BEA WebLogic Integration™

Translating Data with WebLogic Integration

Release 7.0
Document Date: June 2002

Copyright

Restricted Rights Legend

Trademarks or Service Marks

Translating Data with WebLogic Integration

Part Number	Date	Software Version
N/A	June 2002	7.0

Contents

About This Document

What You Need to Know ix
e-docs Web Site x
How to Print the Document x
Related Information x
Contact Us! xi
Documentation Conventions xi

1. Introduction to Data Translation

Understanding XML Translation 1-1
What Is Data Integration? 1-3
Design-Time Component 1-4
Run-Time Component 1-5
Binary-to-XML Translation 1-7
XML-to-Binary Translation 1-7
Plug-In to Business Process Management 1-8
Post-Translation Options and Considerations 1-10

2. Testing Format Definitions

Starting the Format Tester 2-1
Using the Format Tester Dialog Box 2-3
Using the Binary Window 2-3
Using the Data Offset Feature 2-4
Using the Text Feature 2-4
Using the XML Window 2-4
Using the Debug Window 2-5
Using the Resize Bars 2-5

Using the Menu Bar 2-6
File Menu 2-6
Edit Menu 2-7
Display Menu 2-8
Generate Menu 2-9
Translate Menu 2-9
Using the Shortcut Menus 2-10
Testing Format Definitions 2-11
Debugging Format Definitions 2-13
Searching for Values 2-13
Positioning to an Offset 2-14
Using the Debug Log 2-15

3. Building Format Definitions

Understanding Data Formats 3-1
Binary (NonXML) Data 3-2
XML Documents 3-3
DTDs and XML Schemas 3-4
MFL Documents 3-6
Analyzing the Data to Be Translated 3-8
Using the Format Builder 3-9
Starting Format Builder 3-9
Using the Format Builder Window 3-10
Using the Navigation Tree 3-11
Using the Format Builder Menu Bar 3-13
Using the Toolbar 3-14
Using the Shortcut Menus 3-16
Using Drag and Drop 3-17
Creating Message Formats 3-18
XML Element Naming Conventions 3-19
Creating Groups 3-20
Specifying Delimiters 3-23
Creating Fields 3-24
Padding Mandatory Fields 3-31
Creating Comments 3-31

Creating References 3-32
Working with the Palette 3-34
Opening the Palette 3-35
Using the Palette File Menu 3-36
Using the Palette Shortcut Menu 3-36
Copying Items From the Active Message Format to the Palette 3-37
Deleting Items From the Palette 3-38
Copying Palette Items from the Palette to the Active Message Format ... 3-38
Saving or Storing a Message Format 3-38
Opening or Retrieving an Existing Message Format File 3-39
Using Internationalization Features 3-40
Changing Options for a Message Format 3-40
Setting Format Builder Options 3-41
Format Builder Menus 3-43
File Menu 3-44
Edit Menu 3-44
Insert Menu 3-46
View Menu 3-46
Repository Menu 3-47
Tools Menu 3-47
Help Menu 3-48

4. Importing Metadata

Importing a COBOL Copybook 4-1
Importing C Structures 4-3
Sample C Struct Importer Files 4-4
Starting the C Struct Importer 4-5
Understanding Hardware Profiles 4-8
Building the Hardware Profile Utility 4-9
Running the Hardware Profile Utility 4-9
Generating MFL 4-9
Generating C Code 4-12
Importing an FML Field Table Class 4-13
FML Field Table Class Importer Prerequisites 4-13

Sample FML Field Table Class Files 4-14
Creating XML with the FML Field Table Class Importer 4-15

5. Retrieving and Storing Repository Documents

Logging Into the Repository 5-2
Repository Menu Commands 5-3
Retrieving MFL Documents from the Repository 5-4
Storing MFL Documents in the Repository 5-6
Importing Documents into the Repository 5-6
Using the Repository Retrieve and Store Dialog Boxes 5-9
Using the Shortcut Menu 5-11

6. Using the Run-Time Component

Binary to XML 6-2
Generating XML with a Reference to a DTD 6-3
Specifying a Debug Writer 6-4
XML-to-Binary Conversion 6-6
Converting a Document Object to Binary Format 6-7
Specifying a Debug Writer 6-8
XML-to-XML Transformation 6-9
Initialization Methods 6-10
Java API Documentation 6-13
Run-Time Plug-In to Business Process Management 6-13

A. Supported Data Types

MFL Data Types A-1
COBOL Copybook Importer Data Types A-8
C Structure Importer from Importing Metadata A-11

B. Creating Custom Data Types

Samples of User-Defined Types B-2
Registering User-Defined Types B-3
Creating User-Defined Types B-6
Configuring User-Defined Types for the Data Integration Plug-In B-7
Publishing User-Defined Types to the Repository from the Format Builder .. B-8

Publishing User-Defined Types to the Repository Using the Repository Import Utility B-10
User-Defined Type Coding Requirements B-11
Class com.bea.wlxt.bintype.Bintype B-11
Required Interface Routines B-11
Optional Interface Routines B-12
Utility Interface Routines B-14
Class com.bea.wlxt.bintype.BintypeString B-15
Required Interface Routines B-15
Optional Interface Routines B-16
Utility Interface Routines B-16
Class com.bea.wlxt.bintype.BintypeDate B-16
Required Interface Routines B-17
Optional Interface Routines B-17
Utility Interface Routines B-17
Class com.bea.wlxt.mfl.MFLField B-18

C. Running the Purchase Order Sample

What Is Included in the Purchase Order Sample C-2
Prerequisite Considerations C-2
Performing Binary-to-XML Translation C-3
Analyzing the Data to Be Translated C-3
Creating the Format Definition and Testing the Translation C-9
Step 1. Start the Format Builder and Create a Message Format C-9
Step 2. Create the Basic Information Fields C-9
Step 3. Create the Shipping Address and Billing Address Groups C-12
Step 4. Create the Remaining Items C-14
Step 5. Save the Completed Message Format C-15
Step 6. Test the Message Format C-16
Performing XML-to-Binary Translation C-17

Index

About This Document

This document describes how to use WebLogic Integration to translate data from binary format to XML, and vice versa. Specifically, it discusses the following topics:

- Chapter 1, “Introduction to Data Translation”
- Chapter 3, “Building Format Definitions”
- Chapter 2, “Testing Format Definitions”
- Chapter 4, “Importing Metadata”
- Chapter 5, “Retrieving and Storing Repository Documents”
- Chapter 6, “Using the Run-Time Component”
- Appendix A, “Supported Data Types”
- Appendix B, “Creating Custom Data Types”
- Chapter C, “Running the Purchase Order Sample”

What You Need to Know

This document is intended mainly for application programmers and technical analysts who translate data from binary to XML and vice versa.

e-docs Web Site

BEA product documentation is available on the BEA corporate Web site. From the BEA Home page, click on Product Documentation or go directly to the "e-docs" Product Documentation page at http://e-docs.bea.com.

How to Print the Document

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

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

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

Related Information

The following BEA publications are also available:

- *Using the Data Integration Plug-In*
- *Online Help for the Data Integration Plug-In*

Contact Us!

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

In your e-mail message, please indicate that you are using the documentation for WebLogic Integration Release 7.0.

If you have any questions about this release of WebLogic Integration, or if you have problems installing and running WebLogic Integration, contact BEA Customer Support through BEA WebSupport at `http://www.bea.com`. You can also contact Customer Support by using the contact information provided on the Customer Support Card, which is included in the product package.

When contacting Customer Support, be prepared to provide the following information:

- Your name, e-mail address, phone number, and fax number
- Your company name and company address
- Your machine type and authorization codes
- The name and version of the product you are using
- A description of the problem and the content of pertinent error messages

Documentation Conventions

The following documentation conventions are used throughout this document.

Convention	Item
Ctrl+Tab	Indicates that you must press two or more keys simultaneously.
italics	Indicates emphasis or book titles.

Convention	Item

`monospace text`	Indicates code samples, commands and their options, data structures and their members, data types, directories, and file names and their extensions. Monospace text also indicates text that you must enter from the keyboard.
	Examples:
	`#include <iostream.h> void main ( ) the pointer psz`
	`chmod u+w *`
	`\tux\data\ap`
	`.doc`
	`tux.doc`
	`BITMAP`
	`float`

`monospace boldface text`	Identifies significant words in code. *Example*: `void` **`commit`** `( )`
`monospace italic text`	Identifies variables in code. *Example*: `String` *`expr`*
UPPERCASE TEXT	Indicates device names, environment variables, and logical operators. *Example*s: LPT1 SIGNON OR
`{ }`	Indicates a set of choices in a syntax line. The braces themselves should never be typed.

`[ ]`	Indicates optional items in a syntax line. The brackets themselves should never be typed.
	Example:
	`buildobjclient [-v] [-o name ] [-f` *`file-list`*`]...`
	`[-l` *`file-list`*`]...`

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

Convention	Item

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

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

CHAPTER

1 Introduction to Data Translation

In most enterprise application integration (EAI) problem domains, data translation is an inherent part of an EAI solution. XML is quickly becoming the standard for exchanging information between applications; it is invaluable in integrating disparate applications. Most data transformation engines, however, do not support translations between binary data formats and XML. WebLogic Integration enables applications to exchange information by supporting data translations between the binary formats used on legacy systems and XML.

This section provides information about the following topics:

- Understanding XML Translation
- What Is Data Integration?
- Post-Translation Options and Considerations

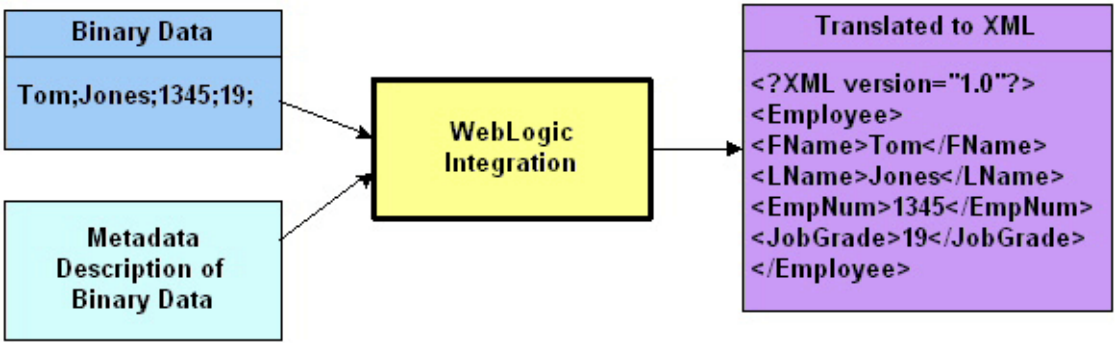
Understanding XML Translation

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

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

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

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



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

What Is Data Integration?

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

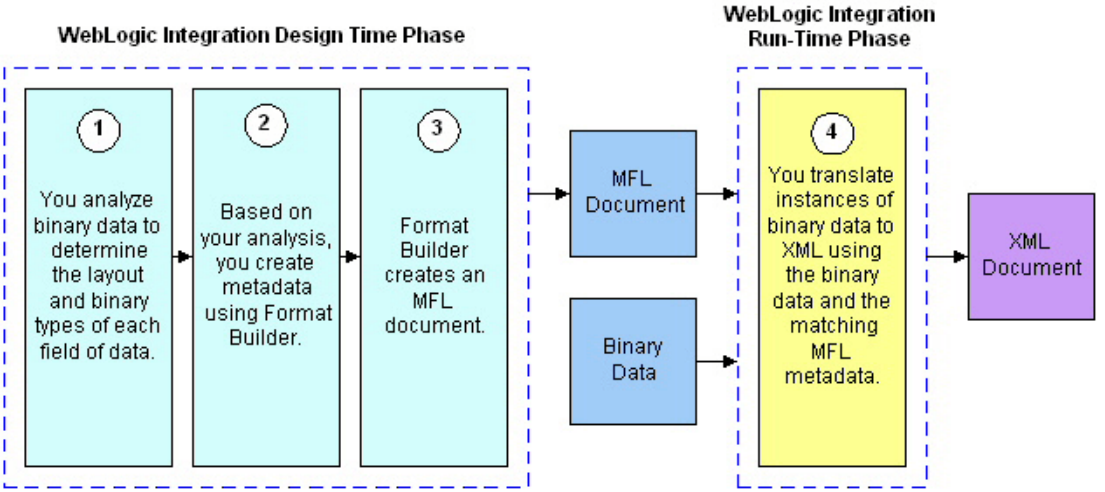
- Design-Time Component
- Run-Time Component
- Plug-In to Business Process Management

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

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

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

Figure 1-2 Event Flow for Translating Data from NonXML to XML Format



Design-Time Component

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

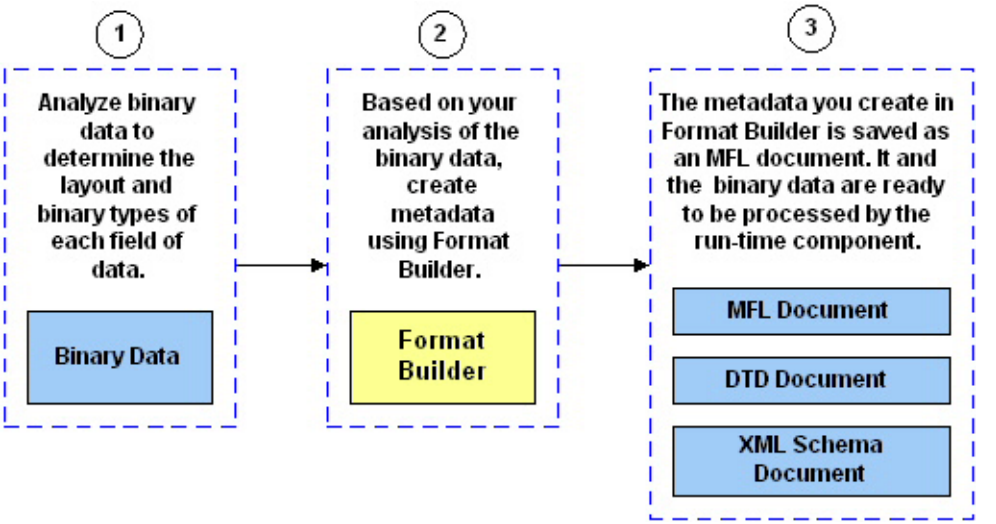
You can describe sequences of bytes as fields and specify, for each field, the type of data (floating point, string, and so on), the size of the data, and the name of the field. You can further define sets of fields (groups), multiple instances of fields and groups, and aggregation.

The description you create is saved in an XML grammar called Message Format Language (MFL). MFL documents contain metadata used by the data integration run-time component and the plug-in to business process management to translate an instance of a binary data record to an instance of an XML document (or vice versa).

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

Figure 1-4 shows the process flow of binary and XML data through Format Builder during the design-time phase.

Figure 1-3 Design-Time Process Flow Through Format Builder



You can also use Format Builder to retrieve, validate, and edit stored MFL documents and to test message format definitions with your own data. MFL documents may be stored using the file system or archived in the repository.

The test feature allows you to verify the MFL documents created in Format Builder by translating a sample XML file to binary format, or translating a sample binary file to XML format. You can save the transformed data in a file for future testing.

Run-Time Component

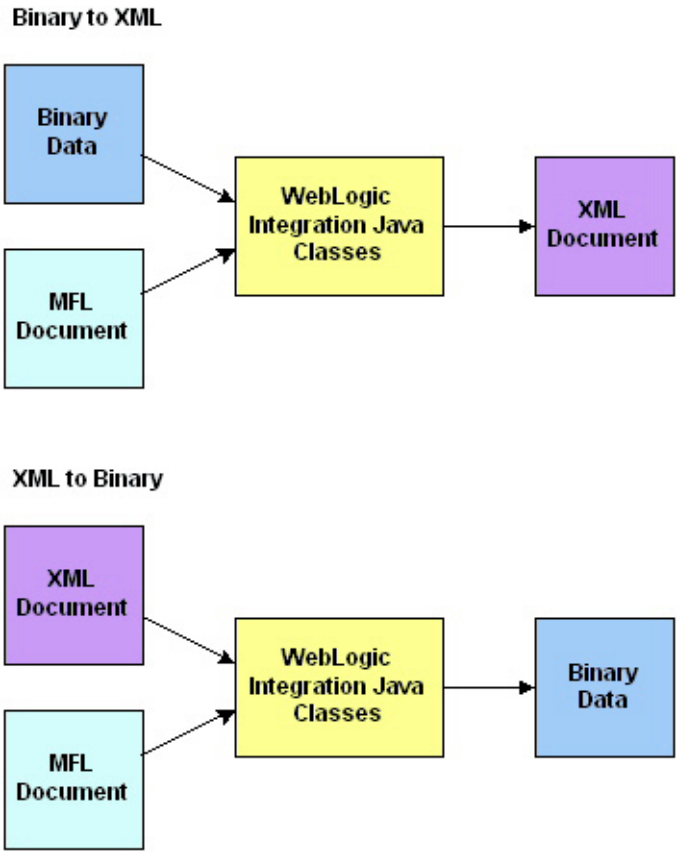
The run-time data integration component of WebLogic Integration is a Java class with various methods that are used to translate data between binary and XML formats. This Java class can be used in several ways. Specifically, it can be:

- Deployed in an EJB using WebLogic Server

- Invoked as a business operation from a workflow in the Studio
- Integrated into any Java application

Figure 1-4 shows the run-time process flow for both binary-to-XML translations and XML-to-binary translations.

Figure 1-4 Run-Time Process Flow



Binary-to-XML Translation

The sample code in Listing 1-1 shows the parsing of a file containing binary data into an XML document object. The MFL file `mymfl.mfl` provides a description of the binary data in `mybinaryfile`.

Listing 1-1 Sample Code for Binary-to-XML Translation

```
import com.bea.wlxt.*;
import org.w3.dom.Document;
import java.io.FileInputStream;
import java.net.URL;

try
{
  WLXT wlxt = new WLXT();
  URL mflDocumentName = new URL("file:mymfl.mfl");
  FileInputStream in = new FileInputStream("mybinaryfile");

  Document doc = wlxt.parse(mflDocumentName, in, null);
}
catch (Exception e)
{
  e.printStackTrace(System.err);
}
```

XML-to-Binary Translation

The sample code in Listing 1-2 shows the translation of the XML data in `myxml.xml` to the binary format specified by the MFL document `mymfl.mfl`. The binary data is written to the file `mybinaryfile`.

Listing 1-2 Sample Code for XML-to-Binary Translation

```
import com.bea.wlxt.*;
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.net.URL;
```

```
try
{
  WLXT wlxt = new WLXT();
  URL mflDocumentName = new URL("file:mymfl.mfl");
  FileInputStream in = new FileInputStream("myxml.xml");
  FileOutputStream out = new FileOutputStream("mybinaryfile");
  wlxt.serialize(mflDocumentName, in, out, null);
}
catch (Exception e)
{
  e.printStackTrace(System.err);
}
```

Plug-In to Business Process Management

The business process management (BPM) tools provided by WebLogic Integration enable you to automate workflows, business-to-business processes, and enterprise application assembly. Designed to run on WebLogic Server, the BPM tools provide a robust J2EE standards-based workflow and process integration solution.

Using an intuitive flowchart paradigm, business analysts use the WebLogic Integration process engine, underlying the BPM tools, to define business processes that span applications or to automate human interaction with applications. Developers can use the process engine to assemble application components and then to execute and manage them, without having to write new code.

The process engine has an extensible architecture that allows new functionality to be plugged in. WebLogic Integration includes a data integration plug-in that allows you to access translation capabilities (both XML-to-binary and binary-to-XML) through a BPM action.

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

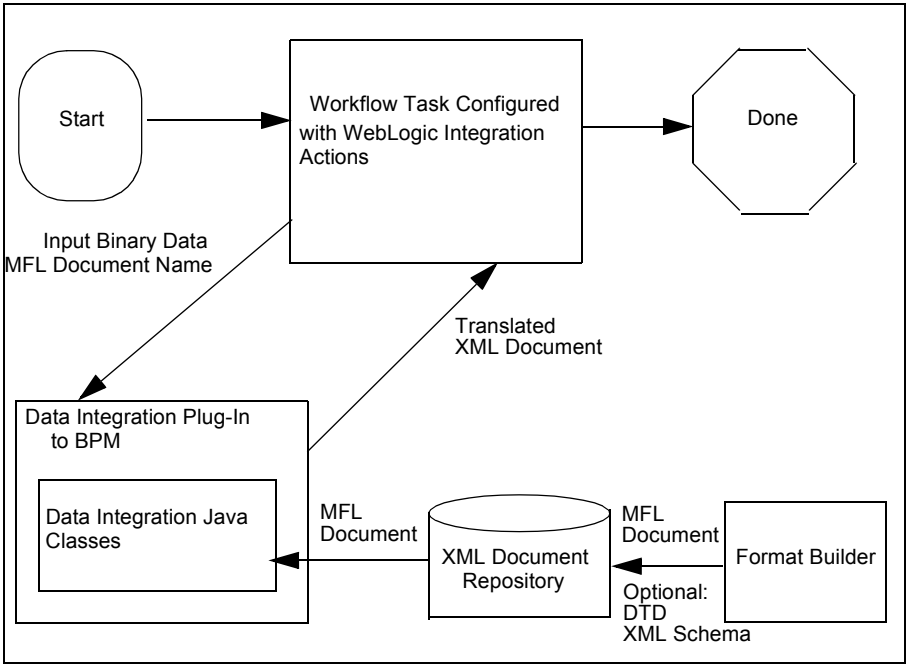
In addition to this data translation capability, the data integration plug-in provides:

- Event data processing in binary format

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

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

Figure 1-5 Data Integration and BPM Relationship



Post-Translation Options and Considerations

After you have successfully translated your binary data to XML, the XML data can be sent to other applications that consume XML, such as the business process management (BPM) client applications (the Studio and the Worklist). Depending on the requirements of the target application, the XML generated may require further transformation; this time, to a different XML grammar, to a display format (HTML), or to another binary format. In this document we refer to the process of transforming an XML document from one XML grammar to another as *XML transformation*.

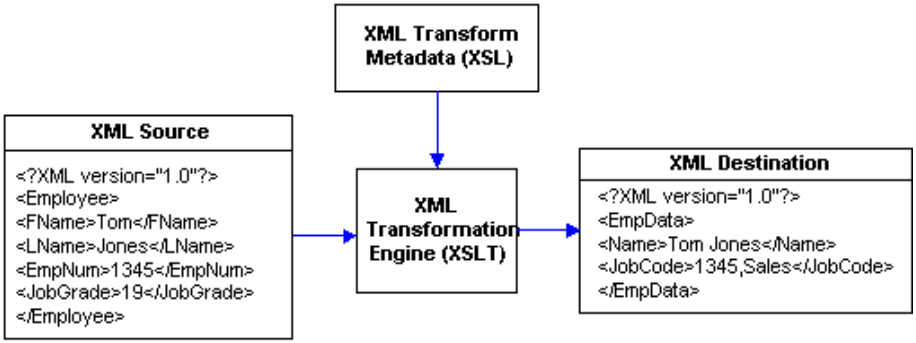
XML transformation can be accomplished via the XML module of WebLogic Server. The BPM functions include an action that allows you to access this module and transform XML documents using XSL stylesheets. You might want to transform XML to any one of several formats:

- A specific XML dialect (such as RosettaNet or ebXML)
- A display format (HTML)
- A format that matches that of another MFL document and can be converted to a different binary format by the data integration component of WebLogic Integration

The eXtensible Stylesheet Language (XSL) is an XML language that describes a series of transformations that can be performed on nodes of an XML document. An XSL stylesheet is an XSL document that can be used by an XSL Transformation (XSLT) engine to map an XML document to another XML dialect or to another text format (such as HTML or PDF). A stylesheet can also be used with the data translation run-time component of WebLogic Integration to transform XML.

Figure 1-6 shows how one XML grammar is converted to another, using an XSLT engine. The transformation metadata in this case is an XSL stylesheet that describes how one XML grammar is mapped onto another.

Figure 1-6 XML Data Transformation of: Tom;Jones,1345;19



2 Testing Format Definitions

After you build a format definition, you can test it using the Format Tester. The Format Tester parses and reformats data as a validation test and then generates sample binary or XML data. This sample data can be edited, searched, and debugged to produce the expected results. Format Tester uses the data translation run-time engine to perform the test translation.

This section discusses the following topics:

- Starting the Format Tester
- Using the Format Tester Dialog Box
- Testing Format Definitions
- Debugging Format Definitions

Starting the Format Tester

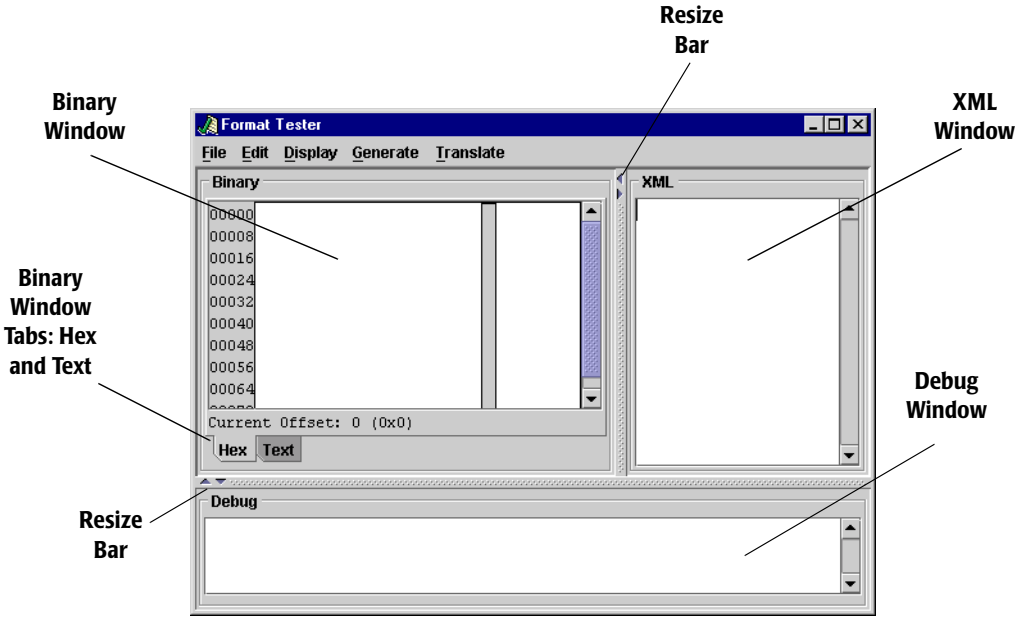
To start Format Tester:

1. If Format Builder is not already running, choose Start→Programs→BEA WebLogic Platform 7.0→WebLogic Integration 7.0→Format Builder to start Format Builder, and then open the MFL document to be tested.

