BEA WebLogic Integration™

Using the Data Integration Plug-In

Version 2.1
Document Date: October 2001

Copyright

Restricted Rights Legend

Trademarks or Service Marks

Using the Data Integration Plug-In

Part Number	Date	Software Version
N/A	October 2001	2.1

Contents

CHAPTER

1 Understanding the Data Integration Plug-In

This guide 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

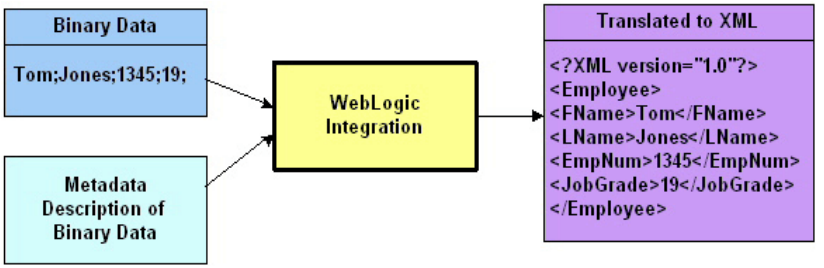
Understanding XML Translation

Data that is sent to or received from legacy applications is often platform-specific binary data that is in the native machine representation. Binary data is not self-describing, so in order to be understood by an application, the layout of this data (metadata) must be embedded within each application that uses the binary data.

XML is becoming the standard for exchanging information between applications because XML embeds a description of the data within the data stream, thus allowing applications to share data more easily. XML is easily parsed and can represent complex data structures. As a result, the coupling of applications no longer requires metadata to be embedded within each application.

When you translate binary to XML data, you convert 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. The translation process converts each field of binary data to XML according to the metadata defined for each field of data. In the metadata you specify the name of the field, the data type, the size, and whether the field is always present or optional. It is this description of the binary data that is used to translate the binary data to XML. Figure 1-1 shows a sample of XML data translation.

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



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

What is Data Integration?

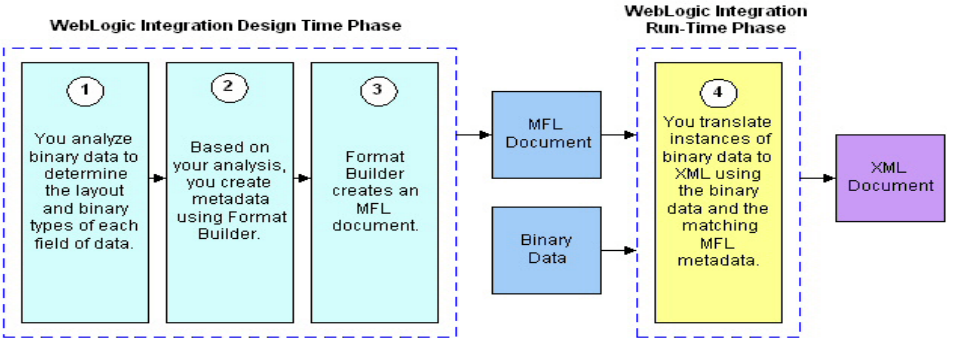
The data integration component of WebLogic Integration facilitates the integration of data from diverse enterprise applications by supporting data translations between binary formats from legacy systems and XML. Data integration normalizes legacy data into XML so it may be directly consumed by XML applications, transformed into a specific XML grammar, or used directly to start workflows in business process management (BPM). Data integration supports non-XML to XML translation and vice versa and is made up of three primary components:

- The Design-Time Component
- The Run-Time Component
- The Plug-In to Business Process Management

To perform a translation, you create a description of your binary data using design-time (Format Builder) in WebLogic Integration. This involves analyzing binary data so that its record layout is accurately reflected in the metadata you create in Format Builder. You then create a description of the input data in Format Builder and save this metadata as a Message Format Language (MFL) document. WebLogic Integration includes importers 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 allows for easy access to configuring translations.

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



The Design-Time Component

The data integration design-time component of WebLogic Integration is a Java application called Format Builder. Format Builder is used to create descriptions of binary data records. Format Builder 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 are metadata used by run-time in data integration and the plug-in to BPM to translate an instance of a binary data record to an instance of an XML document (or vice-versa). Format Builder will also create a DTD or XML Schema document that describes the XML document created from a translation.

The Run-Time Component

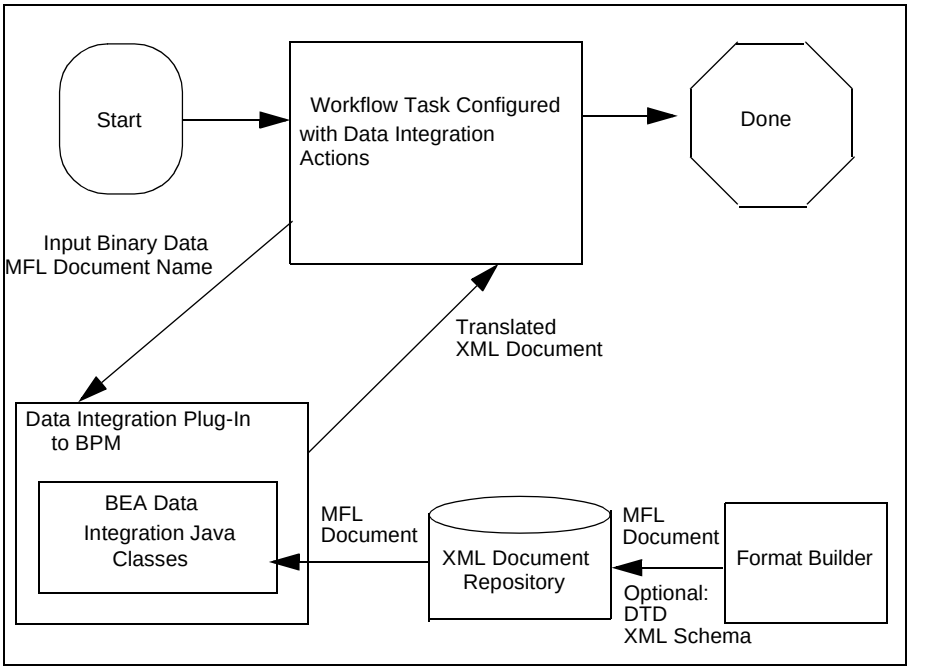
The data integration run-time component of WebLogic Integration is a Java class with various methods used to translate data between binary and XML formats. This Java class can be deployed in an EJB using WebLogic Server, invoked as a business operation from a workflow in BPM, or integrated into any Java application.

The Plug-In to Business Process Management

The Data Integration Plug-In for business process management (BPM) provides for an exchange of information between applications by supporting data translations between binary formats from legacy systems and XML. The Data Integration Plug-In provides BPM actions that allow you to access 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, and execution within the process engine clustered environment.

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



Using the Repository

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

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

The repository provides access to these document types and allows you to share them between data integration, business process management, and B2B integration. 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.

2 Using the Data Integration Plug-In

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, and is invaluable in integrating disparate applications. However, most data transformation engines 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.

In addition to this data translation capability, the data integration plug-in provides a binary data event handler, in-memory caching of MFL documents and translation object pooling to boost performance, a `BinaryData` variable type to edit and display binary data, and execution within the process engine clustered environment.

