BEA WebLogic Integration™

Using the Data Integration Plug-In

Release 2.1 Service Pack 1
Document Date: January 2002

Copyright

Restricted Rights Legend

Trademarks or Service Marks

Using the Data Integration Plug-In

Part Number	Date	Software Version
N/A	January 2002	2.1 Service Pack 1

Contents

Index

CHAPTER

1 Understanding the Data Integration Plug-In

This document describes the functionality and operation of the data integration plug-in. The following topics are discussed:

- Understanding the Data Integration Plug-In
- Using the Data Integration Plug-In
- Running the WebLogic Integration Sample Applications

Understanding XML Translation

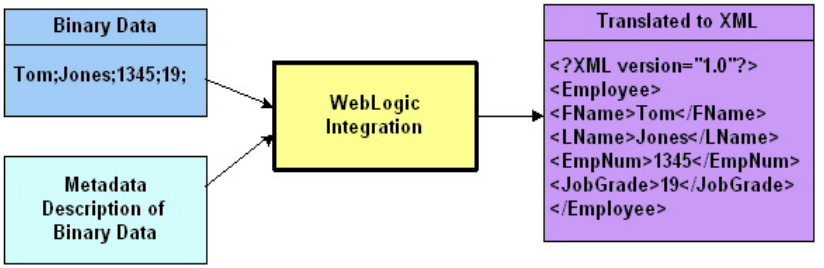
Within most enterprise application integration (EAI) domains, data translation is an inherent part of an EAI solution. XML is quickly becoming the standard for exchanging information between applications; it is invaluable in integrating disparate applications. Most data transformation engines, however, do not support translations between binary data formats and XML. The data integration plug-in provides for an exchange of information between applications by supporting data translations between binary formats from legacy systems and XML.

Data that is sent to or received from legacy applications is often platform-specific information organized in a binary format unique to the machine on which the information originated. Binary data is not self-describing, so in order to be understood by an application, information about the format of this data (metadata) must be embedded within each application that uses binary data from a legacy application.

XML is becoming the standard for exchanging information between applications because XML embeds a description of the data within the data stream, facilitating the exchange of data among applications. Although XML can represent complex data structures, it is easily parsed. As a result, the coupling of applications no longer depends on the embedding of metadata.

Binary-to-XML translation is the conversion of structured binary data to an XML document so that the data can be accessed via standard XML parsing methods. You must create the metadata used to perform the conversion. During the translation process, each field of binary data is converted to XML according to the metadata defined for that field. The metadata you specify must include the name of the field, the data type, the size, and an indication of whether the field is always present or optional. This description of the binary data is used to translate the data to XML. Figure 1-1 shows how a sample piece of binary data is translated into XML.

Figure 1-1 XML Data Translation of: Tom;Jones;1345;19;



Applications developed on a WebLogic Server platform often use XML as the standard data format. If you want the data from your legacy system to be accessible to other applications on a WebLogic Server platform, you may use WebLogic Integration to translate that data from binary to XML format, or from XML to binary format. If you need the data to be converted to a particular XML dialect for end use, you must transform it using an XML data-mapping tool.

What Is Data Integration?

WebLogic Integration facilitates the integration of data from diverse enterprise applications by supporting the translation of binary legacy system data to XML and vice versa. Once legacy data is available as XML, it can be consumed directly by XML applications, transformed into a specific XML grammar, or used directly to start workflows in the WebLogic Integration Studio. WebLogic Integration supports non XML-to-XML translation and vice versa through the use of three data integration tools:

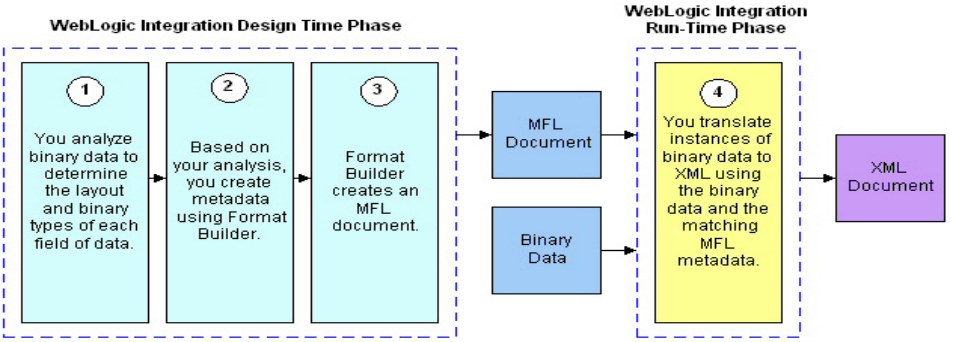
- Design-Time Data Integration Component
- Run-Time Data Integration Component
- Plug-In to Business Process Management Functionality

A translation is a two-step process. First, create a description of your binary data using Format Builder, the design-time tool. This task involves analyzing binary data so that the layout of records in the binary file is accurately reflected in the metadata you create in Format Builder.

Next, you create metadata (a description of the input data) in Format Builder and save this metadata as a Message Format Language (MFL) document. WebLogic Integration provides *importers*, utilities that automatically create message format definitions from common sources of binary metadata, such as COBOL copybooks.

You can then use the run-time component in WebLogic Integration to translate instances of binary data to XML. Figure 1-2 shows the event flow for non XML-to-XML data translation. A plug-in to BPM functionality simplifies the task of configuring translations.

Figure 1-2 Event Flow for Non XML-to-XML Translation Using Data Integration



Design-Time Data Integration Component

The design-time data integration component of WebLogic Integration is a Java application called Format Builder. Format Builder is used to create descriptions of binary data records. It allows you to describe the layout and hierarchy of the binary data so that it can be translated to or from XML.

The description you create in Format Builder is saved in an XML grammar called Message Format Language (MFL). MFL documents contain metadata used at run-time for data integration. The data integration plug-in to BPM functionality also uses this metadata to translate an instance of a binary data record to an instance of an XML document (or vice-versa).

Format Builder also creates a DTD or XML Schema document that describes the XML document created from a translation.

Run-Time Data Integration Component

The run-time data integration component of WebLogic Integration is a Java class with various methods that are used to translate data between binary and XML formats. This Java class can be used in several ways. Specifically, it can be:

- Deployed in an EJB using WebLogic Server
- Invoked as a business operation from a workflow in the Studio
- Integrated into any Java application

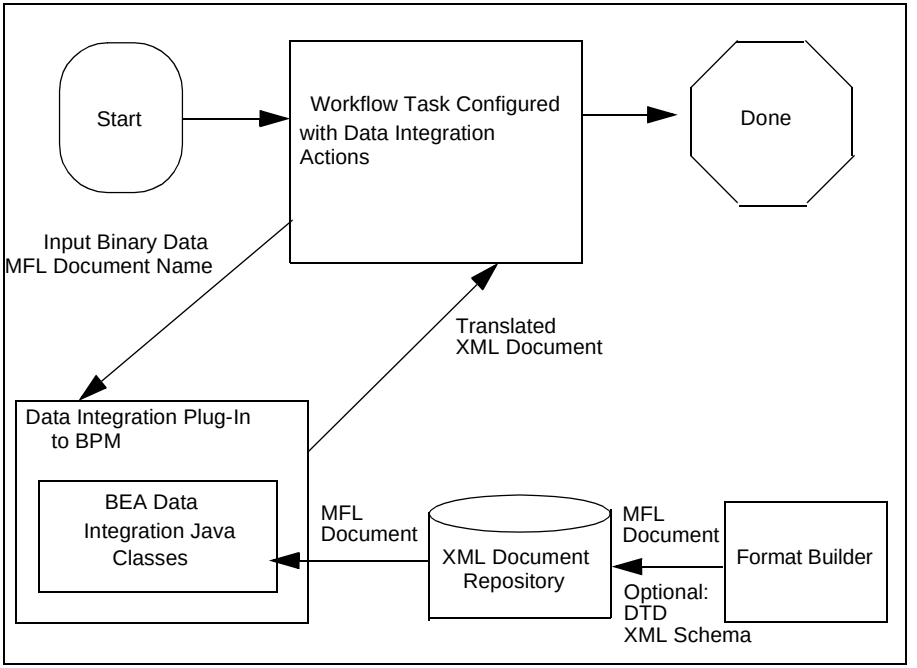
Plug-In to Business Process Management Functionality

The data integration plug-in for business process management (BPM) functionality supports an exchange of information between applications by making it possible to translate binary data from legacy systems into XML. The data integration plug-in provides BPM actions that allow you to perform XML-to-binary and binary-to-XML translations.

In addition to this data translation capability, the data integration plug-in provides:

- Event data processing in binary format
- In-memory caching of MFL documents and translation object pooling to boost performance
- A `BinaryData` variable type to edit and display binary data
- Execution within a clustered WebLogic Server environment.

