BEA WebLogic Collaborate

A Component of BEA WebLogic Integration

Using BEA WebLogic Collaborate Samples

BEA WebLogic Collaborate Release 2.0
Document Edition 2.0
July 2001

Using BEA WebLogic Collaborate Samples

Document Edition	Date	Software Version
2.0	July 2001	2.0

Contents

About This Document

This document describes the WebLogic Collaborate samples, and provides configuration information and instructions for running and verifying each sample.

This document includes the following topics:

- Chapter 1, "Getting Started," describes the WebLogic Collaborate samples, and discusses basic installation and configuration.
- Chapter 2, "Hello Partner Sample," demonstrates communication using the default XOCP messaging protocol.
- Chapter 3, "Trading Partner Lightweight Client Sample," describes how to configure and use the browser and file-sharing client samples.
- Chapter 4, "Messaging API Sample," provides instructions for configuring and executing the WebLogic Collaborate Messaging API sample.

What You Need to Know

This document is intended for independent software vendors (ISVs) who are interested in extending BEA BEA WebLogic Process Integrator. It assumes a familiarity with the BEA BEA WebLogic Process Integrator 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 BEA WebLogic Process Integrator 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 BEA WebLogic Process Integrator 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 BEA WebLogic Process Integrator to meet your company's needs, see the following documents:

- The BEA WebLogic Process Integrator documentation at the following URL:

  `http://edocs.bea.com`

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

  `http://java.sun.com/`

Contact Us!

Your feedback on the BEA BEA WebLogic Process Integrator 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 BEA WebLogic Process Integrator documentation.

In your e-mail message, please indicate that you are using the documentation for the BEA BEA WebLogic Process Integrator 6.0 release.

If you have any questions about this version of BEA BEA WebLogic Process Integrator, or if you have problems installing and running BEA BEA WebLogic Process Integrator, 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.

Convention	Item
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`*
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`*`]...`

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

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

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

CHAPTER

1 Getting Started

The WebLogic Collaborate samples demonstrate implementations of WebLogic Collaborate features. In addition, successful installation and setup of the Hello Partner sample provides verification for installation and basic configuration.

This section includes the following topics:

Samples Overview

The WebLogic Collaborate samples have the following attributes in common:

- They demonstrate step-by-step deployment.
- They are run by workflows that are triggered by XML events.
- They can be customized and redeployed.

With the exception of the Hello Partner sample, which verifies basic communication protocols, the samples demonstrate WebLogic Collaborate solutions to plausible business problems. The following samples are described in this guide:

- Hello Partner Sample—Demonstrates the basic handshake necessary for communication. The Hello Partner sample uses the WebLogic Collaborate default XOCP messaging protocol.
- Trading Partner Lightweight Client Sample—Illustrates a request-reply scenario in which two trading partners without a WebLogic Collaborate installation use

the WebLogic Collaborate 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—Uses the XOCP protocol. Three trading partners are collocated on the same server with the hub. Uses a logic plug-in to simulate delaying message delivery.

Use the WebLogic Collaborate Administration Console to export the sample workflow templates, and then use the WebLogic Process Integrator Studio to change the templates as appropriate. For details, see *Administering BEA WebLogic Collaborate* and *Using the BEA WebLogic Process Integrator Studio*, respectively.

Installing the Samples

Before you can run some samples, you must make unique configuration changes or populate a database. For specific installation and configuration instructions, see the documentation for the appropriate sample. Instructions are provided for both Windows and UNIX systems:

- To install the samples on a Windows system, use the Windows Start menu.
- To install the samples on a UNIX system, enter the appropriate command at the prompt.

Note: The samples installation is configured with a single WebLogic Collaborate instance. However, in an actual peer-to-peer type of installation, each trading partner (with the exception of the lightweight clients) has a WebLogic Collaborate installation, configured for the appropriate delivery channels.

This section describes two types of installations:

- Installing Using the Default Database
- Installing Using Other Supported Databases

Installing Using the Default Database

1. Install and configure WebLogic Process Integrator.

   For instructions, see *Installing and Configuring BEA WebLogic Process Integrator*.

2. Install and configure WebLogic Collaborate.

   The samples, including all associated files, schemas, and scripts, are loaded when you install WebLogic Collaborate. For full installation and configuration instructions, see *Installing BEA WebLogic Collaborate*.

3. Bulkload the sample data file into the repository, using the procedure appropriate for your platform.

   - Windows: From the Start menu, choose Start→Programs→BEAWebLogic EBusiness Platform→WebLogic Integration 2.0/Samples→Bulkload→*sample_name*
   - UNIX: For the Hello Partner and Lightweight Client samples:

     ```
     cd $WLC_HOME/config/samples/shortcuts/
     sh BulkloadHelloPartner.sh
     ```

     For the Messaging API sample:

     ```
     cd $WLC_HOME/config/samples/shortcuts/
     sh BulkloadMessagingAPI.sh
     ```

   The data, schemas, and logic plug-ins required to run the samples are loaded. Wait for the bulkloading process to complete.

   Note: See also "Working With the Bulk Loader" and "Working with the Repository" in *Administering BEA WebLogic Collaborate*.

4. Start WebLogic Collaborate, using the procedure appropriate for your platform:

   - Windows: From the Start menu, choose WebLogic Integration 2.0→Collaborate→Samples→Start Collaborate
   - UNIX: Run the shell script as follows:

     ```
     cd $WLC_HOME/config/samples/
     sh startWeblogic.sh
     ```

   When WebLogic Collaborate is started,WebLogic Process Integrator is started, too.

For the Messaging API sample, skip to Step 7.

5. Start the WebLogic Process Integrator Studio, using the procedure appropriate for your platform:

   - Windows: From the Start menu, choose WebLogic Integration 2.0→Collaborate→Samples→Start Studio
   - UNIX: Run the shell script as follows:

     ```
     cd $WLC_HOME/config/samples/shortcuts
     sh studio.sh
     ```

   To log in, use the following information:

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

   Make sure that `ORG1` is selected in the Studio organization list.

6. Using the WebLogic Process Integrator Studio, import the `JAR` files that contain the workflows and business operations used by the samples.

   Windows:

   - From the Studio menu, choose Tools→Import Package.
   - Browse to the `WLC_HOME/config/samples/workflows` directory.
   - Select `HelloPartnerAndLightWeightClient.jar` from the directory appropriate for your operating system.

     Windows: `%WLC_HOME%\config\samples\workflows`

     UNIX: `$WLC_HOME/config/samples/workflows`
   - Select ORG1 as the Target Organization.
   - Check for active workflows after importing.
   - Import `HelloPartnerAndLightWeightClient.jar`.

7. Run a sample by completing the appropriate procedure.

Installing Using Other Supported Databases

The WebLogic Collaborate samples are configured to use Cloudscape as the default database. In addition to Cloudscape, WebLogic Collaborate supports the Oracle and Microsoft SQL databases. For more information about the databases supported by WebLogic Collaborate, see *Administering BEA WebLogic Collaborate*.

If you prefer to use a database other than Cloudscape when running the samples, you must execute the following steps:

1. Specify an alternate database during product installation. See *Installing BEA WebLogic Collaborate*.
2. Switch the sample database configuration, specifying one of the following as the value of *db_name*: Cloudscape, Oracle, or MSSQL7.

   Windows: From the Start menu, choose WebLogic Integration 2.0→Collaborate→Samples→Switch DB Config→Change Database, *DBname*→Use *db_name*

   UNIX: Run the following shell script:

   ```
   cd $WLC_HOME/config/samples/shortcuts
   sh SwitchDBTodb_name.sh.
   ```

3. Create the schemas for the database you are using.

   - Oracle on Windows:

   a. Run the following script:

```
cd %WLPI_HOME%\ddl
sqlplus ORACLE_USER/ORACLE_PASSWORD@ORACLE_SERVICENAME@wlpi_oracle.ddl
```

   b. Create the common repository by running the following script:

   ```
   cd %WLINT_HOME\repository\oracle
   createDD oracle
   ```

   c. Create the WebLogic Collaborate repository by running the following script:

   ```
   cd %WLC_HOME%\bin
   createDB oracle
   ```

- SQL Server on Windows:

a. Run the following script:

```
cd %WLPI_HOME%\ddl
osql -e -n -S MSSQL_HOSTNAME -U MSSQL_USER -P MSSQL_PASSWORD -i wlpi_mssql.ddl
```

b. Create the common repository:

```
cd %WLINT_HOME%\repository\mssql
createDB mssql
```

c. Create the WebLogic Collaborate repository by running the following script:

```
cd %WLC_HOME%\bin
createDB mssql
```

- Oracle on UNIX:

a. Run the following script:

```
cd $WLPI_HOME/ddl
sqlplus ORACLE_USER/ORACLE_PASSWORD@ORACLE_SERVICENAME@wlpi_oracle.ddl
```

b. Run the next script:

```
cd $WLINT_HOME/repository/oracle
sh createDB.sh oracle
```

c. Run the final script:

```
cd $WLC_HOME/bin:
sh createDB.sh oracle
```

- SQL Server on UNIX: SQL Server runs only on Windows systems. In order to run SQL Server on a UNIX system, you must create the database and schemas on a machine running the Windows operating system, and accessed remotely.