This section provides information about the following topics:

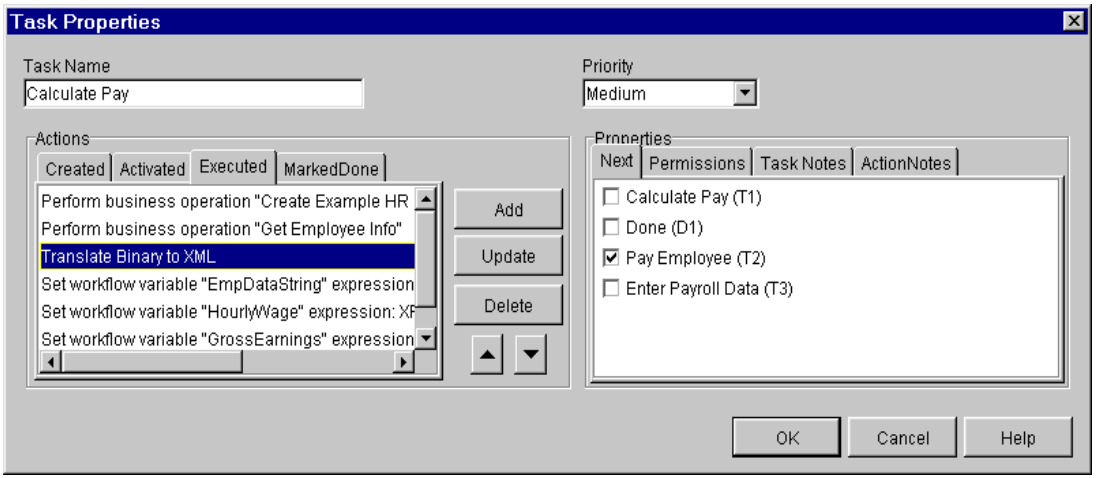
- Data Translation with the Data Integration Plug-In
- Processing Event Data
- Enhancing Data Translation Performance
- Custom Data Types and the Data Integration Plug-In
- WebLogic Server Clustering Support

Data Translation with the Data Integration Plug-In

The data integration plug-in provides XML and non-XML translation capabilities from within business process management (BPM). To perform one of these translation actions, follow the steps below. For more information on the actions specific to BPM, refer to the BPM documentation.

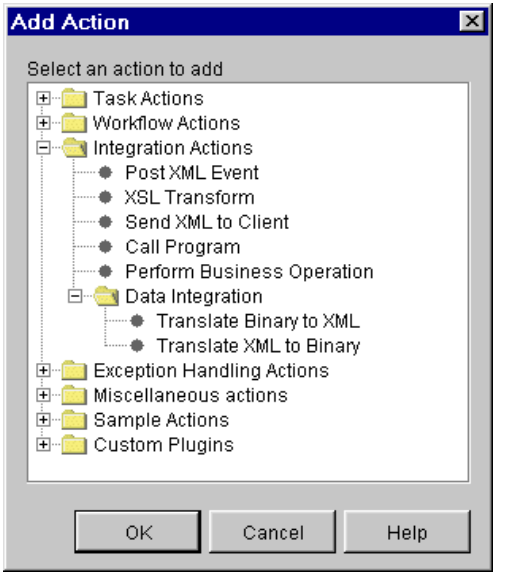
1. Start BPM.
2. Open the desired template definition and double-click a task. The Task Properties dialog box opens (Figure 2-1).

Figure 2-1 Task Properties Dialog Box



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

Figure 2-2 Add Action dialog



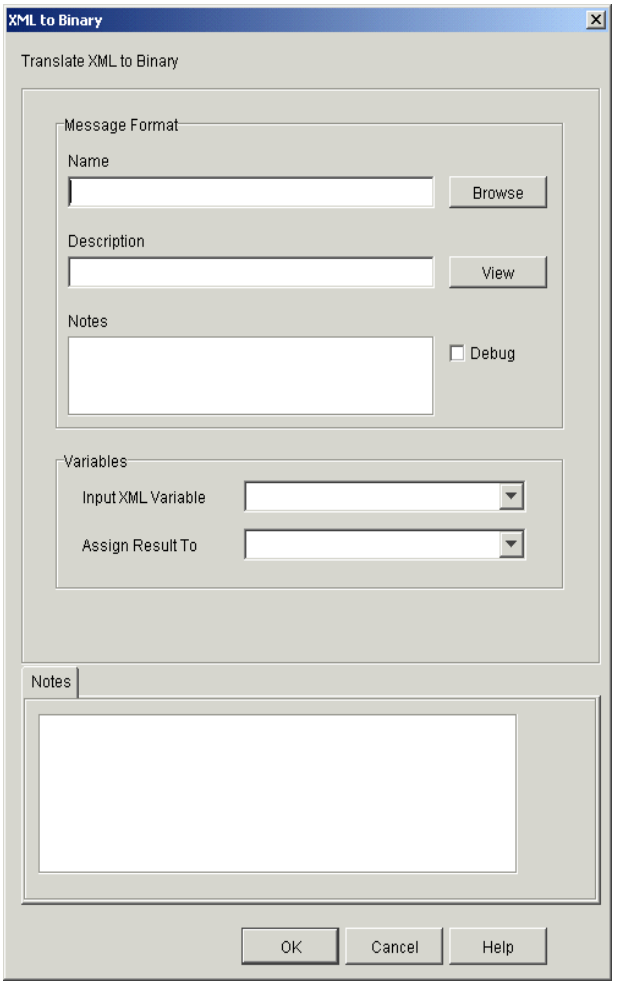
4. Select Integration Actions to expand its action list, then select Data Integration and choose the action you want to perform (Translate XML to Binary or Translate Binary to XML).

Translate XML to Binary

To perform an XML to binary translation:

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

Figure 2-3 Translate XML to Binary Dialog Box



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

Field	Description
Message Format Parameters	
Name	The name of the message format. You can type a name directly in the text box or click Browse to select the document from the repository.

Field	Description
Description	Displays the description of the message format. **Note:** This field is display only. You cannot edit this field.
Notes	Displays the notes attached to the message format. **Note:** This field is display only. You cannot edit this field.
Debug	Enable or disable debug messaging. When you select this option, the translation actions are written to the WebLogic Server log file.
Message Format Action Buttons	
Browse	Allows you to browse MFL documents in the repository.
View	Displays the items contained in the message format so you can verify that you have selected the correct document type for translation.
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 the name you want to assign to the new variable and click OK. A confirmation message box displays. 2. Click Yes to create the new variable.
Assign Result To	Displays the Binary Data workflow variables. Select the variable you want to use to store the translated information or create a new variable as follows: 1. Type the name you want to assign to the new variable and click OK. 2. Click Yes to create the new variable.

