletemsgTag`.

10. To view details about messages, complete the following steps:

    a. Click the link labeled Check all replies.

    b. Select the message in the Message ID column that you want to view. The View Reply Detail window is displayed.
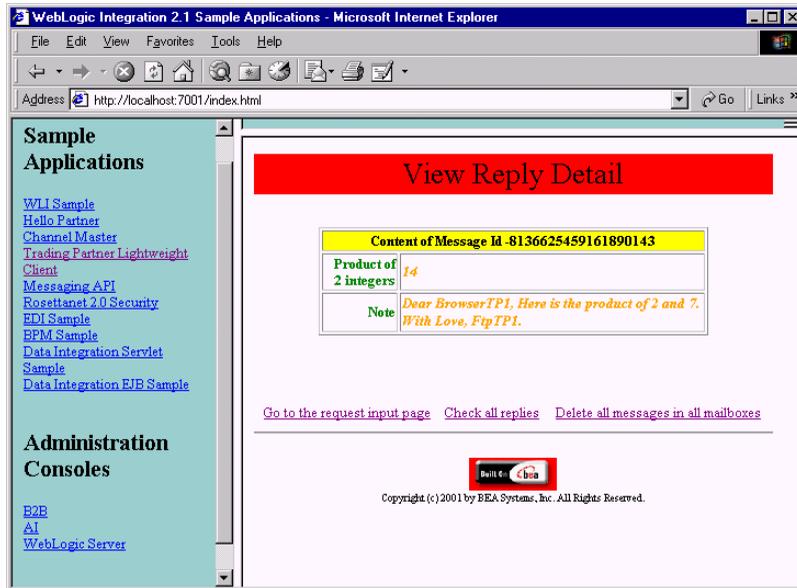
11. If you want to run more B2B samples at this time, keep the samples launcher page open and keep the WebLogic Server running.

    If you do not want to run more B2B samples at this time, exit from your browser and shut down the WebLogic Server by completing the procedure appropriate for your platform:

    ● Windows:

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

    ● UNIX:

       ```
       cd $WLI_HOME/config/samples
       stopWebLogic
       ```

Creating and Using Lightweight Clients

This section provides the following information about creating and configuring lightweight clients:

- Lightweight Client Source Files

- Using the JSP Tag Library

- Configuring a Lightweight Client

Lightweight Client Source Files

WebLogic Integration delivers all the source files you need to create your own lightweight clients. The Lightweight Client sample shares B2B integration information with the Hello Partner sample.

The following table lists all the files delivered specifically for the Lightweight Client sample.

Source Directory	Subdirectories and Files	Contents
WLI_HOME\samples\lightweightClient\src\wlcsamples\lightweightClient		
		Contains the source files for the business operation and its helper, both of which are called by workflow instance.
WLI_HOME\config\samples		
	LwcFileSync.dtd	DTD file for LwcFileSync.xml
	LwcFileSync.xml	File read by the file-sharing client. You must modify the following sections for your environment: - url - wlcfilesync *name* - directoryin - directoryout

Source Directory	Subdirectories and Files	Contents
`WLI_HOME\samples\lightweightClient\lib`		
		Contains original copies of the `LwcFileSync.dtd` and `LwcFileSync.xml` files in `WLI_HOME\config\samples`.
`WLI_HOME\samples\bin\`		
	■ `startFSClient.cmd` ■ `startFSClient`	Script that starts the file-sharing client that is responsible for the following tasks: ■ Transferring a request from the replier's in mailbox to its file-sharing directory ■ Transferring a reply from the replier's out directory to its out mailbox
`WLI_HOME\config\samples\lightweightClient\`		
	`ftpDir\`	Directory created for the execution of the Lightweight Client sample. Contains two directories (`FtpTP1_indir` and `FtpTP1_outdir`) that simulate the in and out directories on the file-sharing client. Names are hard-coded for the sample. **Note:** This directory and its subdirectories are created at run time.
	`web\`	Contains the files for the Web application, such as `*.jpg` and `*.jsp` files.
	`lwcWebApp\`	Contains the JSP tag library description file, and the `web.xml` file used by the Lightweight Client Web application.
`WLI_HOME\samples\lightweightClient\src\wlcsamples\lightweightClient\`		
	`tags\`	Contains the reference implementation of the B2B integration JSP tag libraries. Used for message management by `com.bea.lwclient.LwcMailbox`.

Using the JSP Tag Library

The JSP tag library uses a wrapper to address the WebLogic Integration Messaging API. The JSP tag library is used with predefined workflows to interact with a mailbox, as shown in Figure 5-2.