   Note: Before you can run Format Tester, you must have a message format document open in Format Builder.

2. Choose Tools—Test to display the Format Tester dialog box as shown in the following figure.

Figure 2-1 Format Tester Dialog Box



The Format Tester dialog box is divided into three windows: the Binary window, the XML window, and the Debug window. Resize bars divide the windows. You can drag the resize bar to adjust window size, or click an arrow on the bar to show or hide a window. For example, you can click the left arrow on the bar dividing the Binary and XML windows to hide the Binary window. If a window is hidden, you can drag the bar or click the appropriate arrow to restore the window.

Note: When you open the Format Tester for the first time in a session, only the Binary and XML windows are visible. To open the Debug window, use the resize bar at the bottom of the Format Test dialog box, or choose Display—Debug to toggle the Debug window on and off.

Using the Format Tester Dialog Box

The following topics explain how to use various tools provided in the Format Tester dialog box to navigate and execute commands:

- Using the Binary Window
- Using the XML Window
- Using the Debug Window
- Using the Resize Bars
- Using the Menu Bar
- Using the Shortcut Menus

The following topics explain how to use each of these features to help you accomplish your task.

Using the Binary Window

The Binary window can contain sample data that has been:

- Generated based on the active MFL document.
- Translated from the contents of the XML window.
- Specifically designed to test the active MFL document.

You can open an existing binary data file, edit or save the contents of the window, or clear the window as required for your test situation. For details, see "Using the Menu Bar" on page 2-6 and "Using the Shortcut Menus" on page 2-10.

The Binary window of the Format Tester dialog box serves as a binary file editor. The window contains the following tabs:

- Hex—displays data offsets, the hex value of individual bytes, and the corresponding text, which can be displayed in either ASCII or EBCDIC format.
- Text—Text only display.

The editor allows you to edit a hex byte or a text value. If a hex data value is modified, the corresponding text value is updated, and vice versa.

Using the Data Offset Feature

The data offset feature of the Hex tab allows you to display the data offsets as hexadecimal or decimal addresses.

To change the format of the data offsets:

1. Choose Display→Hex. The following two data offset options are displayed.
   - Offsets as Hexadecimal
   - Offsets as Decimal
2. Select an option that best suits your needs. The data offset portion of the Binary window changes dynamically to reflect your choice.

Using the Text Feature

The Text tab of the Binary window displays the printable characters (usually in the form of words and numbers) and certain control characters (carriage return, tab, and so on). For example, carriage returns are shown as line breaks. Non-printable characters, are displayed as small squares.

Using the XML Window

The XML window can contain sample XML that has been:

- Generated based on the active MFL document.
- Translated from the contents of the Binary window.
- Specifically designed to test the active MFL document.

You can open an existing XML file, edit or save the contents of the window, or clear the window as required for your test situation. For details, see "Using the Menu Bar" on page 2-6 and "Using the Shortcut Menus" on page 2-10.

When XML is generated, the XML Formatting Options specified in the Format Builder options dialog box are used. For additional information, see "Setting Format Builder Options" on page 3-41.

Using the Debug Window

The Debug window displays the actions that occur during a translation operation, any errors that are encountered, and field and group values, along with delimiters. To determine the cause of an error, identify the last field that parsed successfully and examine the properties of the field listed after it in the navigation tree.

When you open the Format Tester for the first time in a session, only the Binary and XML windows are visible. To open the Debug window, choose Display—Debug to toggle the Debug window on and off. The Debug window opens below the Binary and XML windows.

Debug output is restricted to the most recent 64 KB of messages. This restriction prevents large debug output from causing a JVM out of memory event.

The debug log feature allows you to save all debugging information in a file. For details, see "Using the Debug Log" on page 2-15.

Note: Use of the Debug window or log file increases the time required to translate from XML to Binary.

Using the Resize Bars

You can change the dimensions of any window in the Format Tester by using the resize bars located between the Binary, XML, and Debug windows. To change the size of a window, select a resize bar and drag in the appropriate direction (up or down or to the left or right) to enlarge one of the windows and reduce the other.

Each resize bar also contains two directional buttons. Click the appropriate button to show or hide any of the three windows.

Using the Menu Bar

Format Tester functions can be accessed from the five menus listed in the menu bar at the top of the main window.

Figure 2-2 Menu Bar

You can expand a Format Tester menu in either of two ways:

- Click the name of the menu in the menu bar.
- On your keyboard, press Alt + *key*, where *key* is the underlined letter in the menu name.

To execute a command, select it from the menu. Some commands can also be executed via the keyboard shortcut indicated on the menu (For example, a Ctrl + *key* sequence.) The commands available on each menu are described in the following sections.

File Menu

The following commands are available from the File menu.

Table 2-1 File Menu Commands

Command	Description
Open Binary	Displays the Open dialog box to allow you to select and open a file in the Binary window. **Note:** The default file extension for binary files is `.DATA`.
Open XML	Displays the Open dialog box to allow you to select and open a file in the XML window. **Note:** The default file extension for XML files is `.XML`.
Save Binary	Displays the Save dialog box to allow you to save the contents of the Binary window.

Table 2-1 File Menu Commands (Continued)

Command	Description
Save XML	Displays the Save dialog box to allow you to save the contents of the XML window.
Debug Log	Displays the Save dialog box to allow you to save the debugging information in a text file.
Close	Closes the Format Tester window.

Edit Menu

The following commands are available from the Edit menu.

Table 2-2 Edit Menu Commands

Command	Description
Cut	Removes the currently selected text and places it on the clipboard for pasting in another location.
Copy	Copies the currently selected text and places it on the clipboard for pasting in another location.
Paste	Inserts the cut or copied text at the cursor location.
Find	Allows you to search for a hex or text value. This command applies to the content of the Binary window only. **Note:** The text search is case sensitive.
Find Next	Repeats the last Find from the current cursor position. This command applies to the content of the Binary window only.
Go To	Allows you to move the cursor to a specified byte offset in the Binary window.

Display Menu

The following commands are available from the Display menu.

Table 2-3 Display Menu Commands

Command	Description
XML checkbox	Check to display the XML window, uncheck to hide the window. When unchecked, the Binary window expands to fill the Format Tester dialog box.
Debug checkbox	Check to display the Debug window, uncheck to hide the window.
Clear—Binary	Clears the contents of the Binary window.
Clear—XML	Clears the contents of the XML window.
Clear—Debug	Clears the contents of the Debug window.
Hex—Offsets as Hexadecimal option button	Displays the offset values as hexadecimal. Mutually exclusive with the Hex—Offsets as Decimal option.
Hex—Offsets as Decimal option button	Displays the offset values as decimal. Mutually exclusive with the Hex—Offsets as Hexadecimal option.
Text—Values in ASCII option button	Changes the character set used for the text displayed in the binary file editor to ASCII. Mutually exclusive with the Text—Values in EBCDIC option.
Text—Values in EBCDIC option button	Changes the character set used for the text displayed in the binary file editor to EBCDIC. Mutually exclusive with the Text—Values in ASCII option.

Generate Menu

The following commands are available from the Generate menu.

Table 2-4 Generate Menu Commands

Command	Description
Binary	Generates binary data to match the MFL document specification.
XML	Generates XML data to match the MFL document specification.
Prompt while generating data check box	If checked, you are prompted to specify the following during the generation process: ■ Whether or not to include optional fields or groups in the output. ■ Which choice of children to include in the output. ■ The number of times to include repeating fields or groups in the output.

Translate Menu

The following commands are available from the Translate menu.

Table 2-5 Translate Menu Commands

Command	Description
Binary to XML	Based on the MFL document specification, converts the contents of the Binary window to XML. The XML output is displayed in the XML window.
XML to Binary	Based on the MFL document specification, converts the contents of the XML window to binary. The binary output is displayed in the Binary window.

Using the Shortcut Menus

When you right-click in the Binary, XML, or Debug window, a menu of the most frequently used commands for that window is displayed. The following table describes the commands that are available from the shortcut menus.

Table 2-6 Binary, XML, and Debug Shortcut Menu Commands

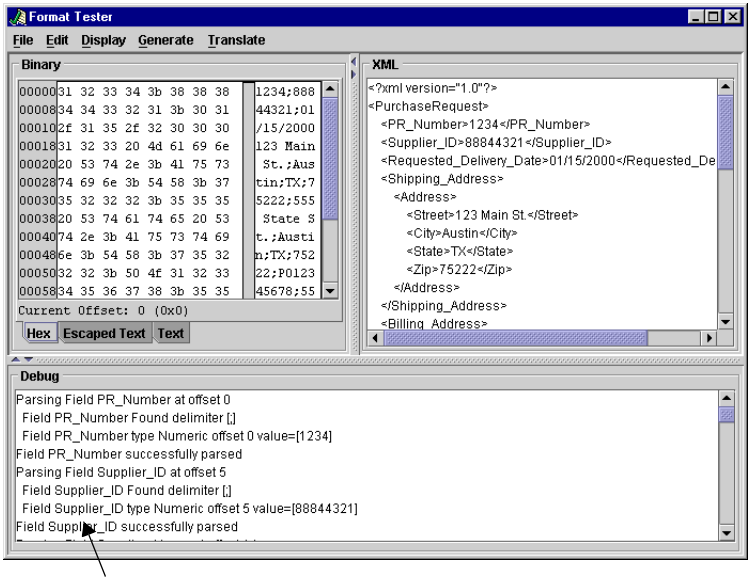
Command	Description
Cut	Removes the currently selected text and places it on the clipboard for pasting in another location.
Copy	Copies the currently selected text and places it in the clipboard for pasting in another location.
Paste	Inserts the cut or copied text at the cursor location.
Clear	Clears the contents of the binary, XML, or Debug window.
Generate	Generates binary or XML data to match the MFL document specification. This command is only available on the Binary and XML shortcut menus.
To XML	Converts the contents of the Binary window to XML. This command is only available on the Binary shortcut menu.
To Binary	Converts the contents of the XML window to binary. This command is only available on the XML shortcut menu.
Text in ASCII	Changes the character set used for the text displayed in the text portion of the Hex tab to ASCII.
Text in EBCDIC	Changes the character set used for the text displayed in the text portion of the Hex tab to EBCDIC.

Testing Format Definitions

To test a message format definition:

1. Start Format Builder.
2. Open a Message Format file.
3. Start Format Tester.
4. Choose File→Open Binary, or File→Open XML to load the file you want to translate and view, or enter your own data in one of the two data windows.
5. Choose Display→Debug if you want to view the actions that take place during the translation operation. This step is optional. If you want to be able to view debugging information later, you must open the Debug window before starting the translation operation.
6. Choose Translate→Binary to XML, or Translate→XML to Binary to translate your data to the appropriate format. The translated data is displayed in the Binary or XML window.

Figure 2-3 Format Tester



7. Correct any errors and test the translation again.
8. Repeat steps 6 and 7 until the translation is successful.

Note: You can leave Format Tester open while you modify the message format document in Format Builder. Changes to the document are detected automatically by Format Tester.

Debugging Format Definitions

The following topics explain how to use three Format Tester features to debug and correct your data:

- Searching for Values
- Positioning to an Offset
- Using the Debug Log

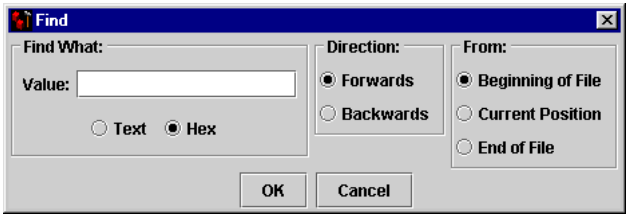
Searching for Values

The Find command allows you to search for hex or text values in the binary data.

To search for a hex or text value:

1. In the Format Tester dialog box, choose File→Open Binary to open the binary data file you want to search.
2. Choose Edit→Find to open the Find dialog box.

Figure 2-4 Find Dialog Box



3. Enter the target of the search in the Value field.
4. Select the Text or Hex option button to specify the value type.
5. Select the Forwards or Backwards option button to specify the search direction.
6. Select the Beginning of File, Current Position, or End of File option button to specify the search starting position.

7. Click OK to dismiss the Find dialog box and execute the specified search.

   If the value is found, the cursor moves the location of the value. If the value is not found, the following message is displayed: The specified search string was not found.

8. To repeat the search from the current cursor position, choose Edit→Find Next.

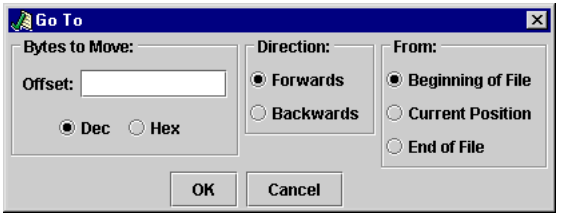
Positioning to an Offset

The Go To command allows you to move the cursor to a specified hexidecimal or decimal address (offset).

To move to a specified offset:

1. In the Format Tester dialog box, choose Edit→Go To to display the Go To dialog box.

Figure 2-5 Go To Dialog Box



2. Enter the target offset in the Offset field.
3. Select the Dec or Hex option button to specify the type of offset.
4. Select the Forwards or Backwards option button to specify the direction.
5. Select the Beginning of File, Current Position, or End of File option button to specify the starting position.
6. Click OK to dismiss the dialog box and move the cursor to the specified offset.

Using the Debug Log

Although debugging information is not saved by default, the Format Tester dialog box allows you to specify a debug log file. When you specify a debug log file, all debugging information generated during your testing session is appended to the specified file.

To specify a debug log file:

1. In the Format Tester dialog box, choose the File→Debug Log to display the Save dialog box.

   Note: The Debug Log checkbox on the File menu is toggled upon selection. If the checkbox is checked, choosing File→Debug Log turns off logging to the file.

2. Select the desired directory, and then do one of the following:

   - To create a new log file, enter the name in the File name field and then click Save.
   - To use an existing log file, select the file and then click Save.

   If you select an existing file, the new debug information is appended to the end of the existing file.

3 Building Format Definitions

Format definitions are the metadata used to parse or create binary data. WebLogic Integration helps you build format definitions for binary data that is to be translated to or from XML.

This section provides information about building format definitions using the Format Builder, WebLogic Integration's design-time component for integrating data. It includes the following topics:

- Understanding Data Formats
- Analyzing the Data to Be Translated
- Using the Format Builder

Understanding Data Formats

To understand how to use the Format Builder, it helps to understand the following format and document types:

- Binary (NonXML) Data
- XML Documents
- DTDs and XML Schemas
- MFL Documents

Binary (NonXML) Data

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

Unlike XML data, binary data is not self-describing. In other words, binary data does not include a description of how the data is grouped, divided into fields, or otherwise arranged. 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 that sequence.

For example, consider the following binary data string:

```
2231987
```

You can interpreted it in many different ways. For example:

- As a date: 2/23/1987
- As a phone number (223-1987)

Without a clear understanding of the purpose of this data string, the application cannot interpret the string appropriately.

In order for binary data to be understood by an application, the layout of the data must be embedded in the application itself. The character set used to encode the character data included in a binary file may also vary. For example, character data on an IBM mainframe is usually encoded using the EBCDIC character set, while data from a desktop computer is either ASCII or unicode.

You can use Format Builder to create a Message Format Language (MFL) file that describes the layout of your binary data. MFL is an XML language that includes elements for describing 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 are expressed in an MFL document. This MFL document is used at run time to translate data to and from an XML document.

Listing 3-1 Example of Binary 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;
```

XML Documents

The eXtensible Markup Language (XML) is fast becoming the universal format for structured documents and data on the Web. Unlike binary data, XML is self-describing; it makes use of *tags* (words bracketed by '<' and '>') that signal the start and end of each block of data. These tags define the hierarchy of related data components which constitute the *elements* in a structured document.

The properties of XML make it suitable for representing and structuring data in a platform-neutral manner. By making the structure explicit, XML can simplify the task of exchanging data between applications. Because the data is presented in a standard form, applications on disparate systems can interpret it using XML parsing tools, instead of having to interpret data in proprietary binary formats.

Listing 3-2 shows an example of an XML document.

Listing 3-2 Example of XML Document

```
<?xml version="1.0"?>
<PurchaseRequest>
 <PR_Number>1234</PR_Number>
 <Supplier_ID>88844321</Supplier_ID>
  <Supplier_Name>Sprockley&apos;s Sprockets</Supplier_Name>
  <Requested_Delivery_Date>2000-01-15T00:00:00:000</Requested_Delivery_Date>
  <Shipping_Address>
    <Address>
      <Street>123 Main St.</Street>
      <City>Austin</City>
      <State>TX</State>
      <Zip>75222</Zip>
    </Address>
  </Shipping_Address>
  </PurchaseRequest>
```

DTDs and XML Schemas

The original XML recommendation only defined one way to describe the elements, attributes, and data types allowed in an XML document instance: the XML Document Type Definition (DTD). Subsequently, it became apparent that a more flexible and powerful way to describe the content model was required, and work began on the XML Schema definition language, which became available as a final recommendation in May of 2001.

An XML document is said to be *valid* if it conforms to the content model described in its associated DTD or XML Schema. Although the metadata required by an XML parser to validate an XML document can be conveyed in either a DTD or XML Schema, an XML Schema definition is more specific than a DTD; it provides much finer grained control over the content than a DTD.

Listing 3-3 shows an example of a Document Type Definition, or DTD.

Listing 3-3 Example DTD

```
<!ELEMENT PurchaseRequest
  (PR_Number,Supplier_ID,Supplier_Name?,
  Requested_Delivery_Date,Shipping_Address,
  Billing_Address,Payment_Terms,Purchase_Items)>
<!ELEMENT PR_Number (#PCDATA) >
<!ATTLIST PR_Number type CDATA #FIXED "nonNegativeInteger">
<!ELEMENT Supplier_ID (#PCDATA) >
<!ATTLIST Supplier_ID type CDATA #FIXED "nonNegativeInteger">
<!ELEMENT Supplier_Name (#PCDATA) >
<!ATTLIST Supplier_Name type CDATA #FIXED "string">
<!ELEMENT Requested_Delivery_Date (#PCDATA) >
<!ATTLIST Requested_Delivery_Date type CDATA #FIXED "timeInstant">
<!ELEMENT Shipping_Address (Address)>
<!ELEMENT Address (Street,City,State,Zip)>
<!ELEMENT Street (#PCDATA) >
<!ATTLIST Street type CDATA #FIXED "string">
<!ELEMENT City (#PCDATA) >
<!ATTLIST City type CDATA #FIXED "string">
<!ELEMENT State (#PCDATA) >
<!ATTLIST State type CDATA #FIXED "string">
<!ELEMENT Zip (#PCDATA) >
<!ATTLIST Zip type CDATA #FIXED "nonNegativeInteger">
```

Listing 3-4 shows an example of an XML Schema definition.

Listing 3-4 Example XML Schema

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/1999/XMLSchema">
<xsd:annotation>
<xsd:documentation>
This schema created for MFL MessageFormat PurchaseRequest.
</xsd:documentation>
</xsd:annotation>

<xsd:element name="PurchaseRequest">
<xsd:complexType content="elementOnly">
<xsd:sequence>
<xsd:element ref="PR_Number" minOccurs="1" maxOccurs="1"/>
<xsd:element ref="Supplier_ID" minOccurs="1" maxOccurs="1"/>
<xsd:element ref="Supplier_Name" minOccurs="0" maxOccurs="1"/>
```

```
<xsd:element ref="Requested_Delivery_Date" minOccurs="1" maxOccurs="1"/>
<xsd:element ref="Shipping_Address" minOccurs="1" maxOccurs="1"/>
</xsd:sequence>
</xsd:complexType>
</xsd:element>

<xsd:element name="PR_Number" type="xsd:nonNegativeInteger"/>

<xsd:element name="Supplier_ID" type="xsd:nonNegativeInteger"/>

<xsd:element name="Supplier_Name" type="xsd:string"/>

<xsd:element name="Requested_Delivery_Date" type="xsd:timeInstant"/>

<xsd:element name="Shipping_Address">
<xsd:complexType content="elementOnly">
<xsd:sequence>
<xsd:element ref="Address" minOccurs="1" maxOccurs="1"/>
</xsd:sequence>
</xsd:complexType>
</xsd:element>

</xsd:schema>
```

MFL Documents

A Message Format Language (MFL) document (also known simply as a message format document) is a specialized XML document used to describe the layout of binary data. An MFL document conforms to the `mfl.dtd`, which includes elements and attributed used to describe each field of data, as well as groupings of fields (groups), repetition, and aggregation. When you use Format Builder to define the hierarchy of a binary record, the layout of fields, and the grouping of fields and groups, the information is saved as an MFL document that can then be used to perform run-time translations. The information captured in the MFL document can also be used to generate DTDs or XML Schemas that describe the content model for the output generated by the MFL document.

The top-level element of a message format document is the `MessageFormat` element, which defines the message format name and version. For example, the following is the root element of the sample `po.mfl` document installed with WebLogic Integration:

```
<MessageFormat name='PurchaseRequest' version='2.01'>
```

WebLogic Integration now supports Message Format Language Version 2.02. This version supports new features related to padding, truncation, and trimming. Message Format Language Version 2.01 is still supported.

The name assigned to the message format document becomes the root element in the XML instances that are generated based on the MFL document. For example, The following is the root element of any XML document generated based on the sample `po.mfl` document:

```
<PurchaseRequest>
```

The other elements and attributes available in an MFL document are used to define the following:

- Fields and Field Formats – A field is a sequence of bytes that is meaningful in the context of an application and that defines the format of a field. (For example, the field `EMPNAME` contains an employee name.) You can define the following formatting parameters:
  - Tagged – Indicates that a literal precedes the data field, denoting the beginning of the field.
  - Length – Indicates that a numeric value precedes the data field, denoting the length of this field.
  - Occurrence – Indicates the number of times the field is shown in the message format. You can specify the number of times the field is to be shown, or define a delimiter that indicates the end of the repeating field.
  - Optional – Indicates that the field may or may not be included in the format of the named message.
  - Code Page – Identifies the type of character encoding used for the data in the field.

- Groups and Group Formats – A group is a collection of fields, comments, and other groups or references that are related in some way (for example, the fields `PAYDATE`, `HOURS`, and `RATE` belong to the `PAYINFO` group). The parameters you can define for a group include:
  - Tagged – Means that a literal precedes the other content of the group, which may be other groups or fields.
  - Occurrence – Indicates either the number of times the group is to be repeated in the message format, or a delimiter that marks the end of the repeated group. For more information about delimiters, see "Specifying Delimiters" on page 3-23.
  - Choice of Children – Indicates that only one item in the group will appear in the message format.
  - Optional – Indicates that the data in this structure may or may not be included in the named message format.
- References and Reference Formats – A reference indicates that another instance of the field or group format exists in the data. The format of a reference field or group is the same as the format original field or group, but you can change the optional setting and the occurrence setting for the reference field or group. For example, if your data includes a *bill to* address and a *ship to* address and the same format is used for both addresses, you can create the address format once, and then reference it. That is, you can create the an address definition for the *bill to* address and reference it for the *ship to* address.
- Comments – Notes containing additional information about the message format.

Analyzing the Data to Be Translated

Before a message format can be created, the layout of the binary data must be understood. Sample data for a legacy purchase order, with corresponding MFL and XML documents for a purchase order record, are installed with ProductName. The sample purchase order illustrates how WebLogic Integration translates data from one format to another. For more information about this sample data, see Appendix C, "Running the Purchase Order Sample."

The key to translating binary data to and from XML is to create an accurate description of it. 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

Use Format Builder to incorporate these elements into the format definitions used for data translations.

Using the Format Builder

Format Builder helps you create format descriptions for binary data and store them in MFL documents. Your description should include hierarchical and structural information derived from a detailed analysis of your data. These format descriptions are stored in an MFL document. You can also use Format Builder to test your format descriptions before applying them to your data.

Starting Format Builder

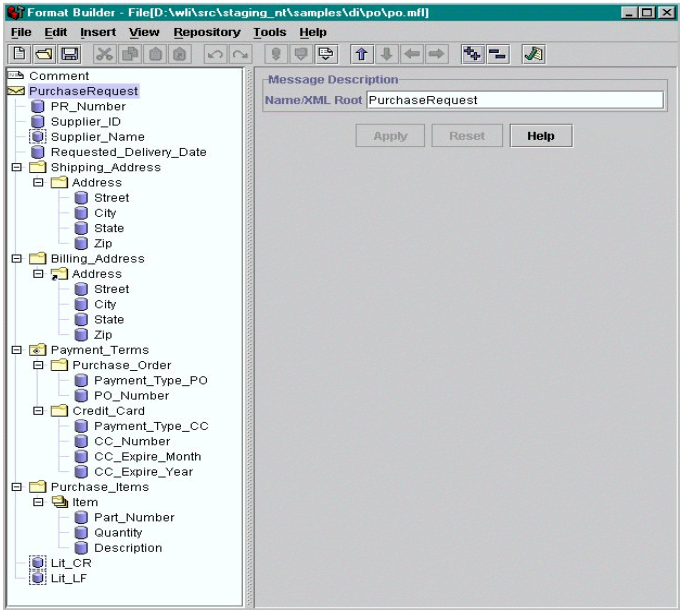
To start Format Builder, choose Start→Programs→BEA WebLogic Platform 7.0→WebLogic Integration 7.0→Format Builder. The Format Builder window is displayed.

