



BEA WebLogic XML/Non-XML Translator Samples Guide

BEA WebLogic XML/Non-XML Translator Samples Guide 1.0
Document Edition 1.0
January 2001

Copyright

Copyright © 2001 BEA Systems, Inc. All Rights Reserved.

Restricted Rights Legend

This software and documentation is subject to and made available only pursuant to the terms of the BEA Systems License Agreement and may be used or copied only in accordance with the terms of that agreement. It is against the law to copy the software except as specifically allowed in the agreement. This document may not, in whole or in part, be copied, photocopied, reproduced, translated, or reduced to any electronic medium or machine readable form without prior consent, in writing, from BEA Systems, Inc.

Use, duplication or disclosure by the U.S. Government is subject to restrictions set forth in the BEA Systems License Agreement and in subparagraph (c)(1) of the Commercial Computer Software-Restricted Rights Clause at FAR 52.227-19; subparagraph (c)(1)(ii) of the Rights in Technical Data and Computer Software clause at DFARS 252.227-7013, subparagraph (d) of the Commercial Computer Software--Licensing clause at NASA FAR supplement 16-52.227-86; or their equivalent.

Information in this document is subject to change without notice and does not represent a commitment on the part of BEA Systems. THE SOFTWARE AND DOCUMENTATION ARE PROVIDED "AS IS" WITHOUT WARRANTY OF ANY KIND INCLUDING WITHOUT LIMITATION, ANY WARRANTY OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. FURTHER, BEA Systems DOES NOT WARRANT, GUARANTEE, OR MAKE ANY REPRESENTATIONS REGARDING THE USE, OR THE RESULTS OF THE USE, OF THE SOFTWARE OR WRITTEN MATERIAL IN TERMS OF CORRECTNESS, ACCURACY, RELIABILITY, OR OTHERWISE.

Trademarks or Service Marks

BEA, ObjectBroker, TOP END, and Tuxedo are registered trademarks of BEA Systems, Inc. BEA Builder, BEA Connect, BEA Manager, BEA MessageQ, BEA Jolt, M3, eSolutions, eLink, WebLogic, WebLogic Enterprise, WebLogic Commerce Server, and WebLogic Personalization Server are trademarks of BEA Systems, Inc.

All other company names may be trademarks of the respective companies with which they are associated.

BEA WebLogic XML/Non-XML Translator Samples Guide

Document Edition	Part Number	Date	Software Version
1.0		January 2001	BEA WebLogic XML/Non-XML Translator 1.0

Contents

1. Running the Purchase Order Sample

What is Included in the Purchase Order Sample	1-2
Prerequisite Considerations	1-2
Understanding the Data Formats Used with XML Translator	1-2
About Binary Data (Non-XML Data)	1-3
About XML Documents	1-3
About MFL Documents	1-4
Performing Binary to XML Translation	1-6
Analyzing the Data to be Translated	1-6
Using the Format Builder To Test the Translation	1-10
Step 1. Starting Format Builder and Creating the Message Format .	1-11
Step 2. Creating Fields	1-11
Step 3. Creating Groups	1-13
Step 4. Creating a Group Reference.....	1-14
Step 5. Creating the Remaining Items	1-15
Step 6. Saving the Message Format	1-16
Step 7. Testing the Message Format	1-17
Performing XML to Binary Translation.....	1-18

2. Running the WebLogic Process Integrator Sample Applications

Prerequisite Considerations	2-1
Running the WebLogic Process Integrator Servlet Sample	2-2
What is Included in the Servlet Sample	2-2
How to Run the Servlet Sample	2-3
Running the WebLogic Process Integrator EJB Sample.....	2-5
What is Included in the EJB Sample	2-5

How to Run the EJB Sample	2-7
Step 1. Configure and Run WebLogic Process Integrator	2-7
Step 2. Create the Business Operations for the Workflow.....	2-7
Step 3. Create the Template and Import the Template Definition	2-16
Step 4. Open and Activate the Template.....	2-17
Step 5. Open WebLogic Process Integrator Worklist and Start the Workflow	2-20

1 Running the Purchase Order Sample

The BEA WebLogic XML/Non-XML Translator (hereafter referred to as XML Translator) software includes a Purchase Order sample designed to illustrate the basic techniques of creating message format definitions for binary data using Format Builder. The Purchase Order sample consists of DTD, MFL, and DATA files. These samples can be used to test your installation of the XML Translator.

The following topics are discussed in this section:

- What is Included in the Purchase Order Sample
- Prerequisite Considerations
- Understanding the Data Formats Used with XML Translator
- Performing Binary to XML Translation
- Performing XML to Binary Translation

What is Included in the Purchase Order Sample

The following table provides a listing and description of the files included in the Purchase Order sample application.

Table 1-1 List of Purchase Order Sample Application Files

Directory	File	Description
samples\po	po_01.data	Purchase order data in binary format
	po_02.data	Additional purchase order data in binary format
	po.dtd	Purchase order document type definition.
	po.mfl	Pre-built message format description of purchase order data.

Prerequisite Considerations

There are certain software applications that must be installed and tasks that must be performed prior to running the Purchase Order sample. Please refer to the *BEA WebLogic XML/Non-XML Translator Release Notes* for more information.

Understanding the Data Formats Used with XML Translator

To understand how the Format Builder is used, it helps to understand the data formats used by XML Translator: binary data, XML, and MFL.

About Binary Data (Non-XML Data)

Because computers are based on the binary numbering system, applications often use a binary format to represent data. A file stored in binary format is computer-readable but not necessarily human-readable. Binary formats are used for executable programs and numeric data, and text formats are used for textual data. Many files contain a combination of binary and text formats. Such files are usually considered to be binary files even though they contain some data in text format.

Unlike XML data, binary data is not self-describing. In other words, binary data does not provide a description of how the data is grouped, divided into fields, or arranged in a layout. Binary data is a sequence of bytes that can be interpreted as an integer, a string, or a picture, depending on the intent of the application that generates the sequence of bytes. For example, the following binary data string can be interpreted many different ways:

```
2231987
```

This could be a date (2/23/1987) or a phone number (223-1987) or any number of other interpretations. Without a clear understanding of the purpose of this data string, the application has no idea how to interpret the string.

In order for binary data to be understood by an application, the format must be embedded within each application that accesses the binary data.

The Format Builder is used to create a Message Format Language (MFL) file that describes the layout of the binary data. MFL is an XML language that includes elements to describe each field of data, as well as groupings of fields (groups), repetition, and aggregation. The hierarchy of a binary record, the layout of the fields, and the groupings of fields and groups are expressed in an MFL document. The MFL document is used by XML Translator at run-time to translate binary data to and from an XML document.

About XML Documents

Extended Markup Language, or XML, is a text format for exchanging data between different systems. It allows data to be described in a simple, standard, text-only format. In contrast to binary data, XML data embeds a description of the data within the data stream. Applications can share data more easily, since they are not dependent on the

layout of the data being embedded within each application. Since the data is presented in a standard form, applications on disparate systems can interpret data in proprietary binary formats.

Instances of XML documents contain character data and markup. The character data is referred to as content, while the markup provides hierarchy for that content. Markup is distinguished from content by angle brackets. Information in the space between the “<” and the “>” is referred to as the tags that markup the content. Tags provide an indication of what the content is for, and a mechanism to describe Parent-child relationships.

An XML document can conform to a content model. A content model allows Metadata (data that is used to describe other data) about XML documents to be communicated to an XML parser. XML documents are said to be “valid” if they conform to a content model. A content model describes the data that can exist in an instance of an XML document. A content model also describes a top-level entity, which is a sequence of subordinate entities. These subordinate entities are further described by their tag names and data content. The two standard formats for XML content models are XML Document Type Definition (DTD) and XML Schema. A Schema is an XML document that defines what can be in an XML document. A DTD also defines what content can exist in an XML document, but the Schema definition is more specific than the DTD, and provides much finer-grained control over the content that can exist in an XML document.

About MFL Documents

Message Format Language (MFL) is an XML language that describes the layout of binary data. This language includes elements to describe each field of data, as well as groupings of fields (groups), repetition, and aggregation. The hierarchy of a binary record, the layout of fields, and the grouping of fields and groups is expressed in an MFL document. MFL documents are created using Format Builder. The Format Builder application allows you to define the structure of binary data and save that information in an MFL document. These MFL documents are then used to perform run-time translation.

The MFL documents you create using Format Builder can contain the following elements:

- Message Format — The top level element. Defines the message name and MFL version.

- **Field** — Sequence of bytes that have some meaning to an application. (For example, the field `EMPNAME` contains an employee name.) Defines the formatting for the field. The formatting parameters you can define include:
 - **Tagged** — Indicates that a literal precedes the data field, denoting the beginning of the field.
 - **Length Field** — Indicates that a length value precedes the data field, denoting the length of this field.
 - **Repeating** — Repeating fields appear more than once in the message format. You can set a specific number of times the field is to repeat, or define a delimiter to indicate the end of the repeating field.
 - **Optional**— The field may or may not be present in the named message format.
- **Groups** — Collections of fields, comments, and other groups or references that are related in some way (for example, the fields `PAYDATE`, `HOURS`, and `RATE` could be part of the `PAYINFO` group). Defines the formatting for all items contained in the group. The formatting parameters you can define include:
 - **Repeating** — Repeating groups appear more than once in the message format: You can set a specific number of times the group is to repeat, or define a delimiter to indicate the end of the repeating group.
 - **Choice of Children** — Defining a group as “Choice of Children” means that only one item in the group will appear in the message format.
 - **Optional**— The group of data within this structure may or may not be present in the named message format.
- **References** — Indicates that another instance of the field or group format exists in the data. Reference fields or groups have the same format as the original field or group, but you can change the optional setting and the occurrence setting for the reference field or group. For example, if you have a “bill to” address and a “ship to” address in your data, you only need to define the address format once. You can create the “bill to” address definition and create a reference for the “ship to” address.
- **Comments** — Notes or additional information about the message format.

