BEA WebLogic Integration™

Translating Data with WebLogic Integration

Version 2.1
Document Date: October 2001

Translating Data with WebLogic Integration

Part Number	Date	Software Version
N/A	October 2001	2.1

Contents

About This Document

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

1. Translating Data with WebLogic Integration Overview

Understanding XML Translation 1-2
What is Data Integration? 1-3
The Design-Time Component 1-4
The Run-Time Component 1-5
The Plug-In to Business Process Management 1-8
Post Translation Options and Considerations 1-9

2. Building Format Definitions

Understanding Data Formats 2-1
About Binary Data (Non-XML Data) 2-1
About XML Documents 2-2
About MFL Documents 2-5
Analyzing the Data to be Translated 2-7
Using Format Builder 2-7
Starting Format Builder 2-8
Using the Format Builder Main Window 2-8
Creating a Message Format 2-16
Creating a Group 2-17

Specifying Delimiters 2-21
Creating a Field 2-22
Creating a Comment 2-26
Creating References 2-27
Working with Palettes 2-29
Saving a Message Format to a File 2-33
Using Internationalization Features 2-34
Opening an Existing Message Format File 2-35
Changing Options for a Message Format 2-36
Working With the Repository 2-36
Setting Format Builder Options 2-38
Format Builder Menus 2-40

3. Testing Format Definitions

Starting Format Tester 3-1
Using the Format Tester Main Window 3-2
Using the Menu Bar 3-3
Using the Shortcut Menus 3-6
Using the Binary Window 3-7
Using the XML Window 3-8
Using the Debug Window 3-8
Using the Resize Bars 3-9
Testing Format Definitions 3-9
Debugging Format Definitions 3-11
Searching for Values 3-11
Positioning to an Offset 3-12
Using the Debug Log 3-13

4. Importing Meta Data

Importing a COBOL Copybook 4-1
Importing C Structures 4-3
C Struct Importer Sample Files 4-4
Starting the C Struct Importer 4-6
Understanding Hardware Profiles 4-9
Generating MFL 4-10

Generating C Code 4-12
Importing an FML Field Table Class 4-13
FML Field Table Class Importer Prerequisites 4-13
FML Field Table Class Sample Files 4-14
Creating XML with the FML Field Table Class Importer 4-14

5. Retrieving and Storing Repository Documents

Accessing the Repository 5-2
Retrieving Repository Documents 5-3
Storing Repository Documents 5-4
Importing Documents into the Repository 5-5
Using the Repository Document Chooser 5-6
Using the Open Document Dialog Box 5-7
Using the Store Document Dialog Box 5-8
Using the Shortcut Menus 5-8

6. Using the Run-Time Component

Binary to XML 6-2
Generating XML with a Reference to a DTD 6-3
Passing in a Debug Writer 6-4
XML to Binary 6-5
Converting a Document object to Binary 6-7
Passing in a Debug Writer 6-8
XML to XML Transformation 6-9
Initialization Methods 6-10
Java API Documentation 6-12
Run-Time Plug-In to Business Process Management 6-12

A. Supported Data Types

MFL Data Types A-1
COBOL Copybook Importer Data Types A-7
C Structure Importer From Importing Metadata A-9

B. Creating Custom Data Types

User Defined Types Sample Files 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 Format Builder .. B-7
Publishing User Defined Types to the Repository Using the Repository Import Utility B-10
User Defined Type Coding Requirements B-10
Class com.bea.wlxt.bintype.Bintype B-10
Class com.bea.wlxt.bintype.BintypeString B-14
Class com.bea.wlxt.bintype.BintypeDate B-15
Class com.bea.wlxt.mfl.MFLField B-17

C. Running the Purchase Order Sample

What is Included in the Purchase Order Sample C-2
Prerequisite Considerations C-2
Understanding the Data Formats C-3
About Binary Data (Non-XML Data) C-3
About XML Documents C-4
About MFL Documents C-4
Performing Binary to XML Translation C-6
Analyzing the Data to be Translated C-6
Testing the Translation C-10
Performing XML to Binary Translation C-18

Index

About This Document

This document describes translating data with WebLogic Integration. You can use the instructions in this document to translate data from binary format to XML and from XML to binary format.

This document covers the following topics:

- Translating Data with WebLogic Integration Overview
- Building Format Definitions
- Testing Format Definitions
- Using the Run-Time Component
- Supported Data Types

What You Need to Know

This document is intended mainly for application programmers and technical analysts who perform data translations from binary to XML and XML to binary.

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 the WebLogic Integration release.

If you have any questions about this version 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. *Examples*: 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.

1 Translating Data with WebLogic Integration Overview

Within 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 and is invaluable in integrating disparate applications. However, most data transformation engines do not support translations between binary data formats and XML. WebLogic Integration provides for an exchange of information between applications by supporting data translations between binary formats from legacy systems and XML.

This section provides information about the following topics:

- Understanding XML Translation
- What is Data Integration?
  - The Design-Time Component
  - The Run-Time Component
  - The Plug-In to Business Process Management
- Post Translation Options and Considerations

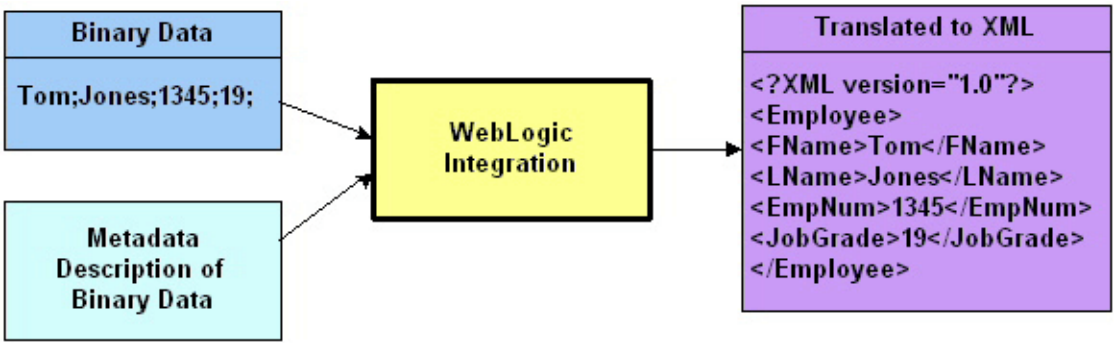
Understanding XML Translation

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

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

When you translate binary to XML data, you convert structured binary data to an XML document so that the data can be accessed via standard XML parsing methods. You must create the metadata used to perform the conversion. The translation process converts each field of binary data to XML according to the metadata defined for each field of data. In the metadata you specify the name of the field, the data type, the size, and whether the field is always present or optional. It is this description of the binary data that is used to translate the binary data to XML. Figure 1-1 shows a sample of XML data translation.