Using the Format Builder Window

The Format Builder window is split into two vertical panes. The left pane contains the navigation tree which shows the structural relationship of the groups and fields defined in the active MFL document. The right pane displays the properties that define the item.

Information about the file you are editing is displayed in the title bar of the Format Builder window.

Figure 3-1 Format Builder Window



The structure of the binary data is defined in the navigation tree through a combination of fields and groups that match the target data.

The following topics explain how to use the various tools provided in the Format Builder window to navigate and execute commands:

- Using the Navigation Tree
- Using the Format Builder Menu Bar
- Using the Toolbar
- Using Drag and Drop
- Using the Shortcut Menus

Using the Navigation Tree

The navigation tree represents the structure of the binary data in a hierarchical layout. The root node of the navigation tree, the *Message node*, corresponds to the MFL document being created or edited. Child nodes are labeled with the names of groups or fields. Fields are represented by leaf nodes in the navigation tree. Groups contain fields or other groups and are represented by nonleaf nodes in the navigation tree.

The icon for each node encapsulates the following information about the node: whether the node represents a message, a group, a field, a comment, or a reference; whether a group or field is repeating; whether a group is a *Choice of Children;* and whether a group or field is optional or mandatory.

You can add, delete, move, copy, or rename nodes in the navigation tree though menus or the toolbar. (For details, see "Using the Format Builder Menu Bar" on page 3-13 and "Using the Toolbar" on page 3-14.)

The following table describes the icons displayed in the navigation tree.

Table 3-1 Navigation Tree Icons

Tree Icon	Icon Name	Description
	Message Format	The top-level element.
	Group	Collections of fields, comments, and other groups or references that are related in some way. (For example, the fields `PAYDATE`, `HOURS`, and `RATE` belong to the `PAYINFO` group.) Defines the formatting for all items in the group.

Table 3-1 Navigation Tree Icons (Continued)

Tree Icon	Icon Name	Description
	Optional Group	A group that may or may not be included in the message format.
	Repeating Group	A group that is included one or more times.
	Optional Repeating Group	A group that may or may not be included, but if included, may occur more than once.
	Group Reference	Indicates the existence of another instance of the group in the data. The format of a reference group is the same as that of the original group, but you can change the optional setting and the occurrence setting for the reference group.
	Group Choice	Indicates that only one of the items in the group is included in the message format.
	Field	Sequence of bytes that is meaningful in the context of the application and that defines the formatting for the field. (For example, the field `EMPNAME` contains an employee name.)
	Optional Field	A field that may or may not be included in the message format.
	Repeating Field	A field is included one or more times.
	Optional Repeating Field	A field that may or may not be included, but, if included, may occur more than once in the message format.
	Field Reference	Indicates the existence of another instance of the field in the data. The format of a reference field is the same as that of the original field, but you can change the optional setting and the occurrence setting for the reference field.

Table 3-1 Navigation Tree Icons (Continued)

Tree Icon	Icon Name	Description
	Comment	Contains notes about the message format or the data translated by the message format.
	Collapse	A minus sign next to an item indicates that the specified item can be collapsed.
	Expand	A plus sign next an item indicates that the specified item can be expanded to show child items.

Using the Format Builder Menu Bar

The menu bar provides quick access to Format Builder functions.

Figure 3-2 Format Builder Menu Bar

The items available in a menu depend on the actions you have taken and the node currently selected in the navigation tree. If a menu item is not available, it is shown in gray in the menu.

You can display a menu in either of two ways:

- Click the name of the menu in the menu bar.
- On your keyboard, press Alt + *key*, where *key* is the underlined letter in the menu name.

To execute a command, select it from the menu. Some commands can also be executed via the keyboard shortcut indicated on the menu (For example, a Ctrl + *key* sequence.) The commands available on each menu are described in "Format Builder Menus" on page 3-43.

Using the Toolbar

The toolbar is a menu of icons that provide alternative ways to access frequently used commands.

Figure 3-3 Format Builder Toolbar

To execute a command, click the appropriate icon in the toolbar. If a command is unavailable, the icon for it appears *grayed-out*.

The following table describes the icons in the Format Builder tool bar.

Table 3-2 Format Builder Toolbar Icons

Toolbar Icon	Name	Description
	New	Creates a new message format.
	Open	Opens an existing message format.
	Save	Saves the current message format.
	Cut	Removes the item currently selected in the left pane, and its child objects, from the navigation tree. The item can be pasted elsewhere in the navigation tree. **Note:** This action is not available if the message format (root) item is selected.
	Copy	Makes a copy of the item currently selected in the left pane for insertion elsewhere in the navigation tree. **Note:** This action is not available if the message format (root) item is selected.

Table 3-2 Format Builder Toolbar Icons (Continued)

Toolbar Icon	Name	Description
	Paste as Sibling	Inserts the cut or copied item as a sibling object of the selected item.
	Paste as Reference	Inserts a reference to the cut or copied item as a sibling object of the selected item.
	Undo	Reverses the previous action. The tool tip indicates the action that can be undone. For example, if you change the name of a field to Address and click Apply, the tool tip displays the following message: Undo Apply Field Address. Format Builder supports multiple undoing of previous actions.
	Redo	Reverses the effects of an Undo command. The tool tip indicates the action that can be redone. For example, if you change the name of a field to Address and then Undo that change, the Redo tool tip displays the following message: Redo Apply Field Address. Format Builder supports multiple redoing of previous actions.
	Insert Field	Inserts a field as a sibling of the item selected in the navigation tree.
	Insert Group	Inserts a group as a sibling of the item selected in the navigation tree.
	Insert Comment	Inserts a comment as a sibling of the item selected in the navigation tree.
	Move Up	Moves the selected item up one position under its parent.

Table 3-2 Format Builder Toolbar Icons (Continued)

Toolbar Icon	Name	Description
	Move Down	Moves the selected item down one position under its parent.
	Promote item	Assigns the selected item to the next highest level in the navigation tree. For example, suppose Field1 is a child object of Group1. If you select Field1 and click the Promote tool, you make Field1 a sibling of Group1.
	Demote item	Assigns the selected item to the next lower level in the navigation tree. For example, suppose Group1 is the sibling of Field1 and it is listed immediately after Group1 in the navigation tree. If you select Field1 and click the Demote tool, you make Field11 a child of Group1.
	Expand All	Expands all the items in the navigation tree to show child items.
	Collapse All	Collapses the navigation tree to show first-level items only.
	Format Tester	Opens the Format Tester window.

Using the Shortcut Menus

When you right-click an item in the navigation tree, a menu of the most frequently used commands for that item is displayed. The following table describes the commands that are available from the shortcut menus.

Note: The availability of a command depends on the item you select and the previous actions you have taken.

Command	Description
Cut	Removes the item currently selected in the left pane, and its child objects, from the navigation tree.
Copy	Makes a copy of the item currently selected in the left pane for insertion elsewhere in the navigation tree.
Paste	Inserts the cut or copied item. An additional menu is displayed when you select Paste. You can paste the item as either a child or a sibling of the selected item. In addition, you can paste a reference to the cut or copied item as a sibling of the selected item.
Insert Group	Inserts a new group as either a child or a sibling of the selected item, depending on your specification.
Insert Field	Inserts a new field as either a child or a sibling of the selected item, depending on your specification.
Insert Comment	Inserts a comment as either a child or a sibling of the selected item, depending on your specification.
Duplicate	Makes a copy of the currently selected item and pastes it as a sibling. The duplicate item contains the same values and child objects as the original. The name of the duplicate is the same as that of the original, with the addition of a prefix: *New*. Thus, for example, if the name of the original item is MyGroup1, then the name of the duplicate is NewMyGroup1.
Delete	Deletes the selected item.

Using Drag and Drop

You can drag and drop to copy and paste, or move items in the navigation tree.

Note: The node being copied or moved is always inserted as a sibling of the selected node during the drag-and-drop process. If you drag and drop the node onto the Message Format node, it is inserted as the last child.

To move an item:

1. Select the item you want to move.
2. Press and hold the left mouse button while you drag the item to the desired node.

3. When the item is in the desired location, release the left mouse button. The item is moved to the new location.

To copy and paste an item:

1. Select the item you want to copy.
2. Press and hold the Ctrl key.
3. Keeping the Ctrl key depressed, press and hold the left mouse button while you drag the item to the desired node.
4. With the sibling object selected, release the left mouse button. A copy of the item is pasted the new location.

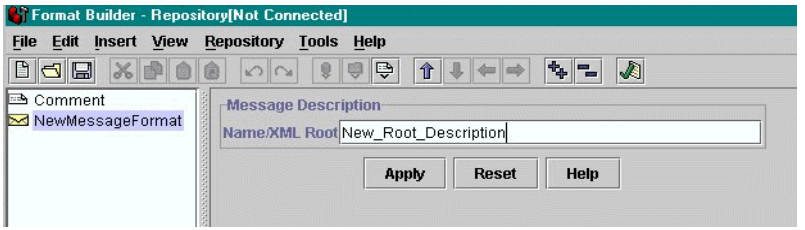
Creating Message Formats

The first step in creating a message format definition file is to create a message format (the root node of a message format file).

To create a message format:

1. Choose File→New. The detail window for the message format is displayed the right pane.

Figure 3-4 Message Format Detail Window



2. Enter the name of the message format in the Name/XML root field.

   Note: The entry in the Name/XML Root field becomes the name of the root element of each XML instance generated based on this message format document. Therefore, the entry must comply with the conventions described in the following section, "XML Element Naming Conventions."

3. Click one of the following:
   - Apply—updates the message format properties.
   - Reset—discards your changes to the detail window and resets all fields to the values that were last applied.
   - Help—displays online help information for the message format detail window.

   Note: The Apply and Reset options are enabled only after changes are made in the detail window.

XML Element Naming Conventions

The names you assign to the root node, fields, groups, and references in a message format document are translated to XML element names in the XML instances generated based on the message format document. Therefore, the names must comply with the following XML naming rules:

- A name must start with a letter or underscore.
- A name can contain letters, digits, periods, hyphens, or underscores.

The following strings are examples of valid names:

- `MyField`
- `MyField1`
- `MyField_again`
- `MyField-again`

The following strings are examples of invalid names:

- `1MyField` (starts with a digit)
- `My>Field` (includes a greater-than sign (`>`), which is an illegal character)
- `My Field` (includes a space, which is not permitted)

Creating Groups

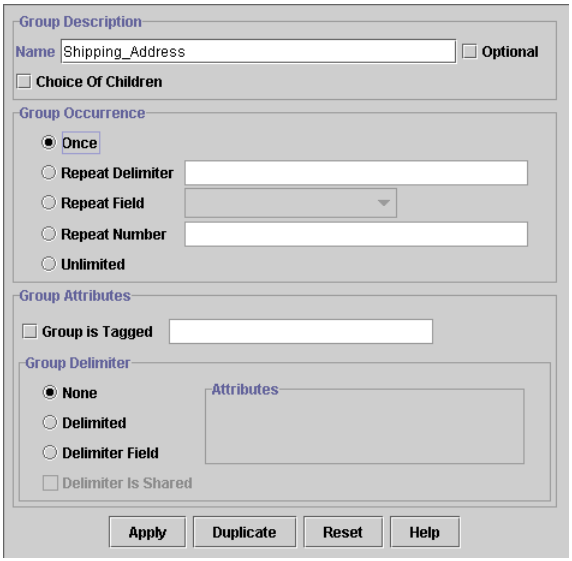
A group is a collection of fields, comments, references, and other groups that are related in some way. For example, the fields `PAYDATE`, `HOURS`, and `RATE` might all belong to the `PAYINFO` group. You can create a group as a child of the message format item, as a child of another group, or as a sibling of a group or field.

To create a group:

1. Select the an item in the navigation tree.
2. Choose one of the following:
   - If the selected item is the root node, or another group, and you want to create the group as the child of the selected item, choose Insert→Group→As Child.
   - If you want to create the group as a sibling the selected item, choose Insert→Group→As Sibling.

   The detail window for the group is displayed the right pane.

Figure 3-5 Group Detail Window



3. Define the properties for the group as described in the following table.

Table 3-3 Group Properties

Category	Property	Description
Group Description	Name	The name of the group. The entry must comply with the conventions described in "XML Element Naming Conventions" on page 19.
	Optional	Select Optional if the group is optional.
	Choice of Children	Select Choice of Children if only one of the items in the group will be included in the message format.
Group Occurrence (Unless defined as Optional in the Group Description, all groups occur at least once.)	Once	Select this option to indicate that the group appears only once.
	Repeat Delimiter	Select this option to indicate that the group will repeat until the specified delimiter is encountered.
	Repeat Field	Select this option to indicate that the group will repeat the number of times specified in the field selected as the repeat field.
	Repeat Number	Select this option to indicate that the group will repeat the specified number of times.
	Unlimited	Select this option to indicate that the group will repeat an unlimited number of times.

Table 3-3 Group Properties (Continued)

Category	Property	Description
Group Attributes	Group is Tagged	Select this option if the group is tagged, that is, if a literal precedes the other content of the group, which may be other groups or fields.
	Group Delimiter	The termination point of a group can be specified by a *delimiter*: a string of characters that marks the end of a group of fields. The group continues until delimiter characters are encountered. **Note:** Normally, groups are not delimited. They are usually parsed by content; the group ends when all child objects have been parsed. For more information about delimiters, see "Specifying Delimiters" on page 3-23. Select from among the following options to specify the group delimiter attributes:
		None — Select this option if there is no delimiter for the group.
		Delimited — Select this option if the termination point of the group is marked with a delimiter character string, then enter the delimiter characters in the Value field.
		Delimiter Field — Select this option if the termination point of the group is marked by a field that contains a delimiter character string. When you select this option, you are prompted to provide the following: Field—select the field that contains the delimiter character string. A list of valid fields is presented in a drop-down list. Default—enter the default delimiter character used if the selected field is not included in the data. This value is required.
		Delimiter is Shared — Select this option to indicate that the delimiter marks both the end of the group of data, and the end of the last field of the group. The delimiter is shared by the group, and the last field of the group, to indicate the end of the data.

4. Click one of the following:
   - Apply—updates the group properties.
   - Duplicate—makes a copy of the group currently displayed and pastes it as a sibling.

     The duplicate group contains the same values and child objects as the original. The name of the duplicate is the same as that of the original, with the addition of a prefix: *New*. Thus, for example, if the name of the original group is MyGroup1, then the name of the duplicate is NewMyGroup1.
   - Reset—discards your changes to the detail window and resets all fields to the values that were last applied.
   - Help—displays online help information for the detail window.

   Note: The Apply and Reset options are enabled only after changes are made in the detail window.

Specifying Delimiters

You can specify delimiters in Format Builder by entering the correct syntax. For example, if you want to specify a tab character as a delimiter ('\u009'), you must enter the construct `\t` to match it.

The following tables maps characters you can use as delimiters to the constructs you must use to designate these characters as delimiters.

Table 3-4 Character Delimiters

Use this construct . . .	To designate the following character as a delimiter . . .
`x`	x
`\\`	\ (backlash)
`\0n`	Character with octal value 0n (<= n <= 7)
`\0nn`	Character with octal value 0nn (0 <= n <= 7)
`\0mnn`	Character with octal value 0mnn (0 <= m <= 3, 0 <= n <= 7)

Table 3-4 Character Delimiters (Continued)

Use this construct . . .	To designate the following character as a delimiter . . .
\xhh	Character with hexadecimal value 0xhh
\uhhhh	Character with hexadecimal value 0xhhhh
\t	Tab character ('\u0009')
\n	Newline (line feed) character ('\u000A')
\r	Carriage-return character ('\u000D')
\f	Form-feed character ('\u000C')
\a	Alert (bell) character ('\u0007')
\e	Escape character ('\u001B')
\cx	Control character corresponding to x

For more information, visit the following URL:

http://java.sun.com/j2se/1.4/docs/api/java/util/regex/Pattern.html

Creating Fields

A field is a sequence of bytes that is meaningful to an application. (For example, the field EMPNAME contains an employee name.) You can create a field as a child of the message format node, as a child of a group, or as a sibling of a group or another field. Field names are used as element names in the XML output; they must comply with the conventions described in "XML Element Naming Conventions" on page 3-19.

To create a field:

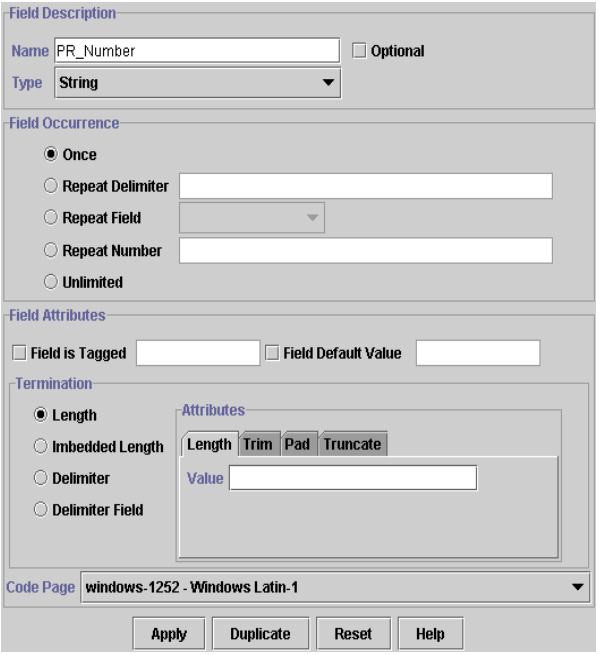
1. Select an item in the navigation tree.

2. Choose one of the following:

   - If you want to create the field as the child of the selected item, choose Insert→Field→As Child.
   - If you want to create the field as the sibling of the selected item choose Insert→Field→As Sibling.

   The detail window for the field is displayed the right pane.

Figure 3-6 Field Detail Window



3. Define the properties for the field as described in the following table.

Table 3-5 Field Properties

Category	Property	Description
Field Description	Name	The name of the field. The entry must comply with the conventions described in "XML Element Naming Conventions" on page 19.
	Optional	Select this option if this is an optional field. Optional means that the data for the field may or may not be present.
	Type	Select the data type of the field from the drop-down list. The default is String. **Note:** Which field type you select dictates which field data options are displayed. For a list of data types supported by ProductName, see Appendix A, "Supported Data Types."
Field Occurrence (Unless defined as Optional in the Field Description, all fields occur at least once.)	Once	Select this option to indicate that the field appears only once.
	Repeat Delimiter	Select this option to indicate that the field will repeat until the specified delimiter is encountered.
	Repeat Field	Select this option to indicate that the field will repeat the number of times specified in the field selected as the repeat field.
	Repeat Number	Select this option to indicate that the field will repeat the specified number of times.
	Unlimited	Select this option to indicate that the field will repeat an unlimited number of times.

Table 3-5 Field Properties (Continued)

Category	Property	Description
Field Attributes (The Field Attributes properties that are displayed are dependent on the Type specified in the Field Description)	Field is Tagged	Select this option if the field is tagged, that is, if a literal proceeds the data, indicating that the data is present. You must also choose the data type of the tag field from the drop-down list. For example in the following: `SUP:ACME INC` `SUP:` is a tag and `ACME INC` is the field data. If you select the Field is Tagged option, enter the tag in the field to the right of the check box.
	Field Default Value	Select this option to specify a value for the data in field that is inserted into the binary data if the field is not included in the XML. If the field is not included in the binary data and it is not optional, then the binary data fails to parse, even if a default value is given.
	Data Base Type	If the field is a date or time field, the base type indicates the type of characters (ASCII, EBCDIC, or Numeric) used to represent the data.
	Year Cutoff	If the field is a date field with a 2-digit year, the year cutoff attribute allows the 2-digit year to be converted to a 4-digit year. If the 2-digit year is greater than or equal to the year cutoff value, a prefix of 19 is added to the year value. Otherwise a prefix of 20 is used.
	Code Page	The character encoding of the field data. The default code page is set by choosing Tools→Options and selecting the default encoding from the Default Field Code Page drop down list.
	Value	The value displayed in a literal field.

Table 3-5 Field Properties (Continued)

Category	Property	Description	
Field Attributes (Continued)	Termination	Select from among the following options to specify the group delimiter attributes:	
		Length	Select this option to set the length of variable-sized data types to a fixed value. When you select this option, you are prompted to provide the following: Length—enter the number of bytes in the field. Trim Leading/Trailing—removes the specified data from the leading or trailing edge of the data. Pad—if the XML data is shorter than the specified length, appends the specified data to correct its length. Select one of the following padding options: ■ Select the Trailing option to append padding at the end of a field. ■ Select the Leading option to append padding at the beginning of a field. Truncate—remove a specified number of characters from a field. Select any combination of the following truncation options: ■ Select the Truncate First option to remove the specified number of characters from the beginning of the field. ■ Select the Truncate After option to remove the specified number of characters from the end of the field. If you select both truncation options, the Truncate First option is implemented initially, and the Truncate After option is invoked on the remaining characters.

Table 3-5 Field Properties (Continued)

Category	Property	Description	
Field Attributes (Continued)	Termination (Continued)	Embedded Length	Select this option to indicate that the termination point of a variable-sized data type is specified by an embedded length. An embedded length precedes the data field and indicates the number of bytes in the data. When you select this option, you are prompted to provide the following: ■ Type—specifies the data type and, if necessary, the length or delimiter for termination. ■ Tag/Length Order—specifies the order of the tag and length fields when both are included. The default order is: tag, length. ■ Trim Leading/Trailing—removes the specified data from the leading or trailing edge of the data. ■ Truncate—remove a specified number of characters from a field. For more information, see the description of the Truncate option for the Length option.
		Delimiter	Select this option to indicate that the termination point of a variable-sized data type is specified by a *delimiter*: a value that marks the end of the field. The field data continues until the delimiter is encountered. When you select this option, you are prompted to provide the following: ■ Value—enter the delimiter that marks the end of the field data. ■ Trim Leading/Trailing—removes the specified data from the leading or trailing edge of the data. ■ Truncate—remove a specified number of characters from a field. For more information, see the description of the Truncate option for the Length option.

Table 3-5 Field Properties (Continued)

Category	Property	Description	
Field Attributes (Continued)	Termination (Continued)	Delimiter Field	Select this option if the termination point of a variable-sized data type is specified by a field that contains a delimiter value. When you select this option, you are prompted to provide the following: ■ Field—select the field that contains the delimiter. ■ Default—enter a default delimiter that can be used when the delimiter field is not present. You must enter a value in this field. ■ Trim Leading/Trailing—removes the specified data from the leading or trailing edge of the data. ■ Truncate—remove a specified number of characters from a field. For more information, see the description of the Truncate option for the Length option. For more information about delimiters, see "Specifying Delimiters" on page 3-23.
		Decimal Position	Specifies the number of digits (0-16) to the left of the decimal point.

4. Click one of the following:
   - Apply—updates the field properties.
   - Duplicate—makes a copy of the field currently displayed and pastes it as a sibling.

     The duplicate field contains the same values as the original. The name of the duplicate is the same as that of the original, with the addition of a prefix: *New*. Thus, for example, if the name of the original field is MyField1, then the name of the duplicate is NewMyField1.
   - Reset—discards your changes to the detail window and resets all fields to the values that were last applied.
   - Help—displays online help information for the field detail window.

   Note: The Apply and Reset options are enabled only after changes are made in the detail window.

Padding Mandatory Fields

In previous releases of WebLogic Integration, no padding was performed on mandatory fields when data for the field did not exist at run time. In WebLogic Integration 7.0, during an XML-to-binary translation, a mandatory field that does not contain data is padded with the default value, if a default value has been specified. If no default value is specified and a field does not contain data at translation time, an error occurs.

Note: Padding of mandatory fields is not supported for binary-to-XML translations.

This feature is useful when a group is specified multiple times, but data is provided for only one occurrence. When padding of mandatory fields is invoked, all occurrences of a group for which data are not provided are padded with default values, if specified.

Creating Comments

Comments are notes about the message format or the data translated by the message format. Comments are included in the message format definition for documentation and informational purposes only; they are unnumbered and are not transformed to XML or binary data. You can create a comment as a child or sibling of any message format, group, or field.

Note: Conventionally, a comment precedes the node it annotates.

To create a comment:

1. Select an item in the navigation tree.
2. Choose one of the following:
   - If you want to create the comment as the child of the selected item, choose Insert→Comment→As Child.
   - If you want to create the comment as the sibling of the selected item, choose Insert→Comment→As Sibling.
3. Enter the comment text in the Comment Details field.

Figure 3-7 Comment Detail Window

4. Click one of the following:
   - Apply—updates the comment text.
   - Reset—discards your changes to the detail window and resets all fields to the values that were last applied.
   - Help—displays online help information for the comment detail window.

   Note: The Apply and Reset options are enabled only after changes are made in the detail window.

Creating References

References allow you to reuse an existing field or group format in a new context. When you create a reference to an existing field or group, the same format is used, but you can modify the optional and occurrence properties for the reference field or group.

For example, if your data includes a *bill to* address and a *ship to* address and the same format is used for both addresses, you can create the address format once, and then reference it. That is, you can create the an address definition for the *bill to* address and reference it for the *ship to* address.

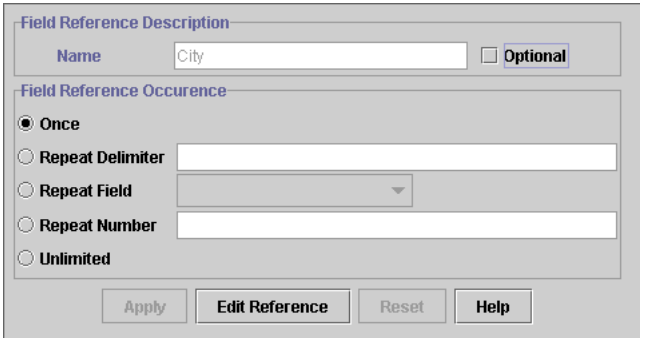
Note: A reference item is given exactly the same name as the original item, therefore, you should use a generic name, such as `address`, when you create a field or group that is be referenced. For instance, in the previous example, you can create an `address` group as a child of the `bill_to` group and then reference the `address` group from within the `ship_to` group.