  To change to SQL Server on a UNIX system, manually edit the SQL database-specific information in the following files, and bulkload the repository before switching databases.

```
$WLC_HOME/config/samples/config.MSSQL7
$WLC_HOME/config/samples/samplesDBManagement/bulkloading/mssql7/HelloPartnerBul
kloadWrapper.xml
$WLC_HOME/config/samples/samplesDBManagement/bulkloading/mssql7/MessagingAPIBul
kloadWrapper.xml
$WLC_HOME/config/samples/samplesDBManagement/mssql7/setDBVars.sh
```

For more information about creating the database schemas required, see "Step 7: Creating the WebLogic Collaborate Tables" in "Installing BEA WebLogic Collaborate" in *Installing BEA WebLogic Collaborate*.

Note: To recreate the Cloudscape database, you must perform the following steps appropriate for your operating system.

Windows: Navigate to the appropriate directories and execute the following commands:

```
cd %WLINT_HOME%\repository\cloudscape
rmdir db
cd %WLPI_HOME%\bin
createCloudscape.cmd
cd %WLC_HOME%\bin
createDB cloudscape
```

UNIX: Navigate to the appropriate directories and execute the following scripts:

```
cd $WLINT_HOME/repository/cloudscape
rm -rf db
cd $WLPI_HOME/bin
sh createCloudscape.sh
cd $WLC_HOME/bin
sh createDBsh cloudscape
```

4. Bulkload the sample data file into the repository, using the procedure appropriate for your platform.

   - Windows: From the Start menu, choose Start→Program→Samples→Bulkload→*sample_name*
   - UNIX: For the Hello Partner and Lightweight Client samples:

     ```
     cd $WLC_HOME/config/samples/shortcuts/
     sh BulkloadHelloPartner.sh
     ```

     For the Messaging API sample:

     ```
     cd $WLC_HOME/config/samples/shortcuts/
     sh BulkloadMessagingAPI.sh
     ```

   The data, schemas, and logic plug-ins required to run the samples are loaded. Wait for the bulkloading process to complete.

   Note: See also "Working With the Bulk Loader" and "Working with the Repository" in *Administering BEA WebLogic Collaborate*.

5. Start WebLogic Collaborate, using the procedure appropriate for your platform:

   - Windows: From the Start menu, choose WebLogic Integration 2.0→Collaborate→Samples→Start Collaborate
   - UNIX: Run the script as follows:

     ```
     cd $WLC_HOME/config/samples/
     sh startWeblogic.sh
     ```

   When WebLogic Collaborate is started, WebLogic Process Integrator is started, too.

   For the Messaging API sample, skip to Step 7.

6. Start the WebLogic Process Integrator Studio, using the procedure appropriate for your platform:

   - Windows: From the Start menu, choose WebLogic Integration 2.0→Collaborate→Samples→Start WLPI Studio
   - UNIX: Run the following script:

     ```
     cd $WLC_HOME/config/samples/shortcuts/
     sh studio.sh
     ```

   The default username and password are `joe` (lowercase) and `password` (lowercase), respectively.

   Make sure that `ORG1` is selected in the Studio organization list.

7. When using the WebLogic Process Integrator Studio, import the `JAR` files that contain the workflows used by the samples.

   Windows:

   - From the Studio menu, choose Tools→Import Package.
   - Select ORG1 as the Target Organization.
   - Check for active workflows after importing.
   - Import `HelloPartnerAndLightWeightClient.jar` from the directory appropriate for your operating system.

     Windows: `%WLC_HOME%\config\samples\workflows`

     UNIX: `$WLC_HOME/config/samples/workflows`

8. Run a sample by completing the appropriate procedure.

CHAPTER

2 Hello Partner Sample

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

Overview of the Hello Partner Sample

Most WebLogic Collaborate conversations take place between trading partners who have both installed WebLogic Collaborate. In such environments, the XOCP protocol is used to exchange messages via a hub. The Hello Partner sample demonstrates all aspects of this process.

What the Sample Demonstrates

The Hello Partner sample demonstrates how two or more trading partners can participate in business communication in a properly configured WebLogic Collaborate environment. Such communication occurs in a networking environment with a hub-and-spoke configuration. XOCP is used to carry messages between two trading partners. A hub and two spokes are created in your local WebLogic Collaborate installation to demonstrate the round-trip path of the message. 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 are implemented using XOCP to carry messages between trading partners.
- Private process - A private process for each trading partner that handles message content processing is shown. The private process interacts with (a) the public process for trading partner message delivery, and (b) an associated Java application for external message processing.

In this sample, a JSP-driven Web page initiates a message to the replying trading partner. The trading partner processes the data in the message and sends a reply to the initiating trading partner, which is displayed by a second Web page.

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, both of which communicate via a hub-and-spoke WebLogic Collaborate configuration. The requestor sends a message and waits for a response from the replier. The requestor uses a Web browser to access a JSP page that initializes and sends a message in XML, using JSP tags that are embedded in the browser pages.

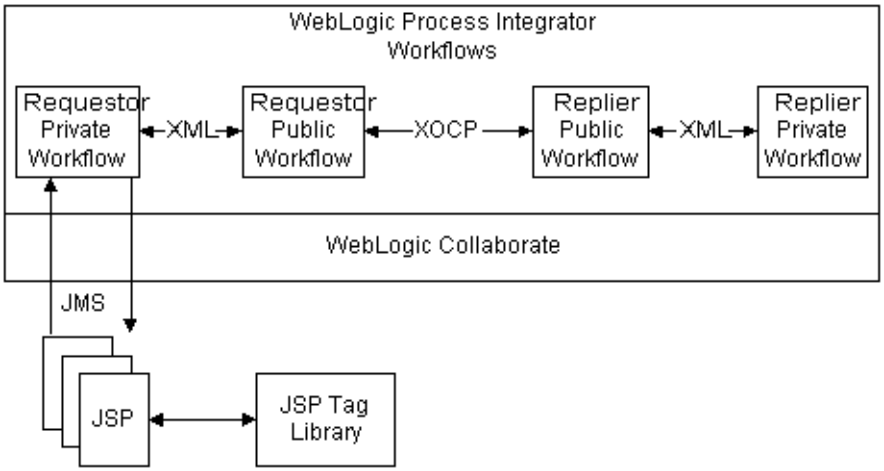
The replier receives the message, generates a response, and publishes the response to the requestor in XML. The response is sent to the requestor, and the contents of the reply are displayed in the requestor's JSP page.

Running the Hello Partner sample demonstrates the following events:

- A private workflow is triggered by an XML event from a JSP tag in the launching Web page.
- The private workflow generates the XML that will be embedded in the message to the replier, which is sent via the public workflow.
- The replier public workflow is triggered by the message.

- When the replier public workflow receives the message, it will extract the XML data contained in the message, and publish an XML event that will trigger the replier's private workflow.
- The replier private workflow generates XML containing the response, sends that XML data back to the replier public workflow using an XML event.
- The replier public workflow packages the XML into a message that is sent back to the requestor.
- The requestor public workflow receives the message from the replier public workflow, extracts the XML containing the response, and then uses the XML to publish an XML event that is received by the requestor private workflow.
- The replier's response is posted to the launching JSP page.

Figure 2-1 Hello Partner Sample Scenario Logic



The actual message content used in the sample is relatively simple. The JSP page allows you to select two numbers. These numbers are sent to the Replier, where they are multiplied. The multiplication result is returned.

Setting Up the Hello Partner Sample

The Hello Partner sample is designed to allow you to verify that you have properly installed WebLogic Collaborate. The Hello Partner sample runs in its own database domain, the Sample domain, rather than in your development or production domain. This isolates the database tables used by the Hello Partner sample from any development or production configurations or data you may load into WebLogic Collaborate before or after running Hello Partner.