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



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

What is Data Integration?

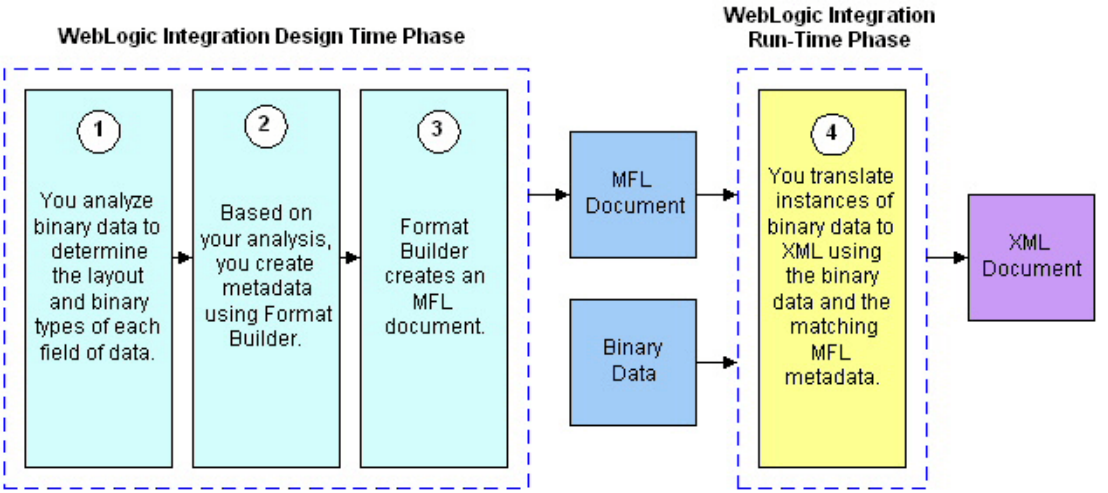
The data integration component of WebLogic Integration facilitates the integration of data from diverse enterprise applications by supporting data translations between binary formats from legacy systems and XML. Data integration normalizes legacy data into XML so it may be directly consumed by XML applications,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

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

To perform a translation, you create a description of your binary data using the design-time component (Format Builder). This involves analyzing binary data so that its record layout is accurately reflected in the metadata you create in Format Builder. You then create a description of the input data in Format Builder and save this metadata as a Message Format Language (MFL) document. Data integration includes importers that automatically create message format definitions.

You can then use data integration's run-time component 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 business process management (BPM) allows for easy access to configuring translations.

Figure 1-2 Event Flow for Translating Data

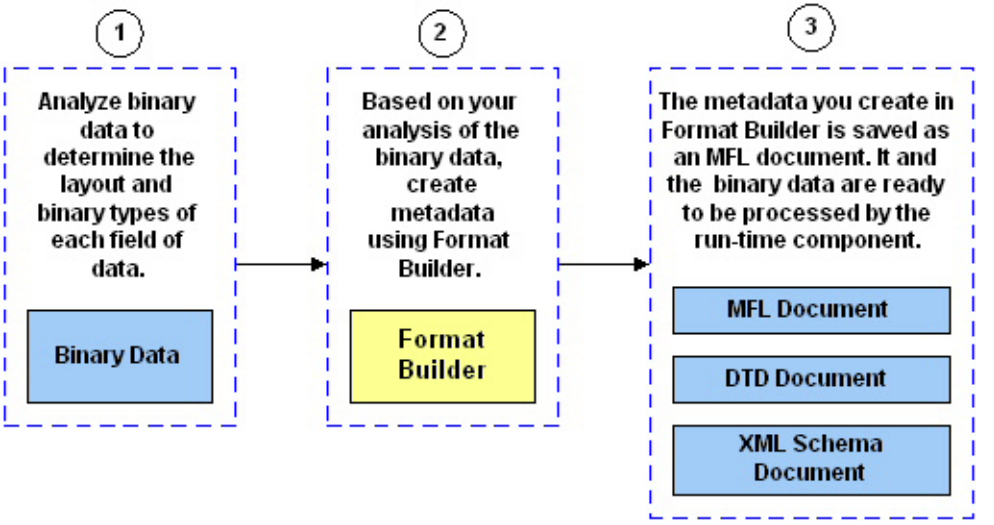


The Design-Time Component

Design-time in data integration is a Java application called Format Builder. You can create descriptions of binary data records, describe the layout and hierarchy of the binary data so it can be translated to or from XML. You can describe sequences of bytes as fields. Each field description includes the type of data (floating point, string, etc.), the size of the data, and the name of the field. You can further define groupings of fields (groups), repetition of fields and groups, and aggregation.

The description you create is saved in an XML grammar called Message Format Language (MFL). MFL documents are 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 will also create 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