To create a reference:

1. Select the item to be referenced in the navigation tree.
2. Choose Edit→Copy.
3. Select an item at the desired location for the reference. When you paste the item as a reference in the next step, the reference is pasted as a sibling of the selected item.
4. Choose Edit→Paste→As Reference.

   The detail window for the reference is displayed. For example, the following figure shows the detail window for a Field Reference.

Figure 3-8 Field Reference Detail Window



5. Define the properties for the reference as described in the following table.

Table 3-6 Reference Properties

Category	Property	Description
Field or Group Reference Description	Name	The name of the field or group for which you created this reference. This value cannot be changed.
	Optional	Select this option if the reference is optional.

Table 3-6 Reference Properties

Category	Property	Description
Field or Group Reference Occurrence (Unless defined as Optional all referenced items occur at least once.)	Once	Select this option to indicate that the referenced item appears only once.
	Repeat Delimiter	Select this option to indicate that the referenced item will repeat until the specified delimiter is encountered.
	Repeat Field	Select this option to indicate that the referenced item will repeat the number of times specified in the field selected as the repeat field.
	Repeat Number	Select this option to indicate that the referenced item will repeat the specified number of times.
	Unlimited	Select this option to indicate that the referenced item will repeat an unlimited number of times.

6. Click one of the following:
   - Apply—updates the reference properties.
   - Reset—discards your changes to the detail window and resets all fields to the values that were last applied.
   - Edit Reference—displays the detail window for the original item to allow you to edit the item.
   - Help—displays online help information for the reference detail window.

   Note: The Apply and Reset options are enabled only after changes are made in the detail window.

Working with the Palette

The Format Builder palette allows you to store commonly used message format components so they are available whenever you need to insert them into your message format definitions.

The default palette, `palette.xml`, is an MFL document which is stored in the WebLogic Integration installation directory. The default palette contains common date formats, literals, and strings. You can use these items in the message formats you create, as well as add your own items to the default palette. You can also create your own MFL documents for use in the palette, or open and use items from any existing MFL document.

The following topics provide the information you need to use the palette:

- Opening the Palette
- Using the Palette File Menu
- Using the Palette Shortcut Menu
- Copying Items From the Active Message Format to the Palette
- Deleting Items From the Palette
- Copying Palette Items from the Palette to the Active Message Format

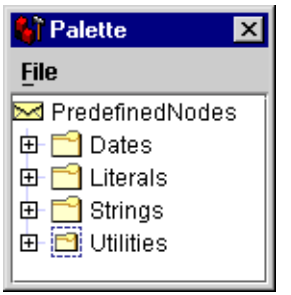
Opening the Palette

To open the palette:

1. Start Format Builder.
2. Choose View—Show Palette.

   The Palette window displays the default palette.

Figure 3-9 Palette



You can copy items from the navigation tree to the palette, and vice versa. You can use drag and drop, or the commands available on the shortcut menu, to organize items in the palette. The contents of the palette are automatically saved when you exit Format Builder.

Note: Only copying items, whether from the navigation tree to the palette or vice versa, is allowed. You cannot move items between the windows.

Using the Palette File Menu

The commands described in the following table are available from the Palette File menu.

Table 3-7 Palette File Menu Commands

Command	Description
Open	Displays the Open dialog box to allow you to select and open an existing MFL document in the palette.
Save	Saves any changes you have made to the MFL document currently open in the palette.
Hide Palette	Closes the Palette window.

Using the Palette Shortcut Menu

A shortcut menu is displayed when you right-click an item or folder in the palette. The following table describes the commands available from the shortcut menu.

Note: Some commands may be unavailable, depending on the item you select.

Table 3-8 Palette Shortcut Menu Commands

Command	Description
Insert	Inserts a new folder. When you select this command, you are prompted to supply the name of the folder.
Rename	Renames a folder. When you select this command, you are prompted to supply the new name.

Table 3-8 Palette Shortcut Menu Commands (Continued)

Command	Description
Delete	Deletes the selected item.
Move Up	Moves the selected item up one position under its parent.
Move Down	Moves the selected item down one position under its parent.
Promote	Assigns the selected item to the next level up in the hierarchy. For example, suppose Field1 is a child of Group1. If you select Field1 and click the Promote tool, then Field1 becomes a sibling of Group1.
Demote	Assigns the selected item to the next lower level in the hierarchy. When you demote an item, it becomes a child of the sibling that immediately precedes it. For example, suppose Field1 is a sibling of Group1, and that it and immediately follows Group1. If you select Field1 and click the Demote tool, Field1 becomes a child of Group1.

Copying Items From the Active Message Format to the Palette

To copy an item from the document currently open in Format Builder to the palette:

1. If it is not already displayed, choose View—Show Palette to display the palette.
2. In the navigation tree, select the item you want to add to the palette.
3. Drag the item to palette window, then drop it in the desired location in the hierarchy.

   The item is copied to the selected location.

Notes: You cannot add an item that depends on the existence of another item to the palette. For example, you cannot add a field or group reference, and you cannot add an item for which a Repeat Field is specified.

Adding comments is possible, but not recommended because comments do not have unique names and therefore are indistinguishable on the palette.

Deleting Items From the Palette

To delete an item from the palette:

1. Right-click the item to be deleted to display the shortcut menu.
2. Select Delete.

   You are prompted to confirm the deletion.
3. Click OK to delete the item.

Copying Palette Items from the Palette to the Active Message Format

To copy an item from the palette to a message format document currently open in Format Builder:

1. If it is not already displayed, choose View→Show Palette to display the palette.
2. In the palette window, select the item you want to add to your message format.
3. Drag the item to navigation tree, then drop it in the desired location in the hierarchy.

   The item is copied to the selected location in the message format.

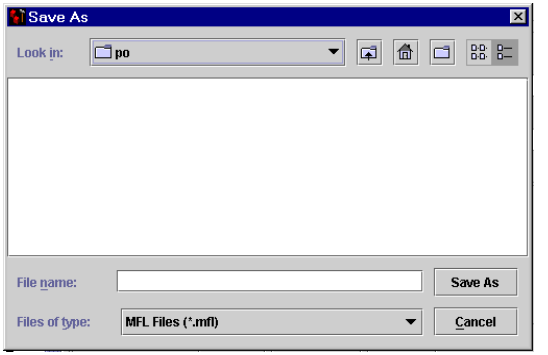
Saving or Storing a Message Format

You can save a message format document to your file system as described in the this section, or you can store the document in the repository, as described in "Storing MFL Documents in the Repository" on page 5-6.

To save a message format file for the first time:

1. Choose File→Save As to display the Save As dialog box.

Figure 3-10 Save As Dialog Box



2. Navigate to the directory in which you want to save the file.
3. Enter the name you want to assign to the file in the File Name field.

   Note: If you do not include an extension in your filename, Format Builder automatically assigns the default extension: `.mfl`.

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

To save changes to an existing file, choose File→Save.

To save an existing file to a new name, choose File→Save As and follow steps 2 through 4 in the preceding procedure.

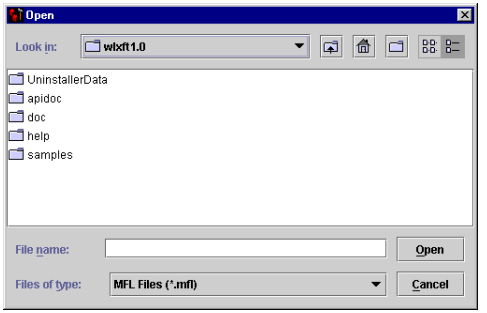
Opening or Retrieving an Existing Message Format File

You can open a message format document on your file system as described in the this section, or you can retrieve the document from the repository, as described in "Retrieving MFL Documents from the Repository" on page 5-4.

To open an existing message format file:

1. Choose File→Open to display the Open dialog box.

Figure 3-11 Open Dialog Box



2. Locate and select the desired file.
3. Click Open to open the file in Format Builder.

Using Internationalization Features

You can use the internationalization features in Format Builder by changing the options for an individual message file or by setting the default Format Builder options to include internationalization. For details, see:

- "Changing Options for a Message Format" on page 3-40
- "Setting Format Builder Options" on page 3-41

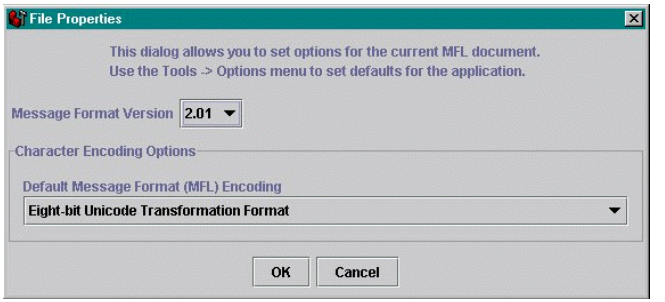
Changing Options for a Message Format

To change options for a message format file:

1. Select the root node of the message format in the navigation tree.
2. Choose File—Properties.

   The File Properties dialog box displays the Message Format Version and the Default Message Format (MFL) Encoding.

Figure 3-12 File Properties Dialog Box



3. Select a type of character encoding for the MFL document from the list of encoding names and descriptions for this file. (To change the default settings for all new message format documents, choose Tools→Options.)
4. Click OK.

   Your changes are reflected in the MFL document when you test it using Format Tester.

Setting Format Builder Options

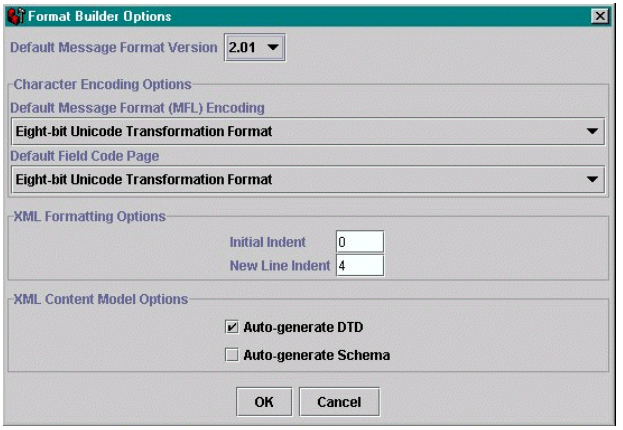
You can set several options to control the overall operation of Format Builder.

To set Format Builder options:

1. Choose Tools→Options.

   The Options dialog box is displayed.

Figure 3-13 Format Builder Options Dialog Box



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

Table 3-9 Format Builder Options

Category	Option	Description
N/A	Default Message Format Version	Select the version to be associated with new MFL documents. **Note:** Each message format document is associated with its own message format version. The version specified for a message format can be changed from the default from the File Properties dialog box described in the preceding section "Changing Options for a Message Format.".
Character Encoding Options	Default Message Format (MFL) Encoding	Select the character encoding to be associated with new MFL documents. The character encoding associated with an MFL document specifies the encoding used for the MFL document itself, and the XML output it generates.
	Default Field Code Page	Select the code page, from the list of binary formats, to be used as the default code page for each field created in your MFL documents. A code page specifies the character encoding of the binary data in the field.
XML Formatting Options	Initial Indent	Enter the number of spaces by which to indent the root element when generating the XML output.
	New Line Indent	Enter the number of spaces by which to indent a new child element when generating the XML output.

Table 3-9 Format Builder Options (Continued)

Category	Option	Description
XML Content Model Options	Auto-generate DTD	Generates a DTD document which captures the content model defined in the MFL document. If you specify Auto-generate DTD: ■ When you save an MFL document to the file system, the DTD is saved in the same directory. ■ When you store an MFL document in the repository, the DTD is also stored.
	Auto-generate Schema	Generates an XML Schema document which captures the content model defined in the MFL document. If you specify Auto-generate Schema: ■ When you save an MFL document to the file system, the XML Schema document is saved in the same directory. ■ When you store an MFL document in the repository, the XML Schema is also stored.

3. Click one of the following:
   - OK—saves your changes and dismisses the dialog box.
   - Cancel—discards your changes and dismisses the dialog box.

Format Builder Menus

The following menus are available in Format Builder: File, Edit, Insert, View, Repository, Tools, and Help.

The commands available on each menu are described in the following sections.

Note: Some commands may be unavailable, depending on which actions you have taken and what is selected in the navigation tree.

File Menu

The following commands are available from the File menu.

Table 3-10 File Menu Commands

Command	Description
New	Creates a new message format document.
Open	Opens an existing message format document.
Close	Closes the current message format document.
Save	Saves the current message format document.
Save As	Saves the current message format under a different name.
Properties	Opens the File Properties dialog box for the active message format document. This dialog allows you to set options for the active MFL document (see "Changing Options for a Message Format" on page 3-40). Choose Tools—Option to set defaults for the application (see "Setting Format Builder Options" on page 3-41).
Exit	Exits the Format Builder.

Edit Menu

The following commands are available from the Edit menu.

Table 3-11 Edit Menu Commands

Command	Description
Undo *action*	Reverses the previous action. The Undo command on the Edit menu is constantly refreshed to indicate the action most recently performed that can be nullified. For example, if you change the name of a field to Field1 and click Apply, the listing for the Undo command contains the following text: Undo Apply Field Field1. Format Builder supports multiple undoing of previous actions.

Table 3-11 Edit Menu Commands (Continued)

Command	Description
Redo *action*	Reverses the effects of the Undo command. The Redo command in the Edit menu is constantly refreshed to indicate the action that can be redone. For example, if you change the name of a field to Field1 and then click Undo, the listing for the Redo command contains the following text: Redo Apply Field Field1. Format Builder supports multiple redoing of actions previously undone.
Cut	Removes the selected item along with its child objects. The item is placed in the clipboard and can be pasted in a new location. **Note:** This action is not available if the Message Format (root) item is selected.
Copy	Makes a copy of the selected item along with its child objects. The copy is placed in the clipboard and can be pasted in a new location. **Note:** This action is not available if the Message Format (root) item is selected.
Paste	Inserts the current contents of the clipboard. When you select Paste, the following Paste menu options are displayed: ■ As Child ■ As Sibling ■ As Reference
Duplicate	Makes a copy of the currently selected item and pastes it as a sibling. The duplicate item contains the same values and child objects as the original. The name of the duplicate is the same as that of the original, with the addition of a prefix: *New*. Thus, for example, if the name of the original item is MyField1, then the name of the duplicate is NewMyField1.
Delete	Deletes the item selected in the navigation tree, as well as all child objects of that item.
Move Up	Moves the selected item up one position under its parent.
Move Down	Moves the selected item down one position under its parent.

Table 3-11 Edit Menu Commands (Continued)

Command	Description
Promote	Assigns the selected item to the next level up in the hierarchy. For example, suppose Field1 is a child of Group1. If you select Field1 and select Promote, then Field1 becomes a sibling of Group1 and is inserted immediately after Group1.
Demote	Assigns the selected item to the next lower level in the hierarchy. When you demote an item, it becomes a child of the group that immediately precedes it. For example, suppose Field1 is a sibling of Group1 and immediately follows Group1. If you select Field1 and select Demote, Field1 becomes a child of Group1.

Insert Menu

The following commands are available from the Insert menu.

Table 3-12 Insert Menu Commands

Command	Description
Field	Inserts a new field. You can insert the field as either a child or sibling of the item selected in the navigation tree.
Group	Inserts a new group. You can insert the group as either a child or sibling of the item selected in the navigation tree.
Comment	Inserts a comment. You can insert the comment as either a child or sibling of the item selected in the navigation tree.

View Menu

The following commands are available from the View menu.

Table 3-13 View menu Commands

Command	Description
Show Palette	Displays the Palette window.

Table 3-13 View menu Commands (Continued)

Command	Description
Expand All	Expands the entire navigation tree to show the child objects of all items in the navigation tree.
Collapse All	Collapses the entire navigation tree to show only the root message format.

Repository Menu

The following commands are available from the Repository menu.

Note: For details about using the repository, see Chapter 5, "Retrieving and Storing Repository Documents."

Table 3-14 Repository Menu Commands

Command	Description
Log In	Displays the ProductName Repository Login dialog box, allowing you to connect to the repository.
Log Out	Disconnects from the repository.
Retrieve	Retrieves a document from the repository.
Store	Stores the current document in the repository.
Store As	Stores the current document in the repository under a different name.

Tools Menu

The following commands are available from the Tools menu.

Table 3-15 Tools Menu Commands

Command	Description
Import	Displays a list of the installed importers. Choose the importer from which you want to import a message.

Table 3-15 Tools Menu Commands (Continued)

Command	Description
Test	Opens the Format Tester.
User Defined Types	Opens the Add/Remove User Defined Types dialog box.
Options	Displays the Format Builder Options dialog box.

Help Menu

The following commands are available from the Help menu.

Table 3-16 Help Menu Commands

Command	Description
Help Topics	Displays the online help in your default browser.
How Do I	Displays a list of common Format Builder tasks. Click a task to view the step-by-step instructions.
About	Displays version and copyright information for the Format Builder and the JDK you are running.

CHAPTER

4 Importing Metadata

WebLogic Integration provides utilities that allow you to import COBOL copybooks, convert C structure definitions, and convert FML field table classes into MFL files. The following topics explain how to perform these import operations:

- Importing a COBOL Copybook
- Importing C Structures
- Importing an FML Field Table Class

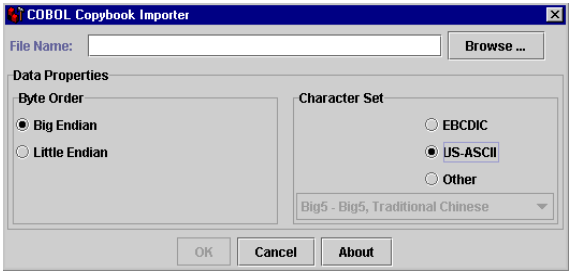
Importing a COBOL Copybook

WebLogic Integration includes a feature that allows you to import a COBOL copybook into Format Builder by creating a message definition to translate the COBOL data. When importing a copybook, you can use comments to document the imported copybook and the Groups and Fields it contains.

To import a COBOL copybook:

1. Choose Tools→Import→COBOL Copybook Importer. The COBOL Copybook Importer dialog box is displayed.

Figure 4-1 COBOL Copybook Importer



2. Designate the properties, as described in the following table.

Table 4-1 COBOL Copybook Importer Properties

Property	Value	Description
File Name	text string	Type the full pathname of the file you want to import or use the Browse button to navigate to the location of the file.
Byte Order	Big Endian	Select this option for IBM 370, Motorola, and most RISC designs (IBM mainframes and most UNIX platforms).
	Little Endian	Select this option for Intel, VAX, and Unisys processors (Windows, VMS, Digital, UNIX, and Unisys)
Character Set **Note:** The character set is an attribute of the originating host machine.	EBCDIC	Select this option to set the character set to EBCDIC.
	US-ASCII	Select this option to set the character set to US-ASCII.
	Other	Select character encoding of the field data by using a list of code pages.

3. Click one of the following:
   - OK—imports the COBOL Copybook using the settings you defined.
   - Cancel—closes the dialog box and returns you to Format Builder without importing.
   - About—displays information about the COBOL Copybook importer, including the version being used and copybook features that are supported.

After you import a copybook, you can work in the same way you work with any message format definition. If you find an error or unsupported data type in the copybook, a message is displayed, informing you of the error. You can choose to have the error displayed or saved in a log file for future reference.

The following table provides a listing and descriptions of the sample files installed for the COBOL copybook importer. All directory names are relative; the specified directories are under under *SAMPLES_HOME*\integration\samples\di where *SAMPLES_HOME* is the samples directory in your WebLogic Platform installation.

Table 4-2 Sample COBOL Copybook Files

Directory	File	Description
COBOL\	emprec5.cpy	Sample copybook file
COBOL\	emprec5.data	Test data corresponding to emprec5.cpy

Importing C Structures

WebLogic Integration includes a C struct importer utility that converts a C struct definition into an MFL message definition by generating the following types of output data:

- MFL document
- C code

Whichever type of output you want, you must first specify a .c or .h input file, which must be parsed, and then select the desired structure. Then you can choose between MFL (default) or C code for your output.

All input to the parser must be valid C code. In addition, all external references, such as #include, #define, and typedef statements, must be resolved before you can use them. You can resolve them by editing them manually or by using the compiler's preprocessor.

Various platform-specific parameters may affect the description of data for C code. For example, the length of a `long` on a particular platform affects the binary data that conforms to a particular structure definition.

Two methods are available for dealing with these platform dependencies, depending on whether or not MFL is generated directly into Format Builder. If you want to generate MFL and have that MFL displayed immediately in Format Builder, you must supply the platform-dependent parameters in a configuration file.

Alternately, if you choose to generate your source in C, you may compile the C code on the desired machine. The compiler on that machine accounts for the necessary platform-dependent information. This approach allows you to produce an executable file that, when run, produces two files: an MFL document and binary data that conforms to that MFL. The MFL document can be opened in Format Builder and the binary data file can be opened in Format Tester.

Generating MFL directly into Format Builder requires platform configuration parameters found in an existing configuration file or a new configuration file created with the hardware profile editor. The hardware profile editor allows you to specify an existing profile that can be loaded, updated, and saved.

The source code for a utility that generates hardware profiles according to your needs is provided in the *SAMPLES_HOME*`\integration\samples\di\cfg` directory.

Sample C Struct Importer Files

The following table provides a listing and descriptions of the sample files installed for the C struct importer. All directory names are relative; the specified directories are under *SAMPLES_HOME*`\integration\samples\di`.

Table 4-3 Sample C Struct Importer Files

Directory	File	Description
`C`	`emprec5.h`	C version of the `emprec5.cpy` sample Copybook file, with some typedefs.
`C`	`emprec5n.h`	Variant of the `emprec5.h` file in which a nested struct definition, but no typedef is used.
`C`	`emprec5s.h`	Simple version of the `emprec5.h` file.

Table 4-3 Sample C Struct Importer Files (Continued)

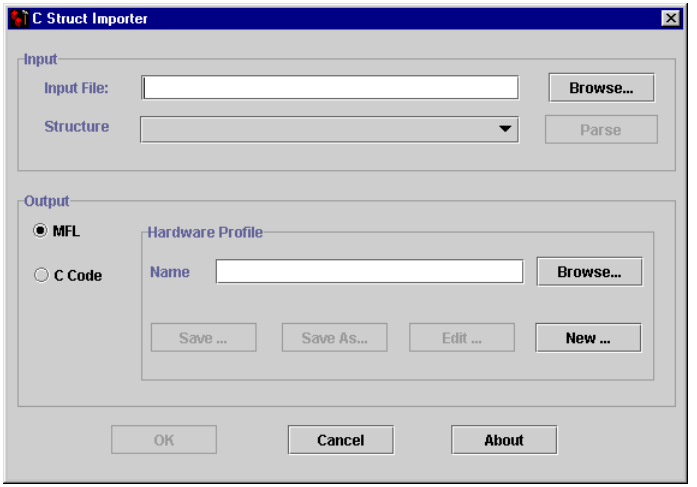
Directory	File	Description
C	ntfsez.h	Small sample, extracted from the ntfs.h file, designed to test recursive typedefs.
Cfg	cprofile.c	Source code for the cprofile.c utility; designed to generate profiles on various platforms.
The following .cfg files are generated by the cprofile program on various platforms. Each .cfg file contains a value for DESCRIPTION.		
Cfg	dec8cc.cfg	DEC Alpha 1091, Digital UNIX 4.0e, cc compiler
Cfg	hp5cc.cfg	HP-UX B.11.00, cc compiler
Cfg	nt4bcc5.cfg	Windows NT 4.0, Borland 5.x compiler, default switches
Cfg	nt4vc6.cfg	Windows NT 4.0, Visual C++ 6.x compiler, default switches
Cfg	sun7cc.cfg	SunOS 5.8, cc compiler
Cfg	w95bcc5.cfg	Windows 95, Borland 5.x compiler, default alignment
Cfg	w95vc5.cfg	Windows 95, Visual C++ 5.x compiler, default alignment

Starting the C Struct Importer

To start the C Struct Importer:

1. Start Format Builder by choosing Start→Programs→BEA WebLogic Platform 7.0→WebLogic Integration 7.0→Format Builder. The Format Builder main window is displayed.
2. Choose Tools→Import→C Struct Importer. The C Struct Importer dialog box is displayed.

Figure 4-2 C Struct Importer Dialog Box



The C Struct Importer dialog box allows you to specify import properties, as described in the following table.

Note: Initially, MFL is specified as the default output type.

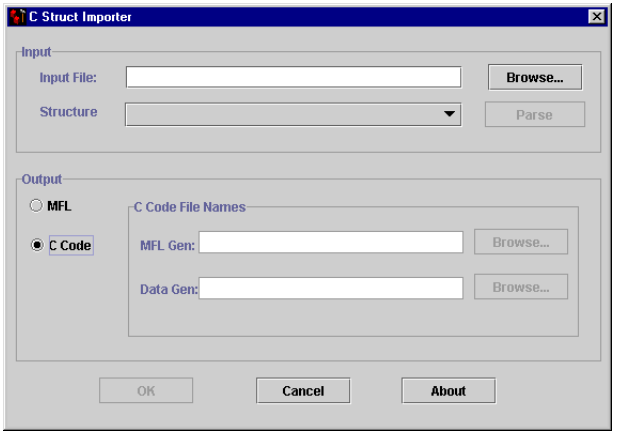
Table 4-4 C Struct Importer Properties

Category	Property	Description
Input	Input File	Type the full pathname of the file you want to import or use the Browse button to navigate to the location of the file.
	Structure	Drop-down list of structures found in the input file after parsing is successful.
	Parse	Select this option to parse the input file. If successful, the Structure list box is populated with the list of structures found in the input file.