Performing Binary to XML Translation

The following sections provide information on building sample purchase order format definitions using the Format Builder utility to test the translation of binary data into XML format.

- Analyzing the Data to be Translated
- Using the Format Builder To Test the Translation

The Format Builder included with XML Translator allows you to build format definitions for binary data that will be translated to or from XML. Format definitions are the metadata used to parse binary data.

Analyzing the Data to be Translated

The key to translating binary data to and from XML is to create an accurate description of the binary data. For binary data (data that is not self-describing), you must identify the following elements:

- Hierarchical groups
- Group attributes, such as name, optional, repeating, delimited
- Data fields
- Data field attributes, such as name, data type, length/termination, optional, repeating

Listing 1-1 shows a sample of binary data. This file is included on the XML Translator CD-ROM and is called `\sample\po\po_01.data`. In this sample, the example data is taken from a fictitious purchase order on a proprietary system that the XYZ Corporation uses. They would like to interchange this information with another system that accepts XML data.

Listing 1-1 Sample Binary Purchase Order Data

```
1234;88844321;SUP:21Sprockley's Sprockets01/15/2000123 Main St.;
Austin;TX;75222;555 State St.; Austin;TX;75222;
PO12345678;666123;150;Red Sprocket;
```

Perform the following steps to analyze the purchase order data:

1. Get the definition of the data. This may involve using printed specifications or internal documentation. For this sample, we have described the purchase order format in Table 1-2.

Table 1-2 Purchase Order Master Record

Field Name	Data Type	Length	Description
Purchase Request Number	Numeric	Delimited by semicolon	The Purchase Request number assigned by the Purchasing department. This number is used to track the status of an order from requisition through delivery and payment.
Supplier ID	Numeric	Delimited by semicolon	The identification of the assigned supplier as defined in the corporate Supplier Data Base. Assignment of an approved supplier is made by the buyer when creating a Purchase Request from a requisition.
Supplier Name	Character	Prefixed by a literal "SUP:". Following this literal is a two digit numeric length field.	The name of the assigned supplier as defined in the corporate Supplier Data Base. This field is prefixed with a literal to indicate that it is present.
Requested Delivery Date	Date MM/DD/YYYY	10 characters	The delivery date specified by the requisitioner.
Shipping Street	Character	Delimited by semicolon	The street address to be used in shipping the requested items.
Shipping City	Character	Delimited by semicolon	The city to be used in shipping the requested items.

1 *Running the Purchase Order Sample*

Field Name	Data Type	Length	Description
Shipping State	Character	Delimited by semicolon	The state to be used in shipping the requested items.
Shipping Zip	Numeric	Delimited by semicolon	The zip code to used in shipping the requested items.
Billing Street	Character	Delimited by semicolon	The street address to be used for billing.
Billing City	Character	Delimited by semicolon	The city to be used for billing.
Billing State	Character	Delimited by semicolon	The state to be used for billing.
Billing Zip	Numeric	Delimited by semicolon	The zip code to used for billing.
Payment Terms			Supported payment terms may be either Purchase Order or Company Credit Card. A literal preceding the payment information identifies the type.
PO Type	Character	Literal "PO"	Indicates PO payment terms.
PO Number	Numeric	Delimited by semicolon	Purchase Order number.
Credit Card Type	Character	Literal "CC"	Indicates Credit Card payment terms.
Credit Card Number	Numeric	Delimited by semicolon	Credit card number.
Credit Card Expiration Month	Numeric	Delimited by semicolon	Expiration month for credit card.
Credit Card Expiration Year	Numeric	Delimited by semicolon	Expiration year for credit card.
Purchase Items			The following fields identify the items to be purchased. This information may be repeated for each item that is part of this Purchase Request. At least one item must be present.
Part Number	Numeric	Delimited by semicolon	The supplier's part number of the requested item.
Quantity	Numeric	Delimited by semicolon	The quantity requested. Must be greater than zero.

Field Name	Data Type	Length	Description
Description	Character	Delimited by semicolon	Description of the requested item.

2. Identify hierarchical groups.

Groups are collections of fields, comments, and other groups or references that are related in some way. In the sample data presented in Table 1-2, notice that two distinct groups can be defined: Payment Terms and Purchase Items. In addition to these groups, notice that there are several fields related to shipping and billing addresses. You can define a group for Shipping Address and a group for Billing Address.

3. Identify group attributes.

You need to define the attributes of the hierarchical groups. Group attributes include the name of the group, whether the group is optional, repeating, delimited, or can be defined as a reference to another group. For example, look at the Address group within the Shipping Address and Billing Address groups. These two groups contain the same fields with the same attributes. Therefore, you can define the Address group within the Shipping Address group and set up the Address group within the Billing Address group as a reference. For more information on references, refer to the *BEA WebLogic XML/Non-XML Translator User Guide*.

4. Identify data fields.

Fields are a sequence of bytes that have some meaning to an application. In the sample data shown in Table 1-2, some of the fields are Purchase Request Number, Supplier ID, Supplier Name, etc.

5. Identify data field attributes.

You need to define the attributes of the data fields. Field attributes include the name of the field, the type of data contained in the field, the length of the field, or the delimiter that denotes the end of the field. For example, the Supplier ID field is delimited by a semicolon (;) indicating the end of the field data, but the Requested Delivery Date has an implied length of 10 characters.

Once you have completed the steps above, it might be helpful to put the data into a spreadsheet form as shown in Table 1-3. This will assist you in entering the data in Format Builder to create your message definitions.

1 Running the Purchase Order Sample

Table 1-3 Analysis of Purchase Order Data

Description	Group	Field	Reference	Optional	Name / Refers To	Data Type	Occurrence	Delimited by
Purchase Request Number		X			PR_Number	Numeric	1	Semicolon
Supplier ID		X			Supplier_ID	Numeric	1	Semicolon
Supplier Name		X	X		Supplier_Name	String	1	Numeric field length 2
Requested Delivery Date		X			Requested_Delivery_Date	Date MM/DD/YYYY	1	Semicolon
Shipping Address	X				Shipping_Address		1	
Street		X			Street	String	1	Semicolon
City		X			City	String	1	Semicolon
State		X			State	String	1	Semicolon
Zip		X			Zip	Numeric	1	Semicolon
Billing Address			X		Address		1	Semicolon
Street			X		Street	String	1	Semicolon
City			X		City	String	1	Semicolon
State			X		State	String	1	Semicolon
Zip			X		Zip	Numeric	1	Semicolon
Payment Terms	X				Payment Terms		1	
Purchase Order	X						0 or 1	
Purchase Order Tag		X			Payment_Type_PO	Literal "PO"	1	Semicolon
Purchase Order Number		X			PO_Number	Numeric	1	Semicolon
Credit Card	X						0 or 1	
Payment Type		X			Payment_Type_CC	Literal "CC"	1	Semicolon
Credit Card Number		X			CC_Number	Numeric	1	Semicolon
Credit Card Expire Month		X			CC_Expire_Month	Numeric	1	Semicolon
Credit Card Expire Year		X			CC_Expire_Year	Numeric	1	Semicolon
Purchase Items	X				Purchase_Items		1-n	Semicolon
Part Number		X			Part_Number	Numeric	1	Semicolon
Quantity		X			Quantity	Numeric	1	Semicolon
Description		X			Description	String	1	Semicolon

Using the Format Builder To Test the Translation

This section walks you through the steps required in Format Builder to create the message definition file for translating the binary Purchase Order data to XML.

Notes: For details on entering data in the detail windows of Format Builder, refer to the *BEA WebLogic XML/Non-XML User Guide*.

The file `\sample\po\po.mfl` included on the CD-ROM contains the message definition created by the steps below. You can use this file for reference to make sure you create the definition correctly.

Step 1. Starting Format Builder and Creating the Message Format

1. Choose Start Menu→Programs→BEA WebLogic XML/Non-XML Translator→Format Builder. The Format Builder main window displays.
2. Choose File→New. A new message definition opens.
3. Enter **PurchaseRequest** as the message format name and click Apply. The tree pane changes to reflect the new message format name.

Step 2. Creating Fields

1. Select PurchaseRequest in the tree pane and choose Insert→Field→As Child. The Field Description detail window opens.
2. Enter the field details as follows:

Field	Value
Name	PR_Number
Type	Numeric
Field Occurrence	Once
Delimiter	; (semi-colon)

These values were determined by our analysis of the raw purchase order data, as you can see in [Table 1-3](#).

3. Click Apply. The PR_Number field is saved to the message format file.

Note: Since the only difference between the PR_Number field and the Supplier_ID field is the name, we will use the Duplicate feature of Format Builder to create the Supplier_ID field.
4. Select the PR_Number field you just created in the tree pane and click Duplicate. A new field description is displayed in the detail pane. This field is always prefixed by the tag ‘SUP:’ and a length that indicates how long the supplier name is.
5. Enter Supplier_ID as the name and click Apply. The Supplier_ID field is created and stored in the message format file.