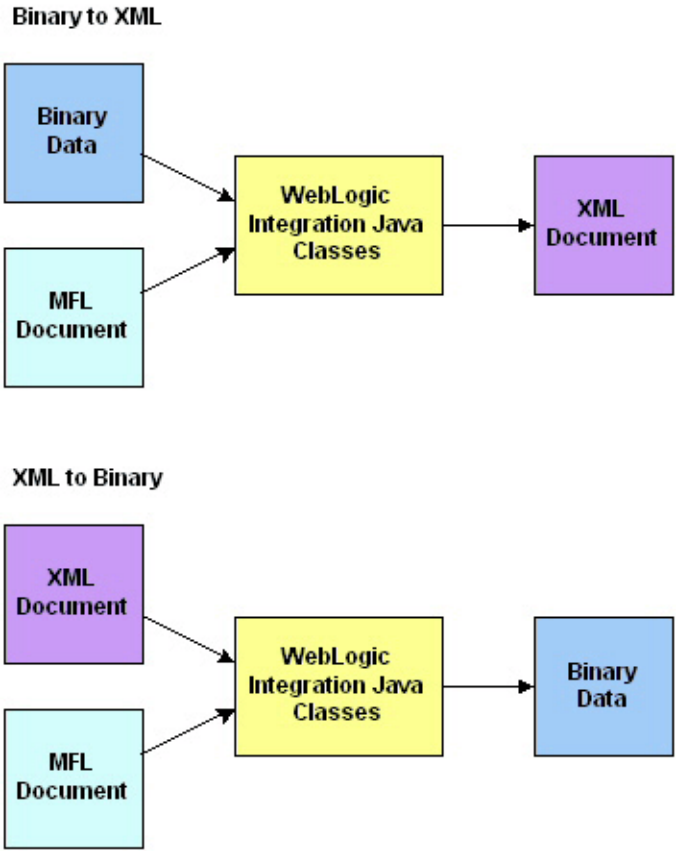


You can also use Format Builder to retrieve, validate, and edit stored MFL documents and to test message format definitions with your own data. MFL documents may be stored using the file system or archived in the repository. The test feature allows you to select the option of testing the translation of XML data to binary format, or binary data to XML format. You may save the transformed data to a file for future testing.

The Run-Time Component

The data integration run-time component is a Java class with various methods used to translate data between binary and XML formats. This Java class can be deployed in an EJB using BEA WebLogic Server, invoked from a workflow in business process management, or integrated into any Java application. Figure 1-4 shows the run-time process flow for binary to XML translations and XML to binary translation.

Figure 1-4 Run-Time Process Flow



Binary to XML Translation

Listing 1-1 is a code sample that shows the parsing of a file containing binary data into an XML document object. The MFL file `mymfl.mfl` is used as the description of the binary data contained in the file `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

Listing 1-2 is a code sample that shows the translation of the XML data contained in the file `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);
}
```

The Plug-In to Business Process Management

Business process management (BPM) in WebLogic Integration automates workflow, business-to-business processes, and enterprise application assembly. It runs on the WebLogic Server and is a robust J2EE standards-based workflow and process integration solution.

Using an intuitive flowchart paradigm, business analysts use the process engine to define business processes that span applications or to automate human interaction with applications. Developers can use the process engine to assemble application components quickly without programming, execute them, and manage them.

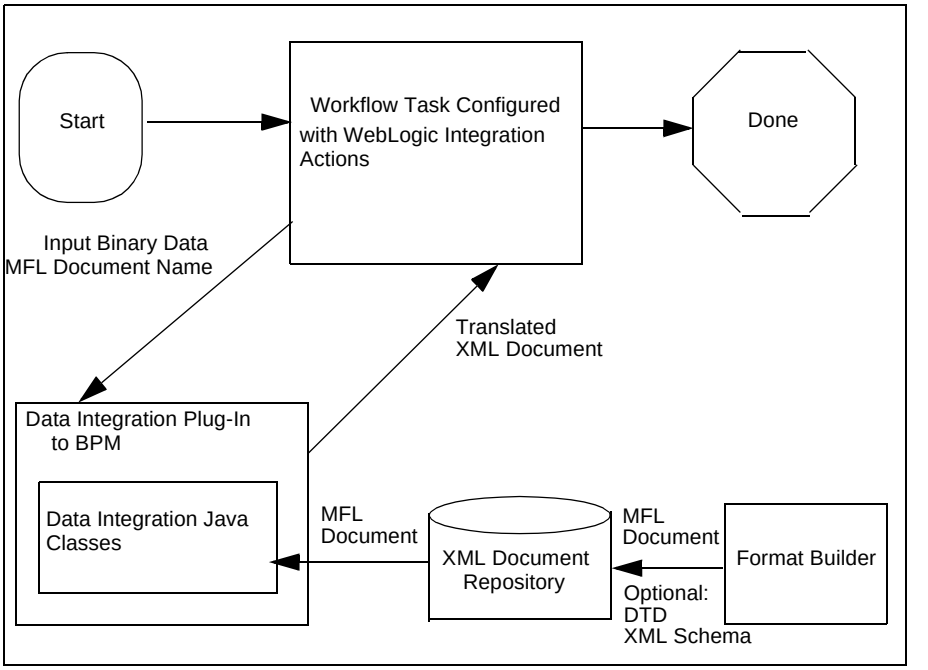
The process engine has an extensible architecture that allows new functionality to be plugged in. WebLogic Integration includes a data integration plug-in that provides XML to binary and binary to XML translation that is accessible through a BPM action.

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

In addition to this data translation capability, the Data Integration Plug-In provides event data processing in binary format, in-memory caching of MFL documents and translation object pooling to boost performance, a `BinaryData` variable type to edit and display binary data, exporting of entirely self-contained workflow definition packages, and execution within the WebLogic Server clustered environment.

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

Figure 1-5 Data Integration and BPM Relationship



Post Translation Options and Considerations

After you have successfully translated your binary data to XML, or vice versa, you have numerous options for additional processing of the XML data. The XML data can be transformed to a specific XML dialect or to a display format. The XML data can be sent to other applications that consume XML such as business process management (BPM). Once your binary data has been put in a self-describing format such as XML, this data is available for use in other applications.

Once you have translated binary data into XML, you may need to transform the XML data to a different XML grammar, to a display format (HTML), or to another binary format. The process of transforming XML to another XML grammar is referred to in this document as XML transformation. XML transformation can be accomplished via

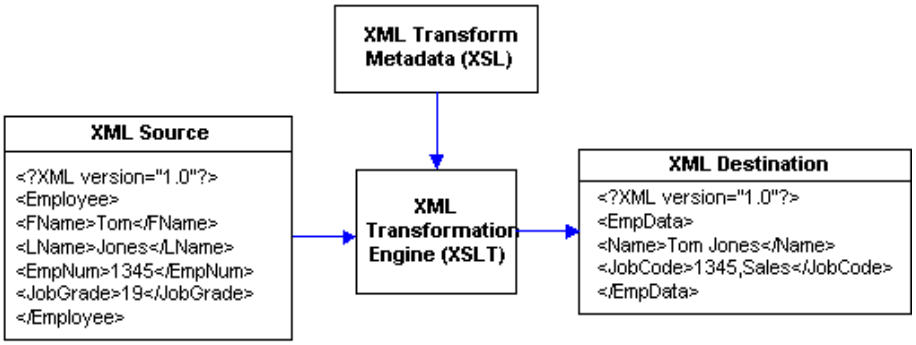
the XML module of the WebLogic Server. BPM provides an action that allows you to access this module and transform XML documents using XSL style sheets. You might want to transform XML for several reasons:

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

XSL (extensible stylesheet language) is an XML language that describes a series of transformations that are to be performed on nodes of an XML document. A stylesheet is an XSL document that can be used 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 demonstrates one XML grammar converted to another using an XSLT engine. The transformation metadata in this case is an XSL style sheet that describes how one XML grammar is mapped into another.

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



2 Building Format Definitions

The following sections provide information on building format definitions using the data integration design-time component of WebLogic Integration (Format Builder):

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

You can build format definitions for binary data that will be translated to or from XML. Format definitions are the metadata used to parse or create binary data.

Understanding Data Formats

To understand how to use Format Builder, it helps to understand the following data formats: binary data, XML, MFL, DTD and Schema.

About Binary Data (Non-XML Data)

Because computers are based on the binary numbering system, applications often use a binary format to represent data. A file stored in binary format is computer-readable but not necessarily human-readable. Binary formats are used for executable programs

and numeric data, and text formats are used for textual data. Many files contain a combination of binary and text formats. Such files are usually considered to be binary files even though they contain some data in a text format.

Unlike XML data, binary data is not self-describing. In other words, binary data does not provide a description of how the data is grouped, divided into fields, or arranged in a layout. Binary data is a sequence of bytes that can be interpreted as an integer, a string, or a picture, depending on the intent of the application that generates the sequence of bytes. In order for binary data to be understood by an application, the layout must be embedded within each application that uses this data. Binary data may also be embedded using different character sets. 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 the binary data. MFL is an XML language that includes elements to describe each field of data, as well as groupings of fields (groups), repetition, and aggregation. The hierarchy of a binary record, the layout of fields, and the grouping of fields and groups are expressed in an MFL document. This MFL document is used at run-time to translate the data to and from an XML document.

Listing 2-1 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;
```