Figure 5-2   Mailbox Scenario Using the JSP Tag Library
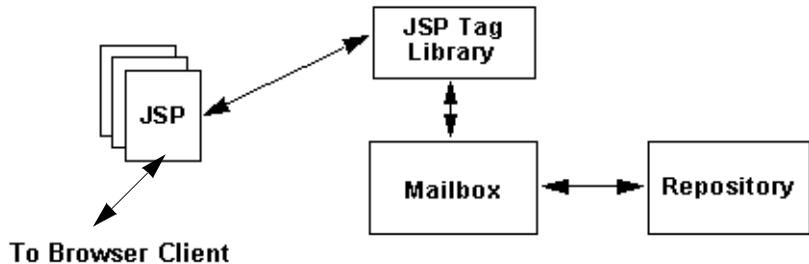
The mailbox shown here supplements the repository. The tag library accesses the mailbox as needed. The repository may also store mailbox messages.

Error handling is based on standard Java exception and error handling via the Mailbox API classes and JSP delivered to the browser client.

For a complete listing of the JSP tags, see Appendix A, "JSP Tag Reference."

Configuring a Lightweight Client

This section is provided only for users who want to move the browser client or the file-sharing client to a machine other than the one on which the samples instance of WebLogic Integration is running. If you move a client in this way, follow the appropriate procedure in this section to configure your remote lightweight client.

If you are running the Lightweight Client sample and the samples instance of WebLogic Integration on the same machine, skip this section.

This section provides procedures for the following tasks:

■ Configuring a File-Sharing Client

■ Configuring a Browser Client

Configuring a File-Sharing Client

A lightweight client is a process that WebLogic Integration runs in a dedicated Java Virtual Machine. As such, it needs no security mechanisms. WebLogic Integration can serve any number of lightweight clients that are defined in the `LwcFileSync.xml` configuration file. For additional information, see "Edit the File-Sharing Configuration File" later in this section.

Edit the WebLogic Integration Configuration File

Add the lightweight trading partner configuration information listed in Listing 5-1 to the configuration file for B2B integration: `config.xml`.

Listing 5-1   File-Sharing Client Configuration

```
<StartupClass
     ClassName="com.bea.lwclient.Startup"
     Name="LwcStartup"
     Targets="myserver"
/>
```

You can edit the configuration file in any of the following domains.

Domain	Pathname
Samples domain	*WLI_HOME*\config\samples\config.xml
WLIDOMAIN (supported production domain)	*WLI_HOME*\config\eaidomain\config.xml
EAIDOMAIN (supported production domain)	*WLI_HOME*\config\wlidomain\config.xml

Edit the File-Sharing Configuration File

To configure a lightweight client, define it in `LwcFileSync.xml`. The file-sharing client configuration in `LwcFileSync.xml` must conform to `LwcFileSync.dtd`.

Note: `LwcFileSync.dtd` must reside in the directory from which the file-sharing client is started.

Listing 5-2 provides a sample DTD file. Replace the values shown in **bold** with values that are appropriate for your lightweight client installation.

Listing 5-2 Sample `LwcFileSync.dtd`

```
<!-- This DTD describes file sharing client configuration file -->
<!ELEMENT wlcfilesyncconfig (wlcfilesync*) >

<!-- maxThreads The upper limit for number of threads permissible -->
<!-- in thread pool at a given point in time  -->
<!ATTLIST wlcfilesyncconfig  maxThreads CDATA #REQUIRED>

<!-- minThreads The lower limit for number of threads permissible -->
<!-- in thread pool at a given point in time -->
<!ATTLIST wlcfilesyncconfig  minThreads CDATA #REQUIRED>

<!-- maxIdleTime The maximum time interval thread could be idle -->
<!-- else removed from the pool -->
<!ATTLIST wlcfilesyncconfig  maxIdleTime CDATA #REQUIRED>

<!-- pollInterval Time interval defining wait interval before polling -->
<!ATTLIST wlcfilesyncconfig  pollInterval CDATA #REQUIRED>

<!-- url Weblogic URL used for JNDI lookup  -->
<!ATTLIST wlcfilesyncconfig  url CDATA #REQUIRED>

<!-- debug flag used to turn debugging on/off  -->
<!ATTLIST wlcfilesyncconfig  debug (TRUE | FALSE) "FALSE">

<!-- The following element is used to configure details of  LWTP -->
<!ELEMENT wlcfilesync EMPTY>

<!-- LWTP name -->
<!ATTLIST wlcfilesync name CDATA #REQUIRED>

<!-- path of the incoming directory on local file system -->
<!ATTLIST wlcfilesync directoryin CDATA #REQUIRED>

<!-- path of the outgoing directory on local file system -->
<!ATTLIST wlcfilesync directoryout CDATA #REQUIRED>
```

Listing 5-3 provides a sample XML file in which file-sharing clients are defined as lightweight trading partners. Replace the strings shown in **bold** with values appropriate for your lightweight client installation.