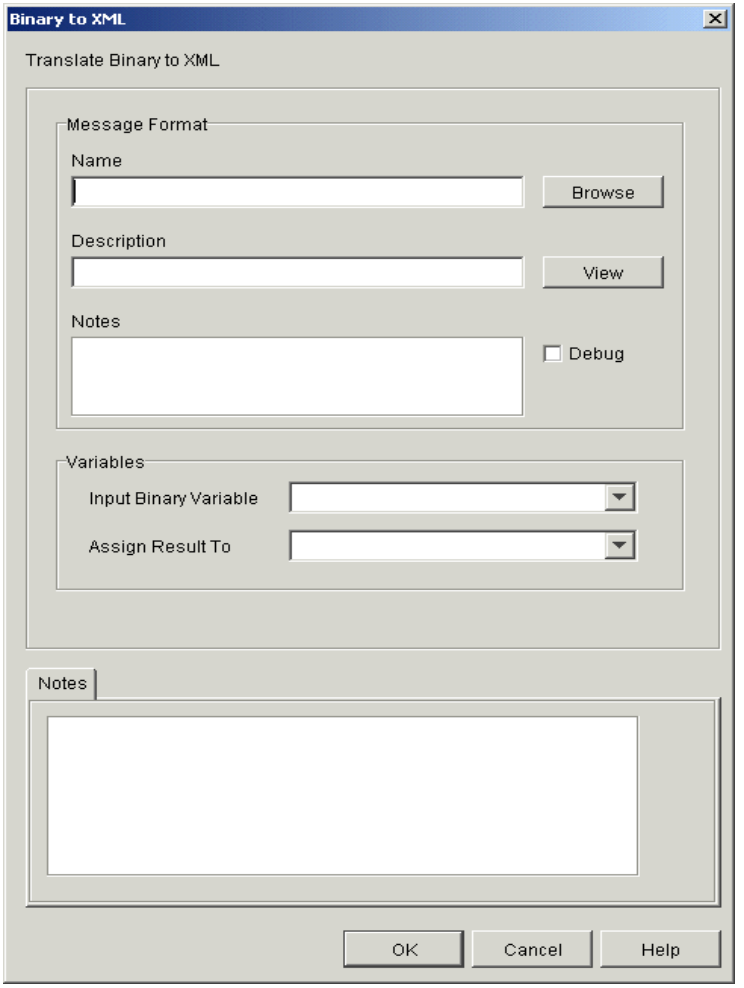
3. Click OK to save the translation information to the workflow.

Translate Binary to XML

To perform a binary to XML translation:

1. From the Add Action dialog (Figure 2-2), choose Integration Actions→Data Integration→Translate Binary to XML. The Translate Binary to XML dialog opens (Figure 2-4).

Figure 2-4 Translate Binary to XML Dialog Box



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

Field	Description
Message Format Parameters	
Name	The name of the message format. You can type a name directly in the text box, or click Browse to select the message format from the repository.
Description	Displays the description of the message format. **Note:** This field is display only. You cannot edit this field.
Notes	Displays the notes attached to the message format. **Note:** This field is display only. You cannot edit this field.
Debug	Enable or disable debug messaging. When you select this option, the translation actions are written to the WebLogic Server log file.
Message Format Action Buttons	
Browse	Allows you to browse MFL documents in the repository.
View	Displays the items contained in the message format so you can verify that you have selected the correct document type for translation.
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 the name you want to assign to the 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 the name you want to assign to the new variable and click OK. 2. Click Yes to create the new variable.

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

Processing Event Data

The data integration plug-in provides functionality that allows binary data to trigger workflows by converting the binary data to XML or pre-processing it at the front end of event processing. This functionality is referred to as the "event handler." Publishing JMS messages to a topic causes the event handler to run.