About XML Documents

Extensible Markup Language, or XML, is a text format for exchanging data between different systems. It allows data to be described in a simple, standard, text-only format. In contrast to binary data, XML data embeds a description of the data within the data stream. Applications can share data more easily, since they are not dependent on the layout of the data being embedded within each application. Since the data is presented in a standard form, applications on disparate systems can interpret the data using XML parsing tools, instead of having to interpret data in proprietary binary formats.

Instances of XML documents contain character data and markup. The character data is referred to as content, while the markup provides hierarchy for that content. Markup is distinguished from text by angle brackets. Information in the space between the "<"and the">" is referred to as the tag that markup the content. Tags provide an indication of what the content is for, and a mechanism to describe parent-child relationships. Listing 2-2 shows an example of an XML document.

Listing 2-2 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>
```

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

Listing 2-3 shows an example of a Document Type Definition.

Listing 2-3 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 2-4 shows an example of an XML Schema.

Listing 2-4 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>
```

About MFL Documents

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

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

- Message Format - The top level element. Defines the message name and MFL version.
- Field - Sequence of bytes that have some meaning to an application. (For example, the field `EMPNAME` contains an employee name.) Defines the formatting for the field. The formatting parameters you can define include:

- Tagged - Indicates that a literal precedes the data field, denoting the beginning of the field.
- Length - Indicates that a length value precedes the data field, denoting the length of this field.
- Occurrence - Repeating fields appear more than once in the message format. You can set a specific number of times the field is to repeat, or define a delimiter to indicate the end of the repeating field.
- Optional - The field may or may not be present in the named message format.
- Code Page - The character encoding of the field data.

- Groups - Collections of fields, comments, and other groups or references that are related in some way (for example, the fields `PAYDATE`, `HOURS`, and `RATE` could be part of the `PAYINFO` group). The parameters you can define include:
  - Tagged - Being tagged means that a literal precedes the other content of the group, which could be other groups or fields.
  - Occurrence - Repeating groups appear more than once in the message format: You can set a specific number of times the group is to repeat, or define a delimiter to indicate the end of the repeating group. For more information on delimiters, see "Specifying Delimiters" on page 2-21.
  - Choice of Children - Defining a group as "Choice of Children" means that only one item in the group will appear in the message format.
  - Optional - The group of data within this structure may or may not be present in the named message format.
- References - Indicate that another instance of the field or group format exists in the data. Reference fields or groups have the same format as the original field or group, but you can change the optional setting and the occurrence setting for the reference field or group. For example, if you have a "bill to" address and a "ship to" address in your data, you only need to define the address format once. You can create the "bill to" address definition and create a reference for the "ship to" address.
- Comments - Notes or additional information about the message format.

Analyzing the Data to be Translated

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

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

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

Format Builder is used to build the format definitions that are used for data translations.

Using Format Builder

Format Builder assists you in creating format descriptions for binary data. You use Format Builder to create hierarchical and detail information derived from structural and 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 you apply them to your actual data.

Starting Format Builder

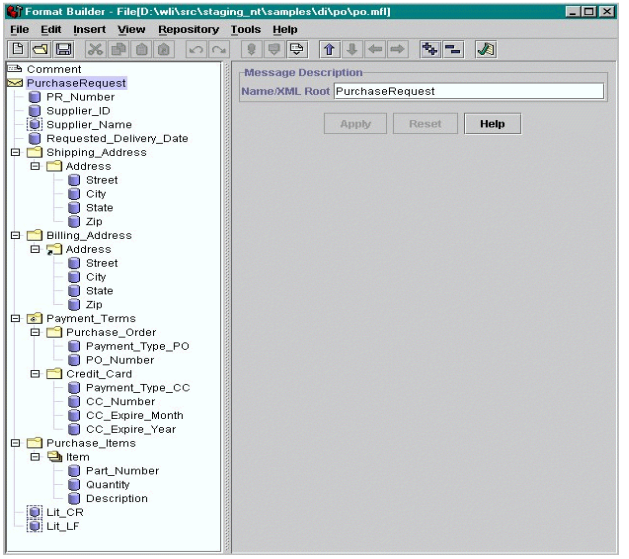
To start Format Builder, choose Start→Programs→BEA WebLogic E-Business Platform→WebLogic Integration 2.1→Format Builder. The Format Builder main window displays. If you did not use the installation directory defaults, your path may be different.