To load the Hello Partner sample:

1. Install WebLogic Collaborate and WebLogic Process Integrator.
2. Verify that user `Joe` has been created during the installation of WebLogic Process Integrator Studio, with password `password`. This user account is created during installation by default.
3. Load the Hello Partner database setup using the Bulk Loader.
   - On Windows systems:

     From the Start Menu, choose Start→Programs→BEA WebLogic E-Business Platform→WebLogic Integration 2.0→Collaborate→Samples→BulkLoad→Hello Partner. This sequence loads all data associated with the Hello Partner sample into the Sample database domain in WebLogic Collaborate. This process may take several minutes.
   - On UNIX systems:

     Start the following shell script from the `$WLC_HOME/config/samples/shortcuts` directory:

     ```
     sh BulkloadHelloPartner.sh
     ```

     This shell script loads all data associated with the Hello Partner sample into the Sample database domain in WebLogic Collaborate. This process may take several minutes.
4. Start WebLogic Collaborate.
   - On Windows systems:

Choose Start→Programs→BEA WebLogic E-Business Platform→WebLogic Integration 2.0→Collaborate→Samples→Start Collaborate.

- On UNIX systems:

  Run the following shell script from the `$WLC_HOME/config/samples` directory:

  ```
  sh startWeblogic.sh
  ```

When you start WebLogic Collaborate, you will eventually see a message similar to `Started Process Integrator`. This does not mean that you have also started the WebLogic Process Integrator Studio. Rather, it means that you have started the WebLogic Process Integrator server, which runs in conjunction with WebLogic Collaborate. To start the WebLogic Process Integrator server, you need a file manager application and a Web browser.

5. Start the WebLogic Process Integrator Studio.

   On Windows systems:

   Choose Start→Programs→BEA WebLogic E-Business Platform→WebLogic Integration 2.0→Collaborate→Samples→Start Studio.

   On UNIX systems:

   Run the following shell script from the `$WLC_HOME/config/samples/shortcuts` directory:

   ```
   sh studio.sh&
   ```

   To log in, use the following information:

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

6. Verify that ORG1 has been selected as the organization.

7. Import the Hello Partner jar file into the WebLogic Process Integrator Studio. Select Import Package... from the Tools menu. Select the following jar file:

WLC_HOME`\config\samples\workflows\HelloPartnerAndLighweightClient.jar`

   Select the Activate workflows after import option before you click Import.

Once you have completed this procedure, WebLogic Collaborate will have loaded all Hello Partner sample data and configurations.

Running the Hello Partner Sample

Once you have loaded the Hello Partner sample, you can start it.

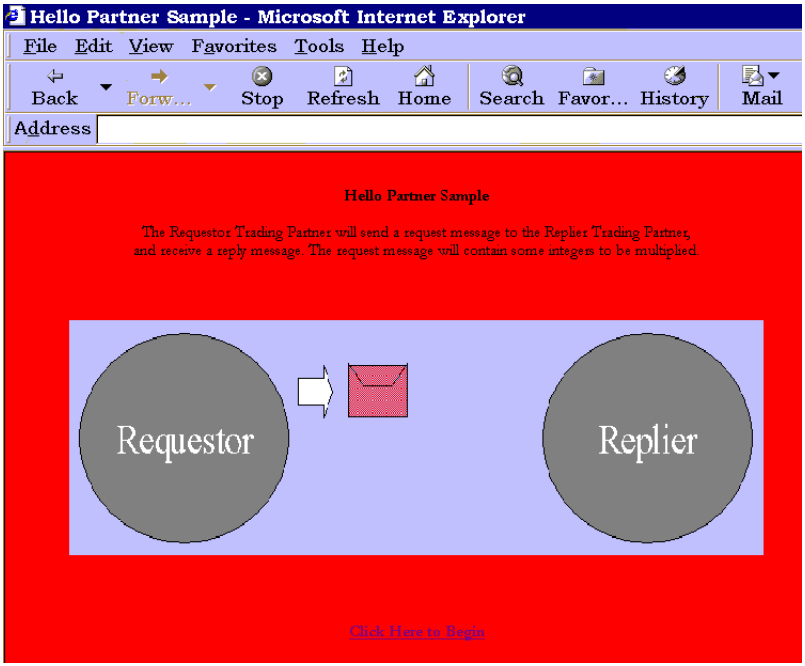
To run the Hello Partner sample:

1. Choose Start→Programs→BEA WebLogic E-Business Platform→WebLogic Integration 2.0→Collaborate→Samples→Run Samples→Hello Partner.

   For UNIX, load the following URL:

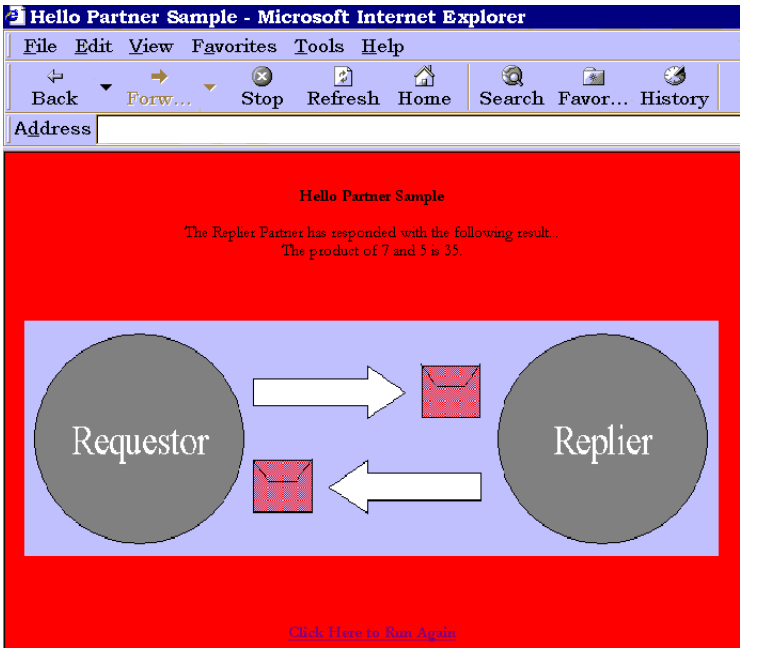
   `HTTP://localhost:7001/HelloPartnerLauncher.html`

Figure 2-2 JSP Launcher Page



2. Select two numbers.
3. Click Click Here to Begin.
4. The example sends the message to the Replier trading partner for multiplication, and then sends the result to the JSP. The Web page displays the following result.

Figure 2-3 JSP Result Page



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, while two private workflows connect the JSP and the requestor's public workflow on one side, and creates the replier's reply data on the other.

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 documents are used in the Hello Partner sample:

- Request WLC Message from Requestor Role
- Reply WLC Message from Replier Role
- XML Message Over JMS from JSP to Trigger Private Workflow
- XML Message Over JMS from Private Workflow to JSP with Result
- XML Event from Replier Public Workflow to Replier Private Workflow
- XML Event from Replier Private Workflows to Replier Public Workflows

Request WLC Message from Requestor Role

This message is sent by the Requestor. It includes the two numbers to be multiplied and is packaged as a document of type multiply-request:

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

Reply WLC Message from Replier Role

The following 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>
```

XML Message Over JMS from JSP to Trigger Private Workflow

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

```
<from-multiply-request-jsp-to-workflow>
       <integer-one>5</integer-one>
       <integer-two>7</integer-two>
</from-multiply-request-jsp-to-workflow>
```

XML Message Over JMS from Private Workflow to JSP with Result

The following message is sent over JMS from the Requestor's private workflow to the JSP. It contains the product of the multiplication, as well as the 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>
```