Table 4-4 C Struct Importer Properties (Continued)

Category	Property	Description
Output	MFL	If you select this option, you can generate MFL from a structure definition and a hardware configuration file. The Hardware Profile dialog box is displayed with the following options. ■ Name—Specify an existing profile either by entering the file name or using the Browse option. The prebuilt hardware profiles may be found in the `samples\di\cfg` directory. ■ Save—saves the current hardware profile. ■ Save As—allows you to save the current hardware profile under another name. ■ Edit—allows you to edit the current hardware profile. ■ New—allows you to create a new hardware profile.
	C Code	If you select this option, you can generate C source code to compile on the target machine and execute to produce MFL. The C Code File Names dialog box is displayed with the following options. ■ MFL Gen—specifies the C source code file name that must be compiled on the target machine to generate MFL. Use the Browse option to navigate to the directory where you want the file to reside. ■ Data Gen—specifies the C source code file name that must be compiled on the target machine for generating test data. Use the Browse option to navigate to the directory where you want the file to reside.

Figure 4-3 C Struct Importer Dialog Box



3. Click one of the following:
   - OK—saves your hardware profile changes.
   - Cancel—dismisses your hardware profile changes.
   - About—displays information about the C Struct Importer, including the version number and the release date.

Understanding Hardware Profiles

The hardware profiles used by the C Struct Importer contain data size and alignment information for specific hardware and compiler combinations and are used to generate MFL for C structures. They are stored in configuration files that can be created, loaded, updated, and saved.

The `cprofile.c` source file in the *`SAMPLES_HOME`*`\integration\samples\di\cfg` directory is used to generate these profiles for any platform. This code is designed to be compiled and executed on the target platform with the compiler normally used. You should be able to compile and execute it on any platform with an ANSI standard C compiler in order to generate a profile configuration file that can be imported into the C Struct Importer.

Building the Hardware Profile Utility

To produce acceptable parser input, execute the appropriate commands for your platform:

- On Windows NT, use the VC++ preprocessor:

  VC++ Compiler
  `cl /P cprofile.c (output in cprofile.i)`

  GNU Compiler
  `gcc -P -E cprofile.c>cprofile.i`

- On UNIX

  `cc -P cprofile.c (output in cprofile.i)`

Running the Hardware Profile Utility

To execute the `cprofile` program and specify a hardware profile name, enter the following text at a command prompt:

`cprofile` *`configfilename`* `[DESCRIPTION]`

A description is optional. If you decide to provide one, put it in the configuration file as the value of `DESCRIPTION`. If the description contains embedded blanks, enclose it in quotes.

Generating MFL

To generate MFL:

1. Enter a filename in the Input File field, or click Browse and select a file from the list that is displayed.
2. Click Parse to parse the file.

   Upon completion, the Structure list is populated with the structures found in the input file.

   Note: If your file does not parse correctly, we recommend that you proceed in one of two ways:

- Run your `.h` or `.c` source code through the compiler, preprocessor, and then run the processor output through the parser.
- Comment out the character creating the parsing failure and attempt to parse again. Please note that the parser fails at the first instance of incompatible data it encounters. Therefore, repetition of this step may be required.

3. Select the desired structure from the Structure drop-down list.

   At this point, you must provide some profile configuration data to generate the MFL directly. You can do this by either creating a new hardware profile or specifying an existing profile.

4. Specify an existing profile or create a new one by performing one of the following procedures:

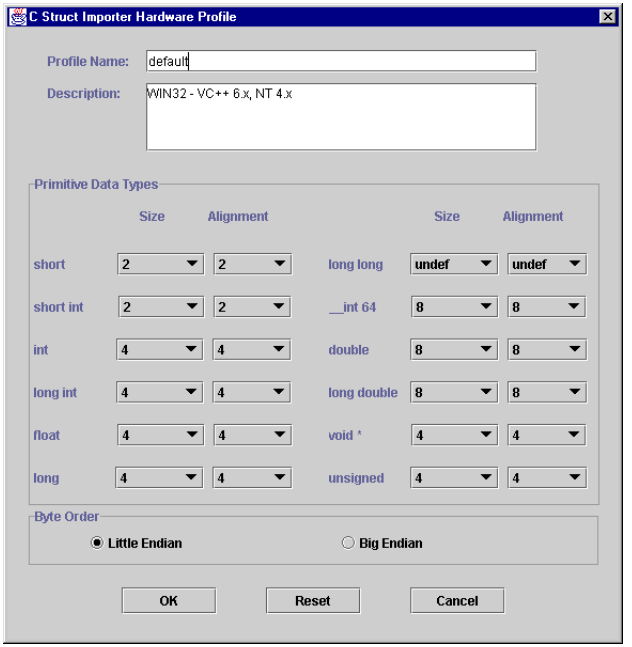
   - Specify an existing profile in one of the following ways: enter the filename in the Hardware Profile Name field, or click Browse to select a file from the list that is displayed.

     Click Edit to open the hardware profile editor if you need to view or edit the profile parameters.

     Note: Hardware profiles for common configurations are prebuilt and may be found in the `samples\c\cfg` directory.

   - Click New to create a new hardware profile. The Hardware Profile editor is displayed with the default parameters loaded. Specify a name and description for the new profile, and modify the primitive data types and byte order as required.

Figure 4-4 C Struct Importer Hardware Profile Dialog Box

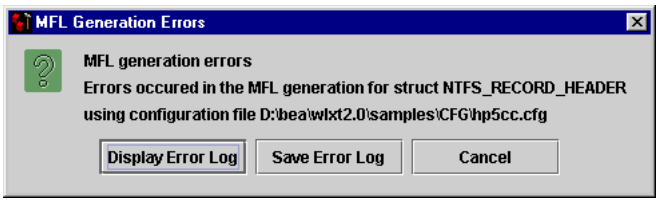


5. Click OK to save your hardware profile changes and return to the C Struct Importer dialog box.

6. Click OK to generate your MFL. If the generation is successful, you are returned to Format Builder with an MFL object listed in the navigation tree. The MFL object reflects the same name as the input file used in the parse operation.

   If errors are detected during the generation process, the MFL Generation Errors dialog box is displayed providing you with an opportunity to view or file the error log.

Figure 4-5 MFL Generation Errors Dialog Box



7. Click one of the following:
   - Display Error Log—to view any errors encountered,
   - Save Error Log—to save the error log to the location of your choice, or
   - Cancel—to dismiss the MFL Generation Errors dialog box.

After you determine which errors were generated, you can return to the C Struct Importer and repeat the applicable steps.

Generating C Code

To generate C code:

1. Enter a filename in the Input File field, or click Browse and select a file from the list that is displayed.
2. Click Parse to parse the file.

   Upon completion, the Structure list is populated with the structures found in the input file.

   Note: If your file does not parse correctly, we recommend that you proceed in one of two ways:
   - Run your `.h` or `.c` source code through the compiler, preprocessor, and then run the processor output through the parser.
   - Comment out the character creating the parsing failure and attempt to parse again. Please note that the parser fails at the first instance of incompatible data it encounters. Therefore, repetition of this step may be required.
3. Select the desired structure from the Structure drop-down list.
4. Select the C Code option.
5. Enter a filename in either the MFL Gen or Data Gen field, or click Browse and select a file from the list that is displayed.
6. Click OK.

   Messages are displayed if you are about to overwrite an existing file or if the code generation has succeeded or failed.

7. Copy the generated source code to the target platform, compile and execute it.

   Note: You must copy the input file containing the struct declarations, as well. When compiled, both programs accept the name of the output file as an argument.

8. Copy the generated MFL or data back to the platform on which Format Builder is running.

Importing an FML Field Table Class

The FML Field Table Class Importer facilitates the integration of WebLogic Tuxedo Connector and business process management (BPM) functionality. Tuxedo application buffers are translated to and from XML by the FML to XML Translator that is a feature of WebLogic Tuxedo Connector.

The integration of Tuxedo with BPM functionality requires the creation of the XML that is passed between the WebLogic Tuxedo Connector Translator and the process engine. To create the necessary XML, use the FML Field Table Class Importer and the XML generation feature of Format Tester.

FML Field Table Class Importer Prerequisites

Before starting Format Builder:

1. Move the field tables associated with the FML buffer from the Tuxedo system to the WebLogic Server/WebLogic Tuxedo Connector environment.
2. Use the `weblogic/wtc/jatmi/mkfldclass` utility to build Java source code representing the field tables. For information about FML Field Table Administration, see the WebLogic Server documentation.
3. Compile the source code. The resulting class files are called `fldtbl` classes because they implement the `FldTbl` interface. These classes must be moved to a location specified in the Format Builder `CLASSPATH`.

The *SAMPLES_HOME*\integration\samples\di\fml directory contains several fldtbl class fields that you can use as samples. These samples allow you to start Format Builder without completing the previous three steps.

Note: Because most users perform these steps when configuring WebLogic Tuxedo Connector, these class files may already exist.

Sample FML Field Table Class Files

The following table provides a listing and descriptions of the sample files installed for the FML Field Table Class Importer. All files are in the *SAMPLES_HOME*\integration\samples\di\fml directory.

Table 4-5 FML Field Table Class Sample Files

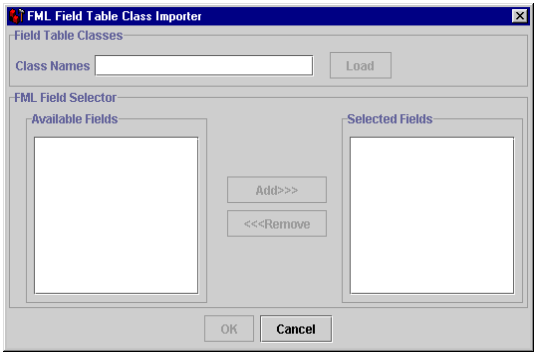
File	Description
bankflds.class	Compiled source file that serves as input to the FML Field Table Class Importer
bankflds.java	fldtbl source file generated by the mkfldclass utility
crdtflds.class	Compiled source file that serves as input to the FML Field Table Class Importer
crdtflds.java	fldtbl source file generated by the mkfldclass utility
tBtest1flds32.class	Compiled source file that serves as input to the FML Field Table Class Importer
tBtest1flds32.java	fldtbl source file generated by the mkfldclass utility

Creating XML with the FML Field Table Class Importer

To create an XML document with the FML Field Table Class Importer:

Note: If you create Java classes using WebLogic Tuxedo Connector, you can place the `.class` files in the `\ext` directory. You can then populate the Available Fields list automatically from the FML Field Table Class Importer dialog box.

1. Start Format Builder. The Format Builder main window is displayed.
2. Choose Tools→Import→FML Field Table Class Importer. The FML Field Table Class Importer dialog box is displayed.



3. In the Class Names field, enter the name of the `fldtbl` class file to be processed.

   Because a single FML buffer may contain fields from several field tables, you can enter one or more `fldtbl` class name files in the Class Names field. Items in the list should be separated by commas. Name are not required to include the `.class` extension.

   Note: If any of the listed classes are not `fldtbl` classes created by the `weblogic/wtc/jatmi/mkfldclass` utility, or if they are not included in the Format Builder CLASSPATH, then an error dialog box is displayed. Even if an error occurs, however, the valid `fldtbl` classes in the list are processed.

4. Click Load. The names of the fields from the field tables are displayed in the Available Fields list. The Available Fields list does not allow duplicate names. Even if the name of a field appears in different field tables, it is included only once in the list.

5. Select the desired fields from the Available Fields list and click Add. The selected fields are displayed in the Selected Fields list. (To remove a field from the Selected Fields list, select it and click Remove.)
6. When your Selected Fields list is complete, click OK. The FML Field Table Class Importer dialog box closes and the name of the generated MFL is added to the Format Builder navigation tree. The selected fields are listed in the order in which they appear in the Selected Fields list.
7. Edit the created MFL document to specify the order and number of occurrences of the fields in the XML document to be passed to the WebLogic Tuxedo Connector FML/XML Translator from business process management (BMP).
8. Choose Tools→Test to open Format Tester.
9. From the Format Tester menu bar, choose Generate→XML.

   Format Tester creates an XML document that conforms to the MFL document in Format Builder.
10. Edit the data content of the fields in the XML document as desired.
11. Choose File→Save XML to save the XML document in a file with a specified name and location.

    The created XML can be imported and used in business process management functions by using the XML instance editor. For information about importing XML, see the BPM documentation.

CHAPTER

5 Retrieving and Storing Repository Documents

The repository provides centralized storage for the following document types:

- Message Format Language (MFL)
- XML Document Type Definition (DTD)
- XML Schema
- XSLT Stylesheet

The repository makes it possible for documents of these types to be shared in WebLogic Integration. The repository provides access to these document types and enables you to manipulate repository documents: you can access the text, description, and notes for a document, and you can remove a document. The repository allows supported documents to be shared by WebLogic Server, business process management, and B2B integration functions. The repository also offers a batch import utility that facilitates the migration of previously constructed MFL, DTD, XML Schema, and XSLT documents residing on a different file system.

This section discusses the following topics:

- Logging Into the Repository
- Repository Menu Commands
- Retrieving MFL Documents from the Repository
- Storing MFL Documents in the Repository
- Importing Documents into the Repository
- Using the Repository Retrieve and Store Dialog Boxes

Logging Into the Repository

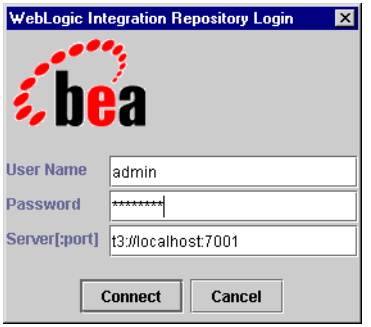
Before you can retrieve documents from, or store documents to the repository, you must be logged in.

To log into the repository:

1. If it is not already started, choose Start→Programs→BEA WebLogic Platform 7.0→WebLogic Integration 7.0→Format Builder to start Format Builder.
2. Choose Repository→Log In to display the WebLogic Integration Repository Login dialog box.
3. Enter your user name, password, and the name of the server on which the repository resides.

   For example, if WebLogic Integration is running on the local machine, you would enter the values shown in the following figure to login as the default `admin` user.

Figure 5-1 WebLogic Integration Repository Login Dialog Box



Note: For default users and passwords see "WebLogic Integration Users and Passwords" in "Getting Started " in *Starting, Stopping, and Customizing BEA WebLogic Integration*.

4. Click Connect.

   If your login is successful, the dialog box is dismissed and the Format Builder Title bar displays the server name and port number. You may now select any of the active repository menu commands.

   Note: The WebLogic Integration Repository Login dialog box allows up to three unsuccessful login attempts, after which a login failure message is displayed. If you fail to log in after three attempts, choose Repository—Log In to repeat the login procedure.

Repository Menu Commands

The following table describes the commands that are available from the Repository menu.

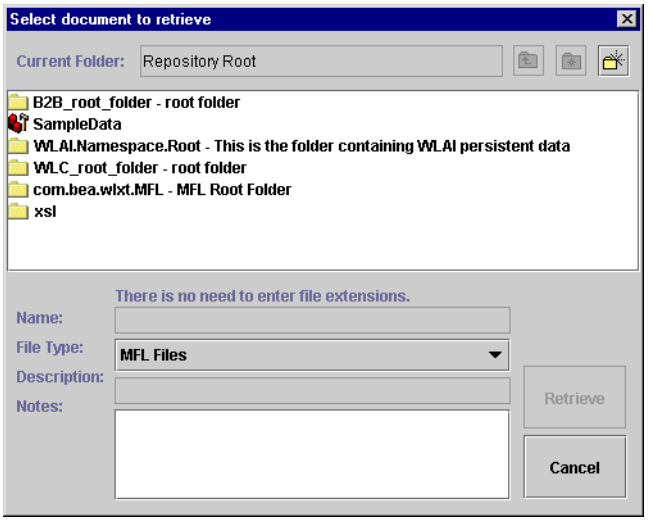
Command	Description
Log In	Displays the WebLogic Integration Repository Login dialog box, allowing you to connect to the repository.
Log Out	Disconnects from the repository.
Retrieve	Retrieves a document from the repository.
Store	Stores the current document in the repository.
Store As	Stores the current document in the repository under a different name.

Retrieving MFL Documents from the Repository

To retrieve an MFL document from the repository:

1. If it is not already started, choose Start—Programs—BEA WebLogic Platform 7.0—WebLogic Integration 7.0—Format Builder to start Format Builder.
2. Log into the repository as described in the preceding procedure, "Logging Into the Repository."
3. Choose Repository—Retrieve to display the Select document to retrieve dialog box.

Figure 5-2 Select document to retrieve dialog box



In the Select document to retrieve dialog box, double-click a folder to view its contents, or click the icon to the left of the Current Folder text box to navigate back to the parent folder. Document names and descriptions are displayed to assist in selection. For detailed information about using this dialog box, see "Using the Repository Retrieve and Store Dialog Boxes" on page 5-9.

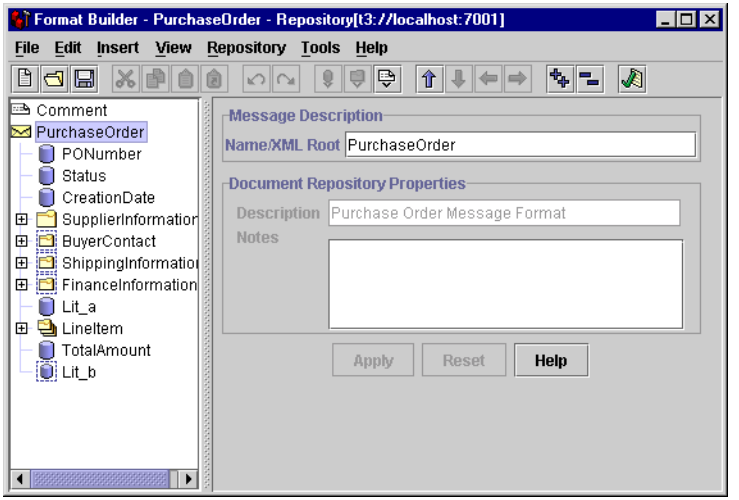
4. Locate and select the MFL document you want to retrieve.

   Note: Only MFL documents can be retrieved in Format Builder.

5. Click Retrieve.

   The selected document is displayed in the navigation tree. You can select the root node to display the detail window for the document as shown in the following figure. Note that when the active document is a repository document, the detail window includes a Document Repository Properties section.

Figure 5-3 Repository Document in Format Builder



When you retrieve a repository document, you can edit it the same way you edit any MFL document in Format Builder. When you are ready to save your changes, do one of the following:

- To save the document to the repository, choose Repository—Store.
- To save the document to the repository with a different name, choose Repository—Store As. For details, see the following procedure, "Storing MFL Documents in the Repository."
- To save the document as a local file, choose File—Save As. For details, see "Saving or Storing a Message Format" on page 3-38.

Storing MFL Documents in the Repository

To store an MFL document to the repository:

1. If it is not already opened, open the document in Format Builder.
2. Log into the repository as described in "Logging Into the Repository" on page 5-2.
3. Choose Repository—Store As.

   The Store As dialog box is displayed. For detailed information about using this dialog box, see "Using the Repository Retrieve and Store Dialog Boxes" on page 5-9.
4. In the Name field, enter the name you want to assign to the document.
5. In the description field, enter a description of the document.
6. In the Notes field, enter any notes you would like to associate with the document.
7. Click Store.

   The document is stored in the repository. When you select the root node, the detail window displays the repository name, description, and notes.

   Note: If your Format Builder options specify the generation of a DTD and/or XML Schema file, the appropriate file, or files, will also be stored in the repository using the specified name.

Importing Documents into the Repository

The batch import utility for the WebLogic Integration repository offers a command-line interface to the repository. This utility makes it easy to import previously built MFL documents into the repository. The batch importer can import MFL, DTD, class, XSLT, and XML Schema documents in any combination. It works with any plug-in repository.

Use of the repository importer with the business process management repository depends on the existence of a `wlxt-wlpi-repository.properties` file in a `CLASSPATH` directory. The property file is created by the Format Builder the first time you log into the repository. Either add the directory for your existing property file to the classpath you are using, or create a `wlxt-wlpi-repository.properties` file containing the following:

`wlxt.repository.url=t3\://`*`host`*`\:`*`port`*

Here, *`port`* is the WebLogic Server port number (the default port is `7001`) and *`host`* is the WebLogic Server computer name or IP address. Specify `localhost` or `127.0.0.1` if the server is running on the local machine.

On Windows systems, a script named `LaunchImport.cmd` is included in *`WLI_HOME`*`\bin`. If you use this script to launch the Import utility, the `CLASSPATH` will be properly set before the utility is launched. Modify the script to reflect the applicable repository path and filelist for the import operation.

To launch the import utility directly, invoke the Batch Import Utility at the command prompt, using the following command.

```
java com.bea.wlxt.repository.Import [-v] [-n] [-t type] [-f folder]
files...
```

The following table describes the options.

Table 5-1 Options for the Import Command

Option	Description
`-v`	Specifies that verbose mode is on. This switch may appear anywhere within the command line and affects all operations listed after it. Verbose mode is disabled by default.
`-n`	specifies that verbose mode is off. This switch may appear anywhere within the command line and affects all operations listed after it. Verbose mode is disabled by default.
`-f`	Optional switch for specifying the parent folder of all the following files. Multiple `-f` switches may be specified to change folders during an import execution. By default, documents are imported into the root folder of the repository. A special `-f` switch argument of @ may be used to specify the root folder. Folder names specified in the `-f` switch are always absolute pathnames from the repository root folder. Folder names within a path should be separated by forward slashes.
`-t`	Optional switch for specifying the default type of all the files listed after it. The default type is assigned to documents when the document type cannot be determined by the file extension. Valid values are `.mfl`, `.dtd`, `.class`, `.xsl`, and `.xsd`.

Table 5-1 Options for the Import Command

Option	Description
`files`	Specifies one or more filenames to be imported. Wildcard characters supported by the shell or operating system can be used.

All options take effect at the point in the command line at which they are encountered and remain in effect until overridden by another option. For example, the following command imports all `.dtd`, `.class`, and `.mfl` files in the current directory, but enables verbose mode only while class files are being imported.

```
java com.bea.wlxt.repository.Import *.dtd -v *.class -n *.mfl
```

The type of an imported document is derived from the extension in the document's filename, as shown in the following table.:

Table 5-2 Supported Document Types and Extensions

File Extension	Document Type Assigned
`.dtd`	DTD
`.xsd`	XML Schema
`.mfl`	MFL
`.class`	Java class
`.xsl`	Extensible Stylesheet Language
Any other extension	Default type (defaults to MFL)

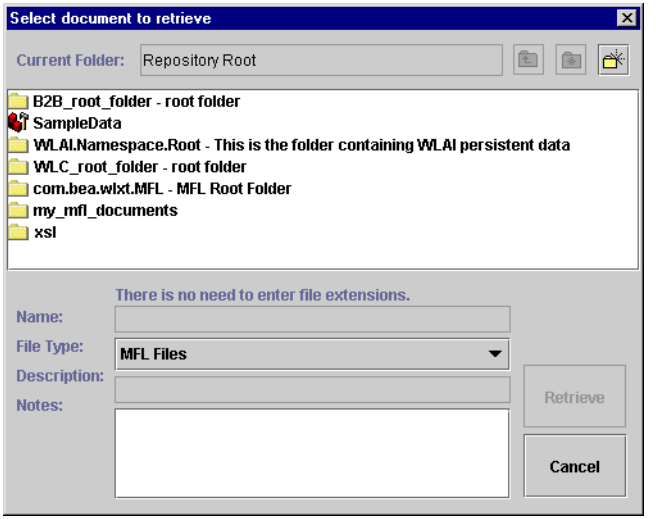
Using the Repository Retrieve and Store Dialog Boxes

The following dialog boxes are used to retrieve and store repository documents:

- Select document to retrieve dialog box
- Store Document dialog

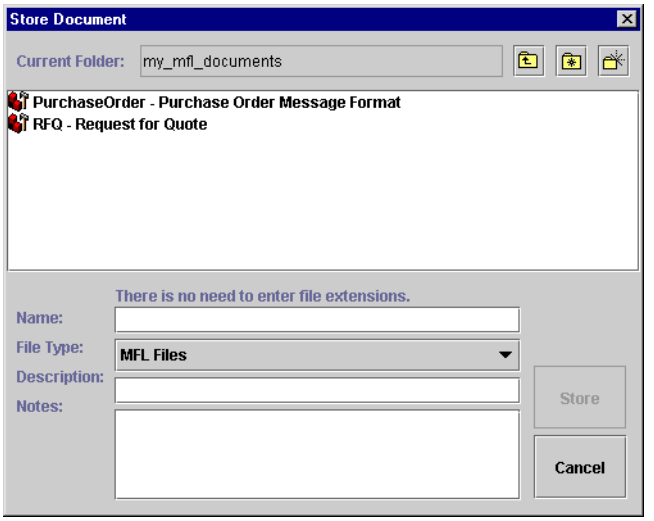
The following figure shows the Select document to retrieve dialog box.

Figure 5-4 Select Document to Retrieve Dialog Box



The following figure shows the Store Document dialog box. This dialog box differs from the Select document to retrieve dialog box in only one respect: it allows you to enter a new document name, description, and notes.

Figure 5-5 Store Document Dialog Box



The following table describes the elements of the Select document to retrieve and Store document dialog boxes.

Table 5-3 Retrieve and Store Dialog Box Elements

Field, Icon, or Button	Definition
Current Folder field	Contains the name of the current repository folder.
Up Folder icon	Click to move up to the parent of the current folder if the current folder is not the root folder of the repository.
Root Folder icon	Click to return to the root folder of the repository.
New Folder icon	Click to create a new folder in the current folder. This icon is disabled if the repository does not support folders.
Contents field	Lists the folders and MFL documents in the current folder. Each entry in the list is prefixed by an icon indicating the type of object. If you select an entry in the list, the information associated with it is displayed in the Name, Description, and Notes fields. If you double-click a folder it becomes the current folder. If you double-click an MFL document in the Select document to retrieve dialog box, the document is opened in Format Builder.