1 Running the Purchase Order Sample

6. Select the Supplier_ID field you just created in the tree pane and choose Insert→Field→As Sibling. This field is always prefixed by the tag ‘SUP:’ and a length that indicates how long the supplier name is.
7. Enter the Supplier_Name field details as follows:

Field	Value
Name	Supplier_Name
Optional	Select this option
Tagged	Select this option and choose “String” from the drop down list
Tag	SUP:
Type	String
Field Occurrence	Once
Termination (Length Field Tab)	
Type	Numeric
Length	Select this option and enter “2” in the text box

8. Click Apply. The Supplier_Name field is created and added to the tree pane.
Note: The dashed box around the field icon in the tree pane indicates this is an optional field.
9. Select the Supplier_Name field you just created in the tree pane and choose Insert→Field→As Sibling.
10. Enter the Requested_Delivery_Date field details as follows:

Note: The Field detail pane changes based on the Type selected. Because this field is a Date type, it has an implied length and does not require you to specify the termination.

Field	Value
Name	Requested_Delivery_Date

Field	Value
Type	Date: MM/DD/YYYY
Field Occurrence	Once
Data Base Type	String

11. Click Apply. The Requested_Delivery_Date field is created and added to the tree pane.

Step 3. Creating Groups

1. Select the Requested_Delivery_Date field in the tree pane and choose Insert→Group→As Sibling. The Group Detail window opens.
2. Enter the group details as follows:

Field	Value
Name	Shipping_Address
Field Occurrence	Once

These values were determined by our analysis of the raw purchase order data, as you can see in [Table 1-3](#).

3. Click Apply. The Shipping_Address group is created and added to the tree pane.
We know from our initial data analysis that the Shipping Address and Billing Address groups contain the same fields with the same attributes. Therefore, we can define the Address group within the Shipping Address group and set up the Address group within the Billing Address group as a reference. For more information on references, refer to the *BEA WebLogic XML/Non-XML Translator User Guide*.
4. Select the Shipping_Address group in the tree pane and choose Insert→Group→As Child.
5. Use the data in [Table 1-3](#) to create the Address group and click Apply to save the data.

1 Running the Purchase Order Sample

6. Follow the steps outlined in [Step 2. Creating Fields](#) to create the Street, City, State, and Zip fields as children of the Address group.

Note: You can use the Duplicate button to create the City and State fields, once the Street field is created.

Step 4. Creating a Group Reference

Now we are going to create the Billing Address group and fields. Since this is a duplicate of the Shipping Address group, we can create a group reference. A reference group has the same format as the original group, but you can change the optional setting and the occurrence setting for the reference group.

1. Select the Shipping_Address group in the tree pane and choose Insert→Group→As Sibling. The Group Detail window opens.
2. Enter the group details as follows:

Field	Value
Name	Billing_Address
Field Occurrence	Once

These values were determined by our analysis of the raw purchase order data, as you can see in [Table 1-3](#).

3. Select the Address group under Shipping_Address in the tree pane and choose Edit→Copy. This copies the Address group details (including child objects) and places them on the clipboard.
4. Select the Billing_Address group you created in step 2 in the tree pane and choose Edit→Paste→As Reference. This pastes the copy of the Address group into the message definition as a sibling of the Billing Address group.

Note: You can identify this Address group as a reference by the icon located to the left of it in the tree pane.



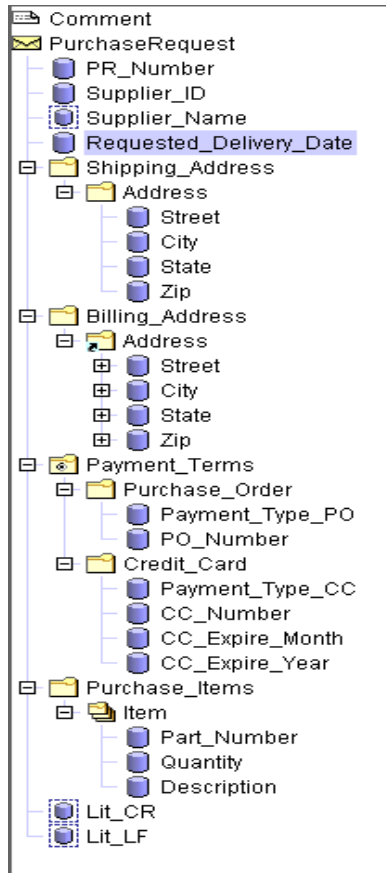
5. Now, we need to change the Address reference group to be a child of the Billing_Address group. Select the Address reference group in the tree pane and choose Edit→Demote. The Address reference group moves “under” the Billing_Address group.

Step 5. Creating the Remaining Items

Follow Steps 1 through 4 above to create the remaining items necessary to complete the message definition for the Purchase Order sample. Use the analysis of the raw purchase order data in [Table 1-3](#) to determine the values you need to enter for each item. You can use the file `\sample\po\po.mfl` for reference if you need assistance.

When you finish entering the items, your tree pane should look similar to Figure 1-1.

Figure 1-1 Completed Tree Pane for Purchase Order Sample



Step 6. Saving the Message Format

To save a message format file:

1. Choose File→Save As. The Save As dialog displays.
2. Navigate to the directory where you want to save the file.
3. In the File Name text box, type the name you want to assign to the file.

Note: Format Builder automatically assigns the .MFL extension to message format files by default if no extension is given.

4. Click Save As to save the file in the specified location with the specified name and extension.

Once you have generated the Message Format definition, you can create a DTD or XML Schema document that describes the XML to be converted. You can set up Format Builder to automatically generate a DTD and/or Schema for your message definitions as follows:

1. From the Format Builder main window, choose Tools→Options. The Format Builder Options dialog opens.
2. Select Auto-generate DTD and/or Auto-generate Schema to have Format Builder automatically create these documents during the translation process.
3. Click OK to activate your selections.

Now, whenever you save message format documents, Format Builder will generate a DTD and/or a Schema for your message format. Refer to the *BEA WebLogic XML/Non-XML User Guide* for more information on DTDs and Schemas.

Step 7. Testing the Message Format

Now, we need to test the message format to identify any errors that exist before using it to translate actual data.

1. Choose Test→Message Format. The Tester opens. This allows you to test the translation of the binary purchase order data into XML.
2. Click Load and navigate to the `SAMPLES\PO` directory.
3. Choose the file `PO_01.DATA` and click Open. The left side of the Format Tester dialog displays the binary data.
4. Click **To XML >**. The binary data is translated, and the right side of the dialog displays the purchase order data in XML format.

Note: You can see the messages output during the translation by selecting Debug.

5. If the translated data appears correct, click Save on the right side of the dialog (under XML).

6. Navigate to the `SAMPLES\PO` directory and enter `PO.XML` as a name for the XML data.

Performing XML to Binary Translation

You can also use Format Builder to create message definitions and test the translation of XML data to binary. The steps required to do this are essentially the same as translating binary data to XML. To translate XML data to binary, first create an MFL description of the binary format. The Purchase Order Record sample provides an MFL document that can be loaded by performing the following steps:

1. Choose File→Open in Format Builder.
 2. Navigate to the directory containing the desired file and select the file name.
 3. Click open. The file is loaded into Format Builder.
 4. Choose Tester→Message Format.
 5. Under the XML panel, click Load, and navigate to the `SAMPLES\PO` directory.
 6. Choose the file `po.xml` and click Open. The right side of the Format Tester dialog displays the XML data.
 7. Click **<To Binary**. The XML data is translated, and the right side of the dialog displays the purchase order data in Binary format.
- Note:** You can see the messages output during the translation by selecting Debug.

2 Running the WebLogic Process Integrator Sample Applications

The BEA WebLogic XML/Non-XML Translator software includes two sample applications designed to illustrate the integration of XML Translator with BEA WebLogic Process Integrator. This section describes these samples and gives you step-by-step instructions for running the samples. The following topics are discussed:

- Prerequisite Considerations
- Running the WebLogic Process Integrator Servlet Sample
- Running the WebLogic Process Integrator EJB Sample

Prerequisite Considerations

There are certain software applications that must be installed and tasks that must be performed prior to running the Purchase Order sample. Please refer to the *BEA WebLogic XML/Non-XML Translator Release Notes* for more information.

Note: The instructions presented in this section assume that you have a good working knowledge of BEA WebLogic Process Integrator and BEA WebLogic Server. You should have successfully installed WebLogic Process Integrator and run a sample workflow.

Running the WebLogic Process Integrator 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, a generated XML file may be posted to a JMS topic to act as a starting event for a WebLogic Process Integrator workflow.

Note: This sample does not require the interface to WebLogic Process Integrator; however, in order to start the workflow, WebLogic Process Integrator is required.

What is Included in the Servlet Sample

The following table provides a listing and description of the files included in the WebLogic Process Integrator Servlet sample application. This sample application can be found in the `samples\wpli\servlet` directory.

Table 2-1 List of Servlet Sample Application Files

Directory	File	Description
<code>\servlet\source</code>	<code>WLPI_sample.java</code>	The source code for the servlet used to present the HTML screen and process binary data to XML. This XML may then, optionally, be placed onto the WebLogic Process Integrator JMS topic.
<code>\servlet</code>	<code>SampleData.mfl</code>	The Message Format Language description of the sample binary data file used to start the sample WebLogic Process Integrator workflow.
<code>\servlet</code>	<code>SampleData.data</code>	The sample data file used as input to start the sample WebLogic Process Integrator workflow.