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>
```

XML Event from Replier Private Workflows to Replier Public Workflows

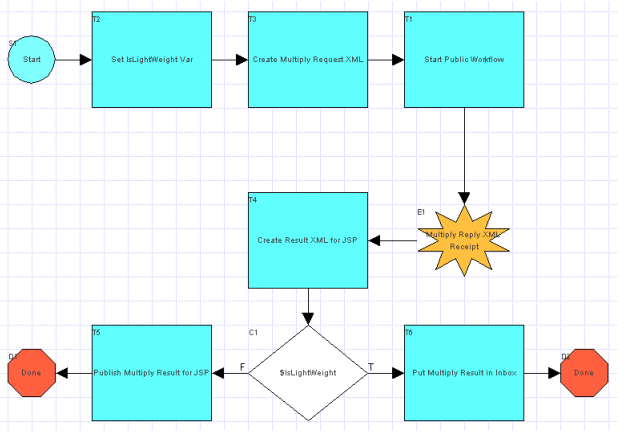
The following XML event contains the reply output of the private workflow:

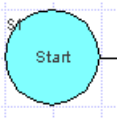
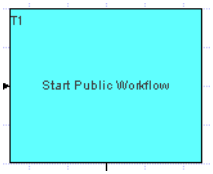
```
<multiply-outputs>
       <integer-product>35</integer-product>
       <note>Dear RequestorPartner: Here is the product of 7 and 5
from ReplierPartner to RequestorPartner.</note>
</multiply-outputs>
```

Requestor Private Workflow

The Requestor private workflow receives the initial request from the JSP, 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 JSP. This process is illustrated by the workflow in the following figure.

Figure 2-4 Requestor Private Workflow

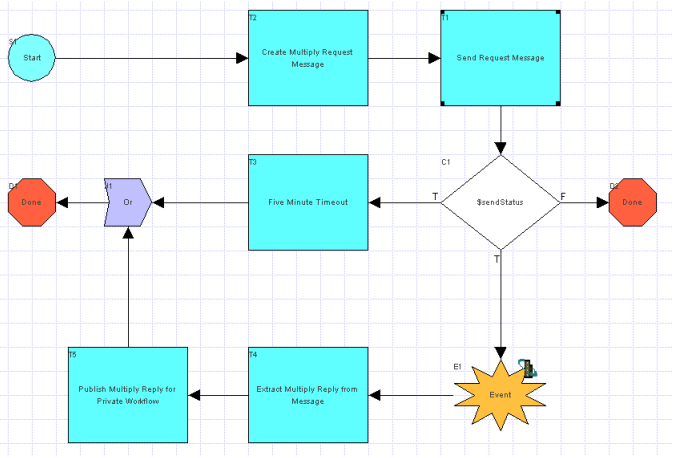


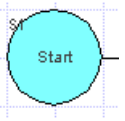
	An XML event received from the JSP triggers the workflow. The XML event is of the form `<from-multiply-request-jsp-to-workflow>` discussed in "XML Message Over JMS from JSP to Trigger Private Workflow." The Start node extracts the conversion string from XML, creates a `<multiply-request>` document, and 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 the `<multiply-reply>` document. A `<from-workflow-to-multiply-request-jsp>` document is created and sent to the JSP.
	A Stop node ends the workflow.

Requestor Public Workflow

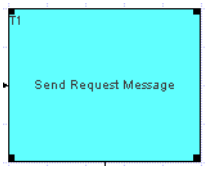
The Requestor public workflow is initiated by the XML event from the private workflow. It creates a message with the request XML, sends the message to the Replier, waits for a response, extracts the response XML from the response message, and then passes the response to the private workflow for processing.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

Figure 2-5 Requestor Public Workflow



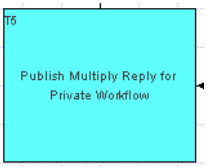


The workflow is called by having a `<multiply-request>` document passed to it. The Start node extracts the message string and stores it in a workflow variable.



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.



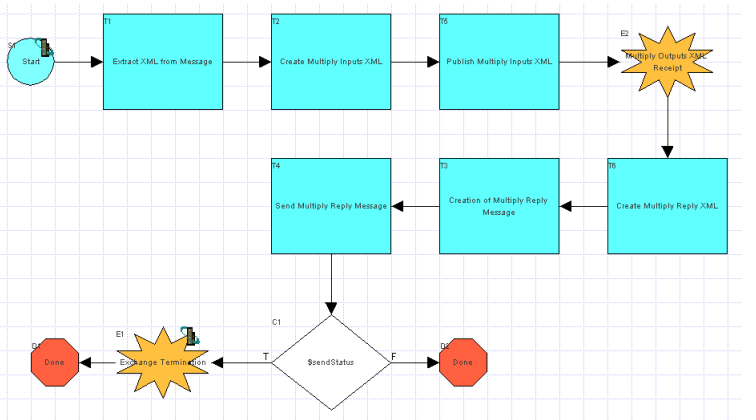
An Action node publishes the `<multiply-reply>` document as an XML Event.

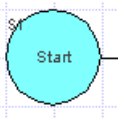
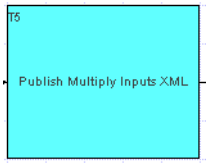
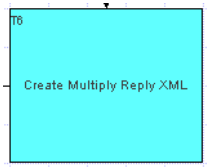
A Stop node ends the workflow.

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 in the following figure.

Figure 2-6 Replier Public Workflow



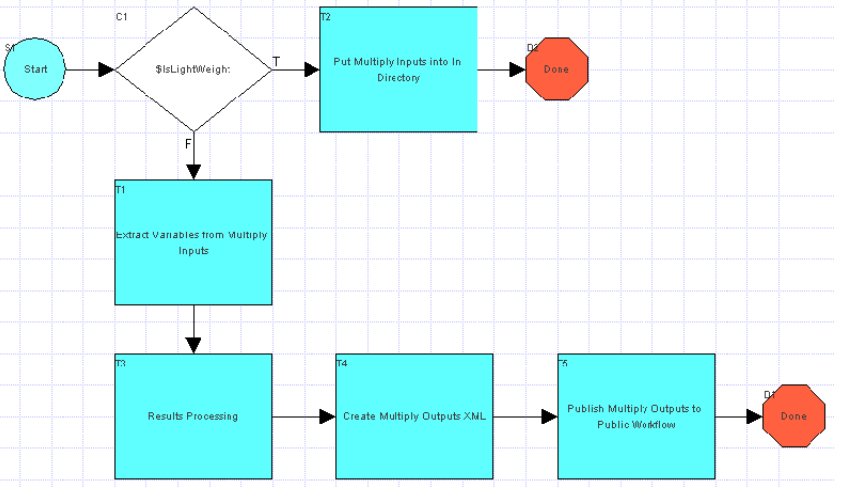
	The workflow is started upon receipt of a `<multiply-request>` document. The Start Node extracts the message content from the message and stores it in a workflow variable. The workflow variable is of format `<multiply-input>`, using the values from inside the `<multiply-request>` document, and the sender 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.

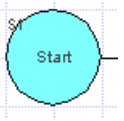
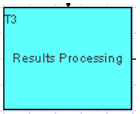
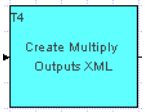
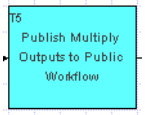
A Stop node ends the workflow.

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 publishes the reply XML back to the replier public workflow using an XML Event.

Figure 2-7 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 of 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 Stop node ends the workflow.

CHAPTER

3 Trading Partner Lightweight Client Sample

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

This section includes the following topics:

Overview of the Lightweight Client Sample

WebLogic Collaborate conversations usually take place between two trading partners who have both installed WebLogic Collaborate. 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 Collaborate installations. This communication is accomplished using two types of lightweight clients that are configured in a remote or host WebLogic Collaborate installation:

- Browser client—Uses JSP served up by predefined WebLogic Collaborate business processes to initiate a conversation, and send and receive messages. The Web host facilitates communication via JSP delivered on demand to the browser client. Uses the WebLogic Collaborate JSP tag libraries to send and check the messages using the WebLogic Collaborate mailbox interface.