Using the Format Builder Main Window

The main window of Format Builder is split into two panes. The left pane (the navigation tree) shows the structural information for the data format. The right pane shows the detail for the item selected in the navigation tree.

Details of the file you are editing display in the Title Bar of the Format Builder main window.

Figure 2-1 Format Builder Main Window



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

The following topics discuss the parts of the main window and provide instructions for navigating and executing commands from the main window of Format Builder:

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

Using the Navigation Tree

The navigation tree represents hierarchical/structural information about the format of the binary data in a navigation tree. The root node of the navigation tree will correspond to the MFL document being created or edited. The root node is referred to as the Message node. Child nodes are labeled with group or field names. Fields are represented by leaf nodes in the navigation tree. Groups contain fields or other groups and are represented by non-leaf nodes in the navigation tree.

The icon for each node encapsulates information about the node. The icon indicates whether the node represents a message, a group, a field, a comment, or a reference. The icon also indicates 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 also have the ability to add, delete, move, copy, or rename nodes in the navigation tree. This is done through the menus or the toolbar (see "Using the Menu Bar" on page 2-11 and "Using the Toolbar" on page 2-11).

The icons that appear in the navigation tree are described in the following table.

Table 2-1 Navigation Tree Icon Descriptions

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` could be part of the `PAYINFO` group). Defines the formatting for all items contained in the group.

Table 2-1 Navigation Tree Icon Descriptions

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 has one or more occurrence.
	Optional Repeating Group	A group that may or may not be included, but if included, may occur more than once.
	Group Reference	Indicates that another instance of the group exists in the data. Reference groups have the same format as 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 will be included in the message format.
	Field	Sequence of bytes that have some meaning to an application. (For example, the field `EMPNAME` contains an employee name.) Defines the formatting for the field.
	Optional Field	A field that may or may not be included in the message format.
	Repeating Field	A field that has one or more occurrences.
	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 that another instance of the field exists in the data. Reference fields have the same format as the original field, but you can change the optional setting and the occurrence setting for the reference field.
	Comment	Contains notes about the message format or the data translated by the message format.

Table 2-1 Navigation Tree Icon Descriptions

Tree Icon	Icon Name	Description
	Collapse	A minus sign next to an object indicates that it can be collapsed.
	Expand	A plus sign next an object indicates that it can be expanded to show more objects.

Using the Menu Bar

The Menu bar displays the menu headings. The menu items that are available depend on what is selected in the navigation tree and the state of the navigation tree. Click a menu heading to open the menu and choose a command.

Figure 2-2 Format Builder Menu Bar

All Format Builder menus are expandable from your keyboard by pressing Alt + mnemonic keys. Some menu commands are also executable using Ctrl + *letter* accelerator keys.

Note: Menu items that appear in gray are unavailable for the current selection.

Using the Toolbar

The toolbar provides buttons that access some of the frequently used commands in the menus. To activate a command, click its toolbar button. If a command is unavailable, its button appears *grayed-out*.

Figure 2-3 Format Builder Toolbar

The toolbar buttons provided with Format Builder are described below:

Table 2-2 Format Builder Toolbar Buttons

Toolbar Button	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-hand 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-hand pane for insertion elsewhere in the navigation tree. **Note:** This action is not available if the Message Format (root) item is selected.
	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 changes to indicate the action that can be undone. For example, changing the name of a field to Address and clicking Apply causes the tool tip to read “Undo Apply Field Address”. Format Builder supports multi-level undoing and redoing.

Table 2-2 Format Builder Toolbar Buttons

Toolbar Button	Name	Description
	Redo	Reverses the effects of an Undo command. The tool tip changes to indicate the action that can be redone. For example, changing the name of a field to Address and then undoing that action causes the tool tip to read “Redo Apply Field Address”. Format Builder supports multi-level undoing and redoing.
	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.
	Move Down	Moves the selected item down one position under its parent.
	Promote item	Promotes the selected item to the next highest level in the navigation tree. For example, Field1 is the child object of Group1. Selecting Field1 and clicking the Promote tool makes it a sibling of Group1.
	Demote item	Demotes the selected item to the next lower level in the navigation tree. For example, Group1 is the sibling of Field1. Field1 immediately follows Group1 in the navigation tree. Selecting Field1 and clicking the Demote tool makes it a child of Group1.
	Expand All	Expands all items in the navigation tree to show child items.
	Collapse All	Collapses the navigation tree to show first level items only.

Table 2-2 Format Builder Toolbar Buttons

Toolbar Button	Name	Description
	Format Tester	Opens the Format Tester window.

Using the Shortcut Menus

Instead of using the standard menus to find the command you need, use the right mouse button to click an item in the navigation tree. The menu that appears shows the most frequently used commands for that item.

The following commands are available from the Shortcut Menus.

Note: Some commands may be unavailable, depending on the item you have selected in the navigation tree, or the state of the navigation tree at the time.

Table 2-3 Shortcut Menus

Menu Command	Description
Cut	Removes the item currently selected in the left-hand pane, and its child objects, from the navigation tree.
Copy	Makes a copy of the item currently selected in the left-hand pane for insertion elsewhere in the navigation tree.
Paste	Inserts the cut or copied item. An additional menu displays when you select Paste. You can choose to paste the item as a child or sibling of the selected item. In addition, you can choose to paste a reference to the cut or copied item as a sibling of the selected item.
Insert Group	Inserts a new group. You select whether to insert the group as a child or sibling of the selected item.
Insert Field	Inserts a new field. You select whether to insert the field as a child or sibling of the selected item.
Insert Comment	Inserts a comment. You select whether to insert the comment as a child or sibling of the selected item.