Table 2-1 List of Servlet Sample Application Files

Directory	File	Description
\servlet	SampleWorkflow.xml	The exported WebLogic Process Integrator workflow used in the sample. This workflow should be imported via the WebLogic Process Integrator Studio GUI to setup the workflow tasks involved in the sample.
\servlet	WLPI_sample.war	A Web Archive file containing all executable sample code and configuration files.
\servlet\images	bealogo.jpg	The BEA logo image displayed on the HTML page rendered by the sample servlet.
\servlet\WEB-INF	hello.html	The HTML page used by the sample servlet to obtain input data from the user.
\servlet\WEB-INF	web.xml	The J2EE configuration file defining deployment information for the sample servlet.
\servlet\WEB-INF	weblogic.xml	The BEA configuration file defining WebLogic-specific information for the sample servlet.
\servlet\WEB-INF \lib	*.jar	Utility libraries, including XML Translator, that are used in the execution of the sample code.

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 WebLogic Process Integrator, refer to the documentation that accompanies those applications.

To run the servlet sample stand-alone (without WebLogic Process Integrator):

1. Copy the `license-group` section from the XML Translator license file (`license.bea`) to the WebLogic Process Integrator license file. For details on this procedure, refer to the BEA WebLogic XML/Non-XML Translator *Release Notes* and the WebLogic Process Integrator documentation.

2. Using a text editor, add the following line to the end of your `weblogic.properties` file:

```
weblogic.httpd.webApp.WLXT=<path to war file>/WLPI_sample.war
```

Note: If you have both WebLogic Server and WebLogic Process Integrator installed, you may have two instances of the `weblogic.properties` file. Make sure you edit the file in the WebLogic Process Integrator directory for this sample exercise.

3. Start WebLogic.
4. Start a web browser and enter the following URL:
`http://<weblogic server>:<weblogic port>/WLXT/WLXTTest`
5. Select an MFL file and a data file from your local machine.
6. Click Submit. The server response displays the generated XML data.

The following additional steps can be run to start the workflow if you are integrating with WebLogic Process Integrator:

1. Using a text editor, open the `weblogic.properties` file in the `wlpi\Server` directory.
2. Modify the `mail.host` parameter to contain the address of your SMTP server.
3. Start WebLogic Process Integrator.
4. Start the WebLogic Process Integrator Studio and login.
5. Select Templates in the left pane, click the right mouse button, and choose Create to create a new template.
6. Select the newly created template, click the right mouse button, and choose Import Template Definition.
7. Select the file `\samples\wlpi\SampleWorkFlow.xml`.
8. Select the newly imported workflow, click the right mouse button, and choose Properties. The Properties dialog displays.
9. Select Active and click OK.
10. Select the workflow, click the right mouse button, and choose Save.
11. Using a text editor, open the file `\samples\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.
12. Open a browser and go to the following URL:

`http://<weblogic server/port>/WLXT/WLXTTest`

13. Enter the following as the MFL file:

`samples\wlpi\SampleData.xml`

14. Enter the following as the data file:

`samples\wlpi\SampleData.data`

15. Select the option to invoke WebLogic Process Integrator and click Submit. A short email message is sent to the address you supplied in the data file.

Running the WebLogic Process Integrator 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 WebLogic Process Integrator EJB sample application. This sample application can be found in the `samples\wpli\ejb` directory.

Table 2-2 List of EJB Sample Application Files

Directory	File	Description
\ejb	Makefile	Make file for building the sample source to a jar file.
\ejb	WLXTEexample.xml	Exported sample workflow from WebLogic Process Integrator

2 *Running the WebLogic Process Integrator Sample Applications*

Table 2-2 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	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\lib	WLXTEJB.jar	Executables for the sample application
\ejb\source	XMLnonXMLTranslator.java	EJB wrapper class for WLXT
\ejb\source	XMLnonXMLTranslatorHome.java	EJB wrapper class for WLXT
\ejb\source	XMLnonXMLTranslatorBean.java	EJB wrapper class for WLXT
\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 WLPI 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

How to Run the EJB Sample

Follow the steps below to run the EJB sample. For specific instructions on performing the tasks in WebLogic Process Integrator and WebLogic Server, please refer to the documentation that accompanies those applications.

Step 1. Configure and Run WebLogic Process Integrator

1. Copy the `license-group` section from the XML Translator license file (`license.bea`) to the WebLogic Process Integrator license file. For details on this procedure, refer to the BEA WebLogic XML/Non-XML Translator *Release Notes* and the WebLogic Process Integrator documentation.
2. Deploy the sample beans by adding an entry to the WebLogic Properties file. For example:

```
weblogic.ejb.deploy=d:/beahome/wlxt1.0/samples/ejb/lib/WLXTEJB.jar
```

3. Add the full path and file name of the following `.jar` files to the `server.cmd` file used to start the WebLogic Process Integrator Server.

```
xml.jar  
dom2.jar  
sax2.jar  
ebase.jar  
wlxt.jar
```

4. In the WebLogic Process Integrator home directory, create a subdirectory called `mflrepos`.
5. Copy the two sample `.mfl` files included with this sample (`Payroll.mfl` and `HR.mfl`) to the new subdirectory.
6. Start the WebLogic Process Integrator Server.
7. Run WebLogic Process Integrator Studio.

Step 2. Create the Business Operations for the Workflow

You need to create seven business operations for the sample. These business operations are referenced by the sample workflow; therefore, their names must exactly match the list below.

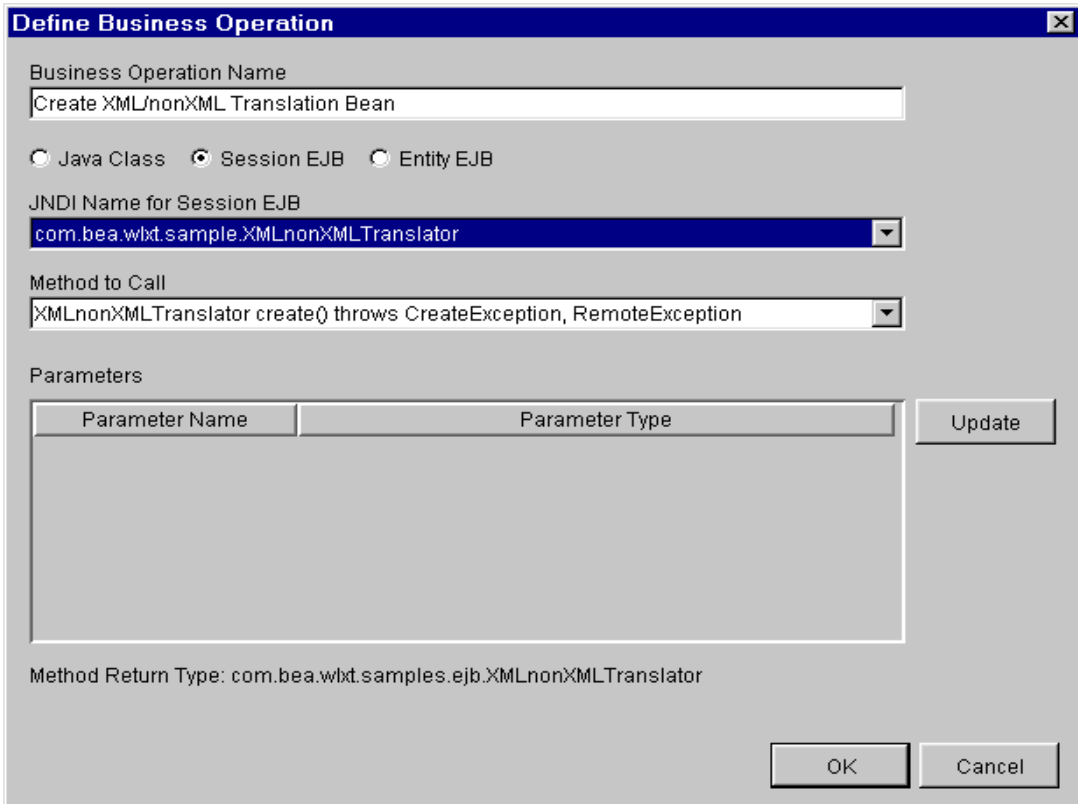
- Create XML/nonXML Translation Bean
- Create Example HR Bean
- Get Employee Info
- Translate Binary to XML
- Create Example Payroll Bean
- Post Payroll Data
- Translate XML to Binary

Notes: The names of these operations must exactly match the above list. Business operation names are case-sensitive.

To create the business operations:

1. Choose Configure→Business Operations and click Add.
2. Enter the information for each business operation as shown in each of the seven dialog illustrations that follow.
3. After you enter the information in the dialog, click OK.
4. Repeat steps 1 through 3 for each business operation.