The following illustration describes the relationship between data integration and BPM functionality.



Using the Repository

The repository provides a centralized document storage mechanism that supports the following four document types:

- MFL - Message Format Language documents
- DTD - XML Document Type Definition documents
- XSD - XML Schema documents
- XSLT - XSLT Stylesheet

The repository provides access to these document types and allows you to use them in data integration, business process management, and B2B integration functions. The repository also includes a batch import utility that allows previously constructed MFL, DTD, XSD, and XSLT documents to be easily migrated into the repository.

CHAPTER

2 Using the Data Integration Plug-In

This section provides information about the following topics:

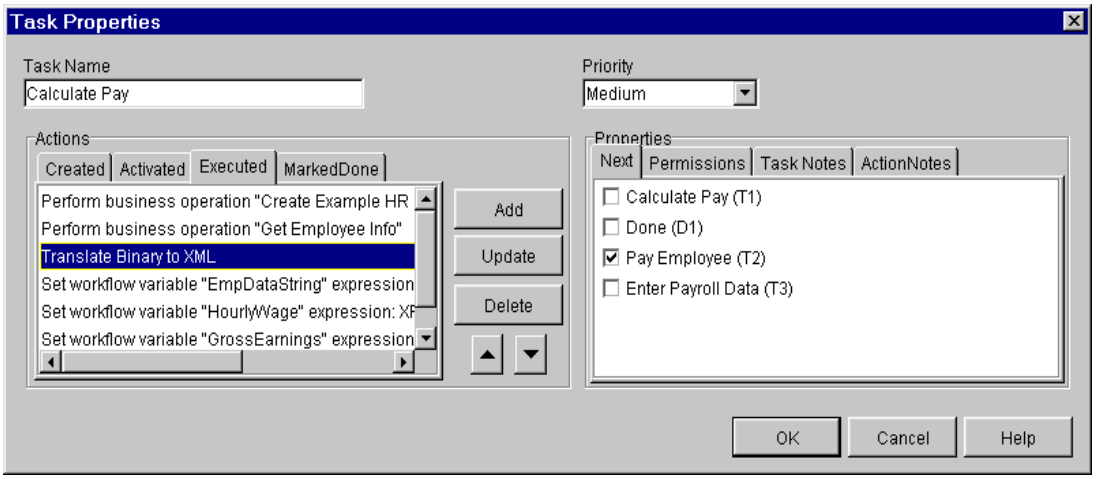
- Data Translation with the Data Integration Plug-In
- Processing Event Data
- Enhancing Data Translation Performance
- Custom Data Types and the Data Integration Plug-In
- Support for WebLogic Server Clustering

Data Translation with the Data Integration Plug-In

The data integration plug-in provides capabilities for translating non XML documents into XML, and vice versa, while executing business process management (BPM) functions. To perform such a translation, complete the following procedure:

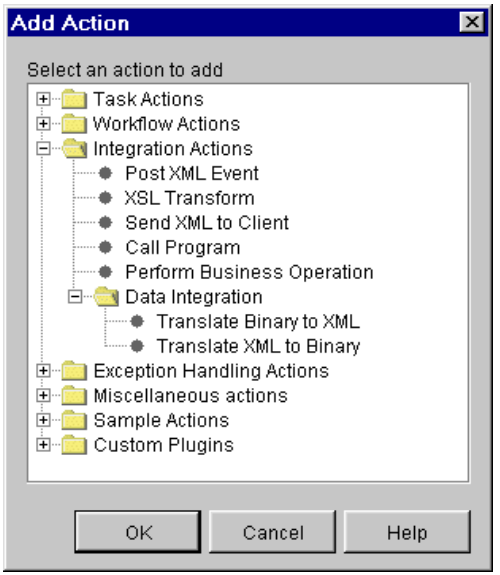
1. Start the WebLogic Integration Studio. For instructions on starting and logging on to the Studio, see Using the Studio Interface.
2. Open the desired template definition and double-click a task. The Task Properties dialog box is displayed (Figure 2-1).

Figure 2-1 Task Properties Dialog Box



3. If the selected task contains the data translation action, select it again from the list, click Update and proceed to step 4. Otherwise, click Add to add a new action. The Add Action dialog box is displayed (Figure 2-2).

Figure 2-2 Add Action Dialog Box



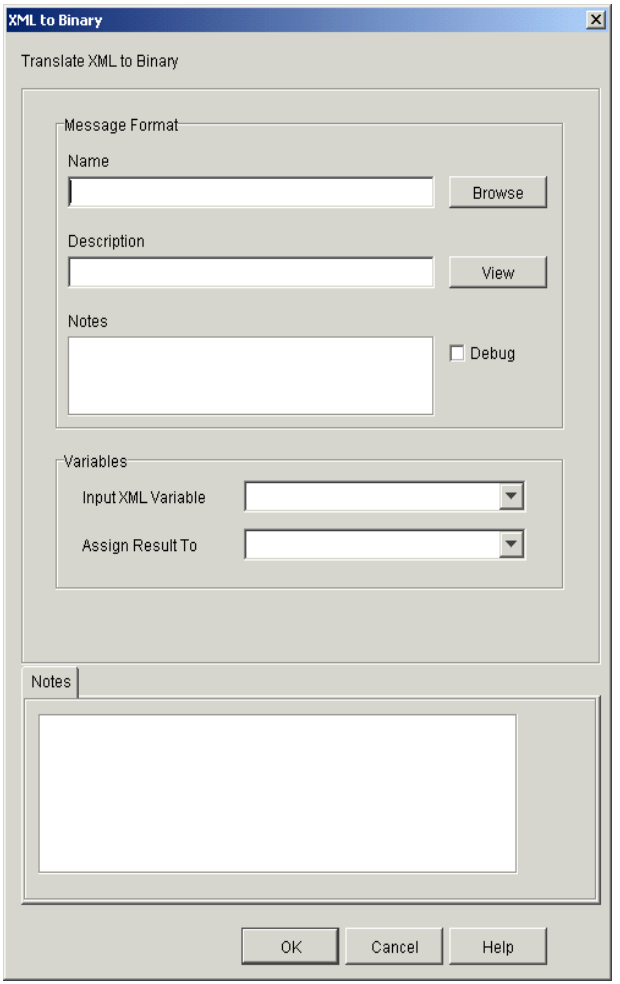
4. Expand the Integration Actions node; a list of actions is displayed. Expand the Data Integration item in that list. A list of translation actions is displayed: Translate XML to Binary and Translate Binary to XML. Select the type of translation you want to perform.

Translate XML to Binary

To perform an XML-to-binary translation:

1. From the Add Action dialog box (Figure 2-2), select Translate XML to Binary. The Translate XML to Binary dialog box is displayed (Figure 2-3).

Figure 2-3 Translate XML to Binary Dialog Box



2. Enter data in the fields as described in the following table.

Table 2-1 Translate XML to Binary Dialog Box Fields

Dialog Box Area	Field	Description
Message Format Parameters	Name	Name of the message format. You can type a name directly in the text field or click Browse to invoke a list of repository documents from which you can make a selection.
	Description	Displays the description of the message format. **Note:** This field is used only for displaying text. You cannot edit text in this field.
	Notes	Displays the notes attached to the message format. **Note:** This field is used only for displaying text. You cannot edit text in this field.
Message Format Action Buttons	Browse	Allows you to browse MFL documents in the repository.
	View	Displays the items in the Message Format area so you can verify that you have selected the correct document type for translation.
	Debug	Enables or disables debug messaging. When you select this option, the translation actions are written to the WebLogic Server log file.
Variable Parameters	Input XML Variable	Displays the XML workflow variables. Select the variable you want to use in the translation, or create a new variable as follows: 1. Type a name for your new variable and click OK. A confirmation message box is displayed. 2. Click Yes to create the new variable.
	Assign Result To	Displays the binary data workflow variables. Select the variable in which you want to store the translated information or create a new variable as follows: 1. Type a name for your new variable and click OK. 2. Click Yes to create the new variable.