Table 5-3 Retrieve and Store Dialog Box Elements (Continued)

Field, Icon, or Button	Definition
Name field	The name of the folder or document.
Description field	A description of the folder or document.
Notes field	Notes associated with to the folder or document.
Retrieve button/Store button	In the Select document to retrieve dialog box, click Retrieve to open the selected document in Format Builder. In the Store Document dialog box, click Store to save the active document to the repository. If you click the Retrieve or Store button while a folder is selected, that folder becomes the current folder.
Cancel button	Click to close the dialog box without applying changes.

Using the Shortcut Menu

When you right-click a folder or document in the Select document to retrieve or Store Document dialog box, a shortcut menu is displayed that allows you to rename, delete, or view and update the properties of the selected object. The following figure shows the shortcut menu.

Figure 5-6 Store Document Shortcut Menu

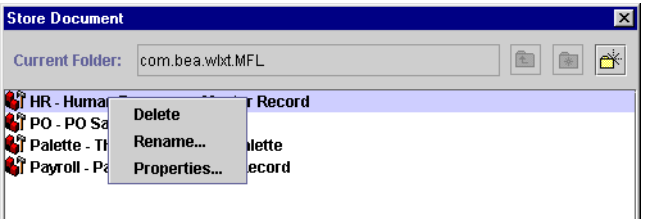


Table 5-4 Shortcut Menu Commands

Command	Description
Delete	Displays the Confirm Delete dialog box. Click Yes to delete the object or No to cancel the operation.

Table 5-4 Shortcut Menu Commands

Command	Description
Rename	Displays the Rename Document dialog box. Enter the new name and click OK to rename the document, or click Cancel to cancel the operation.
Properties	Displays the Modify Properties dialog box. Modify the Description and Notes fields as required, and then click OK to update or Cancel to cancel the operation.

CHAPTER

6 Using the Run-Time Component

WebLogic Integration supports run-time data translation through the `com.bea.wlxt` class, which we refer to, in this chapter, as *the Java class*. The Java class provides various methods that can be used to translate data between binary and XML formats. It can be deployed in an EJB using WebLogic Server, invoked from a business process management (BPM) workflow, or integrated into any Java application.

The data translation Java class provides several `parse()` methods that translate binary data into XML. The data translation run-time component also provides several `serialize()` methods that translate XML data to a binary format. Binary data formats are described via MFL documents. WebLogic Integration uses MFL documents to read and write binary data to or from XML. MFL documents are specified by a URL in a `parse()` or `serialize()` method.

The following sections provide code samples that show how to use the WebLogic Integration run-time data translation tools to parse binary data into XML, and to serialize XML into binary:

- Binary to XML
- XML-to-Binary Conversion
- XML-to-XML Transformation

Binary to XML

The following code listing uses the `parse()` method to parse a binary data file into XML.

Listing 6-1 Sample Binary-to-XML Parse() Method

```
1 import com.bea.wlxt.*;
2 import org.w3c.dom.Document;
3 import java.io.FileInputStream;
4 import java.net.URL;
5
6 public class Example
7 {
8  public static void main(String[] args)
9  {
10   try
11   {
12    WLXT wlxt = new WLXT();
13    URL mflDocumentName = new URL("file:mymfl.mfl");
14    FileInputStream in = new FileInputStream("mybinaryfile");
15
16    Document doc = wlxt.parse(mflDocumentName, in, null);
17    String xml = wlxt.getXMLText(doc, 0, 2);
18    System.out.println(xml);
19   }
20    catch (Exception e)
21  {
22    e.printStackTrace(System.err);
23  }
24 }
25}
```

Note the following sequence in this code:

1. In line 12 a new instance of the Java class is instantiated.
2. In line 13 a Uniform Resource Locator (URL) is created for an MFL file that was created earlier with Format Builder.

3. In lines 14 and 15 a `FileInputStream` is created for binary data in `mybinaryfile`.
4. In line 16 the URL for the MFL document and the stream of binary data are passed into the parse method.
5. From line 17 the parse method converts the binary data into an instance of a W3C Document object which is then converted to XML text via the `getXMLText()` method. (As an alternative, the parse method can also convert the binary data into an instance of a W3C Document object and then manipulate that object directly via the W3C DOM API.)

Generating XML with a Reference to a DTD

WebLogic Integration's data translation tools include `parse()` methods that enable you to include a reference to a Document Type Definition (DTD) or an XML Schema in the XML document generated by either. The following listing illustrates this capability.

Listing 6-2 Example of Generating XML with a DTD Reference

```
1 import com.bea.wlxt.*;
2 import org.w3c.dom.Document
3 import java.io.FileInputStream;
4 import java.net.URL;
5
6 public class Example2
7 {
8  public static void main(String[] args)
9  {
10   try
11   {
12     WLXT wlxt = new WLXT();
13     URL mflDocumentName = new URL("file:mymfl.mfl");
14     FileInputStream in = new FileInputStream("mybinaryfile");
15
16     Documentdoc=wlxt.parse(mflDocumentName,in,"mydtd.dtd",
17     null);String xml = wlxt.getXMLText(doc, 0, 2);
18     System.out.println(xml);
19   }
20     catch (Exception e)
21   {
```

```
22      e.printStackTrace(System.err);
23    }
24   }
25 }
```

The only difference between Listing 6-1 and Listing 6-2 occurs in line 16: a different parse method is invoked that allows a DTD file (`mydtd.dtd`) to be specified, so that a reference to it will be included in the resulting XML document. The following example shows the `DOCTYPE` statement in the resulting XML document that contains the reference to `mydtd.dtd`:

```
<?xml version="1.0"?>
<!DOCTYPE someRootNode SYSTEM 'mydtd.dtd'>
```

A similar parse method allows the resulting XML to refer to an XML Schema.

Specifying a Debug Writer

Every `parse()` method supported by WebLogic Integration allows a PrintWriter to be passed in as the last parameter. If this parameter is not null, WebLogic Integration prints debug messages to the specified PrintWriter. Such printing is helpful if the MFL document and the binary data do not agree and you need to debug the translation.

If you do not want to print debug messages, pass in null for this parameter, as shown in the previous listings.

Listing 6-3 Example of Passing in a Debug Writer

```
1 import com.bea.wlxt.*;
2 import org.w3c.dom.Document
3 import java.io.FileInputStream;
4 import java.io.PrintWriter;
5 import java.net.URL;
6
7 public class Example3
8 {
9  public static void main(String[] args)
10 {
11   try
```

```
12   {
13     WLXT wlxt = new WLXT();
14     URL mflDocumentName = new URL("file:mymfl.mfl");
15      FileInputStream in = new FileInputStream
16     ("mybinaryfile");
17     Document doc=wlxt.parse(mflDocumentName,in,new
       PrintWriter(System.out,true));
18     String xml = wlxt.getXMLText(doc, 0, 2);
19     System.out.println(xml);
20   }
21     catch (Exception e)
22   {
23     e.printStackTrace(System.err);
24   }
25  }
26 }
```

In line 17, as a last parameter to the `parse()` method, a PrintWriter object is created from the `System.out` PrintStream. The creation of this object causes debug messages such as the following to be displayed on the console.

Listing 6-4 Debug Output

```
Parsing FieldFormat NAME at offset 0
  Field NAME Found delimiter [;]
  Field NAME type String offset 0 value=[John Doe]
Done FieldFormat NAME
Group PAYINFO repeat until delim=[*]
  Parsing 1st instance of StructFormat PAYINFO at offset 18
    Parsing FieldFormat PAYDATE at offset 18
.
.
.
```

XML-to-Binary Conversion

The following sample code listing shows how to use WebLogic Integration tools to convert XML text to binary format.

Listing 6-5 Sample XML-to-Binary Conversion

```
1 import com.bea.wlxt.*;
2 import java.io.FileInputStream;
3 import java.io.FileOutputStream;
4 import java.net.URL;
5
6 public class Example4
7 {
8  public static void main(String[] args)
9  {
10   try
11   {
12    WLXT wlxt = new WLXT();
13    URL mflDocumentName = new URL("file:mymfl.mfl");
14    FileInputStream in = new FileInputStream("myxml.xml");
15    FileOutputStream out = new FileOutputStream("mybinaryfile");
16
17    wlxt.serialize(mflDocumentName, in, out, null);
18    out.close();
19   }
20   catch (Exception e)
21   {
22       e.printStackTrace(System.err);
23   }
24  }
25 }
```

Note the following sequence in this code:

1. In line 12 a new instance of the Java class is created.
2. In line 13 a URL is created for an MFL file, and a `FileInputStream` is created for a file containing XML text.

3. A `FileOutputStream` is also instantiated to store the binary data that will be produced by the XML-to-binary translation.
4. In line 17 the `serialize()` method is invoked to translate the XML data in `FileInputStream` 'in' `(myxml.xml)` to the binary format described in '`mymfl.mfl`'. The resulting binary data is written to the FileOutputStream 'out' (which is the file '`mybinaryfile`').

Converting a Document Object to Binary Format

The following sample listing shows how to convert a W3C Document object to a binary format.

Listing 6-6 Converting a Document Object to Binary Format

```
1  import com.bea.wlxt.*;
2  import java.io.FileOutputStream;
3  import java.net.URL;
4
5  import org.w3c.dom.Document;
6
7  import org.apache.xerces.parsers.DOMParser;
8
9  public class Example5
10  {
11    public static void main(String[] args)
12    {
13       // Parse XML into a Document object
14       Document doc = null;
15       try
16       {
17         DOMParser parser = new DOMParser();
18         parser.parse("myxml.xml");
19         doc = parser.getDocument();
20       }
21       catch (Exception e)
22       {
23         e.printStackTrace(System.err);
24         System.exit(1);
25       }
26
27       try
```

```
28        {
29          WLXT wlxt = new WLXT();
30          URL mflDocumentName = new URL("file:mymfl.mfl");
31          FileOutputStream out = new
            FileOutputStream("mybinaryfile");
32
33          wlxt.serialize(mflDocumentName, doc, out, null);
34          out.close();
35        }
36        catch (Exception e)
37        {
38          e.printStackTrace(System.err);
39        }
40    }
41  }
```

This example shows how to specify a Document object to the `serialize()` method of the Java class. This technique is useful when your application already has XML in the form of a Document object, or when it has created a Document object using the DOM API. Lines 14 through 25 convert the XML text in the file `myxml.xml` to a Document object using an XML parser. This Document object is passed, in line 33, to convert it to the binary format specified by the MFL file `mymfl.mfl`.

Specifying a Debug Writer

The serialize methods also support specification of a PrintWriter parameter for the logging of debug messages. The following code example shows how to invoke the serialize method with a PrintWriter object:

```
wlxt.serialize(mflDocumentName, in, out, new
   PrintWriter(System.out, true));
```

This invocation causes the display, on the console, of debug messages such as the following.

Listing 6-7 Debug Output

```
Processing xml and mfl nodes tcp1
Processing xml node NAME
Checking MFL node NAME
```

```
Found matching MFL node NAME
Writing field NAME value John Doe
Processing xml node PAYINFO
Checking MFL node PAYINFO
```

XML-to-XML Transformation

The WebLogic Integration data translation tools also include methods for transforming XML through XSLT, a language designed for such conversions. The Java class provides `transform()` methods that apply an *XSLT stylesheet* to an XML document. An XSLT stylesheet is an XML document that describes transformations to be performed on the nodes of an XML document. Using a stylesheet, you can transform an XML document into HTML, PDF, or another XML dialect.

The following sample code listing shows how to transform an XML document using one of the methods provided by the Java class.

Listing 6-8 XML-to-XML Transformation

```
1  import com.bea.wlxt.*;
2  import java.io.FileInputStream;
3  import java.io.FileOutputStream;
4  import java.net.URL;
5
6  import org.xml.sax.InputSource;
7
8  public class Example7
9  {
10    public static void main(String[] args)
11    {
12
13       try
14       {
15         WLXT wlxt = new WLXT();
16         URL stylesheet = new URL("file:mystylesheet.xsl");
17         FileInputStream in = new FileInputStream("myxml.xml");
18         FileOuputStream out = new FileOutputStream
19         ("myoutputfile")
20
```

```
21          wlxt.transform(new InputSource(in), out, stylesheet);
22
23          out.close();
24        }
25        catch (Exception e)
26        {
27          e.printStackTrace(System.err);
28        }
29    }
30  }
```

Note the following sequence in this code:

1. In line 15 an instance of WLXT is created.
2. In line 16 a URL is created for an XSLT stylesheet created earlier.
3. In line 17 a `FileInputStream` is created for a file containing XML text.
4. A `FileOutputStream` is also created for the text that results from the XSLT transformation.
5. In line 21 a `transform()` method of the Java class is invoked to transform the XML in the file `myxml.xml`, according to the `mystylesheet.xsl` stylesheet. The output of the transformation is written to the `myoutputfile` file.

Initialization Methods

The Java class provides several methods to *preprocess* MFL documents and XSLT stylesheets. Once these documents are preprocessed, they are cached internally and reused when referenced in a `parse()`, `serialize()`, or `transform()` method. Preprocessing greatly improves the performance of these methods, because the MFL document or XSLT stylesheet has already been processed and cached. This technique is particularly useful when data translation is used in an EJB or servlet in which the same MFL documents or XSLT stylesheets are used repeatedly.

The Java class provides two `init()` methods that take either a `java.util.Properties` object or the filename of a Properties file as a parameter. These `init()` methods retrieve the `WLXT.stylesheets` and `WLXT.MFLDocuments` properties from the Properties object. Each property is expected to contain a comma-delimited list of documents to be preprocessed and cached. When these documents are later referenced in a `parse()`, `serialize()`, or `transform()` method, the preprocessed version is retrieved from the cache.

The following sample listing shows how to use an `init()` method to initialize an instance of the Java class.

Listing 6-9 myconfig.cfg Properties File

```
WLXT.MFLDocuments=file:mymfl.mfl
WLXT.stylesheets=file:mystylesheet.xsl
```

Listing 6-10 Sample Source Code for init() Method Using myconfig.cfg File

```
1  import com.bea.wlxt.*;
2  import java.io.FileInputStream;
3  import java.io.FileOutputStream;
4  import java.net.URL;
5
6  import org.xml.sax.InputSource;
7  import org.w3c.dom.Document;
8
9  public class Example8
10  {
11    public static void main(String[] args)
12    {
13
14      WLXT wlxt = null;
15
16      // Initialize WLXT with a properties file
17      try
18      {
19        wlxt = new WLXT();
20        wlxt.init("myconfig.cfg");
21      }
22      catch (Exception e)
23      {
24        e.printStackTrace(System.err);
```

```
25      }
26
27      // Parse binary data into XML
28      Document doc = null;
29      try
30      {
31        URL mflDocumentName = new URL("file:mymfl.mfl");
32       FileInputStream in = new FileInputStream("mybinaryfile");
33
34        doc = wlxt.parse(mflDocumentName, in, null);
35      }
36      catch (Exception e)
37      {
38        e.printStackTrace(System.err);
39      }
40
41      try
42      {
43        URL stylesheet = new URL("file:mystylesheet.xsl");
44        FileOutputStream out = new FileOutputStream
45        ("myoutputfile")
46
47        wlxt.transform(doc, out, stylesheet);
48
49        out.close();
50      }
51      catch (Exception e)
52      {
53        e.printStackTrace(System.err);
54      }
55    }
56  }
```

In line 20 the `init()` method causes the object to preprocess the documents listed in the `myconfig.cfg` file. Therefore two documents, the MFL document and the stylesheet, are processed and cached inside the object before being specified as follows:

- The MFL document is preprocessed before being specified in the `parse()` method in line 34.
- The stylesheet is preprocessed before being specified in the invocation of the `transform()` method in line 46.

Java API Documentation

For complete reference information about using the Java class, see `com.bea.wlxt` in the WebLogic Integration Javadoc. The Javadoc can also be found in the `docs\apidocs` subdirectory of your WebLogic Integration installation.

Run-Time Plug-In to Business Process Management

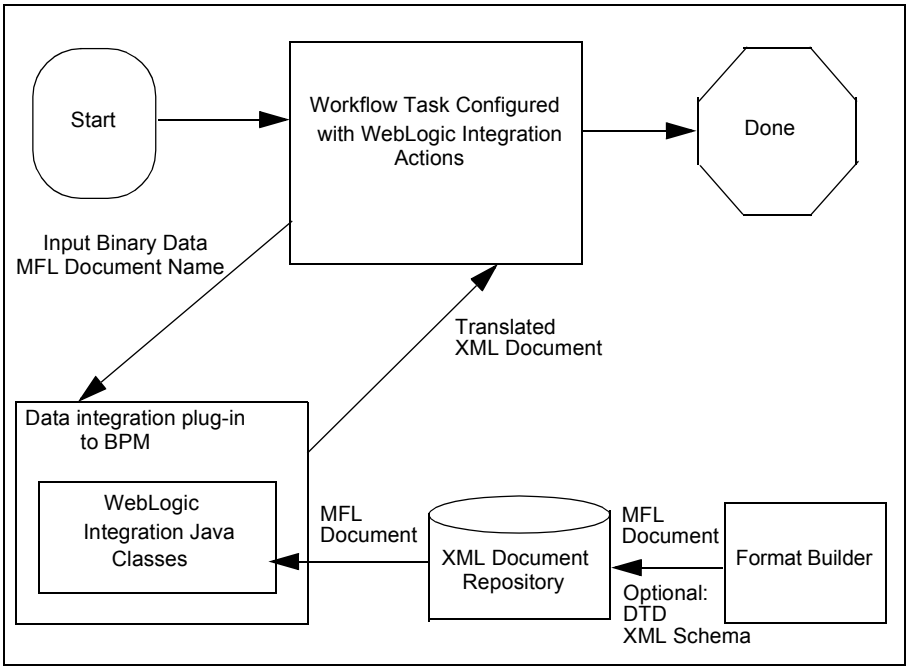
The data integration plug-in for business process management (BPM) facilitates the exchange of information between applications by supporting translations of data from binary formats used on legacy systems to XML, and vice versa. In other words, the BPM actions provided by the data integration plug-in allow you to access XML-to-binary and binary-to-XML translation capabilities.

In addition, the data integration plug-in provides the following features:

- Event data processing in binary format
- In-memory caching of MFL documents and translation object pooling (both of which boost performance)
- `BinaryData` variable type that can be used to edit and display binary data
- Export of entirely self-contained workflow definition packages
- Execution in a clustered WebLogic Server environment

The following illustration shows the relationship between the WebLogic Integration data integration tools and BPM.

Figure 6-1 Run-Time Plug-In to BPM



A Supported Data Types

WebLogic Integration supports the following data types:

- MFL Data Types
- COBOL Copybook Importer Data Types
- C Structure Importer from Importing Metadata

This section describes them.

MFL Data Types

Table A-1 lists the MFL data types supported by WebLogic Integration. These types are specified in the `type` attribute of a FieldFormat element.

Table A-1 Supported MFL Data Types

Data Type	Description	Example
Binary (Base64 encoding)	Any character value accepted. Requires a length, length field, delimiter, or a delimiter field. Resulting XML data for this field is encoded using base-64.	N/A
Binary (Hex encoding)	Any character value accepted. Requires a length, length field, delimiter, or delimiter field. Resulting XML data for this field is encoded using base-16.	N/A

Table A-1 Supported MFL Data Types (Continued)

Data Type	Description	Example
DateTime: *MM/DD/YY hh:mm*	String defining a date and time	01/22/00 12:24
DateTime: *MM/DD/YY hh:mi* AM	String defining a date and time	01/22/00 12:24 AM
DateTime: *MM/DD/YY hh:mm:ss*	String defining a date and time	01/22/00 12:24:00
DateTime: *MM/DD/YY hh:mm:ss* AM	String defining a date and time	01/22/00 12:24:00 AM
DateTime: *DD/MM/YY hh:mm*	String defining a date and time	22/01/00 12:24
DateTime: *DD/MM/YY hh:mm* AM	String defining a date and time	22/01/00 12:24 AM
DateTime: *DD/MM/YY hh:mm:ss*	String defining a date and time	22/01/00 12:24:00
DateTime: *DD/MM/YY hh:mm:ss* AM	String defining a date and tim	22/01/00 12:24:00 AM
DateTime: *MMDDYYhhmm*	A string of numeric digits defining a date and time	0122001224
DateTime: *YYYYMMDDhhmmss*	A 14-byte numeric string of the format *YYYYMMDDHHMISS*. A Base data type may be specified.	
DateTime: *MMDDYYhhmmss*	A string of numeric digits defining a date and time	012200122400
Date: *DDMMMYY*	String defining a date	22JAN00
Date: *DDMMMYYYY*	String defining a date	22JAN2000
Date: *DD/MM/YY*	String defining a date	22/01/00
Date: *DD/MM/YYYY*	String defining a date	22/01/2000
Date: *DD-MMM-YY*	String defining a date	22-JAN-00
Date: *DD-MMM-YYYY*	String defining a date	22-JAN-2000
Date: *MMDDYY*	Six-digit numeric string defining a date	012200

Table A-1 Supported MFL Data Types (Continued)

Data Type	Description	Example
Date: *MMDDYYYY*	Eight-digit numeric string defining a date	01222000
Date: *MM/DD/YY*	String defining a date	01/22/00
Date: *MM/DD/YYYY*	String defining a date	01/22/2000
Date: *MMM-YY*	String defining a date	JAN-00
Date: *MMM-YYYY*	String defining a date	JAN-2000
Date: *MMMYY*	String defining a date	JAN00
Date: *MMMYYYY*	String defining a date	JAN2000
Date: *MMMDDYYYY*	String defining a date	JAN222000
Date: *YYYYMMDD*	Eight-byte numeric string of the format *YYYYMMDD*. Base data of String or EBCDIC type may be specified to indicate the character encoding.	
Date: Wed Nov 15 10:55:37 CST 2000	Default date format of the Java platform	'WED NOV 15 10:55:37 CST 2000'
EBCDIC	A string of characters in IBM Extended Binary Coded Decimal Interchange Code. Requires a length, length field, delimiter, or delimiter field.	
Filler	Sequence of bytes that is not translated to XML. This data field is skipped over when binary data is being translated to XML. When XML is being translated to binary data, this field is written to the binary output stream as a sequence of spaces.	

Table A-1 Supported MFL Data Types (Continued)

Data Type	Description	Example
FloatingPoint: 4-byte, Big-Endian	Four-byte big endian floating point number that conforms to IEEE Standard 754	
FloatingPoint, 4-byte, Little-Endian	Four-byte little endian floating point number that conforms to IEEE Standard 754	
FloatingPoint: 8-byte, Big-Endian	Eight-byte big endian floating point number that conforms to IEEE Standard 754	
FloatingPoint: 8-byte, Little-Endian	Eight-byte little endian floating point number that conforms to IEEE Standard 754	
Floating point IBM 4-byte IBM 8-byte	IBM mainframe floating point	
Integer: Signed, 1-byte	One-byte signed integer	'56' is 0x38
Integer: Unsigned, 1-byte	One-byte unsigned integer	'128' is 0x80
Integer: Signed, 2-byte, Big-Endian	Signed two-byte integer in big endian format	'4660' is 0x1234
Integer: Signed, 4-byte, Big-Endian	Signed four-byte integer in big endian format	'4660' is 0x00001234
Integer: Signed, 8-byte, Big-Endian	Signed eight-byte integer in big endian format	'4660' is 0x0000000000001234
Integer: Unsigned, 2-byte, Big-Endian	Unsigned two-byte integer in big endian format	'65000' is 0xFDE8
Integer: Unsigned, 4-byte, Big-Endian	Unsigned four-byte integer in big endian format	'65000' is 0x0000FDE8
Integer: Unsigned, 8-byte, Big-Endian	Unsigned eight-byte integer in big endian format	'65000' is 0x000000000000FDE8

Table A-1 Supported MFL Data Types (Continued)

Data Type	Description	Example
Integer: Signed, 2-byte, Little-Endian	Signed two-byte integer in little endian format	'4660' is 0x3412
Integer: Signed, 4-byte, Little-Endian	Signed four-byte integer in little endian format	'4660' is 0x34120000
Integer: Signed, 8-byte, Little-Endian	Signed eight-byte integer in little endian format	'4660' is 0x3412000000000000
Integer: Unsigned, 2-byte, Little-Endian	Unsigned two-byte integer in little endian format	'65000' is 0xE8FD
Integer: Unsigned, 4-byte, Little-Endian	Unsigned four-byte integer in little endian format	'65000' is 0xE8FD0000
Integer: Unsigned, 8-byte, Little-Endian	Unsigned eight-byte integer in little endian format	'65000' is 0xE8FD000000000000
Literal	A literal value determined by the contents of the value attribute. When binary data is translated to XML, the presence of the specified literal in the binary data is verified by WebLogic Integration. The literal is read, but it is not translated into XML data. When XML data is translated to a binary format, and a literal is defined as part of that format, WebLogic Integration writes the literal in the resulting binary byte stream.	
Numeric	String of characters containing only digits (0-9). Requires a length, length field, delimiter, or delimiter field.	