Listing 5-3 Sample `LwcFileSync.xml`

```
<?xml version="1.0"?>
<!DOCTYPE wlcfilesyncconfig SYSTEM "LwcFileSync.dtd">
<wlcfilesyncconfig maxThreads="9"
          minThreads="3"
          maxIdleTime="5000"
          pollInterval="2000"
          url="t3://localhost:7001"
          debug="FALSE">
<wlcfilesync name="FtpTP1"
          directoryin="<WLI_HOME>/ftpDir/FtpTP1_indir"
          directoryout="<WLI_HOME>/ftpDir/FtpTP1_outdir"/>
</wlcfilesyncconfig>
```

Configuring a Browser Client

WebLogic Integration supports two methods of configuring a browser client, depending on the security paradigm to be used: HTTP and HTTPS, or SSL. The following sections provide instructions for these methods:

■ Configuring an HTTP Browser Client

■ Configuring an HTTPS (SSL) Browser Client

Configuring an HTTP Browser Client

To configure an HTTP lightweight client, you must edit the web.xml file. To turn on security for a browser client, complete the following procedure:

1. The JSP developer responsible for the HTML user interface for trading partner lightweight client Web applications uses the sendmsg tag and provides the following mandatory argument : security = ON or OFF.

The workflow template developers can also turn security on and off by invoking the `wlcsamples.lightweightClient.LwcBizOp.putMessage()` method. This method also accepts a string as its last argument for SECURITY. To turn security on or off, the developer passes one of the following arguments: ON or OFF.

2. The person who deploys the hosted Web application modifies `WLI_HOME\samples\lightweightClient\web\lwcWebApp\web.xml`, adding security constraints for all the users with permission to use the application.

   For example, to provide a new user with the security privileges required to use the Web application `lwcWebApp.war`, add the following code:

```
<security-constraint>
  <web-resource-collection>
    <web-resource-name>lwcWebApp</web-resource-name>
    <url-pattern>/*</url-pattern>
  </web-resource-collection>
 <auth-constraint>
   <role-name>newpartner</role-name>
 </auth-constraint>
</security-constraint>

<login-config>
 <auth-method>BASIC</auth-method>
</login-config>

<security-role>
  <role-name>newpartner</role-name>
</security-role>
```

3. Make sure the user named `newpartner` is a valid WebLogic Server user. You can assign this username through either the WebLogic Server Administration Console or the WebLogic Integration B2B Console. For details, see the *BEA WebLogic Server Administration Guide* or *Administering B2B Integration*, respectively.

4. Using the B2B Console, map `newpartner` user must be mapped to a valid trading partner lightweight client in two steps:

   a. Create a trading partner lightweight client.

   b. Map the user's identification to the appropriate trading partner record.

   For details, see *Administering B2B Integration*.

5. Using the WebLogic Integration Studio, create a collaboration agreement between *newpartner* and the B2B integration hub. For details, see *Using the WebLogic Integration Studio*.

Configuring an HTTPS (SSL) Browser Client

To configure a lightweight HTTPS (SSL) client, perform the following steps:

1. Configure certificates on both the browser and the B2B integration nodes.

2. Create a WebLogic Server user called `newpartner`, who can run the browser on behalf of a trading partner lightweight client. You can create this user through either the WebLogic Server Administration Console or the WebLogic Integration B2B Console.

3. Using the B2B Console, create the trading partner lightweight client.

4. Map *newpartner* to the appropriate trading partner record.

5. Using the WebLogic Integration B2B Console, create a collaboration agreement between the lightweight client and the B2B integration hub. For details, see *Administering B2B Integration*.

How to Recompile the Sample

If you have made any changes to the sample you must recompile it before you can run it. To recompile your modified sample, complete the following procedure:

1. Set the required environment variables by entering the commands appropriate for your platform:

    - Windows:

      ```
      cd %WLIHOME%\bin
      setenv
      ```

    - UNIX:

      ```
      cd $WLI_HOME/bin
      ./setenv.sh
      ```

2. Go to the following directory:

```
cd samples\lightweightClient\project
```

3. Execute the following command:

```
ant all
```

6 Messaging API Sample

The Messaging API sample shows how the WebLogic Integration Messaging API can be used. Specifically, it demonstrates the use of two message-delivery mechanisms available with the Messaging API and the logic plug-in feature of WebLogic Integration B2B.