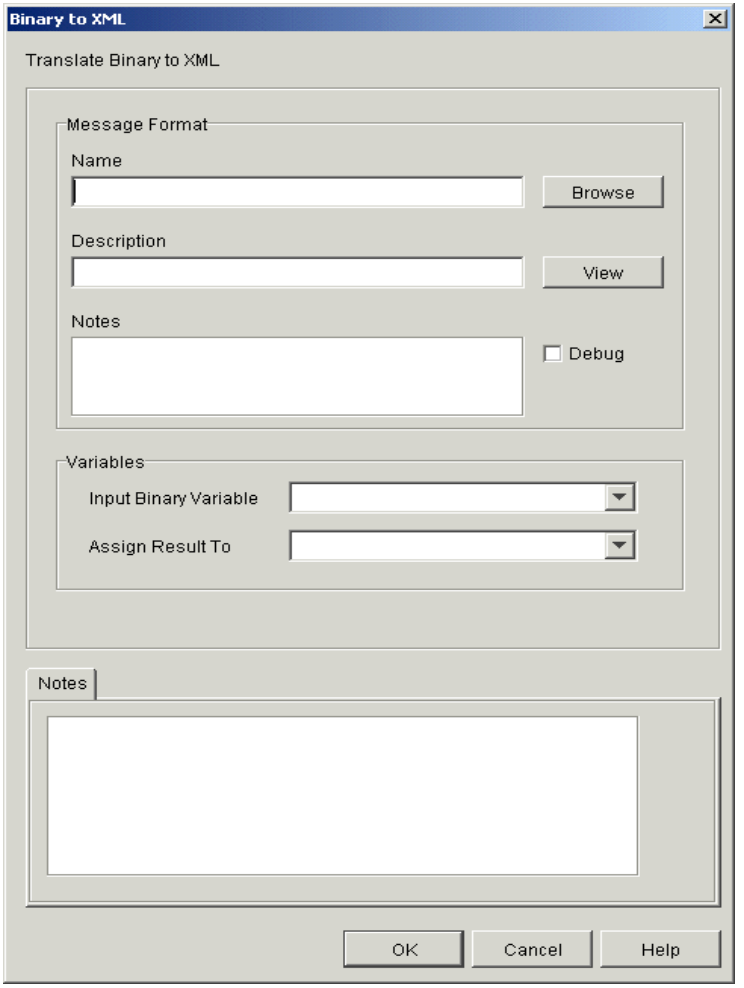
3. Click OK to save the translation information to your workflow.

Translate Binary to XML

To perform a binary-to-XML translation:

1. In the Add Action dialog box (Figure 2-2), choose Integration Actions→Data Integration→Translate Binary to XML. The Translate Binary to XML dialog box is displayed (Figure 2-4).

Figure 2-4 Translate Binary to XML Dialog Box



2. Enter data in the fields as described in the following table.

Table 2-2 Binary to XML Dialog Box Fields

Dialog Box Area	Field	Description
Message Format Parameters	Name	Name of the message format. You can type a name directly in the text field, or click Browse to invoke a list of repository documents from which you can make a selection.
	Description	Displays the description of the message format. **Note:** This field is used only for displaying text. You cannot edit text in this field.
	Notes	Displays the notes attached to the message format. **Note:** This field is used only for displaying text. You cannot edit text in this field.
Message Format Action Buttons	Browse	Allows you to browse MFL documents in the repository.
	View	Displays the items in the Message Format area so you can verify that you have selected the correct document type for translation.
	Debug	Enables or disables debug messaging. When you select this option, the translation actions are written to the WebLogic Server log file.
Variable Parameters	Input Binary Variable	Displays the binary workflow variables. Select the variable you want to use in the translation, or create a new variable as follows: 1. Type a name for your new variable and click OK. A confirmation message box displays. 2. Click Yes to create the new variable.
	Assign Result To	Displays the XML data workflow variables. Select the variable you want to use to store the translated information, or create a new variable as follows: 1. Type a name for your new variable and click OK. 2. Click Yes to create the new variable.

3. Click OK to save the translation information to your workflow.

Processing Event Data

The data integration plug-in provides a function that enables binary data to trigger workflows by converting the binary data to XML or by preprocessing it at the front end of event processing. This function is referred to as the *event handler*. The event handler is executed when JMS messages are published to a topic.

Three JMS properties are required for a message to be preprocessed by the data integration plug-in:

- `WLPIContentType: "binary/x-application/wlxt"`
- `WLPIPlugin: "com.bea.wlxt.WLXTPlugin"`
- `WLPIEventDescriptor: MFL_document_name`

The first two JMS message properties are constant for all messages addressed to the event handler. The third property contains the name of the MFL document that describes the binary data in the message.

Note: The MFL document referenced in the WLPI EventDescriptor must be stored in the repository.

Listing 2-1 is a sample of code used to build a message that can be processed by the event handler.

Listing 2-1 Sample Event Handler Code

```
byte[] bindata = ... the binary data ...
pub = sess.createPublisher(topic);
BytesMessage msg = sess.createBytesMessage();
msg.writeBytes(bindata);
msg.setStringProperty("WLPIPlugin", "com.bea.wlxt.WLXTPlugin");
msg.setStringProperty("WLPIContentType",
 "binary/x-application/wlxt");
msg.setStringProperty("WLPIEventDescriptor", “mymfldoc”);
pub.publish(msg);
```

The servlet sample application shows how the message built by this code is processed by the event handler. See “Running the Servlet Sample” on page 3-2 for information on running the servlet sample.

Enhancing Data Translation Performance

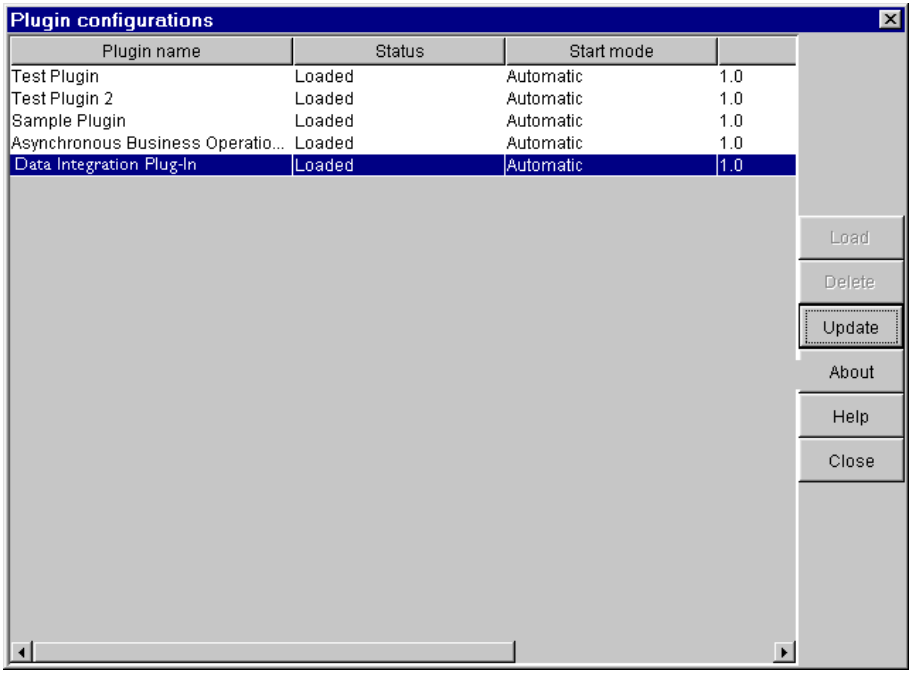
The data integration plug-in provides a configuration panel for monitoring and administering in-memory cache of the MFL document and for enabling or disabling event handler debugging. Using this panel, you can adjust the in-memory cache and translation object pool to enhance the performance of your data translations.

Note: You must clear the MFL document in-memory cache in order for any updates you make to an MFL document to take effect.

To access the configuration panel, follow the steps below. For more information about the actions specific to business process management, refer to the business process management documentation.

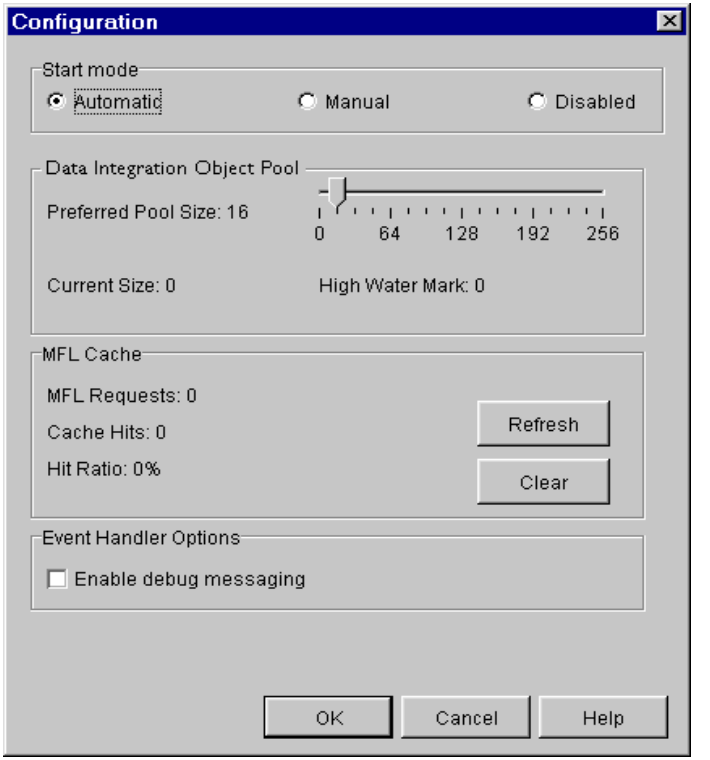
1. Start the WebLogic Integration Studio. For instructions on starting and logging on to the Studio, see Using the Studio Interface.
2. Choose Configuration→Plugins. The Plugin Configuration dialog box is displayed (Figure 2-5).