There are three JMS properties required for the message to be pre-processed 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 the code used to build a message that is to 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);
```

This process is illustrated in the servlet sample application.

Enhancing Data Translation Performance

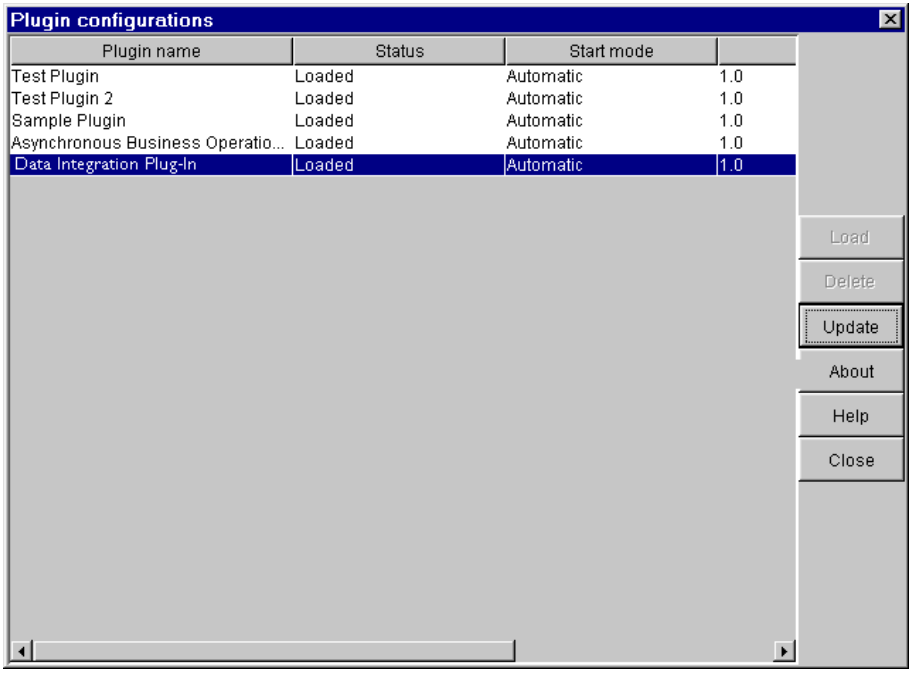
The data integration plug-in provides a configuration panel to administer and monitor the MFL document in-memory cache and enable or disable event handler debugging. Using this panel, you can adjust the in-memory cache and translation object pool to enhance the performance of your data translations.

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 on the actions specific to business process management, refer to the business process management documentation.

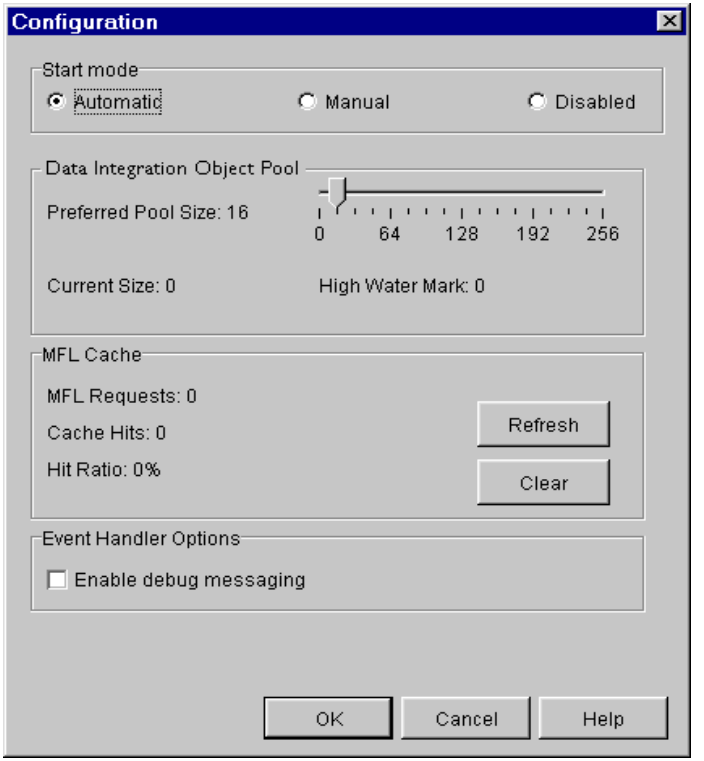
1. Start BPM.
2. Choose Configuration→Plugins. The Plugin Configuration dialog opens (Figure 2-5).

Figure 2-5 Plugin Configuration Dialog Box



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

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



4. Use the fields as described in the table below to monitor and enhance translation performance.

Field	Description
Start mode	
Automatic	Select this option to have the data integration plug-in open automatically when BPM opens.
Manual	Select this option to have the data integration plug-in available from BPM.
Disabled	Select this option to disable the use of the data integration plug-in from BPM.

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 where the MFL document needed 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. This requires all future translation requests to load MFL documents 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 provides additional display and edit capabilities over the 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 group of binary data with additional display capabilities. The `BinaryData` 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

Data integration includes a user defined type feature that allows you to create custom data types specific to 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 in both features and function.

Configuring User Defined Data Types

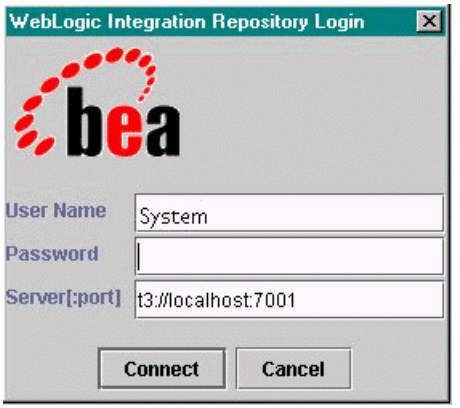
User Defined Types used by the data integration plug-in are stored in the WebLogic Integration repository as `CLASS` documents. At runtime, the data integration plug-in loads user defined type classes from the repository as required. In addition, the data integration plug-in will export the MFL and class files required to support the active template allowing a template to be imported on another business process management instance intact. Class documents may be placed in the repository using one of the following methods:

- Using Format Builder
- Using the Repository Import Utility

Using Format Builder

Perform the following steps to publish a user defined type to the repository using Format Builder.

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



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

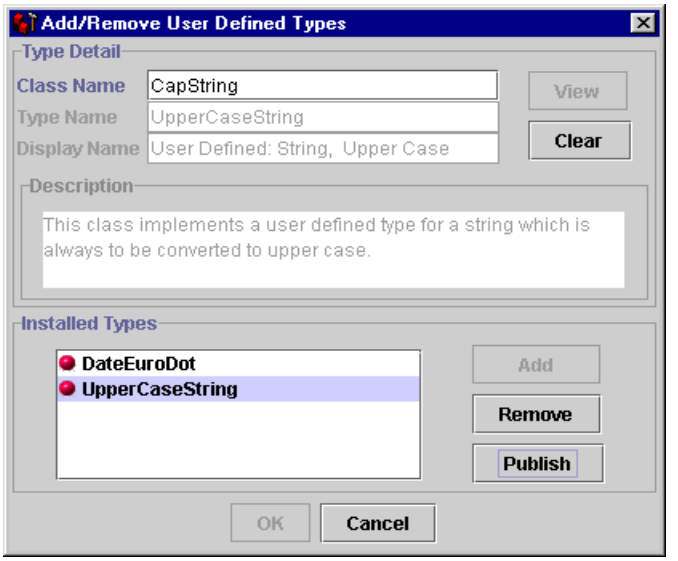
   Note: The WebLogic Integration Repository Login window allows up to three unsuccessful login attempts, after which, a login failure message is