This section includes the following topics:

■ Overview of the Messaging API Sample

■ Before Running the Messaging API Sample

■ Running the Messaging API Sample

Overview of the Messaging API Sample

WebLogic Integration supports two methods of sending business messages:

1. Using application workflows created in the WebLogic Integration Studio. The application workflows contain actions that send business messages. The RosettaNet 2.0 Security and Channel Master samples are examples of workflow applications that send business messages.

2. Using a Java application that invokes the WebLogic Integration Messaging API to send business messages.

This sample uses the second method.

The WebLogic Integration Messaging API supports two message-delivery mechanisms:

- Synchronous—The sending application waits until the published message is delivered to the destination(s). The messaging system returns control to the application once the outcome of the activity of publishing the message is known. The application waits until a time-out occurs or the status of the activity becomes known, whichever happens first.

- Deferred Synchronous—Control returns to the application after a message is published. An XOCPMessageToken object is returned to the application, which the application can later access to check the status of message delivery.

The sample demonstrates the use of both the synchronous and deferred synchronous message delivery mechanisms.

The Messaging API sample contains three trading partners (Partner1, Partner2, and Partner3) that send business messages. The Messaging API sample contains four Java source code files: MdmTp1Servlet.java, MdmTp2Servlet.java, MdmTp3Servlet.java, and WaiterPlugin.java.

This file. . .	Contains the source code for the . . .
MdmTp1Servlet.java	Partner1 trading partner
MdmTp2Servlet.java	Partner2 trading partner
MdmTp3Servlet.java	Partner3 trading partner
WaiterPlugIn.java	Hub filter logic plug-In

For more information about the WaiterPlugIn.java code, see "Tracing the Execution Flow" on page 6-6.

Before Running the Messaging API Sample

Before running the Messaging API sample, complete the following steps:

1. Follow the instructions in "Preparing to Run the Samples" on page 1-2.

2. Make sure the proxy settings on your browser do not prevent you from connecting to the sample 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*.

Running the Messaging API Sample

To run the Messaging API sample, complete the following steps:

1. Run the RunSamples script by completing the procedure appropriate for your platform:

   - Windows:

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

   - UNIX:

     a) Make sure your PATH environment variable includes the directory in which the Netscape (netscape) executable resides.

     b) Go to the WebLogic Integration home directory (the directory in which you installed WebLogic Integration). For example:

     ```
     cd /home/me/bea/wlintegration2.1
     ```

     c) Run the setenv script to set the top-level WebLogic Integration environment variables:

     ```
     . setenv.sh
     ```

     d) Run the RunSamples script:

     ```
     cd samples/bin
     RunSamples
     ```

   Warning: For UNIX systems, the directory in which the netscape executable resides must be included in your PATH environment variable. If it is not included, the samples launcher page cannot be displayed.
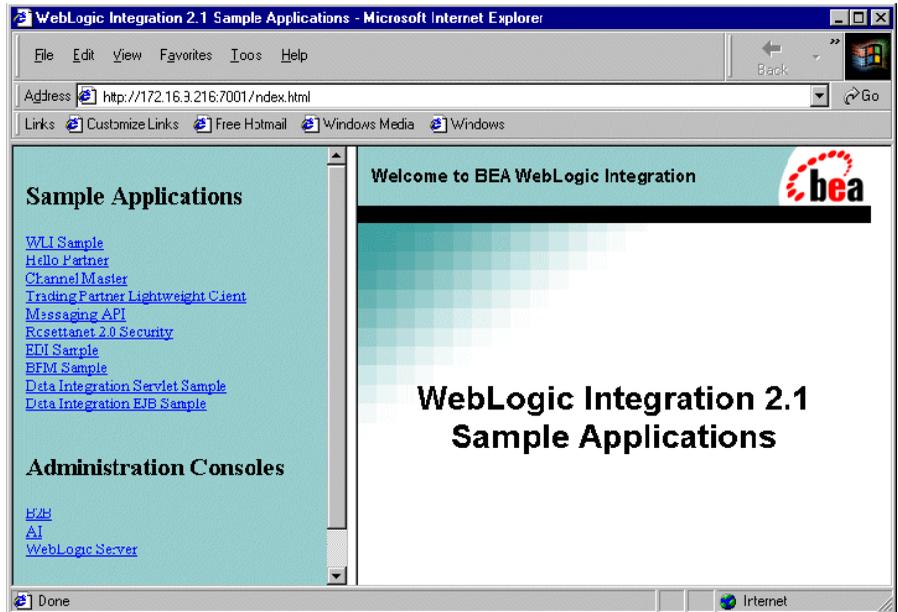
2. If the RunSamples script detects that the configuration section of the RunSamples script has been run before, the following prompt is displayed:

```
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
```