Figure 2-5 Plug-In Configuration Dialog Box



3. Select the Data Integration Plug-In and click Update. The Configuration dialog box for the data integration plug-in is displayed (Figure 2-6).

Figure 2-6 Configuration Dialog Box for the Data Integration Plug-In



4. To monitor and enhance translation performance, enter data in the fields described in the following table.

Dialog Box Area	Field	Description
Start Mode	Automatic	Opens the data integration plug-in automatically when the Studio is opened.
	Manual	Makes the data integration plug-in available from the Studio.
	Disabled	Disables the use of the data integration plug-in from the Studio.

Dialog Box Area	Field	Description
Data Integration Object Pool	Preferred Pool Size	Defines the maximum number of permanent objects in the pool. Use the slider to set the pool size to the desired number. **Note:** The translation engine creates temporary pool objects if the demand exceeds the preferred pool size you have set. These objects are deleted when they are returned to the pool.
	Current Size	Displays the number of objects currently in the pool.
	High Water Mark	Displays the largest number of objects in the pool since the server was started.
MFL Cache	MFL Requests	Displays the total number of requests for translation of MFL documents.
	Cache Hits	Displays the number of requests made while the necessary MFL document was already in the cache.
	Hit Ratio	Displays the percentage of requests satisfied by retrieving MFL documents from the cache, rather than from the database.
MFL Cache Action Buttons	Refresh	Sends a request to the server to update the MFL cache statistics.
	Clear	Clears the MFL document cache. For all subsequent translation requests, MFL documents must be loaded from the repository.
Event Handler Options	Enable Debug Messaging	Enables or disables debug messaging for the event handler. If enabled, debug messages are written to the WebLogic Server log file during translation.

The data integration plug-in extends the capabilities for displaying and editing provided by standard BPM functionality. These capabilities are provided by the Hex Editor component of Format Tester for displaying and editing binary data.

Variable Types and the Data Integration Plug-In

The data integration plug-in provides a `BinaryData` variable type that you can use to edit and display binary data. The `BinaryData` variable acts as a container for a logical set of binary data with additional display capabilities. This variable is used by programs that call the actions provided by the data integration plug-in to pass and receive binary data. It is also used by the Workflow Instance Monitor to display and edit the contents of a binary variable.

Custom Data Types and the Data Integration Plug-In

WebLogic Integration provides a user-defined type feature that allows you to create custom data types that accommodate your unique data type requirements. The user-defined type feature allows these custom data types to be plugged in to the data integration run-time engine. Once a user-defined data type is plugged in, it is indistinguishable from a built-in data type both in terms of features and functionality.

Configuring User Defined Data Types

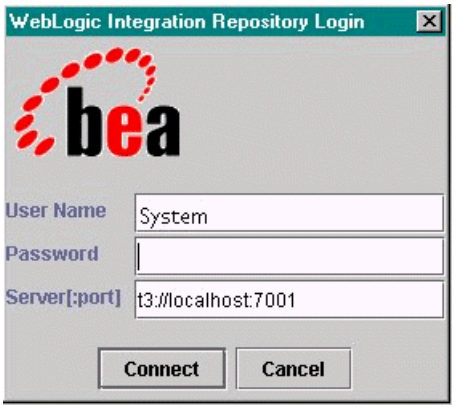
User-defined types used by the data integration plug-in are stored in the WebLogic Integration repository as `CLASS` documents. At run time, the data integration plug-in loads user-defined type classes from the repository as required. In addition, the data integration plug-in exports the MFL and class files required to support the active template, allowing a template to be imported, intact, on another business process management instance. Class documents may be placed in the repository using either of two methods. The following sections describe those methods:

- Using the Format Builder
- Using the Repository Import Utility

Using the Format Builder

To publish a user-defined type to the repository using the Format Builder, complete the following procedure:

1. Start the Format Builder by choosing Start→Programs→BEA WebLogic E-Business Platform→WebLogic Integration 2.1→Format Builder. The Format Builder main window is displayed.
2. Choose Repository→Log In. The WebLogic Integration Repository Login window is displayed.



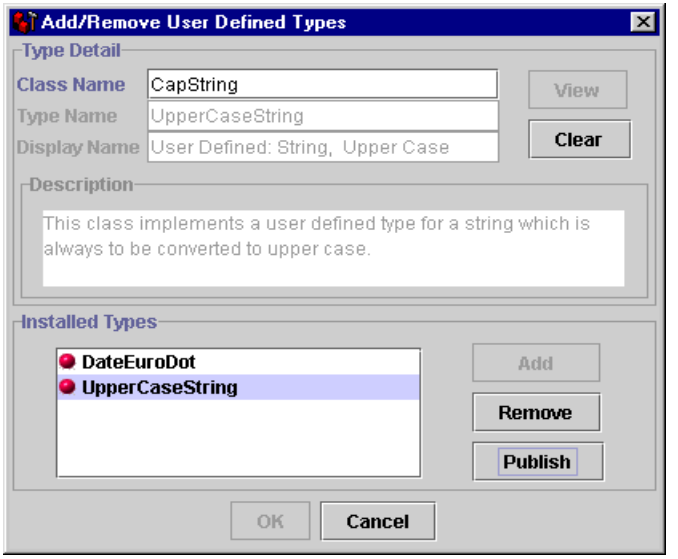
3. Enter the user ID specified for the server users in the User Name field. (The default is `wlipisystem`.)
4. Enter the password specified for the server users in the Password field. (The default is `wlpisystem`.)
5. Enter the name of the server and the number of the associated port in the Server[:port] field.

   Note: The WebLogic Integration Repository Login window allows you to make up to three login attempts. If you are unsuccessful after three attempts, a

login failure message is displayed. If you fail to login three times, choose Repository→Log In to repeat the login procedure.

6. Click Connect. If your login attempt is successful, the Login window is closed and the Format Builder Title bar displays the server name and port number entered in the WebLogic Integration Repository Login window. A menu of active repository items is displayed. Select the one you want to access

7. Choose Tools→User Defined Types. The Add/Remove User Defined Types dialog box is displayed.



Once a connection to the repository is established, the Add/Remove User Defined Types dialog box displays the status of each registered user-defined type and allows for the publication of each such type to the repository. The status of each user-defined type in the repository is indicated by an icon, in the form of a ball, before each entry in the Installed Types area of the dialog box.

The color of the icon before each name of a user-defined type indicates the status of that type:

- Green - The user-defined type has been published to the repository.
- Yellow - The user-defined type has been published to the repository, but the local version of the class differs from the repository version.
- Red - The user-defined type does not exist in the repository.

8. From the list of installed types, select the class you want to publish and click Publish. The icon for the selected entry should become green, indicating that the class was successfully placed in the repository.

Using the Repository Import Utility

To use the repository import utility to import Java class files, including user-defined types complete the following procedure:

1. Create a `wlxt-repository.properties` file in the CLASSPATH. The content of this file should be as follows:

   `wlxt.repository.url=<server url>`

   For example:

   `wlxt.repository.url=t3://localhost:7001`

2. Type the following command to pass the class filename on the Import command line.

   `java com.bea.wlxt.repository.Import <filename>`

   For example, the following command imports all the class files in the current directory:

   `java com.bea.wlxt.repository.Import *.class`

Note: Any Java class file may be imported into the repository using the repository import utility, as well as User Defined Types. This capability is useful if a user-defined type relies on additional class files that do not extend the `com.bea.wlxt.bintype.Bintype` class.

Support for WebLogic Server Clustering

The data integration plug-in can operate successfully in a clustered WebLogic Server environment. In a clustered environment, the plug-in administrator is connected to only one node of the cluster at any given time. Any commands issued by the administrator must be propagated to the other nodes in the cluster.

Communication among the various servers in a cluster is handled through a JMS topic. The topic is used for communication to different nodes in a cluster in a WebLogic Integration environment.

Configuring the Data Integration Plug-In for Clustering

If you want to take advantage of the clustering capability, you must configure the data integration plug-in as follows:

1. Create a JMS topic on one of the servers in the cluster. The JNDI name of this topic must be as follows:

   `com.bea.wlxt.cluster.BroadcastTopic`

   Note: For more information about creating JMS topics, see the WebLogic Server documentation.

2. Open the `config.xml` file in a text editor. This file can be found in the `config\samples\` directory under your WebLogic Integration installation.

   Note: The `config` directory contains separate subdirectories for each domain you have created. Each subdirectory contains its own `config.xml` file. Make sure you open the correct file.