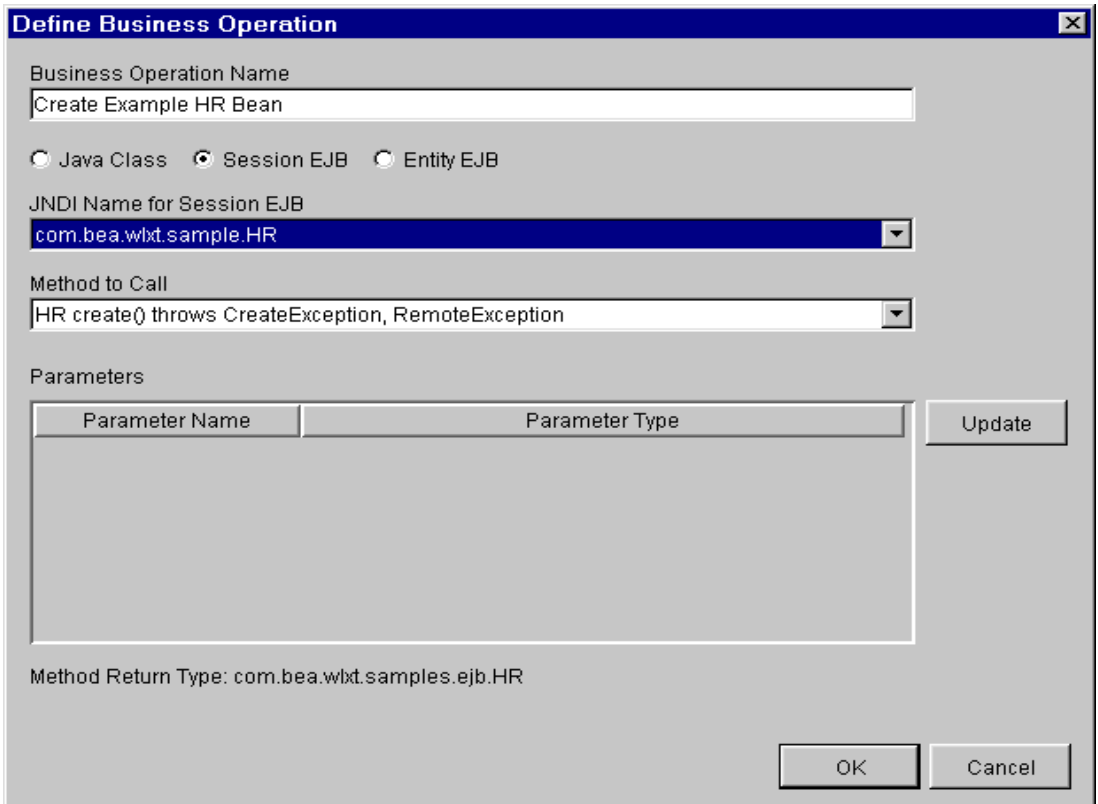
Figure 2-1 Create XML/nonXML Translation Bean Business Operation



The dialog box titled "Define Business Operation" contains the following fields and controls:

- Business Operation Name:** A text field containing "Create XML/nonXML Translation Bean".
- Radio Buttons:** Three options: "Java Class" (unselected), "Session EJB" (selected), and "Entity EJB" (unselected).
- JNDI Name for Session EJB:** A dropdown menu with "com.bea.wlbt.sample.XMLnonXMLTranslator" selected.
- Method to Call:** A dropdown menu with "XMLnonXMLTranslator create() throws CreateException, RemoteException" selected.
- Parameters:** A table with two columns: "Parameter Name" and "Parameter Type". The table is currently empty.
- Method Return Type:** A text label showing "com.bea.wlbt.samples.ejb.XMLnonXMLTranslator".
- Buttons:** "Update", "OK", and "Cancel".

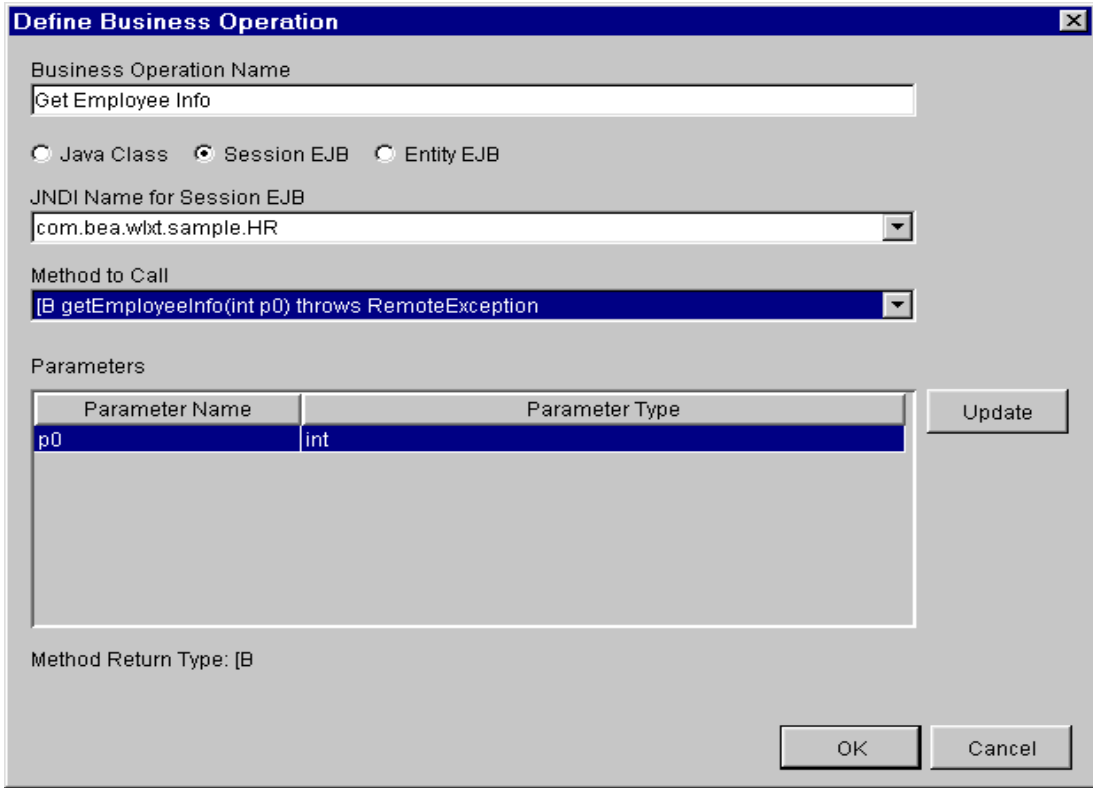
Figure 2-2 Create Example HR Bean Business Operation



The dialog box is titled "Define Business Operation" and contains the following fields and controls:

- Business Operation Name:** A text field containing "Create Example HR Bean".
- Radio Buttons:** Three options are present: "Java Class" (unselected), "Session EJB" (selected), and "Entity EJB" (unselected).
- JNDI Name for Session EJB:** A dropdown menu with "com.bea.wlxt.sample.HR" selected.
- Method to Call:** A dropdown menu with "HR create() throws CreateException, RemoteException" selected.
- Parameters:** A table with two columns: "Parameter Name" and "Parameter Type". The table is currently empty.
- Update:** A button located to the right of the parameters table.
- Method Return Type:** A text label at the bottom left showing "com.bea.wlxt.samples.ejb.HR".
- OK and Cancel:** Two buttons at the bottom right.

Figure 2-3 Get Employee Info Business Operation

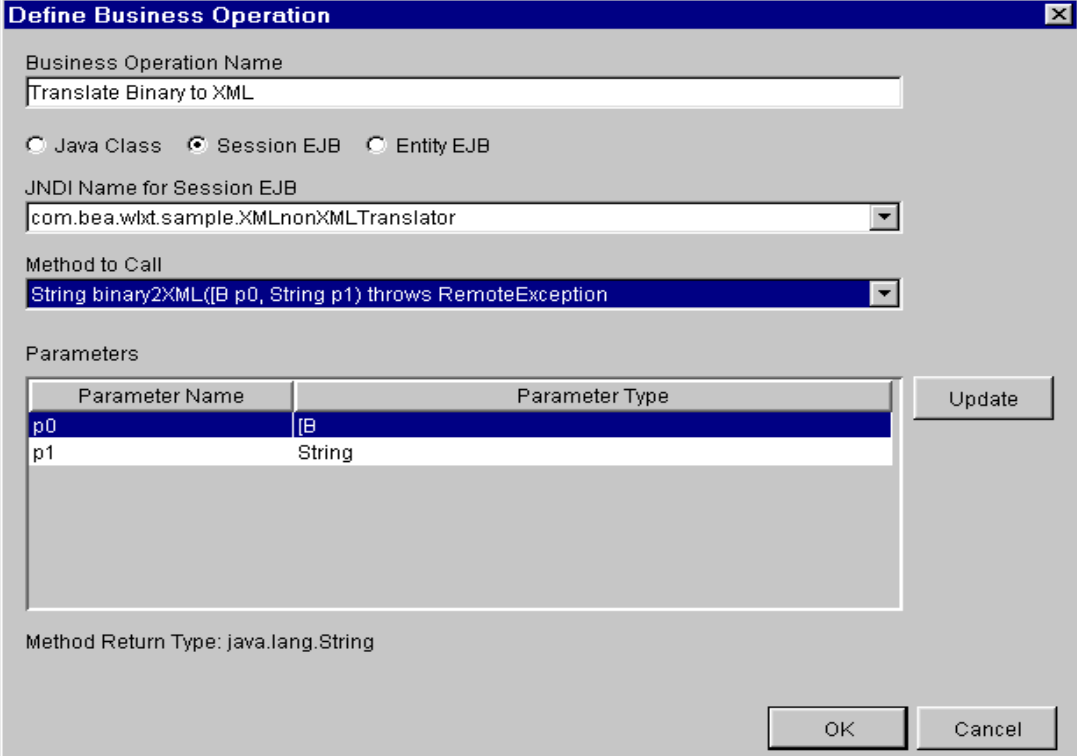


The dialog box is titled "Define Business Operation" and contains the following fields and controls:

- Business Operation Name:** A text field containing "Get Employee Info".
- Radio Buttons:** Three options are present: "Java Class" (unselected), "Session EJB" (selected), and "Entity EJB" (unselected).
- JNDI Name for Session EJB:** A dropdown menu showing "com.bea.wlxt.sample.HR".
- Method to Call:** A dropdown menu showing "[B] getEmployeeInfo(int p0) throws RemoteException".
- Parameters:** A table with two columns: "Parameter Name" and "Parameter Type". One row is visible with "p0" and "int".
- Method Return Type:** A text field showing "[B]".
- Buttons:** "Update", "OK", and "Cancel" buttons are located at the bottom right.