If you answer N to this question, the RunSamples script skips the steps for creating and populating the repository and runs only the step for booting the sample instance of the WebLogic Server.

If you answer Y to this question, the RunSamples script recreates and repopulates the repository, and then it boots the sample instance of the WebLogic Server. When you answer Y the RunSamples script destroys all the data currently in the repository and loads an unaltered version of the sample data into the repository. Answer Y only when the current sample data has been altered or removed and you want a fresh or unaltered version of the sample data in the repository.

Now the RunSamples script starts an instance of the WebLogic Server as a background process and the samples launcher page is displayed.

3. Click the link for the Messaging API under Sample Applications, in the left pane of the samples launcher page. A menu of options for the Messaging API sample is displayed in the right pane.
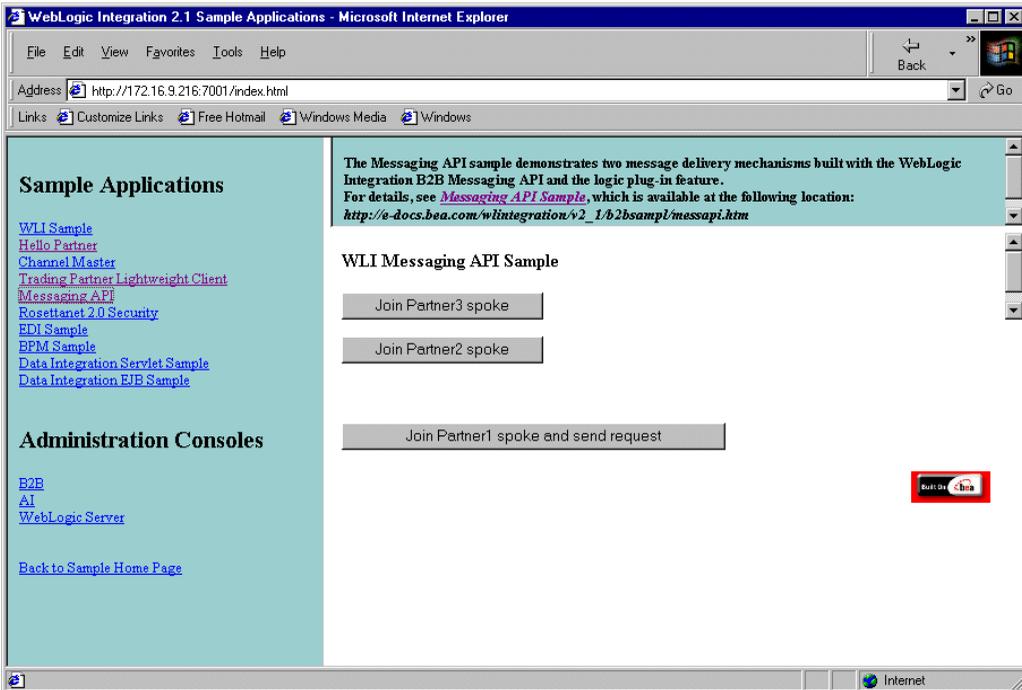
4. Click Join Partner3 spoke.

5. Click Join Partner2 spoke.

6. Click Join Partner1 spoke and send request.

   If the Messaging API sample is executed successfully, the following output is displayed at the bottom of the right pane:

```
Partner1 process flow:
Starting XOCPAppplication... done.
Creating conversation : verifierConversation:1.0:
requestor_Partner1_1001029696695_341001029696695...done.
send string for Message 1 = FIRST MESSAGE
Sending message 1 using synchronous deferred delivery method to Partner 2
Sending a second message before checking for acknowledgment on the first
```

```
send string for Message 2 = SECOND MESSAGE
Sending message 2 using synchronous delivery method to Partner 3
success status for message 2
Waiting for Message 2 response... done.
Processing reply for Message 2:
Received string for Message 2 = partner3 -- second message
```
Verification for Message 2 SUCCESS

```
Doing something else... done
Waiting acknowledgment for Message 1... Acknowledgment received
Success status for message 1
Waiting for Message 1 response... done
Processing reply:
Received string for Message 1 = partner2 -- first message
```
Verification for Message 1 SUCCESS

```
Terminating conversation:verifierConversation:1.0:
requestor_Partner1_1001029696695_341001029696695
success
Shutting down session... done.
```

7. If you want to run more B2B samples at this time, keep the samples launcher page open and keep the sample instance of the WebLogic Server running.