3. Locate the <Application> section for business process management and add the following lines anywhere in this section:

   ```
   <EJBComponent Name="wlxt-cluster"
    DeploymentOrder="99"
    Targets="[server_name]"
    URI="wlxtmb.jar"
   />
   ```

4. Save the `config.xml` file.

Note: You must restart the server in order for the change to the `config.xml` file to be recognized.

3 Running the WebLogic Integration Sample Applications

The data integration software includes two sample applications designed to illustrate the integration of business process management (BPM) in WebLogic Integration. This section describes these samples and give you step-by-step instructions for running them. The following topics are discussed:

- Prerequisites
- Running the Servlet Sample
- Running the EJB Sample

Prerequisites

The instructions presented in this section are based on the assumption that you have a good working knowledge of WebLogic Integration, and that you understand how data integration and the WebLogic Integration process engine work. In addition, you should have successfully installed WebLogic Integration and run a sample workflow before you try to run the sample applications.

Running the Servlet Sample

This sample application implements a Web archive file (`WLPI_sample.war`) that installs a servlet. The servlet accepts requests for conversion of binary data to XML. It is accessed via a browser and responds by displaying the generated XML data. In addition, the application posts the data to the WebLogic Integration event topic in either XML or binary format. The data may then be used to start a workflow.

What Is Included in the Servlet Sample

The servlet sample application resides in the `\samples\di\` subdirectory of the directory in which WebLogic Integration is installed. The following table describes the files included in the Servlet sample application.

Table 3-1 Servlet Sample Application Files

Directory	File	Description
`\wlpi\source`	`WLPI_sample.java`	Source code for the servlet used to display HTML files on the screen and to translate binary data to XML. This XML may be placed, optionally, on the JMS topic.
`\wlpi`	`SampleData.mfl`	Message Format Language description of the sample binary data file used to start the sample workflow.
`\wlpi`	`SampleData.data`	Sample data file used as input when the sample workflow is started.
`\wlpi`	`DI_ServletSample.jar`	Exported workflow used in the sample. This workflow is imported automatically when you configure the samples as described in “Step 1. Start the Sample Application Launcher” on page 3-3.
`\wlpi`	`Makefile`	Make file for building the sample source to a `.jar` file.
`\wlpi`	`build.cmd`	Command that builds the `.jar` file from source.
`\wlpi\images`	`bealogo.jpg`	BEA logo displayed on the HTML page rendered by the sample servlet.

Table 3-1 Servlet Sample Application Files

Directory	File	Description
\wlpi\WEB-INF	hello.html	HTML page used by the sample servlet to obtain input data from the user.
\wlpi\WEB-INF	web.xml	J2EE configuration file defining deployment information for the sample servlet.
\wlpi\WEB-INF	weblogic.xml	BEA configuration file defining WebLogic-specific information for the sample servlet.
wlpi\WEB-INF\lib	cos.jar	Utility libraries used in the execution of the sample code.
wlpi\WEB-INF\lib	HtmlScreen.jar	Utility libraries used in the execution of the sample code.
Under WebLogic Integration Home Directory		
config\samples\applications	WLPI_sample.war	Web archive file containing all executable sample code and configuration files. This file is automatically deployed to the WebLogic Integration application directory upon installation.

How to Run the Servlet Sample

To run the servlet sample, complete the procedure in this section. For instructions about the tasks specific to WebLogic Server and BPM functions, see the BEA documentation at e-docs.bea.com.

Step 1. Start the Sample Application Launcher

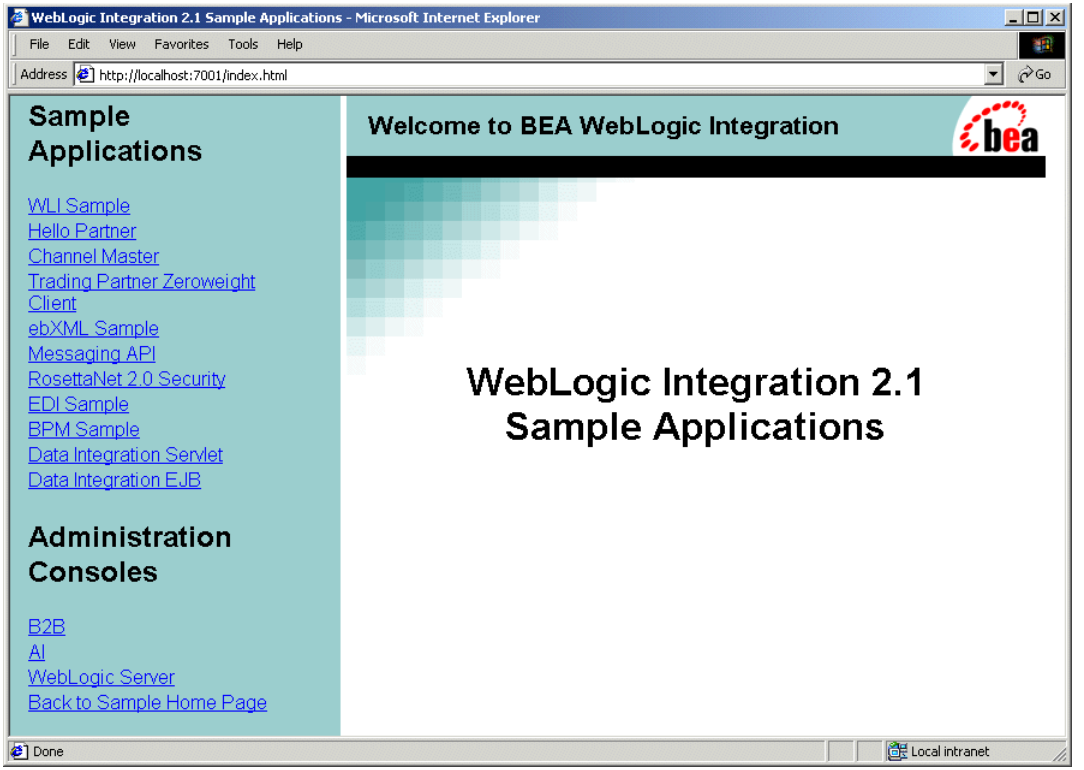
For first-time users:

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

- Windows

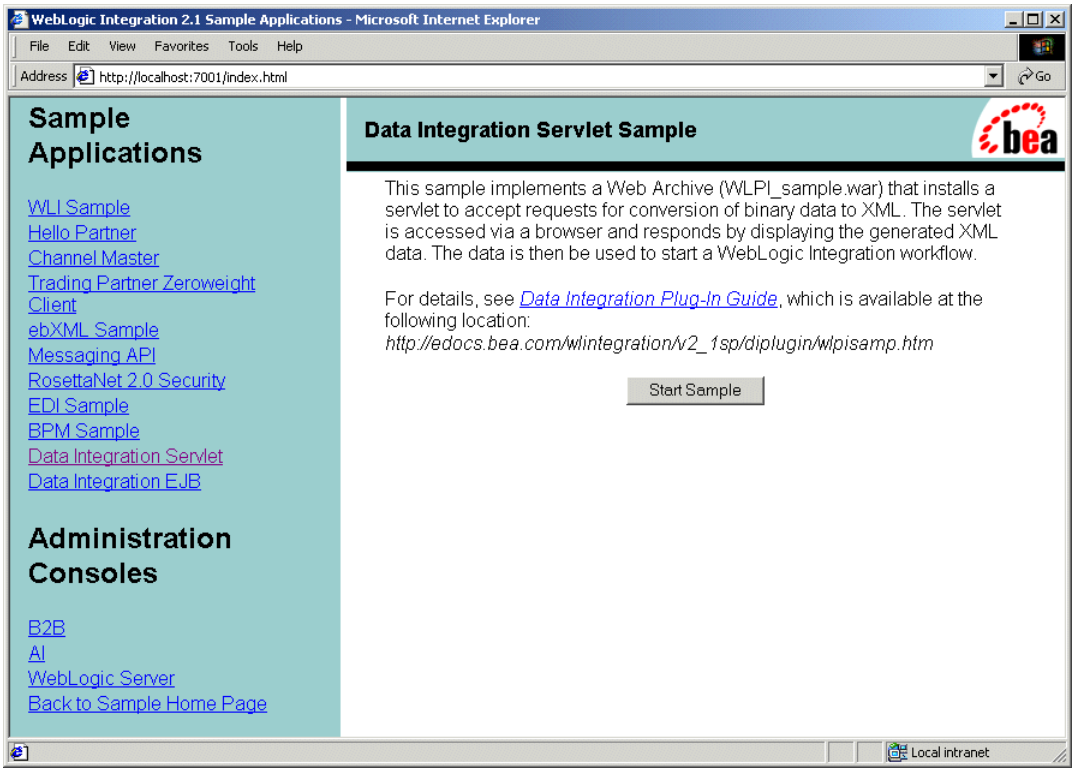
1. Choose Start→BEA WebLogic E-Business Platform→WebLogic Integration 2.1→Samples→Run Samples. All samples are configured all and the samples launcher is displayed. It takes several minutes to configure all samples. Figure 3-1 shows the Sample Application Launcher.