- File-sharing client—Communication is accomplished using a 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 Collaborate 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. This sample does not use an actual FTP server installation; it uses the business operation to simulate the 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 Collaborate installation that has been preconfigured to handle lightweight clients.

The requestor sends a request for multiplication of two integer numbers. Requestor has no local WebLogic Collaborate installation. Instead, it uses a browser to access the JSPs that are served from a remote installation of WebLogic Collaborate. The JSPs create mailboxes and send XML requests using the WLC-provided JSP tag libraries. The same JSPs are also used to check the messages in the requestor's mailbox, as well as to delete messages from the requestor and replier mailboxes.

Note: In an actual business deployment, the requestor would not have access to the replier's mailboxes.

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

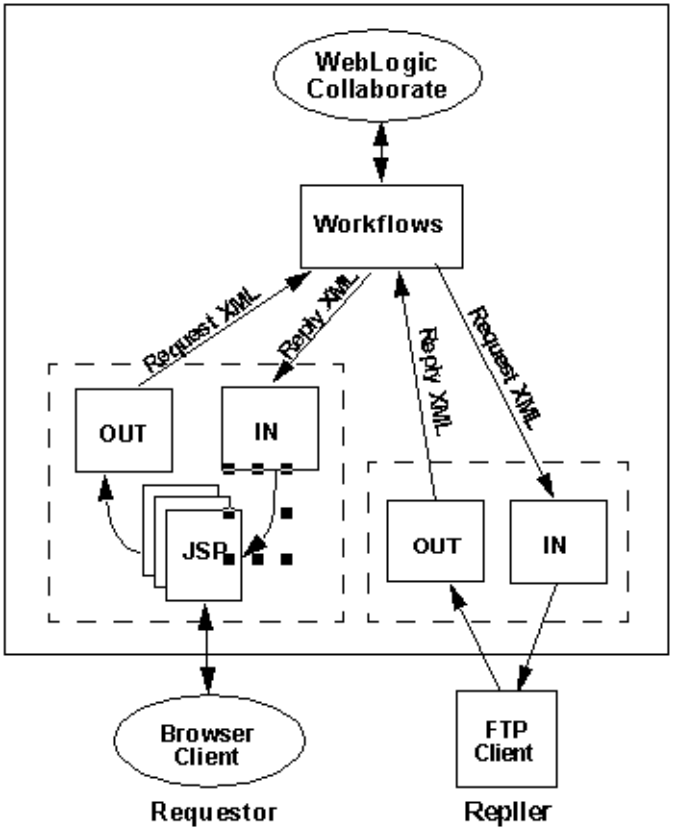
The Lightweight Client sample demonstrates the following events:

- Requestor's private workflow is triggered by an XML event from a JSP tag in the browser client.
- Replier's public workflow is triggered by a reply placed in the replier's out mailbox by the file-sharing client.
- Requestor's message and replier's message are put in the appropriate mailboxes.

In the Lightweight Client sample, there are four preconfigured mailboxes, one inbox and one outbox for each trading partner, as follows:

- Requestor using a browser client: BrowserTP1_Inbox and BrowserTP1_Outbox
- Replier using a file-sharing client: FtpTP1_Inbox, and FtpTP1_Outbox

Figure 3-1 Lightweight Client Deployment Diagram



Running the Lightweight Client Sample

The Lightweight Client sample requires no additional configuration; it can be run as delivered. See "Installing the Samples" on page 1-2.

To run the Lightweight Client sample implementation:

1. If you have already run the Hello Partner sample you have already bulkloaded all of the data required to run the Lightweight Client sample. The Lightweight Client sample shares WebLogic Collaborate information with the Hello Partner sample.

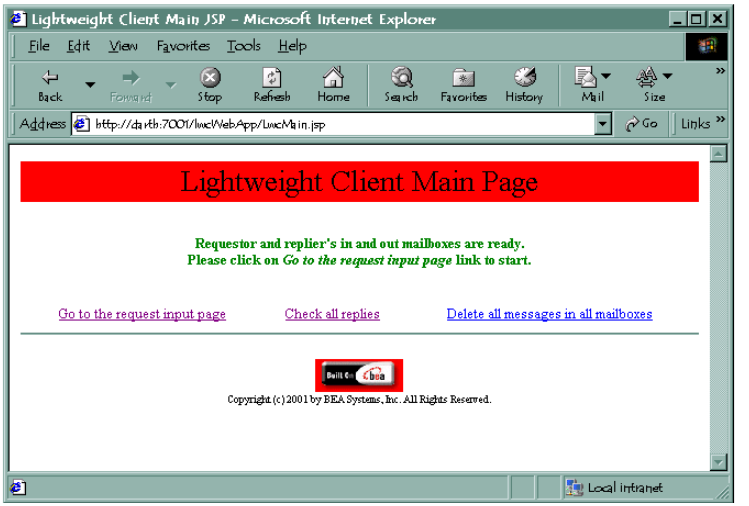
   If you have not run the Hello Partner sample, see "Hello Partner Sample" on page 2-1. See also "Installing the Samples" on page 1-2.

2. Run the Lightweight Client sample as follows:

   - Windows: From the Start menu, choose: WebLogic Integration 2.0→Collaborate→Samples→Run Samples→Lightweight Client
   - UNIX: Load the following sample file:

   ```
   http://localhost:7001/lwcWebApp/LwcMain.jsp
   ```

   The following illustration shows an example of the Lightweight Client main page.



3. Start the file-sharing client.

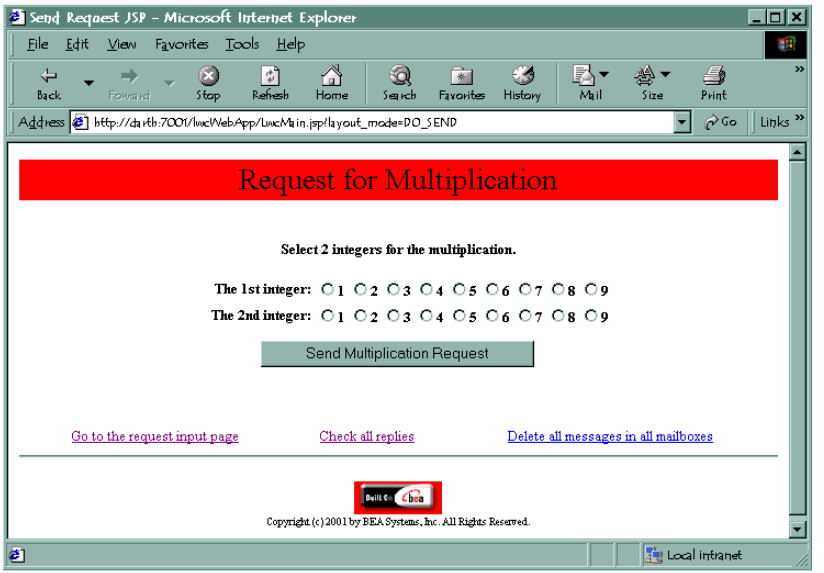
   Windows: Choose WebLogic Integration 2.0→Collaborate→Samples→Lightweight Client→Start File Sharing Client

   UNIX: Execute the following script:

   ```
   cd $WLC_HOME/config/samples/lightweightClient
   sh startFSClient.sh
   ```

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

4. Click Go to the request input page. The Request for Multiplication page is displayed as follows.

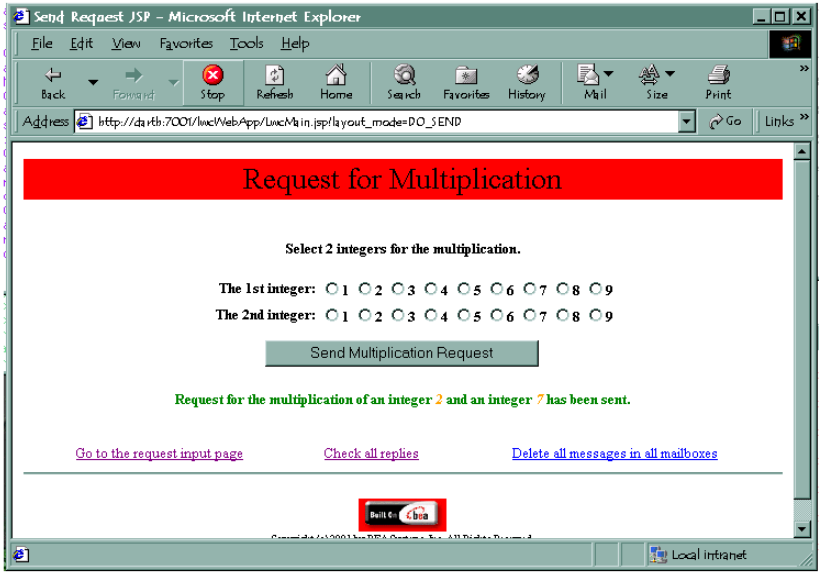


5. Test `SendmsgTag` and the wrapper Mailbox API by manually sending a message to `BrowserTP1_Outbox` as follows:

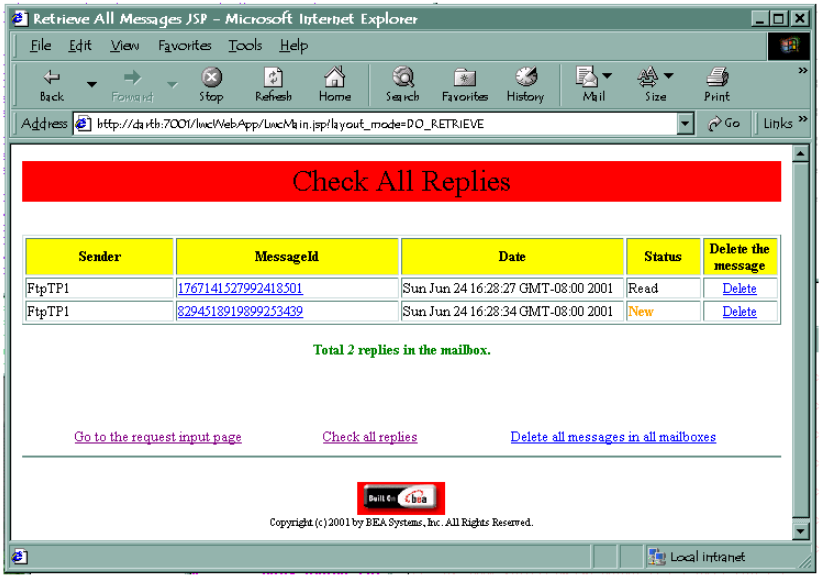
   a. Select two integers.

   b. Click Send Multiplication Request.

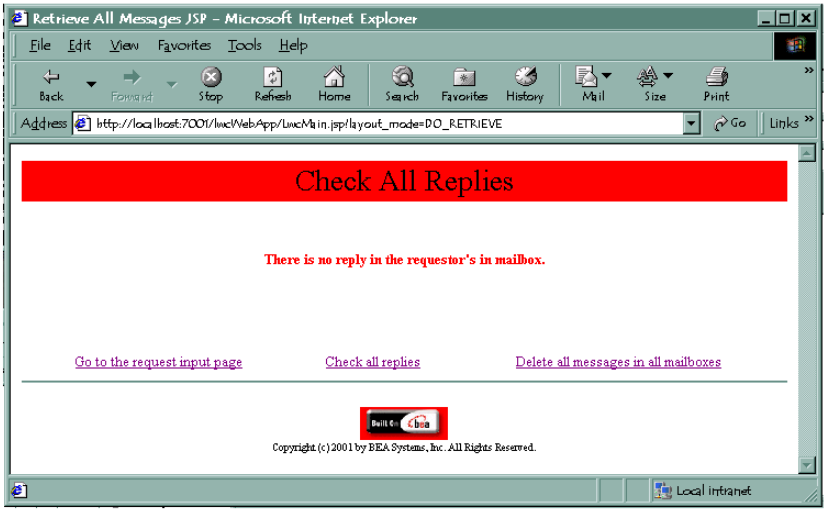
   Your display is updated, as shown in the following figure.



6. Test `CheckallmsgTag` by clicking Check all replies. A list of replies is displayed as shown in the following illustration.

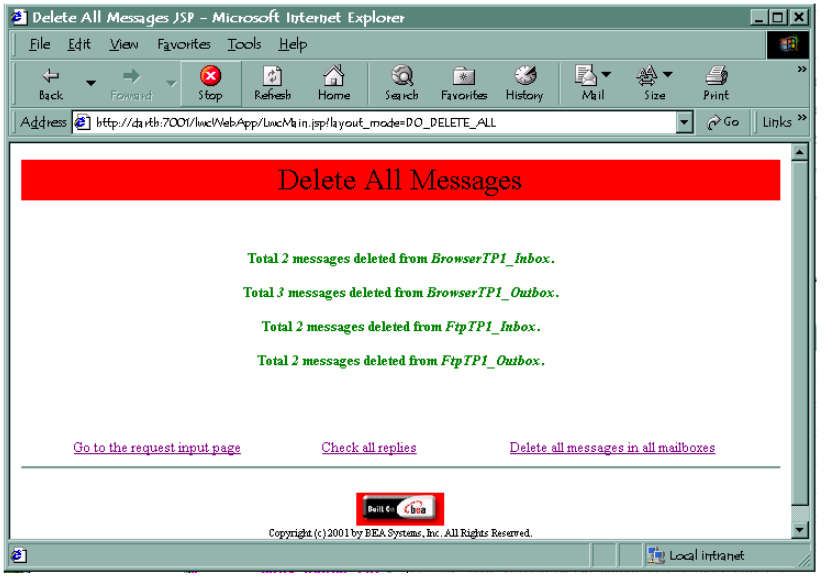


If no replies have been received, the following message is displayed.



7. Test `DeleteallmsgTag` as follows:

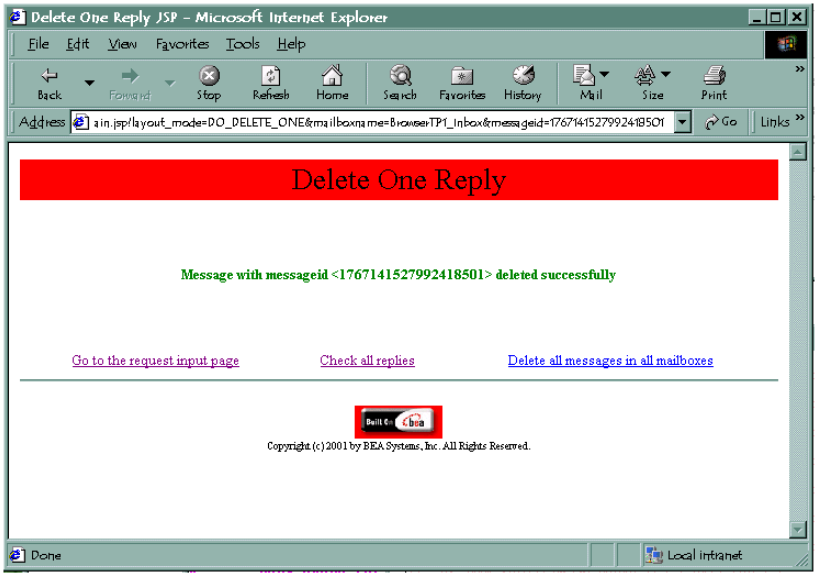
   a. Click Delete all messages in all mailboxes. A list of all the messages in all the WebLogic Collaborate mailboxes is displayed.