displayed. If you experience three login failures, choose Repository→Log In to repeat the login procedure.

6. Click Connect. If your login is successful, the Login window disappears and the Format Builder Title bar displays the server name and port number entered on the WebLogic Integration Repository Login window. You may now choose any of the active repository menu items to access.

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

   With a repository connection established, the Add/Remove User Defined Types dialog box displays the status of each registered user defined type and allows for its publication to the repository. The user defined type repository status is reflected by an icon of a ball preceding the type name of each installed user defined type.



The color of the icon associated with each user defined type indicates its status:

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

8. Select the class you want to publish from the list of Installed Types and click Publish. The icon for the selected entry should become green indicating the class was successfully placed in the repository.

Using the Repository Import Utility

Perform the following steps to use the repository import utility to import Java class files, including User Defined Types.

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 file name on the Import command line.

   `java com.bea.wlxt.repository.Import <file name>`

   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 to the repository using the Repository Import utility, not just User Defined Types. This is useful if a user defined type relies on additional class files that do not extend the `com.bea.wlxt.bintype.Bintype` class.

WebLogic Server Clustering Support

The data integration plug-in can operate successfully in a WebLogic Server clustered 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 the use of a JMS topic. The topic is used for communication on different nodes in a cluster in WebLogic Integration.

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 within the cluster. The JNDI name of this topic must be as follows:

   `com.bea.wlxt.cluster.BroadcastTopic`

   Note: Refer to the WebLogic Server documentation for more information on creating JMS topics.

2. Open the `config.xml` file in a text editor. This file can be found in the `config\samples\` directory where you have WebLogic Integration installed.

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

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

- Prerequisite Considerations
- Running the BPM Servlet Sample
- Running the BPM EJB Sample

Prerequisite Considerations

The instructions presented in this section assume that you have a good working knowledge of WebLogic Integration, specifically data integration and the BPM engine. You should have successfully installed WebLogic Integration and run a sample workflow prior to running the sample applications.

Running the BPM Servlet Sample

This sample application implements a Web Archive (`WLPI_sample.war`) that installs a servlet to accept requests for conversion of binary data to XML. The servlet is accessed via a browser and responds by displaying the generated XML data. In addition, the data is posted 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 following table provides a listing and description of the files included in the BPM Servlet sample application. This sample application can be found where WebLogic Integration is installed in the `\samples\di\` subdirectory.