Figure 3-1 Sample Application Launcher



2. Select the Data Integration Servlet Sample. “Data Integration Servlet Sample Page” on page 3-5 shows the Data Integration Servlet Sample page.

Figure 3-2 Data Integration Servlet Sample Page



- UNIX

1. Make sure your `PATH` environment variable includes the directory in which the Netscape executable (`netscape`) resides. If it is not included, the samples launcher page cannot be displayed.
2. Go to the WebLogic Integration home directory (the directory in which you installed WebLogic Integration). For example:

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

3. Run the `setenv` script to set the top-level WebLogic Integration environment variables:

   ```
   . setenv.sh
   ```

4. Run the `RunSamples` script:

   ```
   cd samples/bin
   RunSamples
   ```

5. 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.

If you have already configured samples:

Start the server and display the Samples Launcher by completing the procedure appropriate for your platform:

- Windows

1. Choose Start→BEA WebLogic E-Business Platform→WebLogic Integration 2.1→Samples→Start Server.
2. Choose Start→BEA WebLogic E-Business Platform→WebLogic Integration 2.1→Samples→Samples Launcher.
3. Select the Data Integration Servlet Sample.

- UNIX

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

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

2. Start the server by entering:

   ```
   . ./startServer
   ```

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

   ```
   StartServer execution successful
   ```

4. Start a Web browser using the following URL:

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

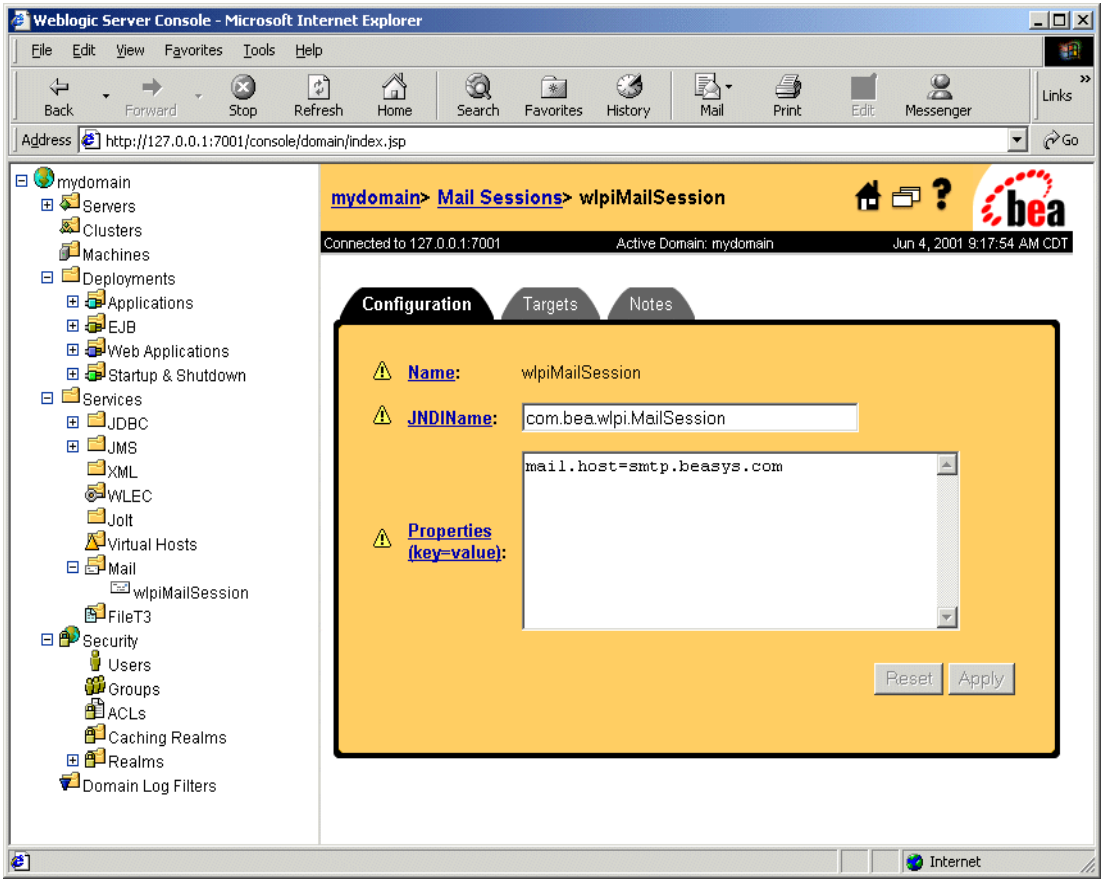
   The Samples Launcher Web page is displayed.

Step 2. Configure the Mail Session

This step is optional if you have already configured your mail host. You may want to perform it anyway, to verify your configuration.

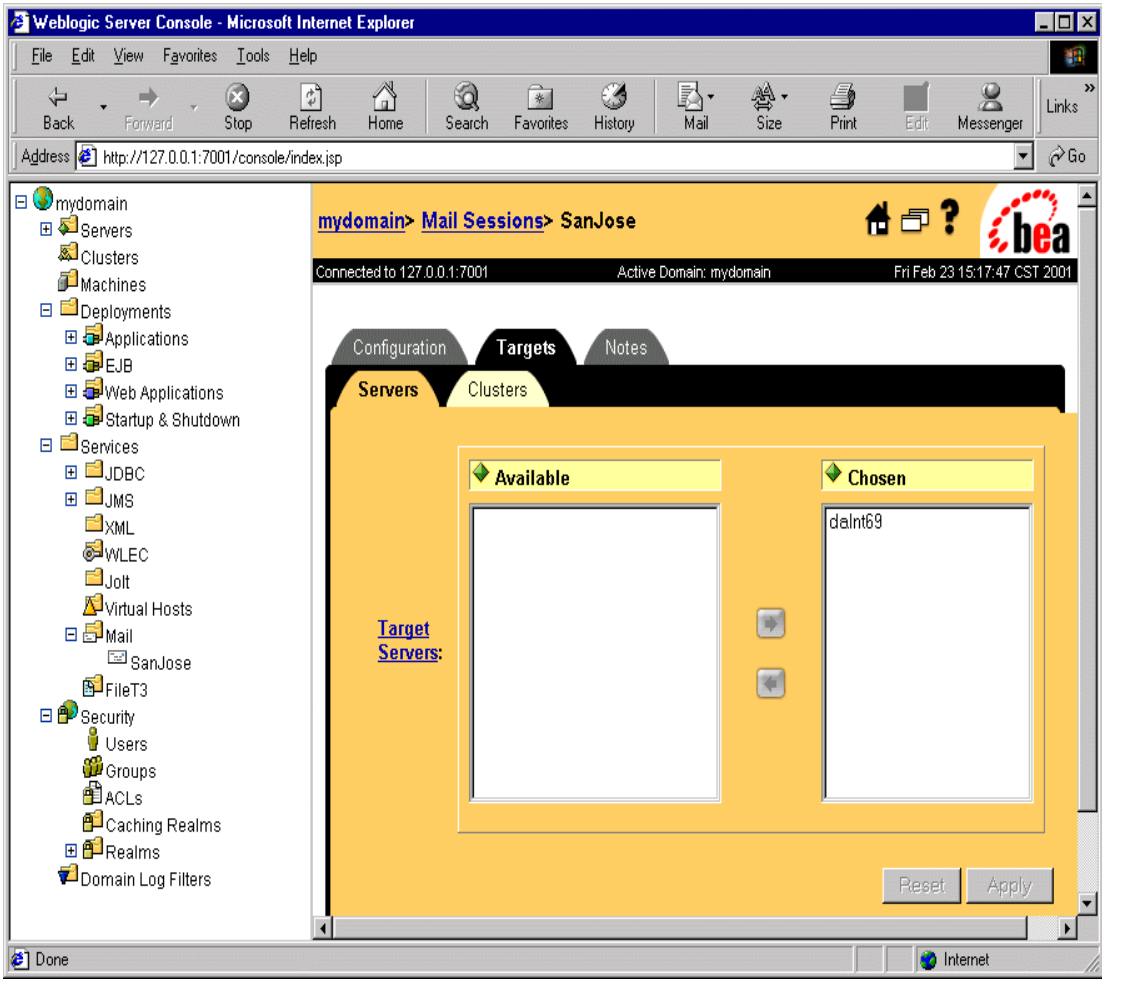
1. Click on WebLogic Server under Administration Consoles on the Samples Launcher. The WebLogic Server Administration Console is displayed.
2. From the navigation tree, choose Services→Mail→wlpiMailSession.
3. Enter the appropriate information to configure your mail host. Make sure that `mail.host=mailserver`. The following figure shows an example of the Configuration tab in the Mail Session window.