Table 2-3 Shortcut Menus

Menu Command	Description
Duplicate	Makes a copy of the currently selected item. The duplicate item contains the same values as the original item. The name of the duplicate item is the same as the original item name, with the word "New" inserted before the original name. For example, duplicating a group called "Group1" results in a group with the name "NewGroup1". When you duplicate an item with a numeric value in its name, the new item name contains the next sequential number. For example, duplicating "NewGroup1" results in a group named "NewGroup2".
Delete	Deletes the selected item.

Using Drag and Drop

You can drag and drop to copy and/or move the 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 use drag and drop 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 use drag and drop to copy 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 placed at the new location.

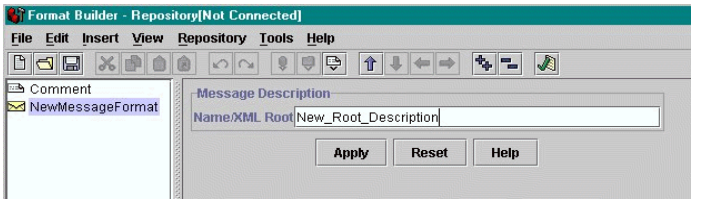
Creating a Message Format

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 Message Format Properties pane displays in the detail window.

Figure 2-4 Message Format Properties



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

Table 2-4 Message Format Properties

Field	Description
Message Format Properties	
Name/XML Root	The name of the message format. This value will be used as the root element in the translated XML document. This name must comply with XML element naming conventions.
Apply	Saves your changes to the message format document.
Reset	Discards your changes to the detail window and resets all fields to the last saved values.
Help	Displays online help information for this detail window.

Message Formats, Fields, and Groups are identified by a Name. The name that is specified is used as the XML tag when binary data is translated to XML by WebLogic Integration. Thus the name must conform to the XML rules for a name.

The rules for names are as follows:

- Must start with a letter or underscore
- Can contain letters, digits, the period character, the hyphen character, or the underscore character.

The following are valid name examples:

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

The following are invalid name examples:

`1MyField` - may not start with a digit
`My>Field` - the greater-than sign (>) is an illegal character
`My Field` - a space is not permitted

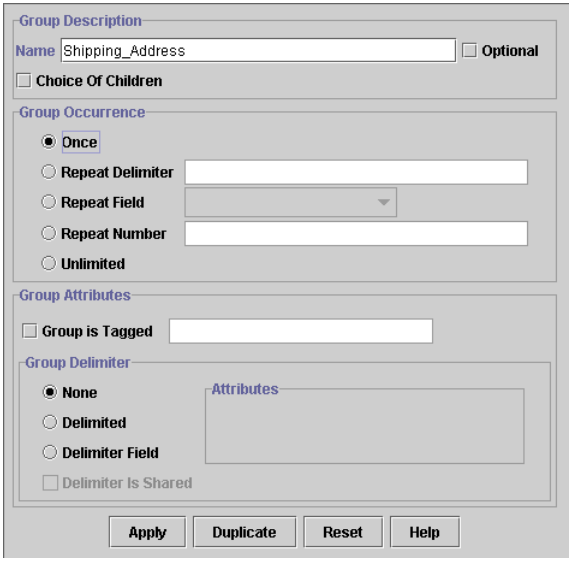
Creating a Group

Groups are collections of fields, comments, references and other groups that are related in some way (for example, the fields `PAYDATE`, `HOURS`, and `RATE` could be part of 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 an item in the navigation tree.
2. Choose Insert→Group→As Child if you want to create the group as the child of the message format or another group. Choose Insert→Group→As Sibling if you want to create the group as the sibling of another group or a field. The Group Details display in the detail window.

Figure 2-5 Group Details



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

Table 2-5 Group Detail Properties

Field	Description
Group Description	
Name	The name of the group. This name must comply with XML element naming conventions.
Optional	Choose Optional if this is an optional group.
Choice of Children	Choose Choice of Children if only one of the items in the group will be included in the message format.

Table 2-5 Group Detail Properties

Field	Description
Group Occurrence	
Occurrence	Choose one of the following to indicate how often this group appears in the message format: ■ **Once** - Indicates the group appears only once. ■ **Repeat Delimiter** - Indicates the group will repeat until the specified delimiter is encountered. ■ **Repeat Field** - Indicates the group will repeat the number of times specified in the repeat field. ■ **Repeat Number** - Indicates the group will repeat the specified number of times. ■ **Unlimited** - Indicates the group will repeat an unlimited number of times. **Note:** Unless a group is defined as Optional, all groups occur at least once.
Group Attributes	
Group is Tagged	Select this option if this is a tagged group. Being tagged means that a literal precedes the other content of the group, which could be other groups or fields.
Group Delimiter	
None	Select this option if the group has no delimiter.
Delimited	Groups can have their termination point specified by a delimiter. A delimiter is a string of characters that marks the end of the group of fields. The group continues until the delimiter characters are encountered. Select this option if the end of the group is marked with a delimiter. **Value** - Enter the delimiter that marks the end of the group of fields. **Note:** Normally, groups are not delimited. They are usually parsed by content (the group ends when all child objects have been parsed).

Table 2-5 Group Detail Properties

Field	Description
Delimiter Field	Groups can have their termination point specified by a field that contains a delimiter character string. A delimiter is a string of characters that mark the end of the group. The group continues until the delimiter character string contained in the specified field is encountered. ■ **Field** - Select the field that contains the delimiter character string. A list of valid fields will be presented in a drop-down list. ■ **Default** - Enter the default delimiter character that will be used if the above field is not present in the data. This value is required. For more information on delimiters, see "Specifying Delimiters" on page 2-21.
Delimiter is Shared	Indicates 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 among the group, and the last field of the group, to delimit the end of the data.
Group Update Buttons	
Apply	Saves your changes to the message format document.
Duplicate	Makes a copy of the group currently displayed. The duplicate group contains the same values as the original group. The name of the duplicate group is the same as the original group name, with the word "New" inserted before the original name. For example, duplicating a group called "Group1" results in a group with the name "NewGroup1". When you duplicate an item with a numeric value in its name, the new item name contains the next sequential number. For example, duplicating "NewGroup1" results in a group named "NewGroup2".
Reset	Discards your changes to the detail window and resets all fields to the last saved values.
Help	Displays online help information for this detail window.

4. Click Apply to save your changes to the message format file, or click Reset to discard your changes to the detail window and reset all fields to the last saved value.