Table A-1 Supported MFL Data Types (Continued)

Data Type	Description	Example
Packed Decimal: Signed	IBM signed packed format. Requires a length, length field, delimiter, or delimiter field to be specified. The length or length field should specify the size of this field in bytes.	
Packed Decimal: Unsigned	IBM unsigned packed format. Requires a length, length field, delimiter, or delimiter field to be specified. The length or length field should specify the size of this field in bytes.	
String	Set of characters. Requires a length, length field, delimiter, or delimiter field. If no length is specified, a length field or delimiter is defined for a String data type, and a delimiter of `\x00` (a NUL character) is assumed.	
String: NUL terminated	String of characters, optionally NUL (`\x00`) terminated, residing in a fixed-length field. This field type requires a length attribute or length field that determines the amount of data read for the field. This data is then examined for a NUL delimiter. If a delimiter is found, the data after the delimiter is discarded. Otherwise, the fixed-length data is used as the value of the field.	
Time: *hhmmss*	String defining a time	122400
Time: *hh:mm* AM	String defining a time	12:24 AM
Time: *hh:mm*	String defining a time	12:24
Time: *hh:mm:ss* AM	String defining a time	12:24:00 AM

Table A-1 Supported MFL Data Types (Continued)

Data Type	Description	Example
Time: *hh:mm:ss*	String defining a time	12:24:00
Zoned Decimal: Leading sign	IBM signed zoned decimal format where the sign indicator is in the first nibble. Requires a length, length field, delimiter, or delimiter field to be specified. The length or length field should specify the size of this field in bytes.	
Zoned Decimal: Leading separate sign	IBM signed zoned decimal format where the sign indicator is in the first byte. The first byte only contains the sign indicator and is separated from the numeric value. Requires a length, length field, delimiter, or delimiter field to be specified. The length or length field should specify the size of this field in bytes.	
Zoned Decimal: Signed	IBM signed zoned decimal format. Requires a length, length field, delimiter, or delimiter field to be specified. The length or length field should specify the size of this field in bytes.	
Zoned Decimal: Trailing separate sign	IBM signed zoned decimal format in which the sign indicator is in the last byte. The last byte contains only the sign indicator; it is separated from the numeric value. Requires a length, length field, delimiter, or delimiter field to be specified. The length or length field should specify the size of this field in bytes.	

Table A-1 Supported MFL Data Types (Continued)

Data Type	Description	Example
Zoned Decimal: Unsigned	IBM unsigned zoned decimal format. Requires a length, length field, delimiter, or delimiter field to be specified. The length or length field should specify the size of this field in bytes.	

COBOL Copybook Importer Data Types

The following table lists the COBOL data types and the support provided by the Importer.

Table A-2 COBOL Data Types

COBOL Type	Support
BLANK WHEN ZERO (zoned)	Supported
COMP-1, COMP-2 (float)	Supported
COMP-3, PACKED-DECIMAL	Supported
COMP, COMP-4, BINARY (integer)	Supported
COMP, COMP-4, BINARY (fixed)	Supported
COMP-5, COMP-X	Supported
DISPLAY (alphanumeric)	Supported
DISPLAY numeric (zoned)	Supported
Edited alphanumeric	Supported
Edited float numeric	Supported

Table A-2 COBOL Data Types (Continued)

COBOL Type	Support
Edited numeric	Supported
Group record	Supported
INDEX	Supported
JUSTIFIED RIGHT	Ignored
OCCURS (fixed array)	Supported
OCCURS DEPENDING (variable-length)	Supported
OCCURS INDEXED BY	Ignored
OCCURS KEY IS	Ignored
POINTER	Supported
PROCEDURE-POINTER	Supported
REDEFINES	Supported
SIGN IS LEADING SEPARATE (zoned)	Supported
SIGN IS TRAILING (zoned)	Supported
SIGN IS TRAILING SEPARATE (zoned)	Supported
SIGN IS LEADING (zoned)	Supported
SYNCHRONIZED	Ignored
66 RENAMES	Ignored
66 RENAMES THRU	Ignored
77 level	Supported
88 level (condition)	Ignored

Support for these data types is limited. Data presented in either of the following formats is converted to an unsigned four-byte integer type:

- `05 pic 9(5) comp-5`
- `05 pic 9(5) comp-x`

Data presented in the following formats generates errors:

- `05 pic X(5) comp-5`
- `05 pic X(5) comp-x`

In these samples, `pic9(5)` can be substituted for `pic x(5)`.

The following table defines the three levels of support for these data types.

Table A-3 Data Type Support Levels

Level of Support	Definition
Supported	Data type is parsed correctly by the importer and converted to a message format field or group.
Unsupported	Data type is not supported; the importer reports an error when the copybook is imported.
Ignored	Data type is parsed and a comment is added to the message format. No corresponding field or group is created.

Some vendor-specific extensions are not recognized by the importer. Any copybook statement that conforms to ANSI standard COBOL, however, is parsed correctly by the importer. The importer's default data model, which is based on the IBM mainframe model, can be changed in the Format Builder to compensate for the endian nature of characters set and data.

When importing copybooks, the importer may identify fields generically, although those fields can easily be identified as specific data types. For this reason, the copybook importer creates comments for each field found in the copybook. This information is useful when you are editing the MFL data, helping you to improve upon the formats used in the original copybook. For example, suppose an original copybook contains the following entry:

```
05 birth-date    picxx/xx/xx
```

After the copybook is imported, this entry is shown in a field of type `EBCDIC` with a length of 8. Close inspection reveals that this entry is intended as a date that can be formatted in either of the following ways:

- *MM/DD/YY*
- *DD/MM/YY*

C Structure Importer from Importing Metadata

The C struct importer does not parse files containing anonymous unions, bit fields, or in-line assembler code. The following samples of unsupported structures are taken from the preprocessor output of a `hello.c` file that contained a `#include <windows.h>` statement:

- Anonymous unions

```
#line 353 "e:\\program files\\microsoft visual
studio\\vc98\\include\\winnt.h"
typedef union_LARGE_INTEGER{
  struct {
    DWORD LowPart;
    LONG HighPart;
  };
  struct {
    DWORD LowPart;
    LONG HighPart;
  } u;
#line 363 "e:\\program files\\microsoft visual
studio\\vc98\\include\\winnt.h"
    LONGLONG QuadPart;
} LARGE_INTEGER
```

- Bit fields

```
typedef struct_LDT_ENTRY {
  WORD LimitLow;
  WORD BaseLow;
  union {
    struct {
      BYTE BaseMid;
```

```
      BYTE Flags1;
      BYTE Flags2;
      BYTE BaseHi;
    } Bytes;
    struct
      DWORD BaseMid : 8;
      DWORD Type : 5;
      DWORD Dpl : 2;
      DWORD Pres : 1;
      DWORD LimitHi : 4;
      DWORD Sys : 1;
      DWORD Reserved_0 : 1;
      DWORD Default_Big : 1;
      DWORD Granularity : 1;
      DWORD BaseHi : 8;
    } Bits;
  } HighWord;
} LDT_ENTRY, *PLDT_ENTRY;
```

- Inline assembler code

```
_inline ULONGLONG
_stdcall
Int64ShrlMod32(
  ULONGLONG Value,
  DWORD ShiftCount
  )
{
  _asm {
   mov ecx, ShiftCount
   mov eax, dword ptr [Value]
   mov edx, dword ptr [Value+4]
   shrd eax, edx, cl
   shr edx, cl
  }
}
```

B Creating Custom Data Types

WebLogic Integration uses a metadata language called Message Format Language (MFL), based on XML, to describe the structure of binary data when data is being integrated. The Format Builder creates and maintains metadata as a data file, or an MFL document in the repository.

In MFL, the following metadata is used to describe a binary field:

- Data type
- Length/Delimiter
- Optional/Mandatory
- Default value
- Code Page encoding

Of this information, the data type is the most critical. The selected data type dictates which metadata attributes are valid and how they are interpreted.

The data integration component of WebLogic 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 translation 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.

Instructions for using user-defined types are provided in the following sections:

- Samples of User-Defined Types
- Registering User-Defined Types
- Creating User-Defined Types
- Configuring User-Defined Types for the Data Integration Plug-In
- User-Defined Type Coding Requirements
- Class com.bea.wlxt.mfl.MFLField

Samples of User-Defined Types

The following table provides describes the sample files installed with user-defined types. All directory names are relative to the WebLogic Integration samples directory (*SAMPLES_HOME*\integration) where *SAMPLES_HOME* is the samples directory in your WebLogic Platform installation..

Table B-1 Sample User-Defined Types

Directory	File	Description
samples\di\userdef	CapString.java	Source for a user data type that converts strings to upper case.
samples\di\userdef	Dddmmyy.java	Source for a User Defined Type that supports a European date format.
samples\di\userdef	Makefile	Make file for building the sample source.
samples\di\userdef	ParseUserDef.java	Source that shows installing the sample User Defined Type and using them with the run-time engine.

Table B-1 Sample User-Defined Types (Continued)

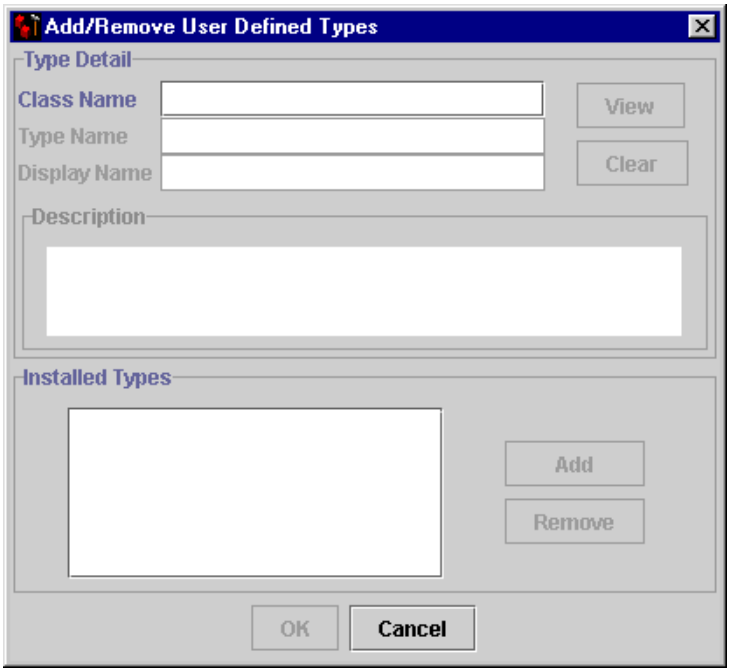
`samples\di\userdef`	`readme.txt`	Explains how to compile and run the sample.
`samples\di\userdef`	`sample.data`	Data for the `ParseUserData` sample.
`samples\di\userdef`	`sample.mfl`	The Message Format Language (MFL) file for the `ParseUserData` sample.

Registering User-Defined Types

Perform the following steps to register a new User Defined Type:

1. Start Format Builder by choosing Start→Programs→BEA WebLogic Platform 7.0→WebLogic Integration 7.0→Format Builder. The Format Builder main window displays.
2. Choose Tools→User Defined Types. The Add/Remove User Defined Types dialog box displays.

Figure B-1 Add/Remove User-Defined Types Dialog Box

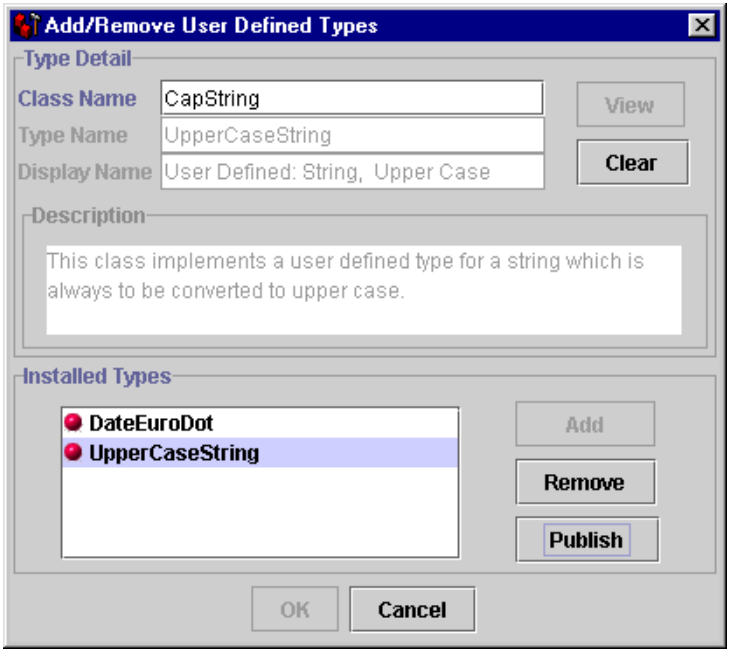


3. In the Class Name field, enter the name of the class implementing the specified type.

   Note: The class name entered must include any package name present in the definition of the module. Additionally, the named class must be found in the Format Builder `CLASSPATH`. The WebLogic Integration installation of the Format Builder creates a *`WLI_HOME`*`/ext` directory specifically for containing user-defined types. The *`WLI_HOME`* directory is the directory in which WebLogic Integration is installed.

4. Click View. Information about the requested class populates several display-only fields, as follows.

Figure B-2 Add/Remove User-Defined Types Dialog Box



- Type Name - returned by a call to `getTypeName()`
- Display Name - return value of `getDisplayName()` prefixed by the literal `User Defined:`.
- Description - returned by `getDescriptionText()`

If a requested class cannot be loaded or does not conform to the requirements for a user-defined type, an error message is displayed. Click OK to return to the Add/Remove User Defined Types dialog box.

For lists of the required and optional interface methods, and utility methods available for use with user-defined types, see the user-defined type coding requirements.

5. Once a valid user-defined type is displayed, click Add to make it available for use within the Format Builder.

After you define the new data type, the new Display Name is shown in the Type drop-down list on the detail window for a field. All user-defined types displayed in the Type drop-down list are prefixed with *User Defined:*, as shown in the DisplayName text field.

Because the Format Builder cannot know the exact type of data represented by a user-defined type, the `xsd::string` type is used to represent the user-defined type in the XML Schema generated to describe the content model of the XML output.

Use of this data type also affects the Format Tester. When data is being generated for an MFL document containing user-defined types, String data is generated for the corresponding fields. You must adjust the generated data so it can be parsed according to the user-defined type.

Creating User-Defined Types

The interface to the data translation engine is an API called by a Java program. Creating a new user-defined type for the process engine is accomplished via a static method of the `com.bea.wlxt.WLXT` class.

Installation of a user-defined type in the data translation engine is not persistent. When the current JVM process terminates, the user-defined type configuration information is discarded. As a result, clients using the stand-alone engine must install all user-defined types at the start of each program.

The following public functions are defined for user-defined types:

- `public static void addNewDataType(String` *`name`*`, Bintype` *`binType`*`)`
  - *`name`* specifies the name of the new data type in MFL.
  - *`binType`* refers to a new user-defined type object.
- `public static void removeDataType (String` *`name`*`)`

  `name` specifies the name of the data type to be removed.

The following example shows how to use these APIs to install and remove the `CapString` user-defined type:

```
import com.bea.wlxt.WLXT;
import com.bea.wlxt.binType.BinType;

// create data type object and install it
Bintype udt = new CapString();
WLXT.addNewDataType("UpperCaseString", udt);
                      .
                      .
                      .

//remove the udt installed above
WLXT.removeDataType("UpperCaseString");
```

Configuring User-Defined Types for the Data Integration Plug-In

The user-defined types used by the data integration plug-in for business process management (BPM) are stored in the WebLogic Integration repository as CLASS documents. At run time, the data integration plug-in loads user-defined type classes from the repository as required. In addition, the data integration plug-in exports the MFL and class files required to support the active template, allowing a template to be imported, intact, on another BPM instance.

At run time, the data integration plug-in retrieves both MFL documents and required user-defined type classes from the repository. Class documents may be placed in the repository by using one of the following methods:

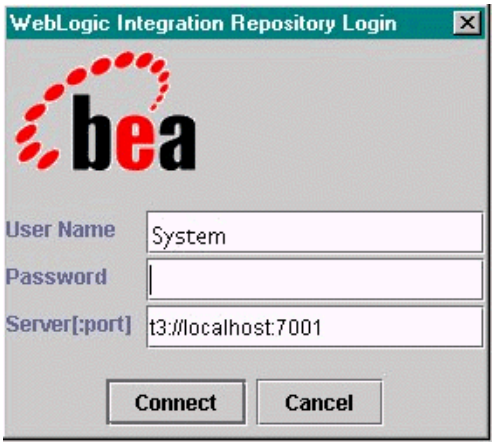
- Publishing user-defined types to the repository from the Format Builder
- Using the Repository Import utility

Publishing User-Defined Types to the Repository from the Format Builder

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

1. Start the Format Builder by choosing Start→Programs→BEA WebLogic Platform 7.0→WebLogic Integration 7.0→Format Builder. The Format Builder main window is displayed.
2. Choose Repository→Log In. The WebLogic Integration Repository Login dialog box is displayed.

Figure B-3 WebLogic Integration Repository Login Dialog Box



3. In the User Name field, enter the user ID specified for the connection.
4. In the Password field, enter the password specified for the connection.
5. In the Server[:port] field, enter the server name and port number.

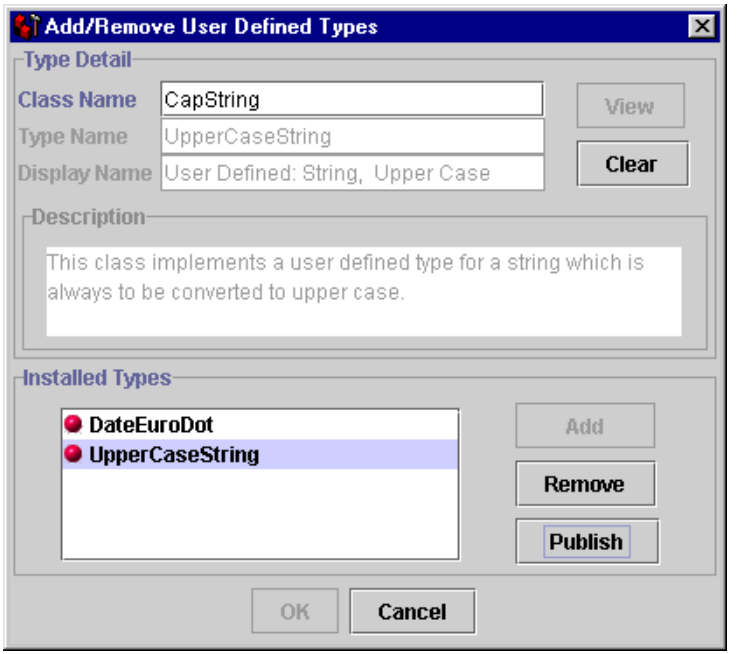
   Note: After three unsuccessful login attempts, the WebLogic Integration Repository Login dialog box is dismissed and a login failure message is displayed. To try again after a failure, choose Repository→Log In to redisplay the Repository Login dialog box.

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

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 a ball-shaped icon before the name of each installed user-defined type.

Figure B-4 Add/Remove User-Defined Types Dialog Box



The status of each user-defined type is indicated by the color of the icon.

Table B-2 Icon Color Description

If the icon is . . .	Then . . .
Green	The user-defined type has been published to the repository.
Yellow	The user-defined type has been published to the repository, but the local and repository versions of the class do not match.
Red	The user-defined type does not exist in the repository.

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

Publishing User-Defined Types to the Repository Using the Repository Import Utility

The repository import utility may be used to import Java class files, including data translation user-defined types. To use the import utility for this purpose, pass the name of the class file on the Import command line. For example, to import all the class files in the current directory:

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

Please note that any Java class file can be imported to the repository. This is not the case for user-defined type class files created in the Format Builder. This capability can be useful if, for example, a user-defined type relies on additional class files that do not extend the `com.bea.wlxt.bintype.Bintype` class. By using the repository import utility, you can put these utility classes in the repository, where they can be accessed by the repository class loader.

User-Defined Type Coding Requirements

User-defined types are required to extend the `com.bea.wlxt.bintype.Bintype` abstract class or one of the classes derived from it. The `Bintype` class provides the basic framework used by the data integration component of WebLogic Integration to interface with a data type. This class also provides utility routines that are useful for processing binary data types. In addition, two subclasses of `Bintype`, `BintypeDate` and `BintypeString`, offer additional utility routines for date and string types, respectively.

The following classes, along with the required and optional interface methods they provide, are available for user-defined types.

Class com.bea.wlxt.bintype.Bintype

Class `com.bea.wlxt.bintype.Bintype` consists of the following required, optional, and utility methods.

Required Interface Routines

The following interface methods are required when using the WebLogic Integration user defined data types utility.

`public String read(InputStream` *`byteStream`*`, MFLField` *`mflField`*`)throws BintypeException`

This routine is called to read a data value from a binary data stream. The read method should read the appropriate number of bytes for the data type, convert them to string representation, and return the string value. The *`mflField`* parameter is a reference to a `com.bea.wlxt.mfl.MFLField` object that describes the attributes of the data field.

If a user-defined type is unable to read the requested data element, it should throw a `com.bea.wlxt.bintype.BintypeException` with a text string that describes the error encountered.

`public void write(BintypeOutputStream byteStream, MFLField mflField, String value)throws BintypeException`

The write method is responsible for converting a string value into the data representation of the binary data type and writing it to a binary output stream.

If a user-defined type is unable to successfully write the requested data element, it should throw a `com.bea.wlxt.bintype.BintypeException` with a text string that describes the error encountered.

Optional Interface Routines

When you use the data translation user-defined data types utility, the following interface routines are optional:

`public boolean canBeFieldType()`

Returns true if the user-defined data type may be used as the type for a data field.

Default: true

`public boolean canBeLenFieldType()`

Returns true if the user-defined type may be used as the data type of a Length Field.

Default: true

`public boolean canBeTagFieldType()`

Returns true if the user-defined type may be used as the type of a Tag Field.

Default: true

`public boolean canBeDelimited()`

Returns true if the user-defined type supports delimited data values.

Default: false

`public boolean isFixedSize()`

Returns true if the user-defined type represents a fixed-size data value. A return of true from this method implies that the data type has an inherent fixed size.

Default: true

`public boolean isDateType()`

Returns true if the user-defined type represents a date data type.

Default: false

`public boolean isCutoffRequired()`

Returns true if the user-defined type is a date data type and requires a cutoff value for adjusting a two-digit year. This method is not used unless `isDateType()` returns true.

Default: false

`public boolean isCodepageOK()`

Returns true if the user-defined type supports code pages.

Default: false

`public boolean isValueOK()`

Returns true if the user-defined type supports an initial value attribute.

Default: false

`public boolean canHaveDecimalPlaces()`

Returns true if the user-defined type represents a number that may include decimal places.

Default: false

`public boolean canBePadded()`

Returns true if the user-defined type padding a fixed-length data value.

Default: false

`public boolean canBeTrimmed()`

Returns true if the user-defined type supports leading and trailing trimming.

Default: false

`public String getDescriptiontext()`

Returns a String that contains a text description of this data types function. This method should be implemented in a user-defined type for documentation purposes.

Default: Empty String

`public String getTypeName()`

Returns a String containing the data type name. This function is used with user-defined types and its return value is the type name used in MFL documents.

Default: The name of the class implementing the user-defined type.

`public String getDisplayName()`

Returns a String containing the Display Name for a data type name. This value appears in the Type drop-down list on the detail window for a field.

Default: The name of the class implementing the user-defined type.

Utility Interface Routines

When you use the data translation user-defined data types utility, the following utility interface routines are available:

`public static byte[] getBinaryBytes (String str)`

Converts a Java String into an array of binary bytes by taking the low order eight bits of each String character. No conversions are performed based on code pages.

`public static String makeString(byte[] b)`

Converts an array of binary bytes into a Java String value. No conversions are performed based on code pages.

`protected void reverseBytes (byte[] data)`

Reverses the order of bytes and an array of binary values.

`protected String readTag(InputStream byteStream, MFLField fld) throws BinTypeexception`

Reads and returns a tag value associated with a data field. This routine compares the tag value read with the expected value in the *fld* object and throws a `Bintypeexception` if they fail to match. If called for a *fld* that does not support tagged values, this method simply returns an empty string.

`protected int readlength(InputStream byteStream, MFLField fld) thorws BintypeException`

Reads and returns the length field associated with a data field. If called for a field that does not support a length field, it simply returns a zero.

`protected void writeTag(BintypeOutputStream byteStream, MFLField fld) throws Bintypeexception`

Writes the tag field associated with *fld* to the `ByteStream` supplied. If *fld* does not support a tag value this method simply returns without writing anything.

`protected void writeLength(BintypeOutputStream byteStream, MFLField fld, int fldLen) throws BintypeException`

Writes the length field associated with *fld* to the `byteStream` supplied. If *fld* does not support a length field this method simply returns without writing anything.

`protected byte[] readDelimitedField(InputStream byteStream, MFLField mflField) throws BintypeException`

Reads data from the supplied input stream until encountering one of the delimiters for field *mflField*. The value returned is the binary data read excluding the delimiter value. If *mflField* is defined as not having a delimiter this method returns a null without reading any data.

`protected String applyPadAndTrim(String value, MFLField fld, boolean applyPad, boolean applyTrim) throws BintypeException`

Applies pad and trim functions as defined for *fld* to the data passed as *value*. The *applyPad* and *applyTrim* parameters control whether padding, trimming, or both are performed. The return value is the result data after having the requested operations performed. By default this method does not do any truncation of the fields.

`protected String applyPadTrimAndTruncate(String value, MFLField fld, boolean applyPad, boolean applyTrim boolean applyTruncate) throws BintypeException`

Applies pad, trim, and truncate functions as defined for *fld* to the data passed as *value*. The *applyPad*, *applyTrim*, and *applyTruncate* parameters control whether padding, trimming, and truncation are performed. The return value is the result data after having the requested operations performed.

Class com.bea.wlxt.bintype.BintypeString

Class `com.bea.wlxt.bintype.BintypeString` consists of the following required, optional, and utility routines.

Required Interface Routines

Same as class `com.bea.wlxt.bintype.Bintype`.

Optional Interface Routines

Same as class `com.bea.wlxt.bintype.Bintype`.

Utility Interface Routines

Same as class `com.bea.wlxt.bintype.Bintype`, plus the following utility interface routines.

`protected String makeString(byte[] data, MFLField mflField) throws BintypeException`

This method converts an array of binary data into a Java String respecting any code page defined in *mflField*.

`protected byte[ ] readField(InputStream byteStream, MFLField mflField) throws Bintypeexception`

This method reads a value representing a String data type from a binary input string. All length/delimiter attributes defined in *mflField* are respected in extracting the returned data value.