Parameter Name	Parameter Type
p0	int

Figure 2-4 Translate Binary to XML Business Operation

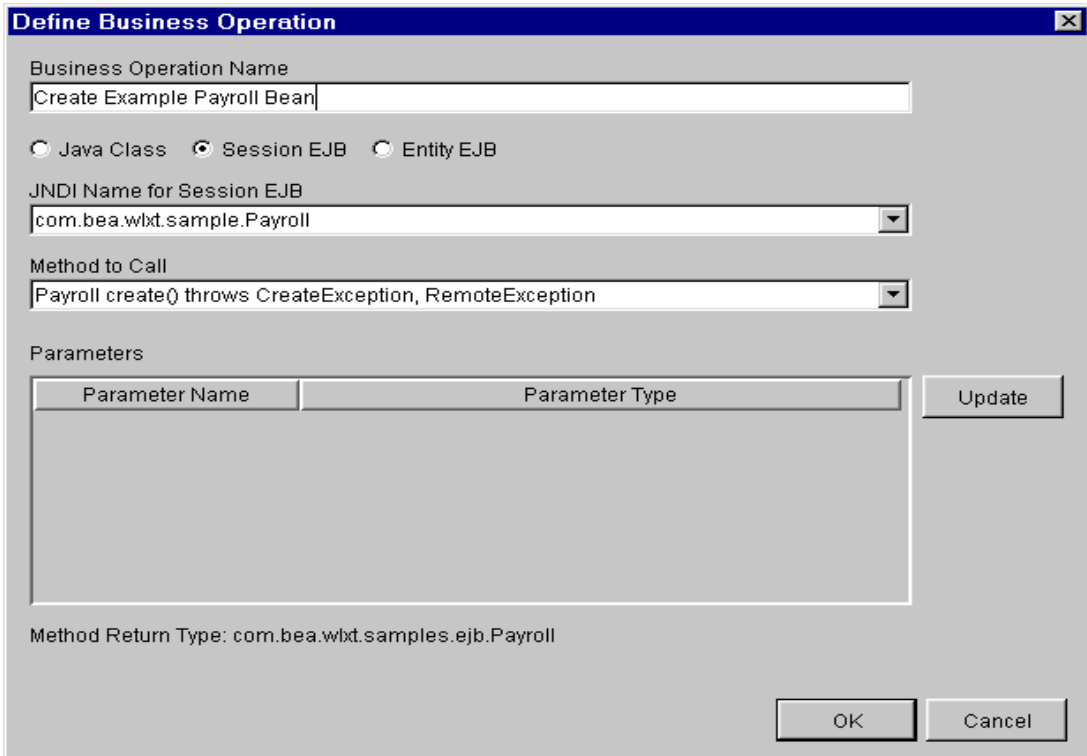


The dialog box titled "Define Business Operation" contains the following fields and controls:

- Business Operation Name:** A text field containing "Translate Binary to XML".
- Radio Buttons:** Three options are present: "Java Class" (unselected), "Session EJB" (selected), and "Entity EJB" (unselected).
- JNDI Name for Session EJB:** A dropdown menu showing "com.bea.wlxt.sample.XMLnonXMLTranslator".
- Method to Call:** A dropdown menu showing "String binary2XML([B p0, String p1) throws RemoteException".
- Parameters Table:** A table with two columns: "Parameter Name" and "Parameter Type".

Parameter Name	Parameter Type
p0	[B
p1	String
- Update Button:** A button located to the right of the parameters table.
- Method Return Type:** A text label showing "java.lang.String".
- OK and Cancel Buttons:** Two buttons located at the bottom right of the dialog.

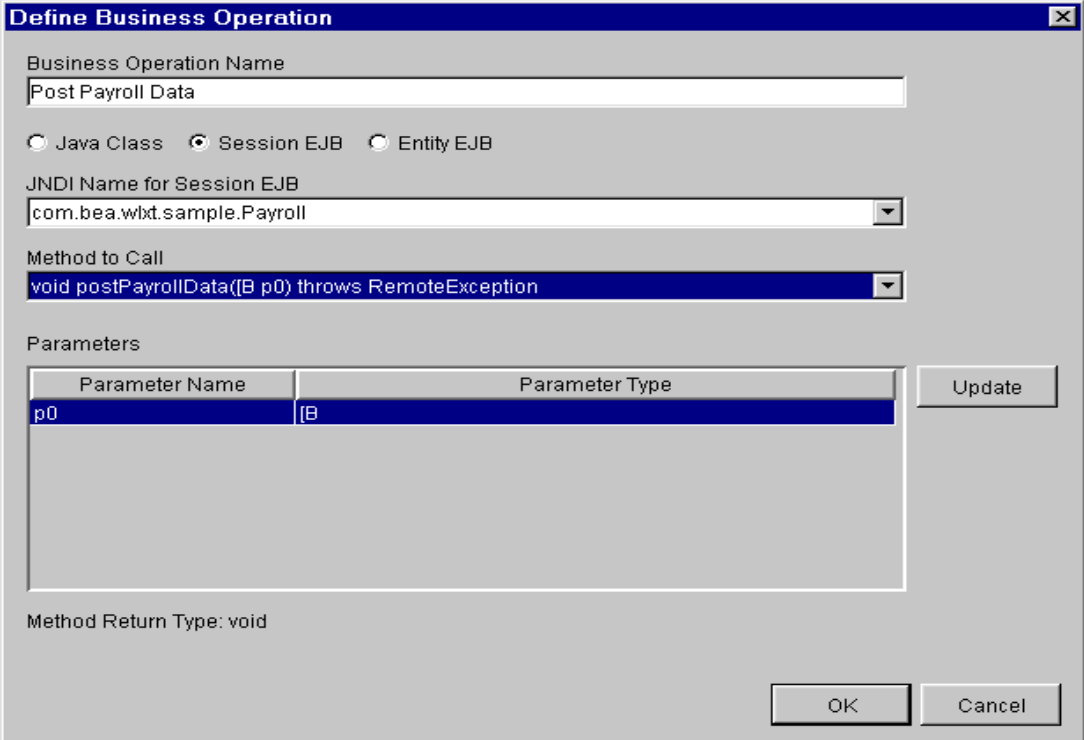
Figure 2-5 Create Example Payroll Bean Business Operation



The dialog box is titled "Define Business Operation" and contains the following fields and controls:

- Business Operation Name:** A text field containing "Create Example Payroll Bean".
- Radio Buttons:** Three options are present: "Java Class" (unselected), "Session EJB" (selected), and "Entity EJB" (unselected).
- JNDI Name for Session EJB:** A dropdown menu showing "com.bea.wlxt.sample.Payroll".
- Method to Call:** A dropdown menu showing "Payroll create() throws CreateException, RemoteException".
- Parameters:** A table with two columns: "Parameter Name" and "Parameter Type". The table is currently empty.
- Method Return Type:** A text label showing "com.bea.wlxt.samples.ejb.Payroll".
- Buttons:** "Update", "OK", and "Cancel" buttons are located at the bottom of the dialog.

Figure 2-6 Post Payroll Data Business Operation



The dialog box is titled "Define Business Operation" and contains the following fields and controls:

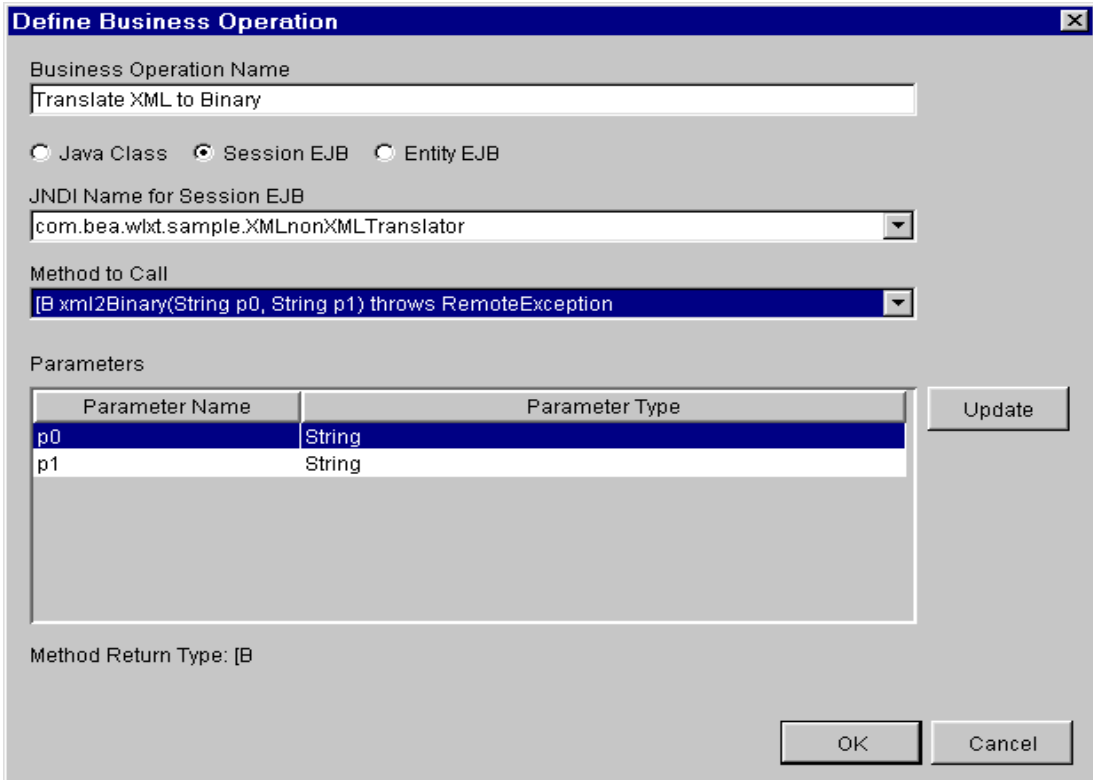
- Business Operation Name:** A text field containing "Post Payroll Data".
- Radio Buttons:** Three options are present: "Java Class" (unselected), "Session EJB" (selected), and "Entity EJB" (unselected).
- JNDI Name for Session EJB:** A dropdown menu showing "com.bea.wlxt.sample.Payroll".
- Method to Call:** A dropdown menu showing "void postPayrollData([B p0] throws RemoteException)".
- Parameters:** A table with two columns: "Parameter Name" and "Parameter Type".