Figure 3-3 Configuration Tab in the Mail Sessions Window



4. Select the Targets tab.
5. Move the name of your mail server name from the Available column to the Chosen column, as shown in the following figure.

Figure 3-4 Server Tab in the Mail Sessions Window



6. Click Apply.

Step 3. Update the Sample XML Data and Send Message

1. In a text editor, open the file `\samples\di\wlpi\SampleData.data`. Replace the text *`user@nowhere.com`* with a valid e-mail address that can be used by the workflow to deliver an e-mail message.
2. In the Input File field on the Sample Application Launcher page, navigate to the following data file:

   `\samples\di\wlpi\SampleData.data`

   The file specified must reside on the local system, which is not necessarily the one on which the server is running. If the file is not on the local system, an error message is displayed.
3. Click Submit. A short e-mail message is sent to the address you supplied in the data file.

Running the EJB Sample

This sample simulates a dataflow from an HR system to a payroll system, initiated by the entry of payroll data. Employee data is obtained from a legacy payroll system in which binary data is used. The data is translated to XML so that a calculation to determine the employee's pay information can be performed. The result of the calculation is translated back to binary format and sent on to the payroll system.

What Is Included in the EJB Sample

The following table describes the files included in the EJB sample application. This application resides in the `\samples\di\ejb` directory.

Table 3-2 EJB Sample Application Files

Directory	File	Description
\ejb	Makefile	Make file for building the sample source to a .jar file.
	WLXTExample.jar	Sample workflow exported from data integration.
	HR.mfl	MFL file for binary data returned from the sample HR Bean.
	Payroll.mfl	MFL file for binary data passed for the sample Payroll Bean.
	Autopay.cmd	Windows NT command script for initiating the workflow from the command line.
	Autopay.sh	UNIX shell script for initiating the workflow from a command line.
	build.cmd	Builds wlxtejb.jar from source.
\ejb\source	Payroll.java	Sample EJBs representing the legacy payroll system.
	PayrollHome.java	
	PayrollBean.java	
	HR.java	Sample EJBs representing the legacy HR system.
	HRHome.java	
	HRBean.java	
	AutoPay.java	Program that places a preformatted message on the Event topic to start the sample workflow.
	HexDump.java	Utility class used by the sample EJBs.
	EmployeeRecord.java	Employee data class used by the sample HR EJB.
Under WebLogic Integration Home Directory		
config\samples\lib	WLXTEJB.jar	Executables for the sample application.

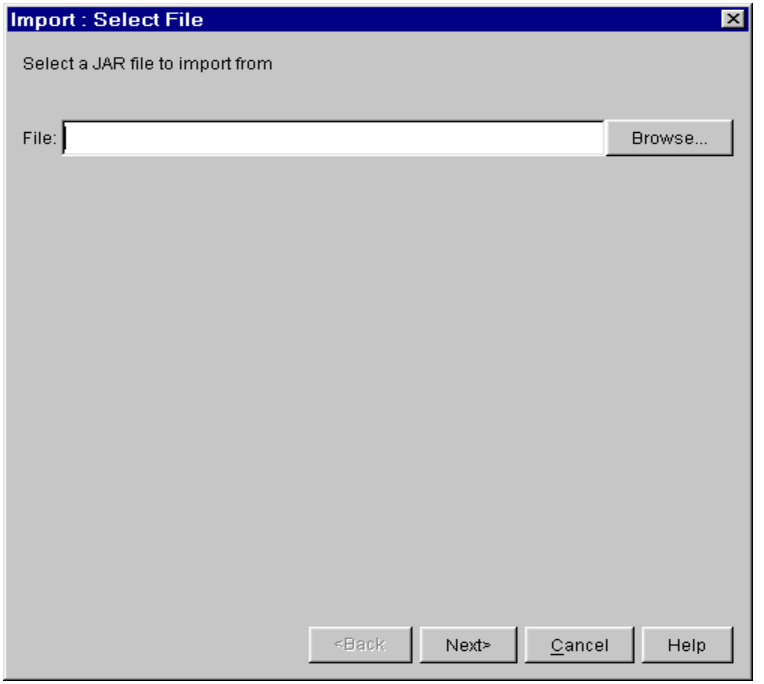
How to Run the EJB Sample

To run the EJB sample, complete the steps described in the following sections.

Step 1. Import the Workflow Definition

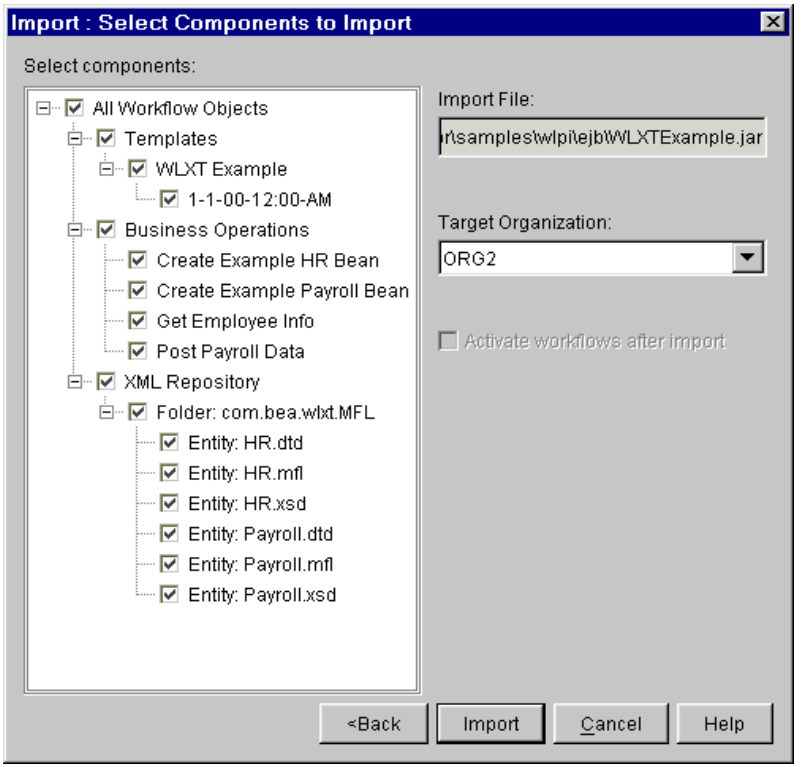
1. Start the WebLogic Integration Studio. For information on starting and logging on to the Studio, see Using the Studio Interface in *Using the WebLogic Integration Studio*.
2. Choose Tools→Import Package. The Import: Select File dialog box is displayed (Figure 3-5).

Figure 3-5 Import: Select File Dialog Box



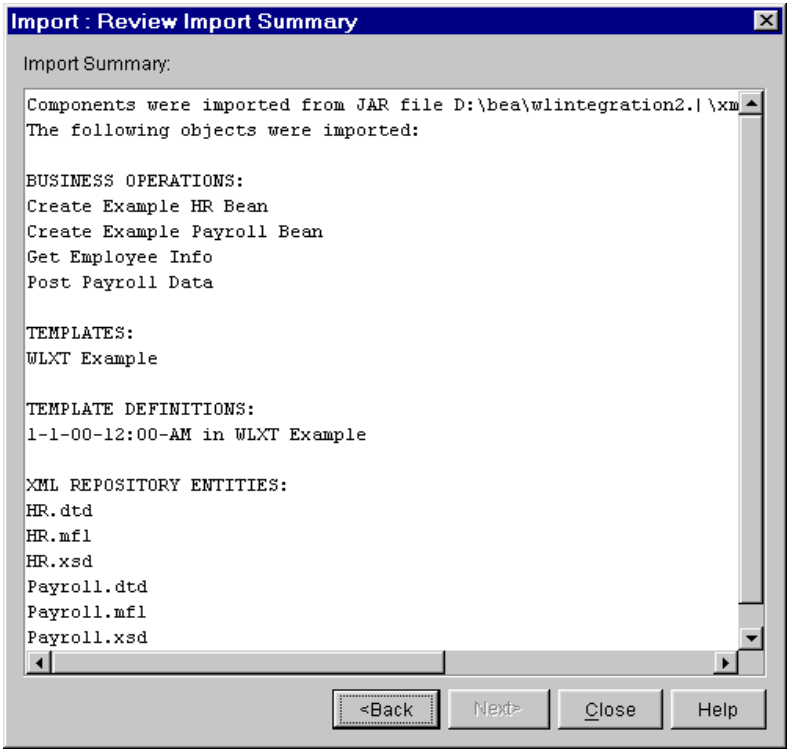
3. Click Browse, select the definition file called `WLXTExample.jar`, and click Open. Click Next, the Import: Select Components to Import dialog box is displayed (Figure 3-6).