   If you do not plan on running more B2B samples at this time, shut down the instance of the WebLogic Server by completing the procedure appropriate for your platform:

   - Windows:

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

   - UNIX:
     ```
     cd $WLI_HOME/config/samples
     stopWebLogic
     ```

Tracing the Execution Flow

The following steps trace the execution flow of the Messaging API sample:

1. The RunSamples script is started, with the following results:

   a. The sample instance of the WebLogic Server is started.

b. A browser opens and displays the samples launcher page.

c. On the samples launcher page, the Messaging API link is clicked: the Messaging API sample page is displayed. This page in turn, displays three options: Join Partner3 spoke, Join Partner2 spoke, and Join Partner1 spoke and send request.

2. The Join Partner3 spoke option is selected, with the following results:

   a. An HTTP request is posted to the `MdmTp3` servlet.

   b. `MdmTp3`, in turn, invokes the `doPost` method from `MdmTp3Servlet.java`.

   c. `doPost` method starts trading partner Partner3's XOCPApplication.

3. The Join Partner2 spoke option is selected, with the following results:

   a. An HTTP request is posted to the `MdmTp2` servlet.

   b. `MdmTp2` in turn, invokes the `doPost` method from `MdmTp2Servlet.java`.

   c. The `doPost` method starts trading partner Partner2's XOCPApplication.

4. The Join Partner1 spoke and send request option is selected, with the following results:

   a. An HTTP request is posted to the `MdmTp1` servlet.

   b. `MdmTp1`, in turn, invokes the `doPost` method from `MdmTp1Servlet.java`.

   c. The `doPost` method starts trading partner Partner1's XOCPApplication.

   d. The `doPost` sends the first message, triggering a series of messages among the three trading partners (Partner1, Partner2, and Partner3). All three trading partners in this sample register with the WebLogic Integration repository to use the VerifierHubChannel delivery channel and to participate in the verifierConversation conversation. Partner1 is registered with the repository in the role of requester. Partner2 and Partner3 are registered in the role of replier.

The following figure shows the sequence in which these business messages are most likely to be sent and received. (The exact sequence is timing related and depends on the thread scheduling of the Java virtual machine.)

Figure 6-1   Interactive Diagram of the Flow of Messages Among Trading Partners
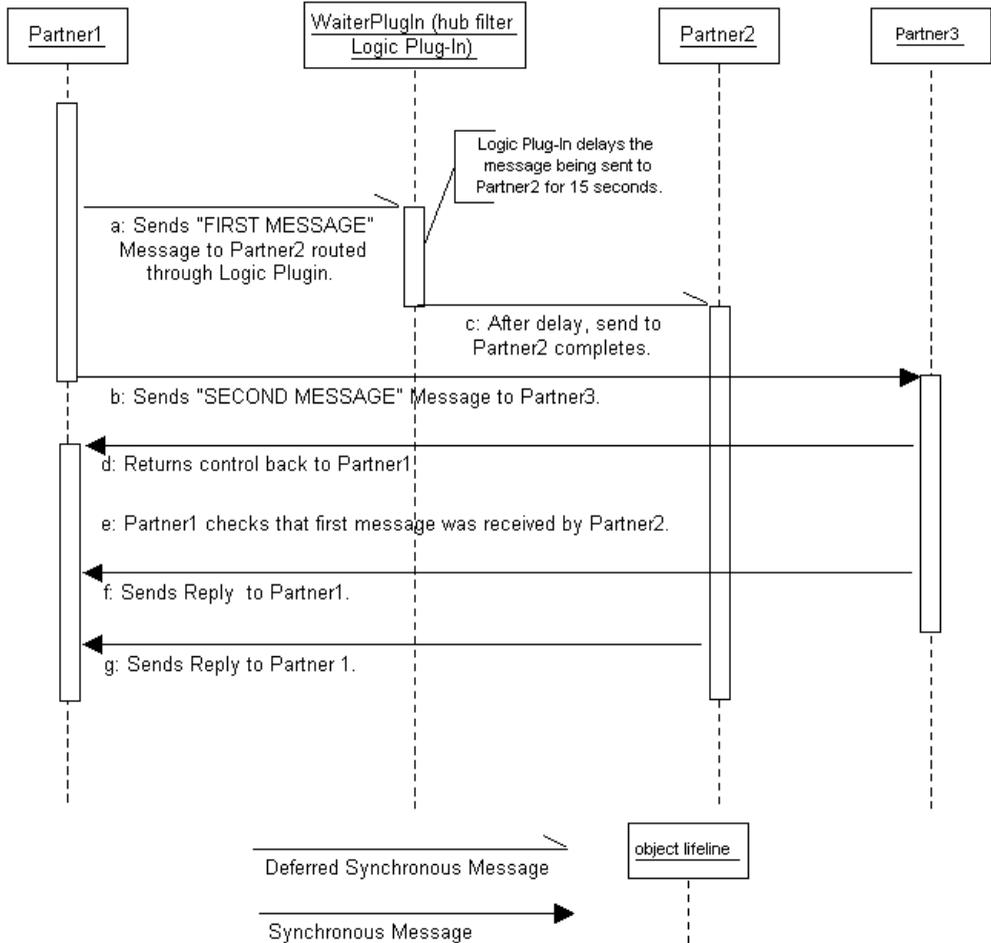
The following sequence provides details about each of the steps indicated by a corresponding letter in Figure 6-1:

a. Partner1 sends a message with the text FIRST MESSAGE to Partner2. This message is sent deferred synchronously. Consequently, Partner1 does not wait or block for a return from Partner2, but it continues executing tasks.

    The nonblocking aspect of the message is represented in the Interaction diagram (Figure 6-1) by Partner1's Object lifeline. When Partner initially becomes active, the representation of Partner1's lifeline changes from a dashed line (indicating an inactive state) to a narrow rectangle (indicating an active state). The lifeline remains active after Partner1 sends the first message. By remaining active, Partner1 can do other tasks, such as sending another message, as shown in the next step.

    All the messages sent in this sample are routed through the hub. A logic plug-in called WaiterPlugIn has been added to the filter chain of the hub. When messages are routed though the hub, the process method of the WaiterPlugIn class is executed. The process method checks the target recipient of the outgoing message. If the target recipient is Partner2, process sleeps for 15 seconds. Otherwise, it sends the message immediately. In the example shown in Figure 6-1, the first message, which was sent by Partner1 to Partner2, is delayed by 15 seconds.

b. While the first message is still being processed, Partner1 sends the second message, with the text SECOND MESSAGE, to Partner3. This message is sent synchronously. Consequently, Partner1 must block or wait until Partner3 returns before Partner1 can process other tasks.

c. Notice how this blocking of Partner1 is represented in Figure 6-1. Because the second message is synchronous, the representation of Partner1's lifeline changes from a narrow rectangle (indicating an active state) to a dashed line (indicating an inactive state) after Partner1 sends the second message. The change to an inactive state means that Partner1 must block or wait until the message is sent and acknowledged by Partner3 before it can become active and start doing other tasks again.

    The second message and all the reply messages of this sample are routed through the hub, where the process method of WaiterPlugIn is executed on them. Each of these messages passes through the hub without delay because the recipient of the message is not Partner2. Routing of these messages through the hub filter logic plug-in is not shown in Figure 6-1.

d. After the first message is delayed in the hub for 15 seconds, the hub routes it to Partner2.

e. Control is returned to Partner1 after the synchronous send to Partner3 completes. At this point Partner1 becomes active again, as shown by the change in its lifeline, in Figure 6-1, from a dashed line (representing an inactive state) to a narrow rectangle (representing an active state).

f. Partner1 verifies that the first message was received by Partner2.

g. Partner3 takes the text of the second message, converts it to lowercase, and adds the prefix `partner 3--` to it. It then sends the modified message back to Partner1.

h. Partner2 takes the text of the first message, converts it to lowercase, and adds the prefix `partner 2--` to it. It then sends the modified message back to Partner1.

5. The results are displayed by Partner1 on an HTML page. This step is not represented in Figure 6-1.

A JSP Tag Reference

A Java Server Pages (JSP) tag library is provided for trading partner lightweight clients. It uses a wrapper to address the WebLogic Integration Messaging API. Error handling uses standard Java exception and error handling via WebLogic Integration Messaging API classes and JSP pages delivered to the lightweight client.