   Note: The Apply and Reset buttons are only enabled once changes are made to the detail panel's components.

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 the delimiter ('\u009'), you would enter the construct `\t` to match it.

Table 2-6 Character Delimiters

Construct	Matches
`x`	The character x
`\\`	The backlash
`\0n`	The character with octal value 0n (<= n <= 7)
`\0nn`	The character with octal value 0nn (0 <= n <= 7)
`\0mnn`	The character with octal value 0mnn (0 <= m <= 3, 0 <= n <= 7)
`\xhh`	The character with hexadecimal value 0xhh
`\uhhhh`	The character with hexadecimal value 0xhhhh
`\t`	The tab character ('\u0009')
`\n`	The newline (line feed) character ('\u000A')
`\r`	The carriage-return character ('\u000D')
`\f`	The form-feed character ('\u000C')
`\a`	The alert (bell) character ('\u0007')
`\e`	The escape character ('\u001B')
`\cx`	The 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

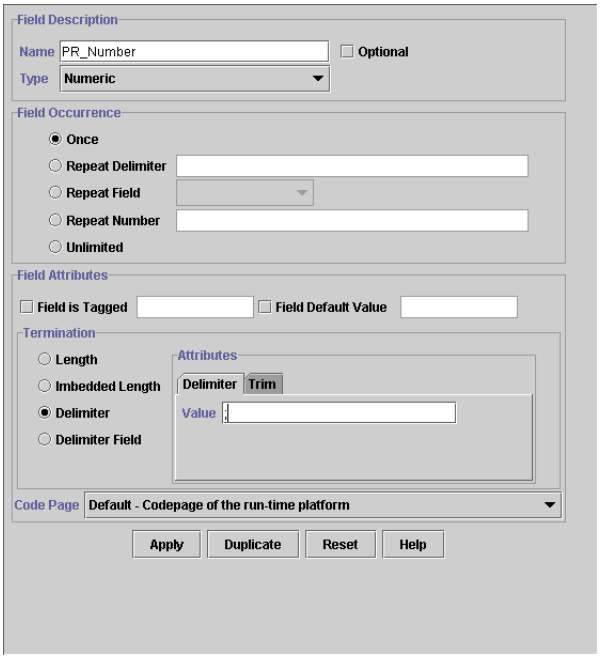
Creating a Field

Fields are a sequence of bytes that have some meaning to an application. (For example, the field `EMPNAME` contains an employee name.) You can create a field as a child of the message format item, as a child of a group, or as a sibling of a group or another field. The field name is used as the element name in the XML document and must comply with XML naming conventions.

To create a field:

1. Select an item in the navigation tree.
2. Choose Insert→Field→As Child if you want to create the field as the child of the message format or group. Choose Insert→Field→As Sibling if you want to create the field as the sibling of another field or a group. The Field Details display in the detail window.

Figure 2-6 Field Details



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

Table 2-7 Field Detail Properties

Field	Description
Field Description	
Name	The name of the field. This name must comply with XML element naming conventions.
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:** The Field Type you select dictates the Field Data Options that appear on the dialog box. Refer to Appendix A, "Supported Data Types," for a list of data types supported by WebLogic Integration.
Field Occurrence	
Occurrence	Choose one of the following to indicate how often this field appears in the message format: ■ **Once** - Indicates the field appears only once. ■ **Repeat Delimiter** - Indicates the field will repeat until the specified delimiter is encountered. ■ **Repeat Field** - Indicates the field will repeat the number of times specified in the field denoted as the repeat field. ■ **Repeat Number** - Indicates the field will repeat the specified number of times. ■ **Unlimited** - Indicates the field will repeat an unlimited number of times. **Note:** Unless a field is defined as optional, the field will occur at least one time.
Note: The fields that display in the following sections of the detail window depend on the Field Type selected.	
Field Attributes	
Field is Tagged	Select this option if this is a tagged field. Being tagged means that 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 box. For example: `SUP:ACME INC, SUP:` is a tag. ACME INC is the field data. If you selected the Field is Tagged option, enter the tag in the text box to the right of the checkbox.

Table 2-7 Field Detail Properties

Field	Description
Field Default Value	Select this option to specify a value for the data of the field that will be inserted into the binary data if the field is not present in the XML. **Note:** If the field does not occur in the binary data and it is not optional, then the binary data will fail to parse even if there is a default value given.
Data Base Type	If the field is a date or time field, the base type indicates what type of characters (ASCII, EBCDIC, or Numeric) make up the data.
Year Cutoff	If the field is a date field that has a 2-digit year, the year cutoff 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 '19' prefix will be added to the year value. Otherwise a '20' prefix will be used.
Code Page	The character encoding of the field data. The default code page is set by choosing Tools→Options dialog box.
Value	The value that appears in a literal field.
Field Termination	
Length	Variable-sized data types can have their length set to a fixed value. ■ **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, enter the necessary value to the data to correct its length.
Imbedded Length	Variable-sized data types can have their termination point specified by an imbedded length. An imbedded length precedes the data field and indicates how many bytes the data contains. ■ **Type** - Specifies the data type and Length or Delimiter for termination if needed. ■ **Tag/Length Order** - Specifies the order of tag and length fields when both are present. Default is tag before length. ■ **Trim Leading/Trailing** - Removes the specified data from the leading or trailing edge of the data.

Table 2-7 Field Detail Properties

Field	Description
Delimiter	Variable-sized data types can have their termination point specified by a delimiter. A delimiter is a value that marks the end of the field. The field data continues until the delimiter is encountered. ■ **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.
Delimiter Field	Variable-sized data types can have their termination point specified by a field that contains a delimiter value. A delimiter is a value that marks the end of the field. The field data continues until the field containing the delimiter is encountered. ■ **Field** - Select the field that contains the delimiter. ■ **Default** - Enter the default delimiter. You must supply a default value. The default is used when the delimiter field is not present. ■ **Trim Leading/Trailing** - Removes the specified data from the leading or trailing edge of the data. For more information on delimiters, see "Specifying Delimiters" on page 2-21.
Decimal Position	Specifies the number of digits (0-16) to the left of the decimal point.
Field Update Buttons	
Apply	Saves your changes to the message format file.
Duplicate	Makes a copy of the field currently displayed. The duplicate field contains the same values as the original field. The name of the duplicate field is the same as the original field name, with the word "New" inserted before the original name. For example, duplicating a field called "Field1" results in a field with the name "NewField1". When you duplicate an item with a numeric value in its name, the new item name contains the next sequential number. For example, duplicating "NewField1" results in a group named "NewField2".
Reset	Discards your changes to the detail window and resets all fields to the last saved values.
Help	Displays online help information for this detail window.