`protected void writeField (BintypeOutputStream byteStream, MFLField mflField, String value, String codepage) throws BintypeException`

Writes a data value to the supplied *byteStream* respecting the attributes defined for *mflField* and the passed *codepage*. If *codepage* is passed as null, the default system code page is used.

`protected String trimBoth (String data, char trimChar)`

Trims a specific character from both ends of the supplied data and returns the resulting string.

`protected String trimLeading (String data, char trimChar)`

Trims a specific character from the front of the supplied data and returns the resulting string.

`protected String trimTrailing(String data, char trimChar)`

Trims a specific character from the end of the supplied data and returns the resulting string.

Class com.bea.wlxt.bintype.BintypeDate

Class `com.bea.wlxt.bintype.BintypeDate` consists of the following required, optional, and utility routines.

Required Interface Routines

Same as class `com.bea.wlxt.bintype.Bintype`.

Optional Interface Routines

Same as class `com.bea.wlxt.bintype.Bintype`.

Utility Interface Routines

Same as class `com.bea.wlxt.bintype.Bintype`, plus the following utility interface routines.

`protected static String readDate(String` *`date`*`, SimpleDateFormat` *`fmt`*`)`

Parses a date from the string parameter date according to the *`fmt`* format. The return value is a valid date in string format.

`protected static String readDate(String` *`date`*`, SimpleDateFormat` *`fmt`*`, MFLField` *`fld`*`, int` *`yearpos`*

Parses a date containing a two-digit year from the string parameter date, according to the *`fmt`* format.

`protected static String writeDate(String` *`date`*`, SimpleDateFormat` *`fmt`*`)`

Returns a date string suitable for output from the string parameter *`date`* using the *`fmt`* date format.

`protected static String rawDateToString(byte[ ]` *`data`*`, String` *`baseType`*`)`

Converts a date in raw binary format into a Java String using the base data type specified.

`protected static byte[ ] stringDateToRaw(String` *`date`*`, String` *`baseType`*`)`

Converts a date in Java String format into a binary array of bytes using the supplied base data type.

Class com.bea.wlxt.mfl.MFLField

The `MFLField` class is passed into all read and write methods for user-defined types. It encapsulates all the attributes defined for the field being read or written. `MFLField` supplies various methods that allow these attributes to be queried, and their respective values returned. Attributes that are not supported by a user-defined type are never included. For example, if a user-defined type returns false to the `isValueOK()` method, it is never passed to an `MFLField` object that returns true for the `MFLField.hasValue()` method.

Class `com.bea.wlxt.mfl.MFLField` supports the following MFLField methods:

`public String getName()`
: Returns the name of the data field.

`public String getType()`
: Returns the name of the data type for this field. This name is returned by the `getTypeName()` method of the user-defined type.

`public boolean hasBaseType()`
: Returns true if a base type is defined for the field. This method returns true only for date data types.

`public String getBaseType`
: Returns the name of the base data type for this field.

`public boolean hasDelimRef()`
: Returns true if a delimiter reference is defined for this field.

`public String getDelimRef()`
: Returns the name of the field that contains the delimiter value for this field.

`public boolean hasdelimRefValue()`
: Returns true if the delimiter reference field contains a value.

`public boolean hasDefaultValue()`
: Returns true if a default value is defined for this field.

`public String getDefaultValue()`
: Returns the default value for this field.

`public boolean hasPad()`
: Returns true if a pad value is defined for this field.

`public String getPad()`
Returns the pad value for this field.

`public boolean hasTrimLeading()`
Returns true if a leading trim value is defined for this field.

`public String getTrimLeading()`
Returns the leading trim value for this field.

`public boolean hasTrimTrailing()`
Returns true if a trailing trim value is defined for this field.

`public String getTrimTrailing()`
Returns the trailing trim value for this field.

`public boolean isOptional()`
Returns true if this field is defined as optional.

`public boolean hasCutoff()`
Returns true if a date cutoff value is defined for this field.

`public int getCutoff()`
Returns the value of the date cutoff defined for this field.

`public boolean hasLength()`
Returns true if an exact length is defined for this field.

`public int getLength()`
Returns the exact length defined for this field.

`public boolean hasDelim()`
Returns true if a delimiter value is defined for this field.

`public String getDelim()`
Returns the delimiter value defined for this field.

`public boolean hasValue()`
Returns true if an initial value is defined for this field.

`public String getValue()`
Returns the initial value defined for this field.

`public boolean hasCodepage()`
Returns true if a code page is defined for this field.

`public String getCodepage()`
Returns the name of the codepage defined for this field.

`public boolean hasTagField()`
Returns true if a tag field is defined for this field.

`public boolean hasLenField()`
Returns true if a length field is defined for this field.

`public boolean isTagBeforeLength()`
Returns true if the field tag value (if any) appears before the length field.

`public boolean hasDecimalPlaces()`
Returns true if decimal places are defined in this field.

`public int getDecimalPLaces()`
Returns the number of decimal places defined for this field.

`public String getPadType() { return(padType); }`
Returns the type of padding set for this field (`leading` or `trailing`)

`public boolean hasTruncateFirst() { return(0 != truncateFirst); }`
Returns true if *n* number of truncate characters at the start of the field are defined.

`public int getTruncateFirst() { return(truncateFirst); }`
Returns the number of truncate characters defined at the start of this field.

`public boolean hasTruncateAfter() { return(0 != truncateAfter); }`
Returns true if *n* number of truncate characters at the end of the field are defined.

`public int getTruncateAfter() { return(truncateAfter); }`
Returns the number of truncate characters defined at the end of this field.

C Running the Purchase Order Sample

The WebLogic Integration 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 data integration component of WebLogic Integration.

The following topics are discussed in this section:

- What Is Included in the Purchase Order Sample
- Prerequisite Considerations
- Performing Binary-to-XML Translation
- Performing XML-to-Binary Translation

What Is Included in the Purchase Order Sample

The following table describes the files provided with the purchase order sample application. All directory names are relative to the WebLogic Integration samples directory (*SAMPLES_HOME*\integration) where *SAMPLES_HOME* is the samples directory in your WebLogic Platform installation..

Table C-1 List of Purchase Order Sample Application Files

Directory	File	Description
samples\di\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	Prebuilt message format description of purchase order data.

Prerequisite Considerations

Before you can run the purchase order sample, you must make sure that certain software applications are installed and certain tasks are complete. For more information, see *BEA WebLogic Integration Release Notes*.

To understand how the Format Builder is used, it helps to understand the data formats used by the data integration tools provided by WebLogic Integration: binary data, XML, and MFL. If you have not already done so, please review "Understanding Data Formats" on page 3-1.

Performing Binary-to-XML Translation

The following sections provide information about building a sample purchase order format definition and testing the translation of binary data into XML format:

- Analyzing the Data to Be Translated
- Creating the Format Definition and Testing the Translation

You can build format definitions that provide the information required to translate binary data to or from XML. Format definitions are the metadata used to parse the content of a binary data file.

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:

- Data fields
- Data field attributes, such as name, data type, length/termination, optional, repeating
- Groups of related fields
- Group attributes, such as name, optional, repeating, delimited
- Hierarchical structure (groups of fields and/or other groups)

Listing C-1 shows sample binary data that is included on the WebLogic Integration CD-ROM (and the download) and is called `\samples\di\po\po_01.data`. In this sample, the data is taken from a fictitious purchase order on a proprietary system used by the XYZ Corporation. XYZ wants to exchange information with another system that accepts XML data.

Listing C-1 Sample Purchase Order in Binary Format

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

To analyze the purchase order data:

1. Generate a definition of the data. To do so, you may need to use printed specifications or internal documentation. For this sample, we have described the purchase order format in Table C-2.

Table C-2 Purchase Order Master Record

Category	Field Name	Data Type	Length	Description
Basic Information	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.

Table C-2 Purchase Order Master Record (Continued)

Category	Field Name	Data Type	Length	Description
Shipping Address	Street	Character	Delimited by semicolon	The street address to be used in shipping the requested items.
	City	Character	Delimited by semicolon	The city to be used in shipping the requested items.
	State	Character	Delimited by semicolon	The state to be used in shipping the requested items.
	Zip	Numeric	Delimited by semicolon	The zip code to be used in shipping the requested items.
Billing Address	Street	Character	Delimited by semicolon	The street address to be used for billing.
	City	Character	Delimited by semicolon	The city to be used for billing.
	State	Character	Delimited by semicolon	The state to be used for billing.
	Zip	Numeric	Delimited by semicolon	The zip code to be used for billing.

Table C-2 Purchase Order Master Record (Continued)

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

2. Identify fields.

   A field is a sequence of bytes that is meaningful to an application. For example, in Table C-2, Purchase Request Number, Supplier ID, Supplier Name, and so forth, are all fields.

3. Identify field attributes.

   Field attributes include the name of the field, the type of data stored in the field, the length of the field, and the delimiter that indicates 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.

4. Identify hierarchical groups.

   Groups are collections of fields, comments, and other groups or references that are related in some way. In Table C-1, notice that the sample data defines a number of distinct groups: Shipping Address, Billing Address, Payment Terms, and Purchase Items.

5. Identify group attributes.

   You must define the attributes of the hierarchical groups. Group attributes include the name of the group, whether the group is optional, repeating, or delimited, or whether it is defined as a reference to another group. For example, because the same fields are required to define both a Shipping Address and a Billing Address (`Street`, `City`, `State`, `Zip`), you can define an `Address` group within the `Shipping_Address` group and set up a reference to it from within the `Billing_Address` group.

After you complete the preceding procedure, you may want to put the data into a spreadsheet, as shown in the example in Figure C-1. This spreadsheet can then serve as a guide when you create your purchase order message definition.

Figure C-1 Analysis of Purchase Order Data

1	Description	Group	Field	Reference	Optional	Name / Refers To	Data Type	Occurrence	Delimited by
2	Purchase Request Number		X			PR_Number	Numeric	1	Semicolon
3	Supplier ID		X			Supplier_ID	Numeric	1	Semicolon
4	Supplier Name		X		X	Supplier_Name	String	1	Numeric field length 2
5	Requested Delivery Date		X			Requested_Delivery_Date	Date MM/DD/YYYY	1	Semicolon
6	Shipping Address	X				Shipping_Address		1	
7	Address	X				Address		1	
8	Street		X			Street	String	1	Semicolon
9	City		X			City	String	1	Semicolon
10	State		X			State	String	1	Semicolon
11	Zip		X			Zip	Numeric	1	Semicolon
12	Billing Address	X				Billing_Address		1	
13	Address			X		Address		1	
14	Street			X		Street	String	1	Semicolon
15	City			X		City	String	1	Semicolon
16	State			X		State	String	1	Semicolon
17	Zip			X		Zip	Numeric	1	Semicolon
18	Payment Terms	X				Payment Terms		1	
19	Purchase Order	X						0 or 1	
20	Purchase Order Tag		X			Payment_Type_PO	Literal "PO"	1	
21	Purchase Order Number		X			PO_Number	Numeric	1	Semicolon
22	Credit Card	X						0 or 1	
23	Payment Type		X			Payment_Type_CC	Literal "CC"	1	
24	Credit Card Number		X			CC_Number	Numeric	1	Semicolon
25	Credit Card Expire Month		X			CC_Expire_Month	Numeric	1	Semicolon
26	Credit Card Expire Year		X			CC_Expire_Year	Numeric	1	Semicolon
27	Purchase Items	X				Purchase_Items		1	
28	Item	X				Item		1- n	
29	Part Number		X			Part_Number	Numeric	1	Semicolon
30	Quantity		X			Quantity	Numeric	1	Semicolon
31	Description		X			Description	String	1	Semicolon
32	Carriage Return		X		X	Lit_CR	Literal "\r"	1	
33	Line Feed		X		X	Lit_LF	Literal "\n"	1	

Creating the Format Definition and Testing the Translation

This section walks you through the process of creating a message format for translating the binary data described in yeah to XML. To make sure you create your purchase order message format correctly, you can compare the file you create with the `\samples\di\po\po.mfl` file on the product CD-ROM.

Step 1. Start the Format Builder and Create a Message Format

To start Format Builder and create a message format:

1. Choose Start→Programs→BEA WebLogic Platform 7.0→WebLogic Integration 7.0→Format Builder. The Format Builder main window is displayed.
2. Choose File→New.

   A new message format root node is created and displayed in the navigation tree.
3. Enter `PurchaseRequest` in the Name/XML Root field.
4. Click Apply.

   The name of the message format root node is updated.

Step 2. Create the Basic Information Fields

To create the fields required to capture the basic identifying information for the purchase order:

1. To create the `PR_Number` field, choose Insert→Field→As Child.

   A new field is added to the navigation tree. The default field properties are displayed in the detail window.
2. Define the properties for the field as described in the following table.

In this section . . .	Do the following . . .
Field Description	Enter `PR_Number` in the Name field.
	Select `Numeric` from the Type drop down list.
Field Occurrence	Verify that the Once option button is selected.
Field Attributes: Termination	Select the Delimiter option button
	Enter a semi-colon (`;`) in the Value field on the Delimiter tab.
Field Attributes: Code Page	Select the desired code page from the Code Page drop down list. A code page specifies the character encoding of the binary data in the field.

Note: These values are determined by the analysis of the raw purchase order data shown in Figure C-1.

3. Click Apply.

   The properties of the field are updated.

   Note: Because the only difference between the `PR_Number` field and the `Supplier_ID` field is the name you can use the Format Builder Duplicate feature to create the `Supplier_ID` field.

4. Right click the `PR_Number` field.
5. Select Duplicate from the shortcut menu.

   A duplicate field (`NewPR_Number`) is added as a sibling and becomes the current selection in the navigation tree.

6. Enter `Supplier_ID` in the Name field and click Apply to update.
7. To save your changes to the message format document, do the following:

   a. Choose File—Save As to display the Save As dialog box.

   b. Navigate to the `samples/di/po` directory.

c. Enter the desired filename. For example enter `my_po.mfl`.

Note: The Format Builder automatically assigns the `.mfl` extension to message format files if no extension is specified.

d. Click Save As.

8. To add a the `Supplier_Name` field, choose Insert→Field→As Sibling.

   A new field is added to the navigation tree. The default field properties are displayed in the detail window.

9. Define the properties for the `Supplier_Name` field as described in the following table.

In this section . . .	Do the following . . .
Field Description	Enter `Supplier_Name` in the Name field.
	Select the Optional checkbox.
	Select `String` from the Type drop down list.
Field Occurrence	Verify that the Once option button is selected.
Field Attributes	Select the Field is Tagged checkbox.
	Enter the following in the Field is Tagged field: `SUP:`
Field Attributes: Termination	Select the Embedded Length option button.
	Select `Numeric` from the Type drop down list on the Description tab.
	Verify that the Length option button is selected, and then enter `2` in the Length text box.

10. Click Apply.

    The properties of the field are updated.

    Note: The dotted-line box around the field icon in the navigation tree indicates that this field is optional.

11. To add the `Requested_Delivery_Date` field, choose Insert→Field→As Sibling.

A new field is added to the navigation tree. The default field properties are displayed in the detail window.

12. Define the properties for the `Requested_Delivery_Date` field as described in the following table.

In this section . . .	Do the following . . .
Field Description	Enter `Requested_Delivery_Date` in the Name field.
	Select `Date: MM/DD/YYYY` from the Type drop down list.
Field Occurrence	Verify that the Once option button is selected.
Field Attributes	Verify that String is selected in the Data Base Type drop down list.

Note: The contents of the detail window for a field is determined by the Type setting. When you select a date type from the drop down list, the length is implicitly determined. Therefore, the Field Attributes: Termination properties are no longer displayed.

13. Click Apply.

    The properties of the field are updated.

14. Choose File→Save to save your changes.

Step 3. Create the Shipping Address and Billing Address Groups

To create the shipping address and billing address groups:

1. Select any field in the navigation tree and choose Insert→Group→As Sibling.

   A new group is added to the navigation tree. The default group properties are displayed in the detail window.

2. Define the properties for the `Shipping_Address` group as described in the following table.

In this section . . .	Do the following . . .
Group Description	Enter `Shipping_Address` in the Name field.
Group Occurrence	Verify that the Once option button is selected.
Group Attributes	Verify that the None option is selected for Group Delimiter.

Note: These values are determined by the analysis of the raw purchase order data shown in Figure C-1.

3. Click Apply.

   The properties of the group are updated.

   Note: Because the only difference between the `Shipping_Address` group and the `Billing_Address` group is the name, you can use the Format Builder Duplicate feature to create the `Billing_Address` group.

4. Right click the `Shipping_Address` group.
5. Select Duplicate from the shortcut menu.

   A duplicate group (`NewShipping_Address`) is added as a sibling and becomes the current selection in the navigation tree.

6. Enter `Billing_Address` in the Name field and click Apply to update.

   Because the `Shipping_Address` and `Billing_Address` groups contain the same fields with the same attributes, we can define an `Address` group within the `Shipping_Address` group and set up the `Address` group within the `Billing_Address` group as a reference.

7. Select the `Shipping_Address` group in the navigation tree and choose Insert→Group→As Child.
8. Use the data in Figure C-1 to create the `Address` group and click Apply to update the group properties.
9. Follow the steps outlined in Step 2. Create the Basic Information Fields, and the data provided in Figure C-1, to create the `Street`, `City`, `State`, and `Zip` fields as children of the `Address` group.

Note: Once the `Street` field is created, you can use the Duplicate button to create the `City` and `State` fields.

10. To create a reference to the `Address` group, right-click the `Address` group under `Shipping_Address`, and then select Copy from the shortcut menu.

    The `Address` group properties (including the child fields) are copied and placed on the clipboard.

11. Right-click the `Billing_Address` group and select Paste As Reference from the shortcut menu.

    The `Address` reference, is pasted immediately after the `Billing_Address` group. The arrow in the icon is identifies the group as a reference group:

12. To make the `Address` group reference a child of the `Billing_Address` group, select the reference, and then choose Edit→Demote.

    The `Address` reference becomes a child of the `Billing_Address` group.

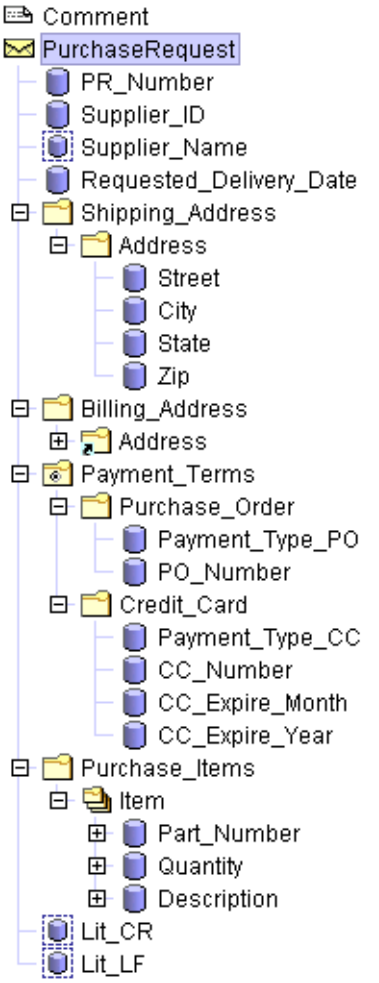
13. Choose File→Save to save your changes.

Step 4. Create the Remaining Items

Repeat the process of creating or duplicating fields and groups as required to complete the purchase order message format document. Use the analysis of the raw purchase order data presented in Figure C-1 to determine the values you need to enter for each item. For assistance, refer to the `\samples\di\po\po.mfl` file.

When you finish entering the required items, your navigation tree should look similar to the one shown in Figure C-2.

Figure C-2 Completed Navigation Tree for Purchase Order Sample



Step 5. Save the Completed Message Format

Choose File—Save to save your completed message format.

The completed MFL document is saved. Depending on the Format Builder options currently set, the DTD and/or XML Schema that describes the content model captured in the MFL document are also saved.

To set up the Format Builder to automatically generate a DTD and/or XML Schema for your message format documents:

1. Choose Tools→Options to display the Format Builder Options dialog box.
2. In the XML Content Model section at the bottom of the dialog box, check Auto-generate DTD and/or Auto-generate Schema, as required.
3. Click OK to update the options.

Now, whenever you save message format documents, Format Builder generates the specified file(s) for each message format document.

Step 6. Test the Message Format

To test the message format to identify any errors before using it to translate data:

1. Choose Tools→Test to display the Format Tester dialog box.
2. Choose File→Open Binary to display the Open dialog box.
3. Navigate to the `samples/di/po` directory.
4. Select the `PO_01.DATA` file and click Open.

   The binary data is displayed in the Binary window.
5. Click Translate→Binary To XML.

   The content of the `PO_01.DATA` file is translated to XML based on the active MFL document. The XML output is displayed in the XML window.

   Note: To view a description of each translation step, choose Display→Debug to open the Debug window, and then choose Translate→Binary To XML. A message is displayed for each step of the process.
6. If the translated data appears to be correct, choose File→Save XML.
7. Type the name `PO.XML` in the File name field, and then click Save to save the XML output.

Performing XML-to-Binary Translation

You can also use the 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 those you follow to translate binary data to XML. To translate XML data to binary, first create an MFL description for the binary format. The purchase order sample files can be used to test the process as described in the following procedure.

1. Choose File→Open from the Format Builder menu.
2. Select the purchase order message format document.
3. Click Open.

   The message format document is displayed in the navigation tree.
4. Choose Tools→Test to display the Format Tester dialog box.
5. Choose File→Open XML from the Format Tester menu.
6. Navigate to the `samples\di\po` directory.
7. Select the `po.xml` file and click Open. The XML data is displayed in the right pane.
8. Choose Translate→XML to Binary.

   The XML data is translated, and the purchase order data is displayed, in binary format, in the left pane.

   Note: To view a description of each translation step, choose Display→Debug to open the Debug window, and then choose Translate→XML to Binary. A message is displayed for each step of the process.
9. If the translated data appears to be correct, choose File→Save Binary.
10. Type the name (for example `test_po.data`) in the File name field, and then click Save to save the binary output.

Index

B

batch import utility 5-7

C

C struct importer
  hardware profiles 4-8
  starting 4-5
C structures 4-3
Character encoding options 3-42
choice of children 3-21
COBOL copybook
  importer data types A-8
  importing 4-1
COBOL data types A-8
code page 3-27
com.bea.wlxt.bintype.Bintype class B-11
  optional interface routines B-12
  required interface routines B-11
  utility interface routines B-14
com.bea.wlxt.bintype.BintypeDate
  optional interface routines B-17
  required interface routines B-17
  utility interface routines B-17
com.bea.wlxt.bintype.BintypeDate class B-16
com.bea.wlxt.bintype.BinTypeString
  optional interface routines B-16
  required interface routines B-15
  utility interface routines B-16
com.bea.wlxt.bintype.BinTypeString class B-15
com.bea.wlxt.mfl.MFLField class B-18
comments 3-8, 3-31
customer support contact information xi

D

data base type 3-27
data fields 3-9
Data Gen 4-7
data offsets 2-4
data types
  COBOL A-8
  MFL A-1
  support A-1
debug writer 6-4
Delimiter 3-30
delimiter
  field 3-29
  group 3-22
delimiter shared 3-22
document type definition 6-3
documentation, where to find it x

E

edit menu 3-44
  copy 3-45
  cut 3-45
  delete 3-45
  demote 3-46
  duplicate 3-45

move down 3-45
move up 3-45
paste 3-45
promote 3-46
redo 3-45
undo 3-44, 3-45

F

field 3-7
creating 3-24
data type 3-26
name 3-26
optional 3-26
parameters 3-7
field data options
code page 3-27
data base type 3-27
value 3-27
year cutoff 3-27
field delimiter
delimiter 3-29
delimiter field 3-30
file menu 3-44
close 3-44
exit 3-44
new 3-44
open 3-44
save 3-44
save as 3-44
FML Field Table Class
importing 4-13
sample files 4-14
Format Builder
setting options 3-41
starting 3-9
using 3-9, 3-10
Format Tester
debug log 2-15
debug window 2-5
starting 2-1

G

group
attributes 3-9
creating 3-20
delimiter 3-22
occurrence 3-21
group delimiter
delimited 3-22
delimiter is shared 3-22
group occurrence
once 3-21, 3-26, 3-34
repeat delimiter 3-21, 3-26, 3-34
repeat field 3-21, 3-26, 3-34
repeat number 3-21, 3-26, 3-34
unlimited 3-21, 3-26, 3-34

H

hardware profiles 4-8
building 4-9
help menu 3-48
about 3-48
help topics 3-48
how do I 3-48
hierarchical groups 3-9

I

importing C structures 4-3
insert menu 3-46
comment 3-46
field 3-46
group 3-46

M

menu bar 3-13
Message Format
adding palette items 3-38
default version 3-42
opening 3-39, 3-40

saving 3-38
MFL data types A-1
MFL documents, about 3-6
MFL Gen 4-7

N

name
field 3-26
group 3-21

O

offsets, positioning 2-14
once
group occurrence 3-21, 3-26, 3-34
optional
field 3-26
group 3-21

P

palette 3-34
adding items 3-37
deleting items 3-38
printing product documentation x

R

references
creating 3-32
description 3-8
related information x
repeat delimiter
group occurrence 3-21, 3-26, 3-34
repeat field
group occurrence 3-21, 3-26, 3-34
repeat number
group occurrence 3-21, 3-26, 3-34
repository
accessing 5-2
importing documents 5-6
logging in 5-2
menu commands 5-3
repository document chooser 5-9
repository documents
retrieving 5-4
storing 5-6
root node 3-11

S

shortcut menus 3-16
support, technical xi

T

toolbar 3-14
tools menu 3-47
import 3-47
options 3-48
Tree Pane 3-10
using 3-11

U

unlimited
group occurrence 3-21, 3-26, 3-34
user defined types B-1
coding requirements B-11
registering B-3
sample files B-2

V

value, field data option 3-27
view menu 3-46
collapse all 3-47
expand all 3-47
show pallet 3-46

X

XML content model options 3-43

XML documents, about 3-3
XML formatting options 3-42