Figure 3-6 Import: Select Components to Import



4. Make sure that the Activate workflows after import check box is selected, and that all components are selected. Click Import. The Import: Review Import Summary dialog box is displayed (Figure 3-7).

Figure 3-7 Import: Review Import Summary

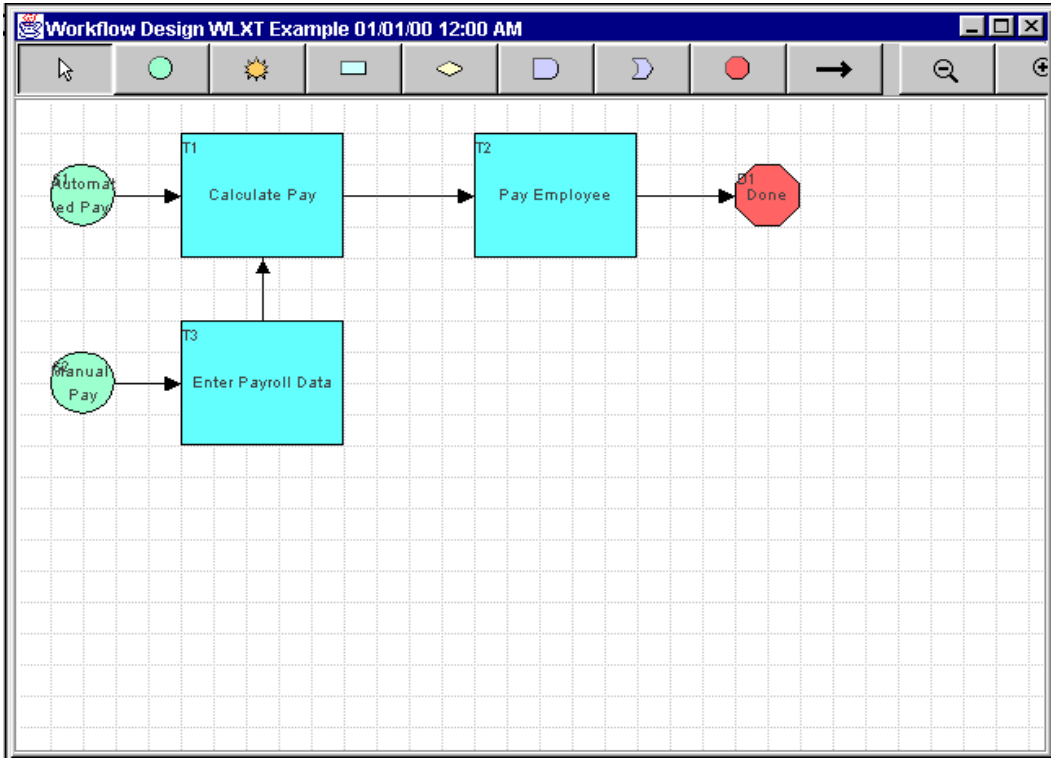


5. Confirm that the correct components are listed. If they are not, click Back and select the components again. If the list of components is correct, click Close. You are now ready to open the template.

Step 2. Open the Template

1. In the navigation tree, expand the `WLXT Example` template imported in the previous step. Right-click the template definition `1-1-00-12:00-AM`.
2. Select Open. The workflow created for this sample application is displayed (Figure 3-8).

Figure 3-8 Workflow for WebLogic Integration Example



Step 3. Start the Workflow

You can start the workflow created in the sample in either of two ways:

- From the WebLogic Integration Worklist
- From the Command Line

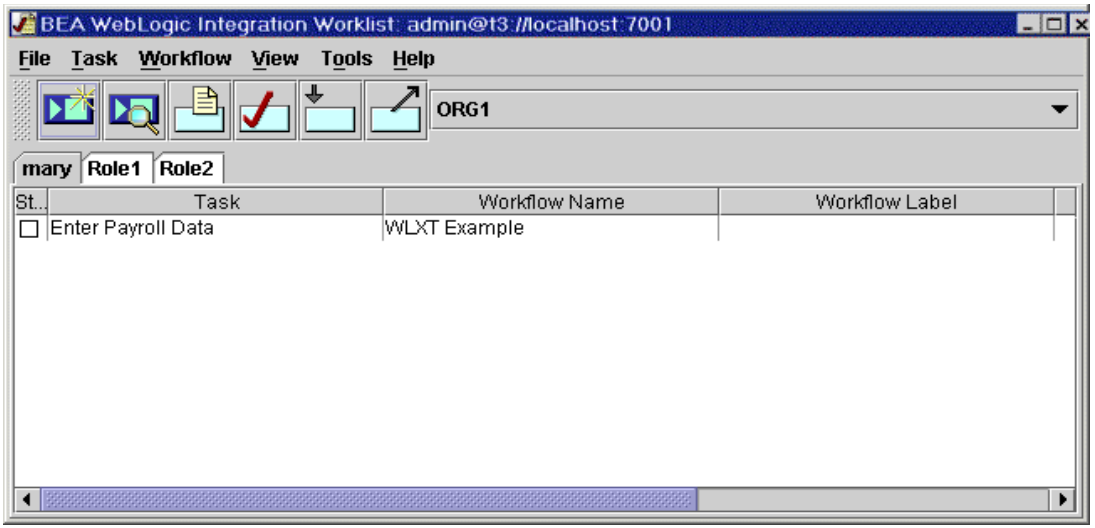
Once you start the workflow, you can use the Studio to monitor the simulated flow of data between the HR system and the payroll system.

From the WebLogic Integration Worklist

To start the sample workflow from the WebLogic Integration Worklist:

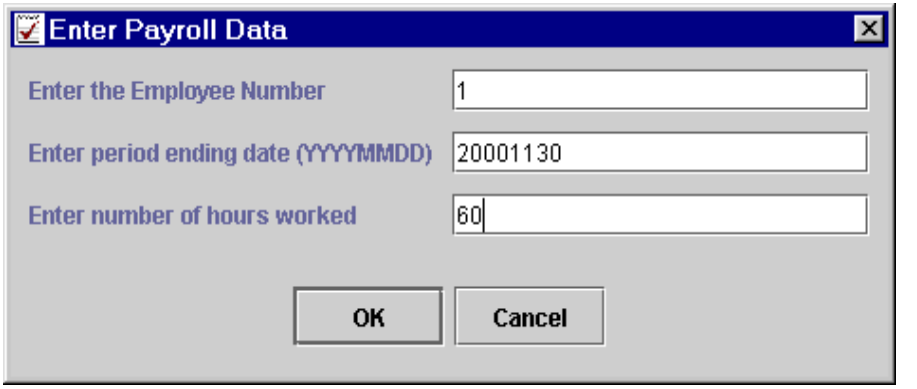
1. Start the WebLogic Integration Worklist and choose Workflow→Start a Workflow.
2. Select `WLXT Example`. Click OK.

Figure 3-9 WLXT Example Worklist



3. Right-click the Enter Payroll Data task and select Execute. The Enter Payroll Data dialog box is displayed (Figure 3-10).

Figure 3-10 Enter Payroll Data



4. Enter the payroll data and click OK. The task and the workflow are started.

   Note: For this example, only the numbers 1 through 4 are valid as employee numbers. You can enter any value, however, as the period ending date and the number of hours worked.

From the Command Line

To start the sample workflow from a command-line prompt:

1. In a text editor, open the script (`Autopay.cmd` on Windows NT systems or `Autopay.sh` on Unix systems) and check the location of the `WebLogic Integration process engine`. By default, the location is `localhost` and the number of the port is `7001`.
2. Change the location information to match the host and port for your system.
3. Set the environment variable `WL_HOME` to the pathname of the directory in which WebLogic Server is installed on your system. For example:

   ```
   set WL_HOME=c:\bea\wlserver6.1
   ```

4. Set the environment variable `WLI_HOME` to the pathname of the directory in which WebLogic Integration is installed on your system. For example:

   ```
   set WLI_HOME=c:\bea\wlintegration2.1
   ```

5. Run the command script for your system (Windows NT or UNIX), passing the same parameters shown in Figure 3-10. For example:

   ```
   Autopay 1 2000-11-30 60
   ```

   The workflow is started.

Step 4. Examine the Variable Values

To monitor the sample dataflow, complete the following procedure:

1. In the Studio, display the Workflow Instances dialog box. For information on monitoring workflows, see Monitoring Workflows in *Using the WebLogic Integration Studio*.
2. Right-click the desired workflow instance and, from the pop-up menu, select Variables. The Workflow Variables dialog box is displayed.
3. Examine each variable and verify that it is set to the values entered in “Step 3. Start the Workflow” on page 3-16.

Index

S

U

W