Parameter Name	Parameter Type
p0	[B

Below the table, the text "Method Return Type: void" is displayed.

Buttons: "Update", "OK", and "Cancel".

Figure 2-7 Translate XML to Binary Business Operation



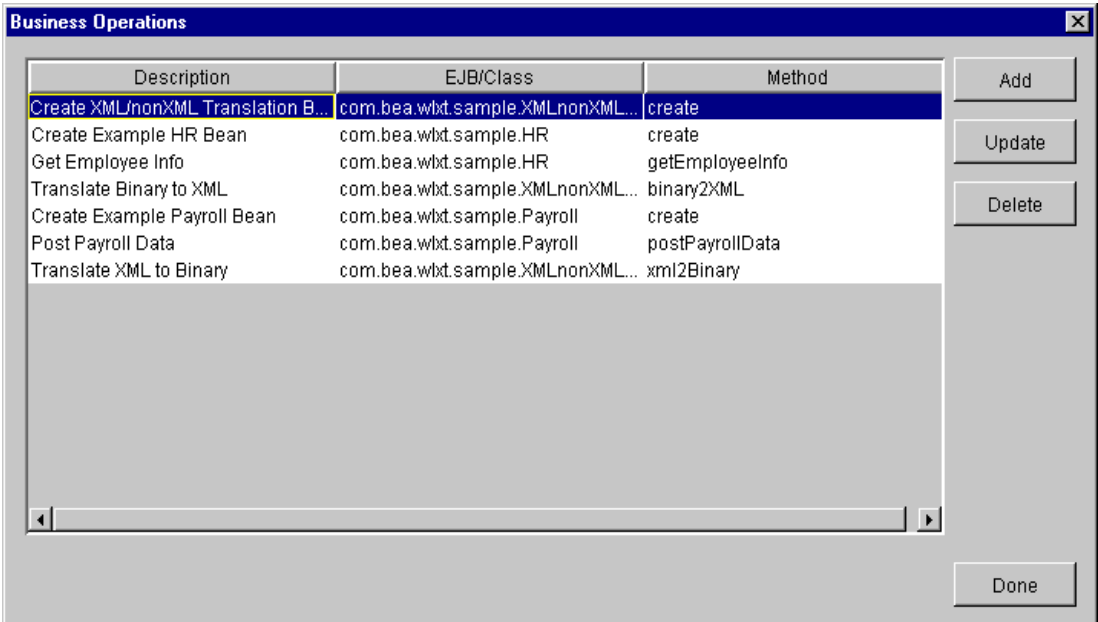
The dialog box titled "Define Business Operation" contains the following fields and controls:

- Business Operation Name:** A text field containing "Translate XML to Binary".
- Radio Buttons:** Three options: "Java Class" (unselected), "Session EJB" (selected), and "Entity EJB" (unselected).
- JNDI Name for Session EJB:** A dropdown menu showing "com.bea.wlxt.sample.XMLnonXMLTranslator".
- Method to Call:** A dropdown menu showing "[B xml2Binary(String p0, String p1) throws RemoteException".
- Parameters:** A table with two columns: "Parameter Name" and "Parameter Type".

Parameter Name	Parameter Type
p0	String
p1	String
- Method Return Type:** A text field containing "[B".
- Buttons:** "Update", "OK", and "Cancel".

The following dialog shows a list of the completed business operations.

Figure 2-8 List of Completed Business Operations

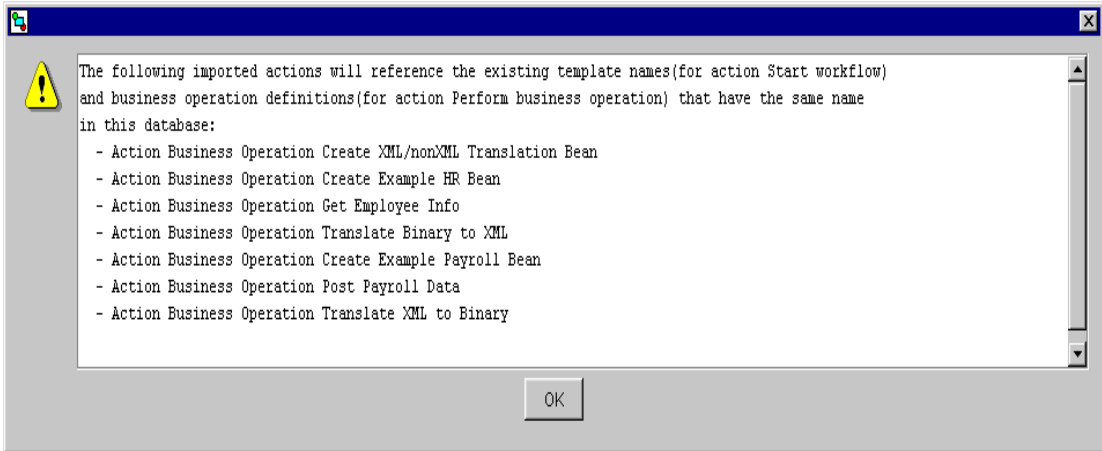


5. Click Done when you have finished creating the Business Operations.

Step 3. Create the Template and Import the Template Definition

1. Create a new template called `WLXT Example`.
2. Select the template from the tree view in the left pane and click the right mouse button.
3. Choose Import Template Definition.
4. Select the definition file `WLXTExample.xml`. A message informing you that the business operations you just created will be executed during this workflow.

Figure 2-9 Import Message



5. Click OK to close the message box.

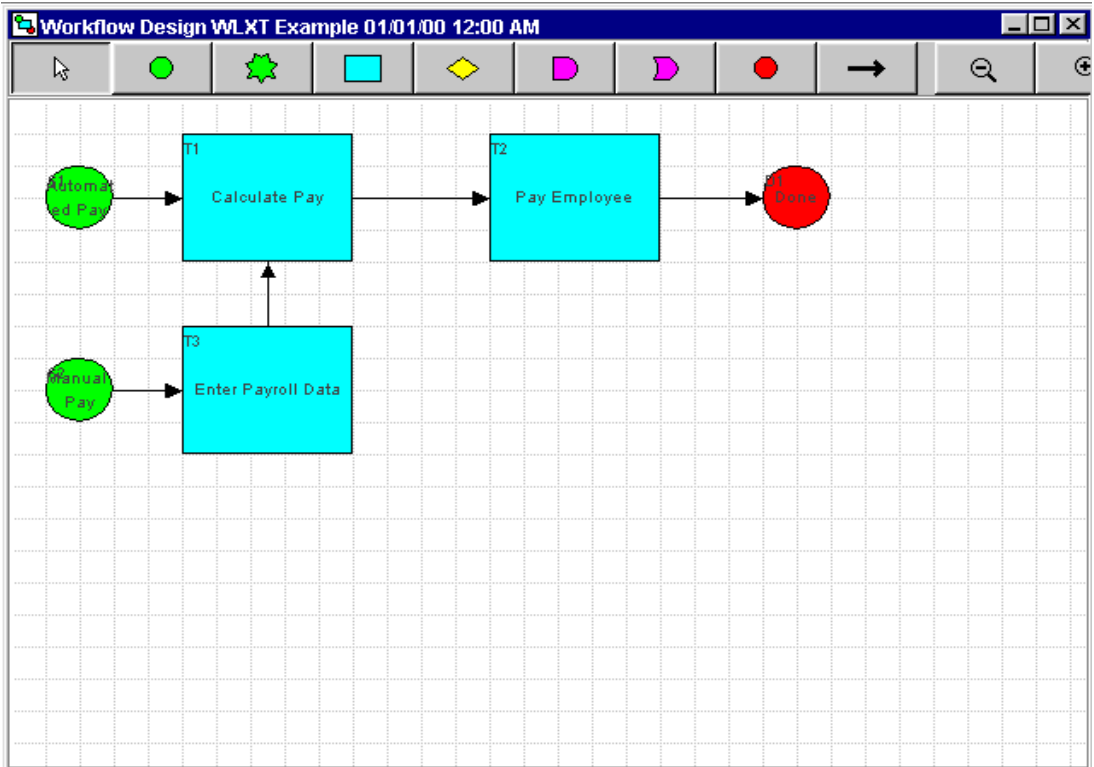
Step 4. Open and Activate the Template

1. Select the WLXT Example template definition imported in the previous step from the tree view and click the right mouse button.