This section provides reference information for the following JSP tags:

- SendmsgTag

- ChecknewmsgTag

- CheckallmsgTag

- ReadmsgTag

- DeletemsgTag

- DeleteallmsgTag

- CreatemboxTag

- RemovemboxTag

SendmsgTag

Passes a business message to a mailbox for outgoing mail; provides message persistence for reliability.

Syntax
```
SendMsgTag (String mboxName, String sender, String message, String
URL, String security)
```

Returns
Returns `Message has been sent successfully` if the message was successfully sent to the JSP page in which `SendMsgTag` is embedded.

Variables

Variable	Description
String mboxName	Name of the mailbox. To run the sample, use one of the following names for the appropriate mailbox: ■ For incoming mail: *trading_partner_name_*Inbox ■ For outgoing mail: *trading_partner_name_*Outbox
String message	Message content.
String sender	Mailbox address for the sender. This may be an e-mail address or an FTP server address.
String URL	URL for the hosting trading partner.
String security	Value may be ON or OFF.

Example
```
sendmsg mboxname="<%=outboxName_browserTP1%>"
sender="<%=SENDER%>" message="<%=domAsStr%>" url="<%=url%>"
security="ON"/>
```

ChecknewmsgTag

Checks for new messages in the mailboxes for incoming and outgoing mail. Does not check for stored messages (see `CheckallmsgTag`).

Syntax   `ChecknewmsgTag (String mboxName)`

Returns   If the mailbox is empty, returns `No new message found in mailbox`. If the mailbox contains one or more new messages, the messages are displayed in HTML format.

Variables

Variable	Description
mboxName	Name of the mailbox. To run the sample, use one of the following names for the appropriate mailbox: ■ For incoming mail: *trading_partner_name*_Inbox ■ For outgoing mail: *trading_partner_name*_Outbox

CheckallmsgTag

Checks all messages in the mailbox, including stored messages.

Syntax `CheckallmsgTag (String mboxName)`

Returns If there are messages in the mailbox, they are displayed in HTML format. If the mailbox is empty, returns `No message found in mailbox`.

Variables

Variable	Description
mboxName	Name of the mailbox. To run the sample, use one of the following names for the appropriate mailbox: ■ For incoming mail: *trading_partner_name_*Inbox ■ For outgoing mail: *trading_partner_name_*Outbox

ReadmsgTag

Gets details about a specific message from the mailbox.

Syntax ReadmsgTag (String mboxName, String msgId)

Returns Message details are displayed in HTML format.

Variables

Variable	Description
mboxName	Name of the mailbox. To run the sample, use one of the following names for the appropriate mailbox: ■ For incoming mail: *trading_partner_name_*Inbox ■ For outgoing mail: *trading_partner_name_*Outbox
msgID	Unique message identifier.

DeletemsgTag

Deletes a specified message from the mailbox.

Syntax   `DeletemsgTag (String mboxName, String msgID)`

Returns   Returns `Message with messageID msgID deleted successfully.`

Variables

Variable	Description
mboxName	Name of the mailbox. To run the sample, use one of the following names for the appropriate mailbox: ■ For incoming mail: *trading_partner_name_*Inbox ■ For outgoing mail: *trading_partner_name_*Outbox
msgID	Unique message identifier.

DeleteallmsgTag

Deletes all messages from the mailbox.

Syntax   `DeleteallmsgTag (String mboxName)`

Returns   Returns `All messages were deleted successfully` when successful.

Variables

Variable	Description
mboxName	Name of the mailbox. To run the sample, use one of the following names for the appropriate mailbox: ■ For incoming mail: *trading_partner_name_*Inbox ■ For outgoing mail: *trading_partner_name_*Outbox

CreatemboxTag

Creates a mailbox.

Syntax   `CreatemboxTag (String mboxName)`

Returns   Returns no message.

Variable

Variable	Description
mboxName	Name of the mailbox. To run the sample, use one of the following names for the appropriate mailbox: ■ For incoming mail: *trading_partner_name*_Inbox ■ For outgoing mail: *trading_partner_name*_Outbox

RemovemboxTag

Removes a specific mailbox.

Syntax   `RemovemboxTag (String mboxName)`

Returns   Returns `Mailbox removed successfully.`

Variables

Variable	Description
mboxName	Name of the mailbox. To run the sample, use one of the following names for the appropriate mailbox: ■ For incoming mail: *trading_partner_name_*Inbox ■ For outgoing mail: *trading_partner_name_*Outbox

Index

W

WaiterPlugIn class 6-2, 6-9

X

XOCPMessageToken 6-2