b. Click Delete all messages in all mailboxes again. Displays the total number of messages deleted.

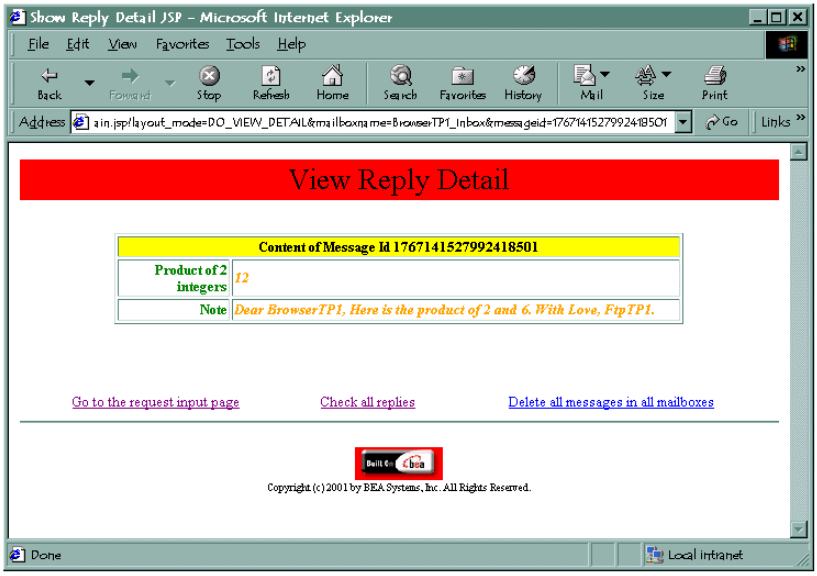
8. Test `DeletemsgTag` as follows:

- On the Check All Replies screen shown in Step 6, select a message
- Click Delete. The Delete One Reply screen confirms that the message was successfully deleted.



9. You can view message details as follows:

- On the Check All Replies screen shown in Step 6, click on the message in the Message ID column that you want to view.
- The View Reply Detail screen is displayed.



Creating and Using Lightweight Clients

This section provides information about creating and configuring lightweight clients:

- Lightweight Client Source Files
- Using the JSP Tag Library
- Configuring a Lightweight Client
- Configuring a Browser Client

Lightweight Client Source Files

The source directories included with WebLogic Collaborate contain all of the files required for creating your own lightweight clients. The Lightweight Client sample shares WebLogic Collaborate information with the Hello Partner sample.

Files and directories specific to the Lightweight Client sample are listed in the following table.

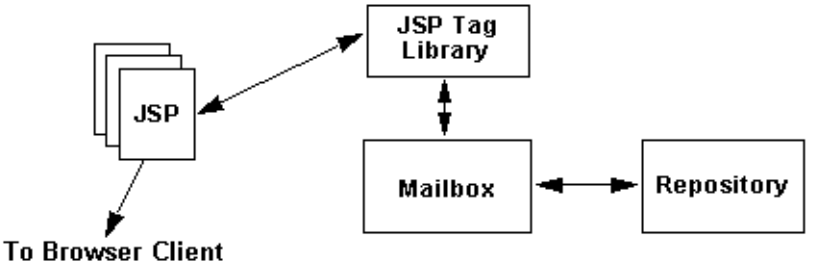
Source Directory	Contents
`WLC_HOME\dtd\`	
`lwcFileSync.dtd`	TheDTD file for `LwcFileSync.xml`.
`setenv.cmd` `setenv.sh`	Makes sure that `lwclient.jar`, applications, and classes are included in `JAVACLASSPATH` for both the spoke configuration and WebLogic Server.
`build.cmd` `build.sh`	Builds all of the samples. Once this script is executed, `lwcWebApp.war` is created and placed under `WLC_HOME\lib`. All of the class files are placed under `WLC_HOME\config\samples\applications\DefaultWebApp_myserver\WEBINF\classes`. **Note:** The `WAR` file is prepackaged and installed when you install WebLogic Collaborate.
`WLC_HOME\config\samples\lightweightClient\`	Contains the source files for the business operation and its helper to be called by a workflow instance.
`startFSClient.cmd` `startFSClient.sh`	A script that starts the file-sharing client that is responsible for transferring a request from the replier's in mailbox to its file sharing directory, and for transferring a reply from the replier's out directory to its out mailbox.

Source Directory	Contents
`LwcFileSync.xml`	File read by the file-sharing client. You must modify the following sections for your environment: ■ `url` ■ `wlcfilesync` *`name`* ■ `directoryin` ■ `directoryout`
`ftpDir\`	A directory created for the Lightweight Client sample runtime execution. It simulates the file-sharing client's in and out directories. Contains `FtpTP1_indir` and `FtpTP1_outdir` to simulate the in and out directories on the file-sharing client. Names are hard-coded for the sample. **Note:** This directory and its associated subdirectories are created at runtime.
`lwcWebApp\`	Contains the files for the Web application, such as `*.jpg` and `*.jsp` files.
`WEB-INF\`	Contains the JSP tag library description file, and `web.xml` used by the Lightweight Client web application.
`taglib\`	Contains the reference implementation of the WebLogic Collaborate 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 Collaborate Messaging API. The JSP tag library is used with predefined workflows to interact with a mailbox, as shown in Figure 3-2.

Figure 3-2 Mailbox Scenario Using the JSP Tag Library



The mailbox shown in Figure 3-2 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 WebLogic Collaborate 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

Additional steps are required when configuring a WebLogic Collaborate installation to support customers using lightweight clients.

A lightweight client does not need any security mechanisms; it is a daemon process that is independently run in its own Java Virtual Machine by WebLogic Collaborate. It serves any number of lightweight clients that have been configured through the `LwcFileSync.xml` configuration file. For additional information, see "Edit the File-Sharing File" later in this document.

Edit the Configuration File

Add the lightweight trading partner configuration information, shown in Listing 3-1 and Listing 3-2, to the WebLogic Collaborate `config.xml` file.

Listing 3-1 Web Application Configuration