2 Running the WebLogic Process Integrator Sample Applications

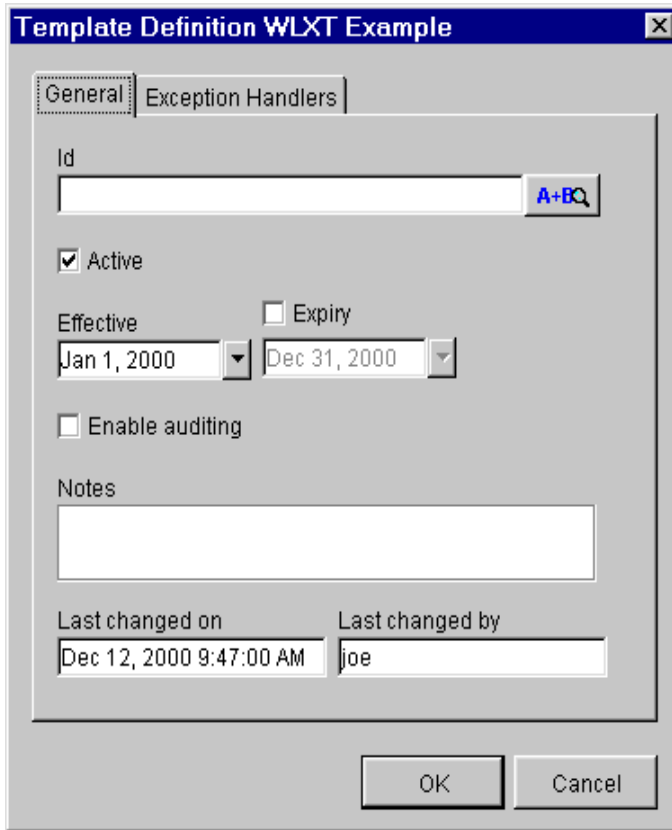
2. Choose Open. The workflow created for this sample application displays.

Figure 2-10 Workflow for WLXT Example



3. Select the `WLXTExample` template definition again from the tree view and click the right mouse button.
4. Choose Properties. The Template Definition properties dialog displays.

Figure 2-11 Template Definition

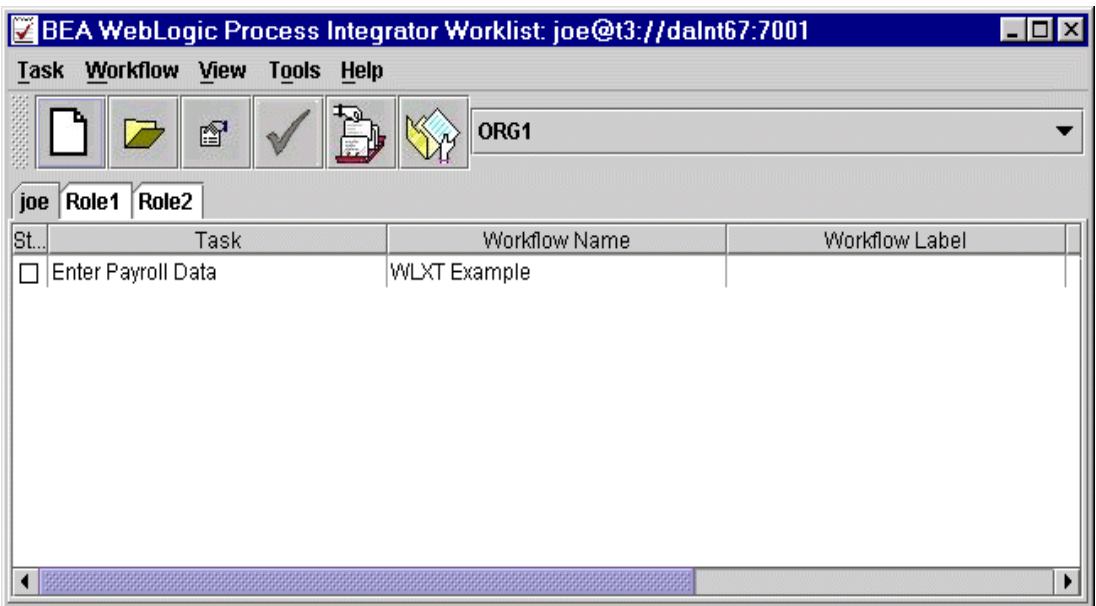


5. Click Active to activate the template and click OK.
6. Select the WLXTExample template definition a third time from the tree view and click the right mouse button again.
7. Choose Save to save the template definition with the changes you made.

Step 5. Open WebLogic Process Integrator Worklist and Start the Workflow

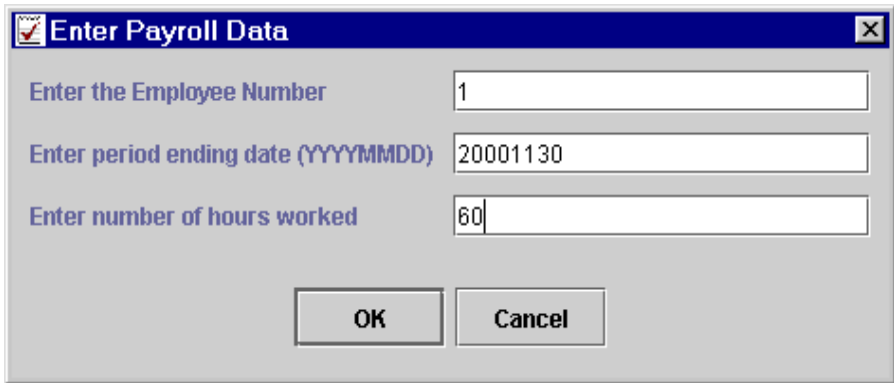
1. Start WebLogic Process Integrator Worklist and choose Workflow→Start a Workflow.
2. Select WLXT Example. The template opens and the tasks are displayed.

Figure 2-12 WLXT Example Worklist



3. Select the Enter Payroll Data task and click the right mouse button.
4. Choose Execute. The Enter Payroll Data dialog displays.

Figure 2-13 Enter Payroll Data



The screenshot shows a dialog box titled "Enter Payroll Data". It has a blue title bar with a small icon on the left and a close button (X) on the right. The dialog contains three text input fields, each with a label to its left. The first field is labeled "Enter the Employee Number" and contains the text "1". The second field is labeled "Enter period ending date (YYYYMMDD)" and contains the text "20001130". The third field is labeled "Enter number of hours worked" and contains the text "60". Below the input fields are two buttons: "OK" and "Cancel".

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.

To start the sample workflow from a command line prompt:

Run one of the following command scripts, passing the same parameters as listed in Step 5 above:

```
Autopay.cmd (NT)
Autopay.sh (Unix)
```

Figure 2-14 shows the Autopay script being run from a command prompt.

Figure 2-14 Command Prompt

```

Command Prompt - server
-----
HR getEmployeeInfo called with EmpNum 1
HR returning:
00000001456477617264732020202020      ...Edwards
2020202020202020205065746520202020      Pete
20202020202020202020202020202036313220      612
4d61696e2053742e202020202020202020      Main St.
202020202020202020202020202020202020      Purchase
202020202020202020202020202020202020      Purchase
20202020202020202020205075726368617365
202020202020202020202020202020202020      NY..?.Ah..
202020202020202020202020202020202020      4e5900003fcd4168cccd
-----
XMLnonXMLTranslator: binary2XML called with:
binaryData:
00000001456477617264732020202020      ...Edwards
202020202020202020202020202020202020      Pete
20202020202020202020202020202036313220      612
4d61696e2053742e202020202020202020      Main St.
202020202020202020202020202020202020      Purchase
20202020202020202020205075726368617365      Purchase
202020202020202020202020202020202020      NY..?.Ah..
202020202020202020202020202020202020      4e5900003fcd4168cccd
MFL File:
HR.mfl
XMLnonXMLTranslator: Returning data from binary2XML:
<?xml version="1.0"?><EmployeeRecord><EmpNum>1</EmpNum><LastName>Edwards</LastNa
me><FirstName>Pete</FirstName><Address>>612 Main St.</Address><City>Purchase</Cit
y><State>NY</State><Zip>16333</Zip><HourlyWage>14.55</HourlyWage></EmployeeRecor
d>
-----
XMLnonXMLTranslator: xml2Binary called with:
xmlData:
<?xml version="1.0" encoding="UTF-8"?>
<PayData>      <EmpNum>1</EmpNum>      <PeriodEnding>20001130</PeriodEnding>      <
HoursWorked>60</HoursWorked>      <GrossEarnings>873.0</GrossEarnings>      </PayDat
a>
MFL File: Payroll.mfl
XMLnonXMLTranslator: Returning data from xml2Binary:
0000000132303030313133300000003c      ...20001130...<
445a4000      DZE.
-----
Payroll postPayrollData called:
EmpNum: 1
PeriodEnding: 20001130
HoursWorked: 60
GrossEarnings: 873.0

```