Table 3-1 List of Servlet Sample Application Files

Directory	File	Description
`\wlpi\source`	`WLPI_sample.java`	The source code for the servlet used to present the HTML screen and process binary data to XML. This XML may be placed, optionally, onto the JMS topic.
`\wlpi`	`SampleData.mfl`	The Message Format Language description of the sample binary data file used to start the sample workflow.
`\wlpi`	`SampleData.data`	The sample data file used as input to start the sample workflow.
`\wlpi`	`SampleWorkflow.jar`	The exported workflow used in the sample. This workflow should be imported via the WebLogic Integration Studio GUI to setup the workflow tasks involved in the sample.
`\wlpi`	`Makefile`	Make file for building the sample source to a .jar file.
`\wlpi`	`build.cmd`	Builds the .jar file from source.
`\wlpi\images`	`bealogo.jpg`	The BEA logo image displayed on the HTML page rendered by the sample servlet.

Table 3-1 List of Servlet Sample Application Files

Directory	File	Description
`\wlpi\WEB-INF`	`hello.html`	The HTML page used by the sample servlet to obtain input data from the user.
`\wlpi\WEB-INF`	`web.xml`	The J2EE configuration file defining deployment information for the sample servlet.
`\wlpi\WEB-INF`	`weblogic.xml`	The BEA configuration file defining WebLogic-specific information for the sample servlet.
`wlpi\WEB-INF\lib`	`cos.jar`	Utility libraries that are used in the execution of the sample code.
`wlpi\WEB-INF\lib`	`HtmlScreen.jar`	Utility libraries that are used in the execution of the sample code.
Under (WebLogic Integration Home)		
`config\samples\applications`	`WLPI_sample.war`	A 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

Follow the steps below to run the servlet sample. For instructions on the tasks specific to WebLogic Server and BPM, refer to the documentation that accompanies those applications.

Step 1. Start the WLI Sample Application Launcher

For first-time users:

1. Choose Start→BEA WebLogic E-Business Platform→WebLogic Integration 2.1→Samples→Run Samples. This will configure all samples and will open the samples launcher. It takes several minutes to configure all samples.
2. Select the Data Integration Servlet Sample.

If you have already configured samples before:

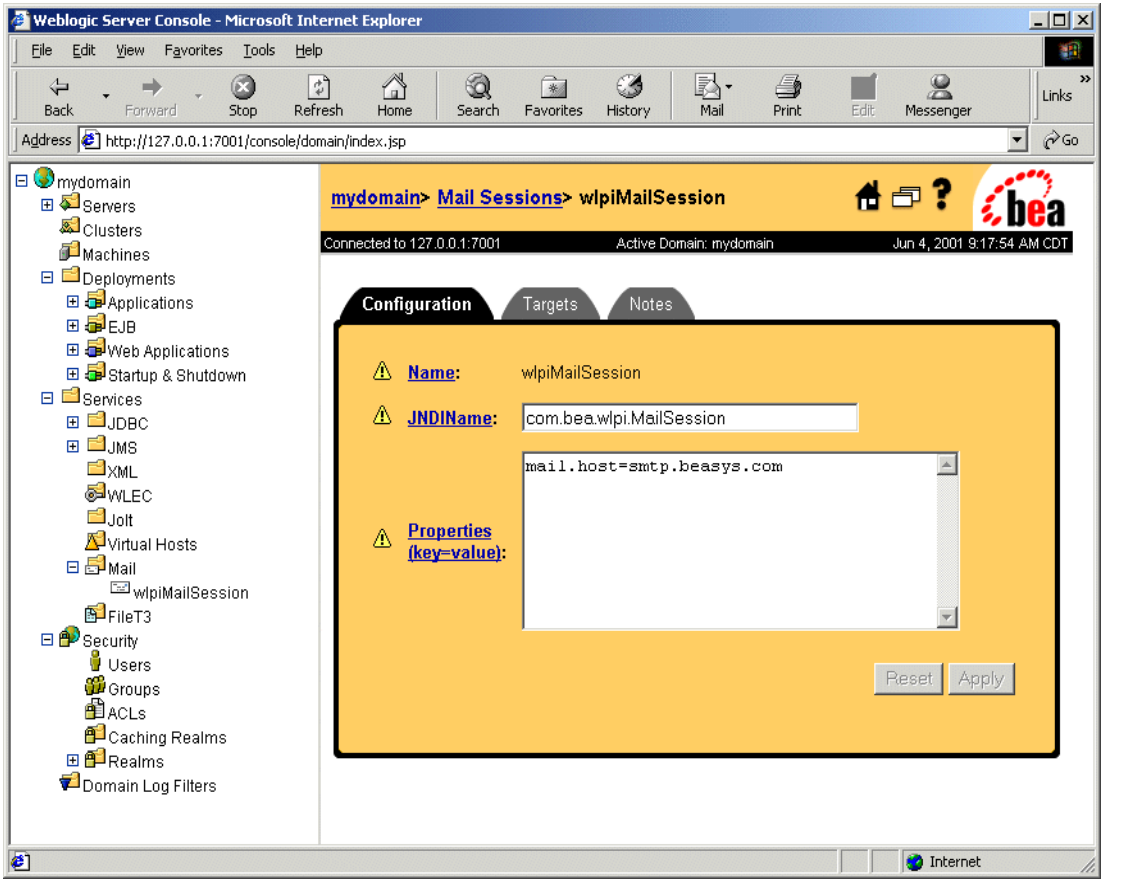
1. Choose Start→BEA WebLogic E-Business Platform→WebLogic Integration 2.1→Samples→Start Server.
2. Choose Start→BEA WebLogic E-Business Platform→WebLogic Integration 2.1→Samples→Samples Launcher.
3. Select the Data Integration Servlet Sample.

Step 2. Configure the Mail Session

This step is optional if you have already configured your mail host. You may also use this step to verify your configuration.

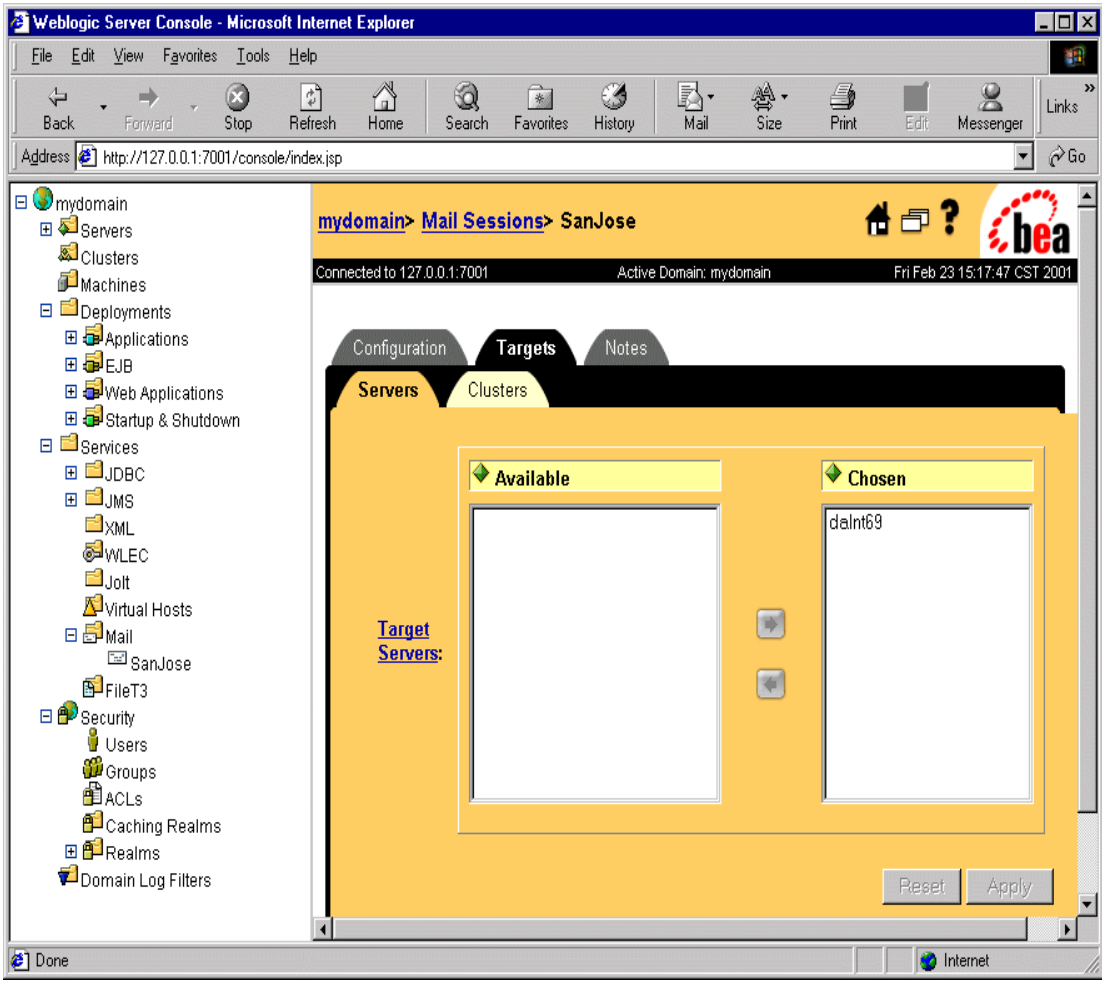
1. From the navigation tree, choose Services→Mail→wlpiMailSession.
2. Enter the appropriate information to configure your mail host. Make sure that `mail.host=mailserver`. The following figure shows an example of the Mail Session Configuration screen.

Figure 3-1 WebLogic Server Console Mail Session Configuration Tab



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

Figure 3-2 WebLogic Server Console Mail Session



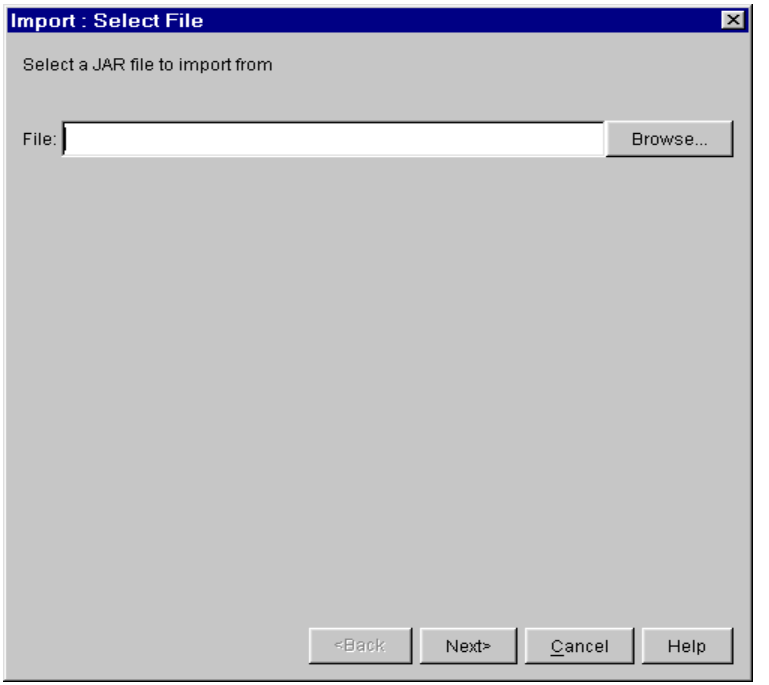
Step 3. Create a New Template and Activate the Workflow