```
<!-- LightweightClient settings -->
    <Application
```

```
      Name="LwcWebApp"
      Path="<WLC_HOME>/lib"
    >
      <WebAppComponent
        Name="LwcWebApp"
        ServletReloadCheckSecs="1"
        Targets="myserver"
        URI="LwcWebApp.war"
      />
    </Application>
```

Listing 3-2 File-Sharing Client Configuration

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

Edit the File-Sharing File

Configure the lightweight client by editing `lwcFileSync.xml`. The file-sharing client configuration is defined in `lwcFileSync.xml`, and conforms to `lwcFileSync.dtd`.

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

Listing 3-3 contains the DTD file. Listing 3-4 provides a sample XML file that sets up the lightweight trading partners. Replace the values shown in **`bold`** with values that are appropriate for your lightweight client installation.

Listing 3-3 Example of `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 3-4 Example of `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="<WLC_HOME>/ftpDir/FtpTP1_indir"
            directoryout="<WLC_HOME>/ftpDir/FtpTP1_outdir"/>
</wlcfilesyncconfig>
```

Configuring a Browser Client

WLC supports two methods of configuring a browser client, depending on the security paradigm desired:

- 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 application(s) 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 using the `examples.lightweightClient.LwcBizOp.putMessage()` method that 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 responsible for the hosted Web application deployment modifies *`WLC_HOME`*`\lwcWebApp\WEB-INF\web.xml` to add security constraints for all the users with permission to use the application.

   For example, to provide a new user with security privileges 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. The *newpartner* user must be a valid WebLogic Server user. The user can be created using either the WebLogic Server Administration Console or the WebLogic Collaborate Administration Console. For details, see the *BEA WebLogic Server Administration Guide*, or *Administering BEA WebLogic Collaborate*, respectively.
4. The *newpartner* user must be mapped to a valid trading partner lightweight client. This mapping is done using the WebLogic Collaborate Administration Console. Through the Console, you create the trading partner lightweight client and map the user's identification to the appropriate trading partner record. For details, see *Administering BEA WebLogic Collaborate*.
5. You must create a collaboration agreement between *newpartner* and the WebLogic Collaborate hub. Use the WebLogic Process Integrator Studio to create the collaboration agreement. For details, see *Using the BEA WebLogic Process Integrator Studio*.

Configuring an HTTPS (SSL) Browser Client

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

1. Configure certificates at both the browser and the WebLogic Collaborate nodes.
2. Create a WebLogic Server user called newpartner, who will 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 Collaborate Administration Console.
3. Use the WebLogic Collaborate Administration Console to create the trading partner lightweight client.
4. Map *newpartner* to the trading partner record.
5. Create a collaboration agreement between the lightweight client and the WebLogic Collaborate hub, using the WebLogic Collaborate Administration Console. For details, see *Administering BEA WebLogic Collaborate*.

CHAPTER

4 Messaging API Sample

This section describes the Messaging API sample, which demonstrates two message delivery mechanisms built with the WebLogic Collaborate messaging API and logic plug-in features. It includes the following topics:

- Overview of the Messaging API Sample
- Configuration of the Messaging API Sample
- Running the Messaging API Sample

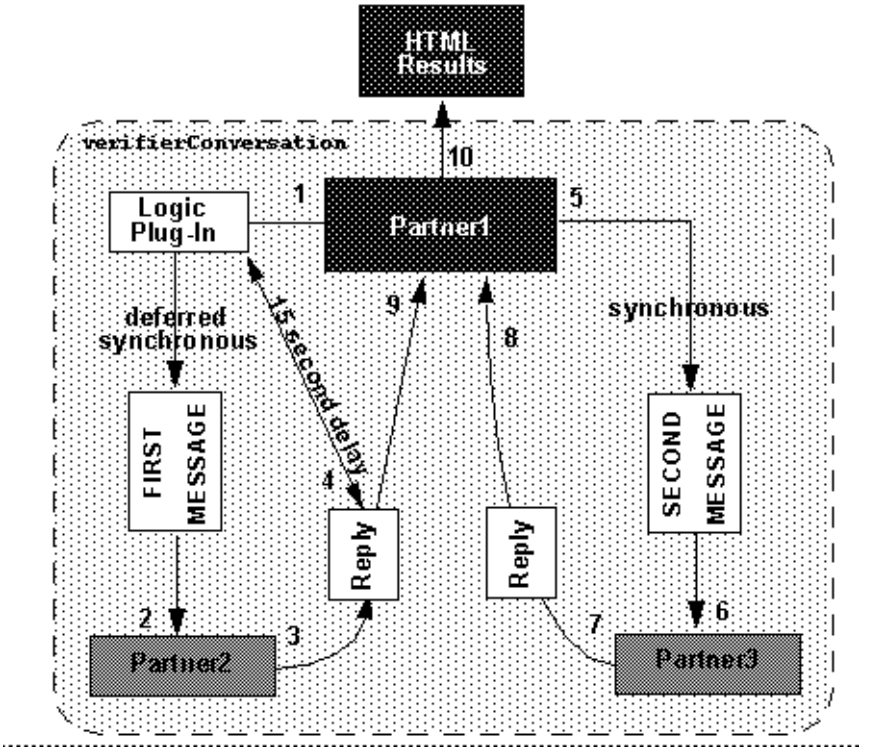
Overview of the Messaging API Sample

WebLogic Collaborate supports two message delivery methods:

- 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 timeout occurs or status is known, whichever happens first.
- Deferred Synchronous—Control returns to the application after a message is published. A `messageToken` object is returned to the application, which the application can later access to check the status of message delivery. Once a token is accessed, the application must wait until the status of the message delivery is known or a timeout occurs, whichever happens first.

Figure 4-1 illustrates the Messaging API sample conversation.

Figure 4-1 Diagram of the Messaging API Sample Conversation



1. Partner1 sends a message (1, 2) to Partner2, who replies (3, 4).
2. The logic plug-in holds the reply for 15 seconds.
3. Before looking for a reply from Partner2, Partner1 sends a second message to Partner3 (5, 6).
4. Partner3 replies to Partner1 (7, 8).
5. Partner1 now retrieves the reply from Partner2 (9).
6. The results are displayed by Partner1 in an HTML page (10).