To import the workflow definition:

1. Run WebLogic Integration Studio by choosing Start→Programs→BEA WebLogic E-Business Platform→WebLogic Integration 2.1→Studio.

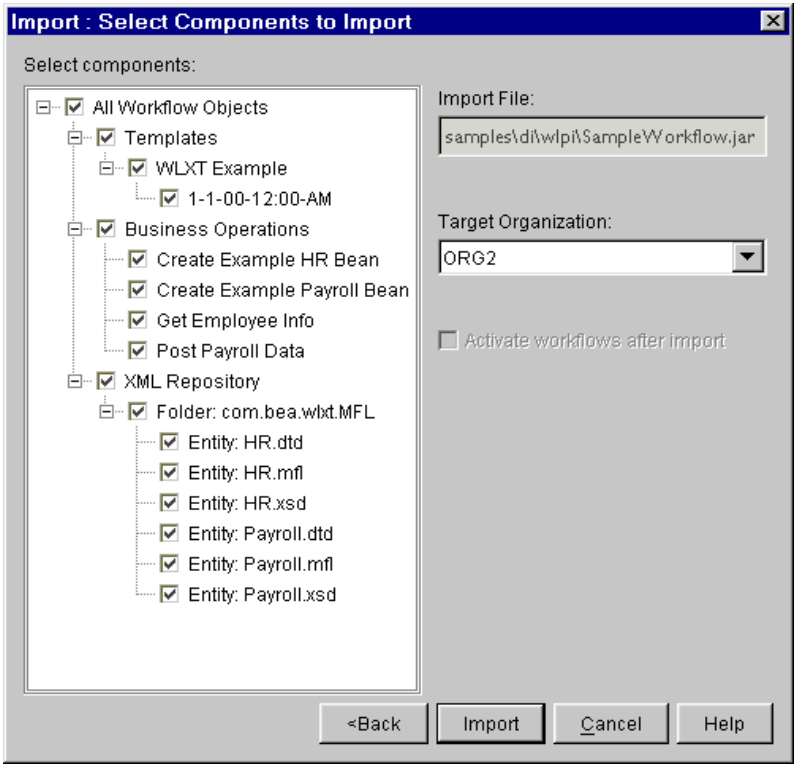
2. Choose Tools→Import Package. The Import: Select File dialog box opens (the following figure).

Figure 3-3 Import: Select File Dialog Box



3. Click Browse, select the definition file `SampleWorkflow.jar`, and click Open. Click Next, the Import: Select Components to Import dialog box opens (the following figure).

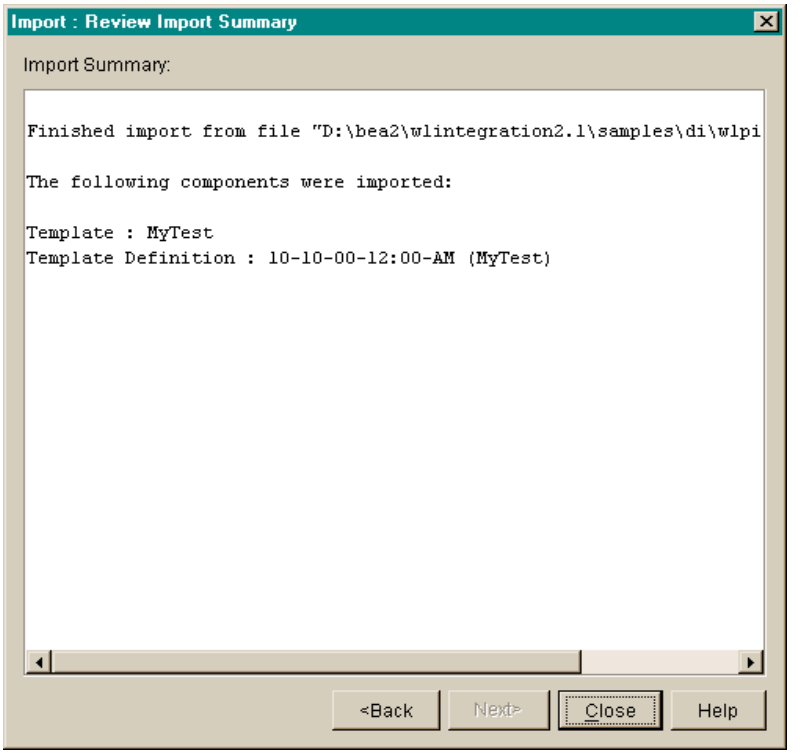
Figure 3-4 Import: Select Components to Import



4. Make sure that the Activate workflows after import check box is selected, make sure all components are selected, and click Import. The Import: Review Import Summary dialog box opens (the following figure).

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

Figure 3-5 Import: Review Import Summary



Step 4. Store the SampleData.mfl File in the Repository

1. Start Format Builder by choosing Start→Programs→BEA WebLogic E-Business Platform→WebLogic Integration 2.1→Format Builder.
2. Open the `\samples\di\wlpi\SampleData.mfl` file.
3. Choose Repository→Log In.
4. Choose Repository→Store As to store the file as SampleData in the repository.

Step 5. Update the Sample XML Data and Send Message

1. Using a text editor, open the file `\samples\di\wlpi\SampleData.data`. Replace the text *`user@bea.com`* with a valid email address. This is the address the workflow uses to deliver the email message.
2. From the servlet sample enter `SampleData` into the MFL text field.
3. Navigate to the following data file:

   `\samples\di\wlpi\SampleData.data`
4. Select the option to invoke the process engine and click Submit. A short email message is sent to the address you supplied in the data file.

Running the BPM EJB Sample

This sample simulates a dataflow from an HR system to a payroll system, initiated by the entry of payroll data. The employee data is obtained from a legacy payroll system that uses binary data. The data is translated to XML in order to perform a calculation to determine the employee's pay information. The result of the calculation is translated back to binary and sent on to the payroll system.

What is Included in the EJB Sample

The following table provides a listing and description of the files included in the EJB sample application. This sample application can be found in the `\samples\di\ejb` directory.

Table 3-2 List of EJB Sample Application Files

Directory	File	Description
`\ejb`	`Makefile`	Make file for building the sample source to a .jar file.
`\ejb`	`WLXTExample.jar`	Exported sample workflow from data integration.

Table 3-2 List of EJB Sample Application Files

Directory	File	Description
\ejb	HR.mfl	MFL file for binary data returned from the Sample HR Bean.
\ejb	Payroll.mfl	MFL file for binary data passed to the Sample Payroll Bean.
\ejb	Autopay.cmd	Windows NT command script to initiate the workflow from the command line.
\ejb	Autopay.sh	UNIX shell script to initiate the workflow from a command prompt.
\ejb	build.cmd	Builds wlxtejb.jar from source.
\ejb\source	Payroll.java	Sample EJB to represent legacy payroll system.
\ejb\source	PayrollHome.java	Sample EJB to represent legacy payroll system.
\ejb\source	PayrollBean.java	Sample EJB to represent legacy payroll system.
\ejb\source	HR.java	Sample EJB to represent legacy HR system.