A hub delivery channel (`VerifierCSpace`) is registered in the WebLogic Collaborate repository. The three trading partners in the delivery channel participate in a conversation (`verifierConversation`), assuming the following roles:

- Requestor (Partner1)
- Replyer (Partner2)
- Replyer (Partner3)

Message definitions are stored in two repository files: `request.dtd` and `reply.dtd`.

Partner1, the requestor trading partner, sends two different messages to Partner2 and Partner3. The first message contains the string `FIRST MESSAGE`, and is sent to Partner2 using the deferred synchronous delivery method.

Once the first message has been sent, and before checking for a reply from Partner2, Partner1 sends a second message containing the string `SECOND MESSAGE` to Partner3 using the synchronous delivery method.

At this point, Partner1 must wait for acknowledgment from Partner3 before performing any other action. When the acknowledgment from Partner3 is received and the reply is processed, Partner1 is able to retrieve the `messageToken` received when the first message was sent, and check for the acknowledgment received from Partner2.

The reply to the second message is then processed, and the results of the conversation are displayed for Partner1 in an HTML page produced by the `Post` method of the invoking servlet.

The invoking servlet used by Partner1 is represented by the `examples.MessageDeliveryMethods.MdmTp1Servlet.java` servlet, which is registered in the repository as `MdmTp1`. Partner2 and Partner3 use similar servlets, registered in their respective repositories as `MdmTp2` and `MdmTp3`.

The servlets used by Partner2 and Partner3 are `examples.MessageDeliveryMethods.MdmTp2Servlet.java` and `examples.MessageDeliveryMethods.MdmTp3Servlet.java`. They are registered in WebLogic Collaborate as `MdmTp2` and `MdmTp3`. These servlets are invoked by the HTML page `MessageDeliveryMethods.html`, which is located in the WebLogic Collaborate directory `%WLC-HOME%\config\samples\applications\DefaultWebApp_myserver.`

In order to simulate a delay in the reception of the first message by `Partner2`, a logic plug-in called `WaiterPlugIn` has been added to the WebLogic Collaborate `filter` chain; the `WaiterPlugIn` class checks for the target recipient of the outgoing message. If the target recipient is Partner2, the corresponding thread sleeps for 15 seconds; if the target recipient is not Partner2, the message is sent immediately to the target trading partner (Partner3).

The first message is sent to the target recipient using a deferred synchronous delivery method. As a result of the sending operation, an `XOCPMessageToken` object (`token1`) is returned to the sender. This token is used to check for the acknowledgment of the reception by Partner2, as follows:

```
// send the message in synchronous deferred mode
XOCPMessageToken token1 = (XOCPMessageToken)smsg1.send();
debug("Sending message 1: conversation id="+c.getId()+" done");
```

Configuration of the Messaging API Sample

The following data is bulkloaded during samples installation and configuration:

- One conversation definition: `verifierconversation`.
- Two messaging protocols:
  - `XOCP-hub`
  - `XOCP-spoke`
- Four trading partner definitions:
  - Hub
  - Partner1
  - Partner2
  - Partner3
- Four document exchange definitions:
  - One for the hub with its own transport, end point, and a routing proxy
  - One for each of three spokes, Partner1, Partner2, and Partner3, each with its own transport, end point, and delivery channel
- Four delivery channels:
  - One for the hub
  - One for each spoke

Running the Messaging API Sample

1. Install the samples as described in “Installing the Samples” on page 1-2.

   Make sure that you have bulkloaded the schemas and data, and have started WebLogic Collaborate.

2. Start the Messaging API sample.

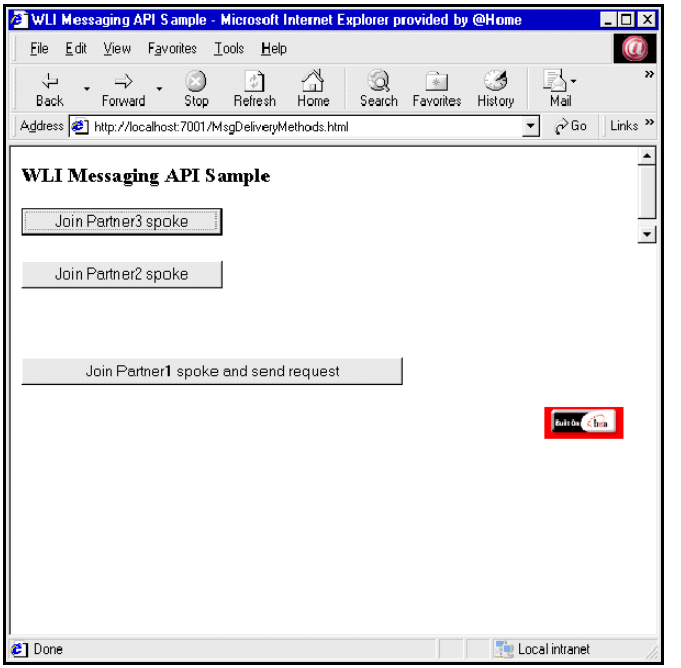
   Windows: Choose WebLogic Integration 2.0→Collaborate→Samples→Run Samples→Messaging API

   UNIX: Load the following HTML page:

   `HTTP://localhost:7001/MsgDeliveryMethods.html`

   A browser window is opened with a Messaging API screen, as shown in the following illustration.

Figure 4-2 Messaging API Sample First Screen



3. Submit a request to start WebLogic Collaborate for Partner3, Partner2, and Partner1.
4. Click Join Partner 3 spoke.

   A WebLogic Collaborate session is started for `Partner3`, and Partner3 is registered in the `replyer` role for the `verifierConversation` conversation.
5. Click Join Partner 2 spoke.

   A WebLogic Collaborate session is started for Partner2, who is registered with the `replyer` role for the `verifierConversation` conversation.
6. Click Join Partner 1 spoke and send request.

   A WebLogic Collaborate session is started for Partner1, and Partner1 is registered with the `requestor` role for the `verifierConversation` conversation. Partner1 acts as the conversation initiator, and sends `message1` and `message2` to Partner2 and Partner3, respectively.

   Note: You can follow the progression of the sample by checking the `weblogic.log` file displayed in the WebLogic Collaborate Administration Console.
7. The outgoing message is analyzed by the `WaiterPlugIn` logic plug-in. If the target recipient is Partner2, the corresponding thread sleeps for 15 seconds before delivering the message to the trading partner; otherwise the message is delivered immediately.
8. Partner3 receives message2 from Partner1 in the form of a string: `SECOND MESSAGE`. Partner3 converts the string to lowercase, and adds the following prefix to it: `partner3 --` . The result is the string `partner3 -- second message`. At this point, the string is added to a business document and sent back to Partner1.
9. Partner1 receives the message, processes a response, and then loops for ten seconds.
10. After the 10-second interval, Partner1 receives the token for sending message1, and then waits for a response from Partner2.
11. Partner2 receives the response after `WaiterPlugIn` has slept for 15 seconds.

12. Partner2 receives message1 from Partner1 in the form of a string: `FIRST MESSAGE`. Partner2 converts the string to lowercase, and adds the following prefix to it: `partner2 -- `. The result is the string `partner2 -- first message`. At this point the string is added to a business document and sent back to Partner1.
13. Partner1 receives the message and processes the response.
14. Partner1 terminates the conversation and shuts down the session. When the session is stopped, Partner2 and Partner3 automatically leave the conversation. The console displays a log message similar to the following:

```
Wed Oct 25 17:47:42 PDT 2000:<I <WLC <XOCP-Hub INFO: Trading Partner, Partner1
left C-Space, VerifierCSpace

Wed Oct 25 17:47:42 PDT 2000:<I <WLC <User ***MdmTp1Servlet: done.
```

Messaging API Sample Output

When the Messaging API sample is executed successfully, the following output is displayed in the browser window:

```
Partner1 process flow:
Starting enabler... done.
Creating conversation :
verifierConversation:1.0:requestor_http://127.0.0.1:7001/Enabler1_0_97252124232
9...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_http://127.0.0.1:7001/Enabler1_
0_972521242329
success
Shutting down session... done.
```

CHAPTER

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 Collaborate Messaging API. Error handling uses standard Java exception and error handling via WebLogic Collaborate 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
- CreateboxTag
- RemoveboxTag

SendMsgTag

Used to pass 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 the following names for the appropriate mailbox: ■ For incoming mail: *`trading_partner_name`*`_Inbox` ■ For outgoing mail: *`trading_partner_name`*`_Outbox`
`String message`	The message content.
`String sender`	Mailbox address for the sender. This may be an e-mail address or FTP server address.
`String URL`	The URL for the hosting trading partner.
`String security`	Values may be `ON` or `OFF`.

Example

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

ChecknewmsgTag

Used to check 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 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 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 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 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 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

Variable

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