\ejb\source	HRHome.java	Sample EJB to represent legacy HR system.
\ejb\source	HRBean.java	Sample EJB to represent legacy HR system.
\ejb\source	AutoPay.java	Program to place a pre-formatted message on the Event Topic to start the sample workflow.
\ejb\source	HexDump.java	Utility class used by the sample EJBs.
\ejb\source	EmployeeRecord.java	Employee data class used by the sample HR EJB.
Path = (WebLogic Integration Home)		
config\samples\applications	WLXTEJB.jar	Executables for the sample application.

How to Run the EJB Sample

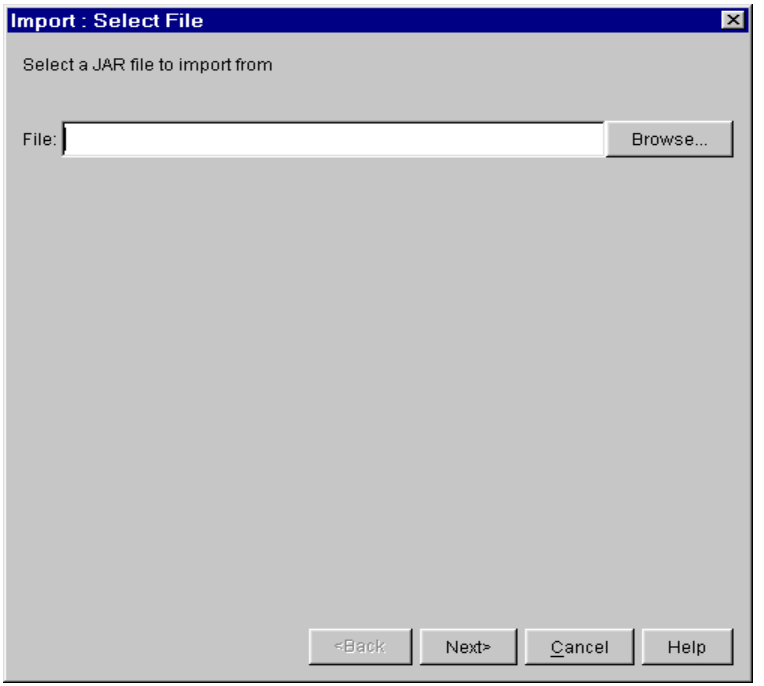
Follow the steps below to run the EJB sample.

Step 1. Import the Workflow Definition

To import the workflow definition:

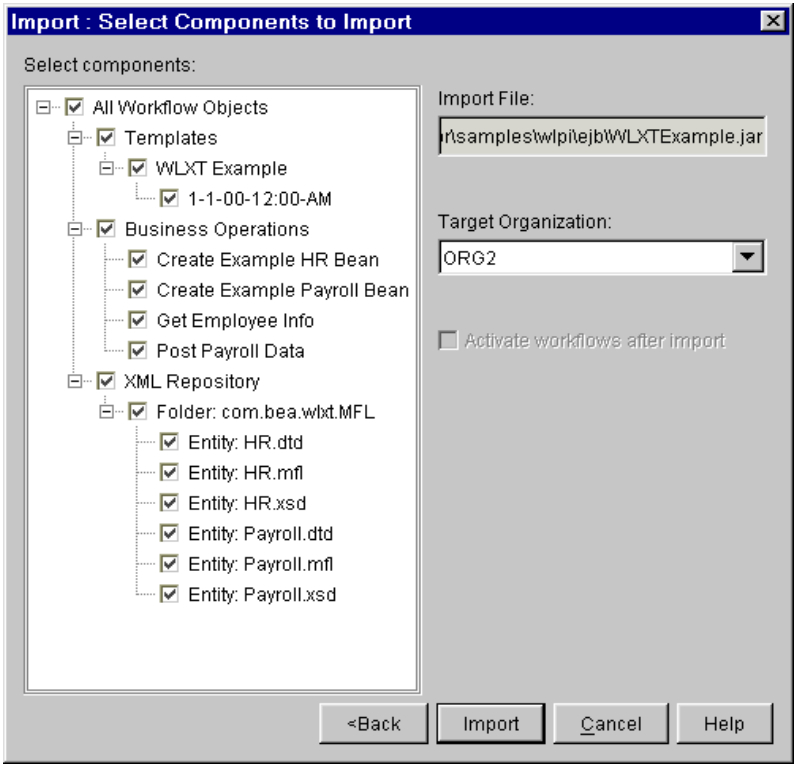
1. Run WebLogic Integration Studio.
2. Choose Tools→Import Package. The Import: Select File dialog box opens (Figure 3-6).

Figure 3-6 Import: Select File Dialog Box



3. Click Browse, select the definition file `WLXTExample.jar`, and click Open. Click Next, the Import: Select Components to Import dialog box opens (Figure 3-7).

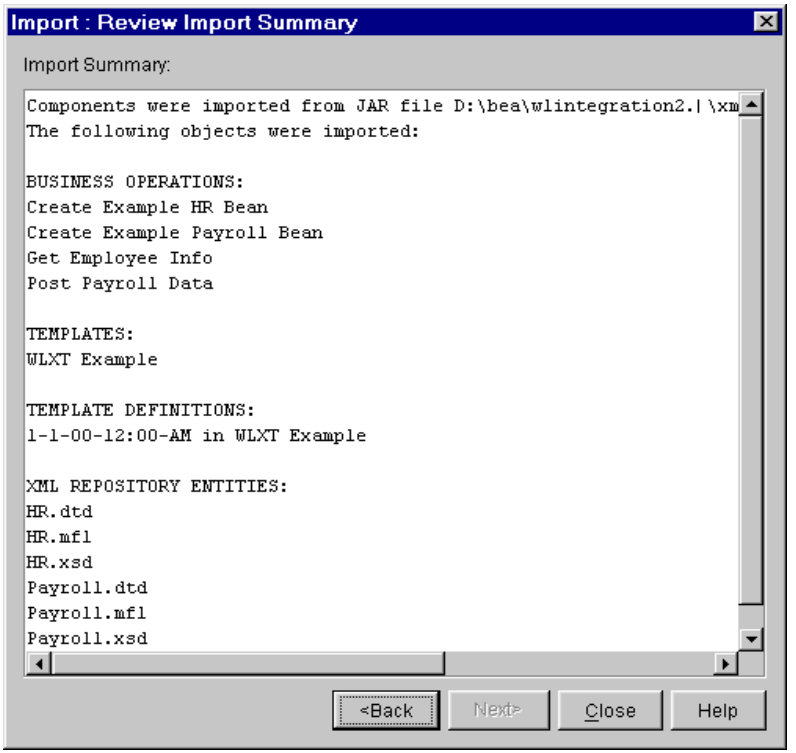
Figure 3-7 Import: Select Components to Import



4. Make sure that the Activate workflows after import check box is selected, make sure all components are selected, and click Import. The Import: Review Import Summary dialog box opens (Figure 3-8).

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

Figure 3-8 Import: Review Import Summary

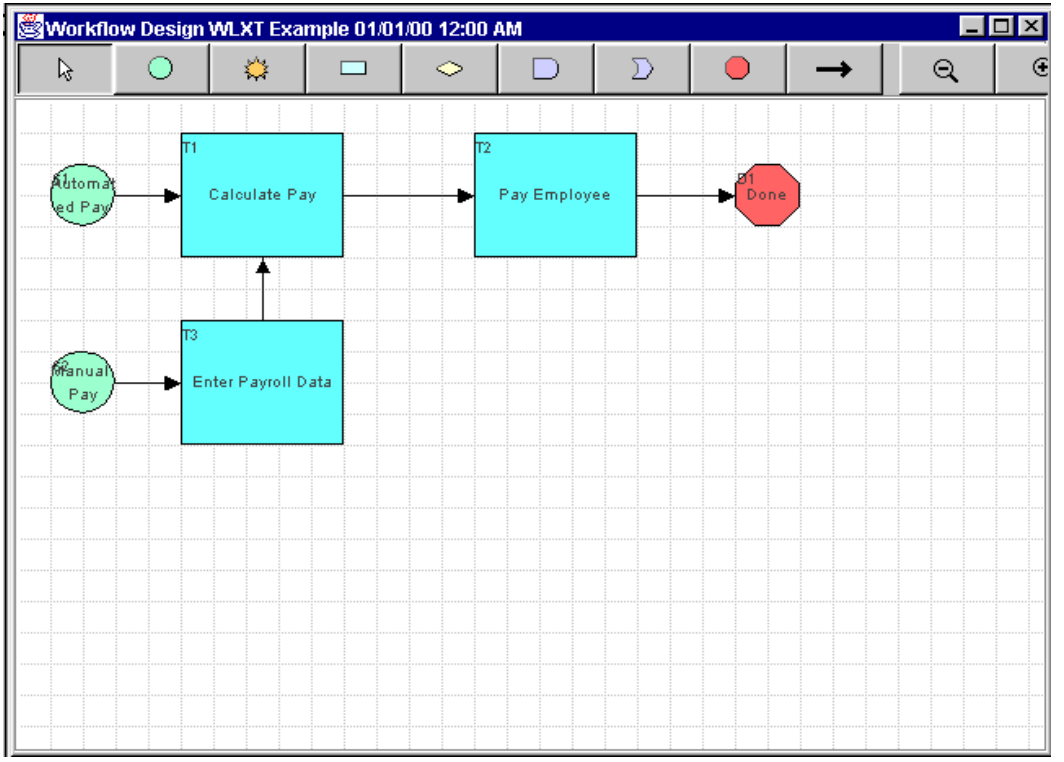


Step 2. Open the Template

To open the template:

1. Expand the `WLXT Example` template imported in the previous step in the navigation tree. Right-click the template definition `1-1-00-12:00-AM`.
2. Choose Open. The workflow created for this sample application displays.

Figure 3-9 Workflow for WebLogic Integration Example



Step 3. Start the Workflow

There are two ways to start the workflow created in the sample:

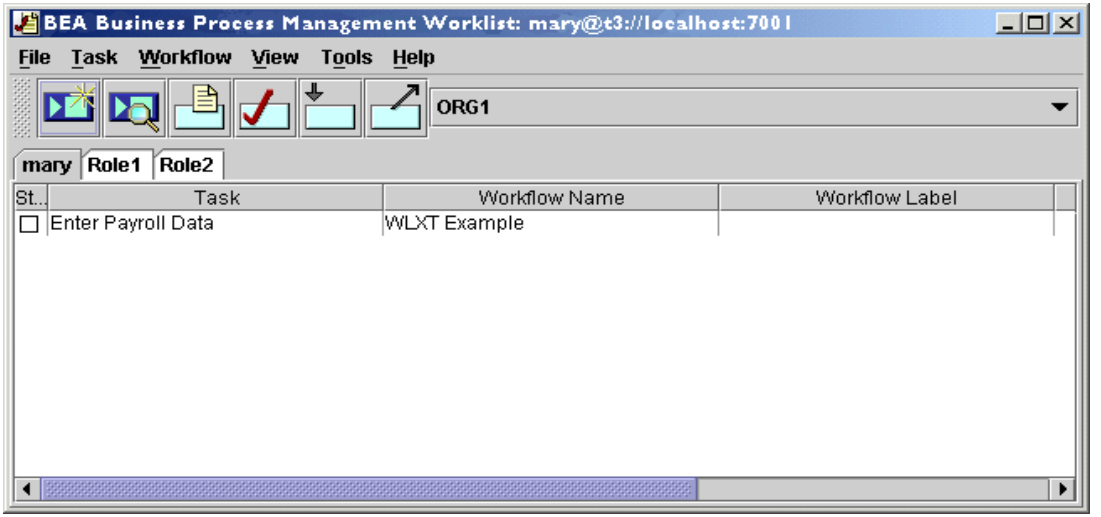
- From the WebLogic Integration Worklist
- From the Command Line

From the WebLogic Integration Worklist

To start the sample workflow from the WebLogic Integration Worklist:

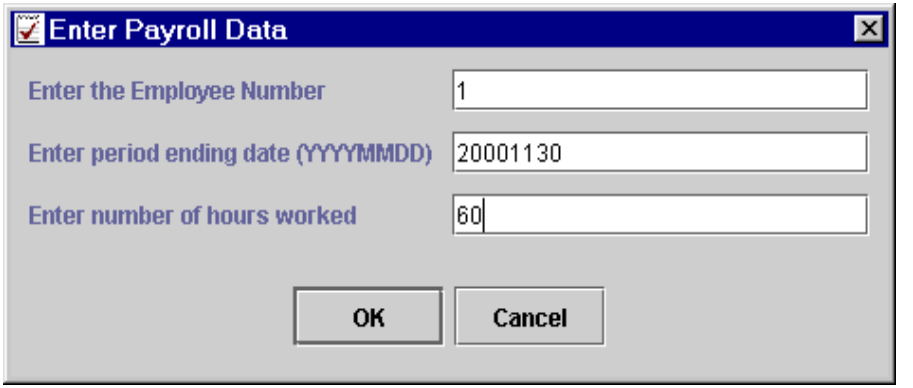
1. Start WebLogic Integration Worklist and choose Workflow→Start a Workflow.
2. Select `WLXT Example`. Click OK.

Figure 3-10 WLXT Example Worklist



3. Right-click the Enter Payroll Data task.
4. Choose Execute. The Enter Payroll Data dialog box displays.

Figure 3-11 Enter Payroll Data



5. Enter the payroll data and click OK. The task is started and the workflow runs.

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

From the Command Line

To start the sample workflow from a command line prompt:

1. Open the script file (`Autopay.cmd` on Windows NT systems; `Autopay.sh` on Unix systems) in a text editor and check the location of the `BPM` Server. By default, the location is `localhost` and `port:7001`.
2. Change the location information to match the host and port for your system.
3. Set the environment variable `WL_HOME` to the home directory for WebLogic Server on your system. For example:

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

4. Run the command scripts for your system (Windows NT or UNIX), passing the same parameters shown in Figure 3-11. For example:

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

Index

U

W