4. Click Apply to save your changes to the message format file, or click Reset to discard your changes to the detail window and reset all fields to the last saved value.

Note: The Apply and Reset buttons are only enabled once changes are made to the detail panel's components.

Creating a Comment

Comments contain 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. You can create a comment as a child or sibling of any message format, group, or field. Comments are unnumbered in the MFL document and are not transformed to the XML or Binary data.

Note: Conventionally, the comment usually precedes the node it intends to document.

To create a comment:

1. Select an item in the navigation tree.
2. Choose Insert→Comment→As Child if you want to create the comment as the child of the selected item. Choose Insert→Comment→As Sibling if you want to create the comment as the sibling of the selected item. The Comment Details display in the detail window.
3. Enter data in the fields as described in the following table.

Figure 2-7 Comment Details

Table 2-8 Comment Detail Properties

Field	Description
Comment Details	Enter the comment text.
Apply	Saves your changes to the message format document.
Reset	Discards your changes to the detail window and resets the field to the last saved value.
Help	Displays online help information for this detail window.

4. Click Apply to save your changes to the message format file, or click Reset to discard your changes to the Comment Details window and reset the field to the last saved value.

   Note: The Apply and Reset buttons are only enabled once changes are made to the detail panel's components.

Creating References

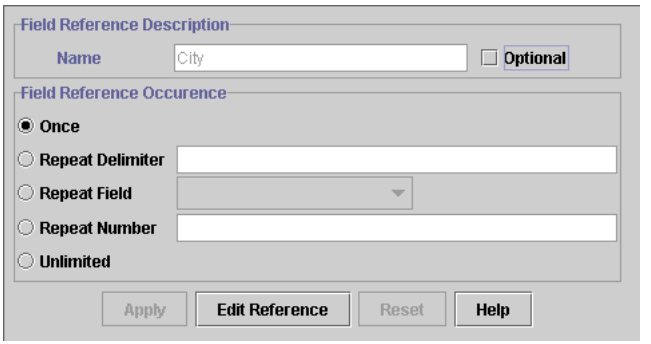
References indicate that the description of the field or group format has been previously defined and you want to reuse this description without re-entering the data. Reference fields or groups have the same format as the original field or group, but you can change only the optional setting and the occurrence setting for the reference field or group. For example, if you have a "bill to" address and a "ship to" address in your data and the format for the address is the same, you only need to define the address format once. You can create the "bill to" address definition and create a reference for the "ship to" address.

Note: References are named exactly the same as the original item. For example, the "bill to" address definition and the "ship to" address definition would be named the same. If you want to reuse a group definition, create a generic group and embed it within a specific group. For example, in the previous example, you can create an *address* group within a *bill_to* group and reference *address* within a *ship_to* group.

To create a reference:

1. Select a field or group in the navigation tree.
2. Choose Edit→Copy.
3. Choose the proper sibling in the navigation tree.
4. Choose Edit→Paste→As Reference.

Figure 2-8 Reference Details



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

Table 2-9 Reference Detail Properties

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

Table 2-9 Reference Detail Properties

Field	Description
Occurrence	
Occurrence	Choose one of the following to indicate how often this reference field or group appears in the message format: ■ **Once** - Indicates the reference appears only once. ■ **Repeat Delimiter** - Indicates the reference will repeat until the specified delimiter is encountered. For more information on delimiters, see "Specifying Delimiters" on page 2-21. ■ **Repeat Field** - Indicates the reference will repeat the number of times specified in the field denoted as the repeat field. ■ **Repeat Number** - Indicates the reference will repeat the specified number of times. ■ **Unlimited** - Indicates the reference will repeat an unlimited number of times.
Field Update Buttons	
Apply	Saves your changes to the message format document.
Edit Reference	Displays the detail window for the original item so you can edit the details of the referenced field or group.
Reset	Discards your changes to the detail window and resets all fields to the last saved values.
Help	Displays online help information for this detail window.

6. Click Apply to save your changes to the message format file, or click Reset to discard your changes to the detail window and reset all fields to the last saved value.

   Note: The Apply and Reset buttons are only enabled once changes are made to the detail panel's components.

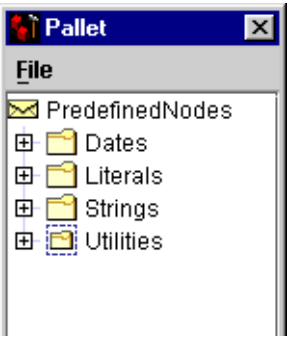
Working with Palettes

The palette allows you to store commonly used message format items and insert them into your message format definitions. These items are stored in an MFL document, and you can use the drag and drop feature (see "Using Drag and Drop" on page 2-15) to copy items from the palette into your message format definition. You can also open any MFL document in the palette and reuse any message format items.

Opening the Palette

To open the palette:

1. Start Format Builder.
2. Choose View→Show Palette. The palette window displays.



You may reorder or change the hierarchy within the palettes by using drag and drop or the Context menu. The contents of the palette are automatically saved when you exit Format Builder.

Note: You can only copy items from the navigation tree to the palette and vice versa. You cannot move items between the windows.

The WebLogic Integration palette contains some common date formats, literals, and strings. You can use these items in the message formats you create, as well as adding your own items to the palette.

Using the File Menu

The following commands are available from the palette's File menu.

Table 2-10 Reference Detail Properties

Menu Command	Description
Open	Opens an existing message Format.
Save	Saves any message format items you have added to the palette, or any existing items you have modified. The default palette is named palette.xml and is stored in the Format Builder installation directory.
Hide Palette	Closes the Palette window.

Using the Shortcut Menu

The following commands are available from the palette's shortcut menu. You can access the shortcut menu by right-clicking within the palette window.

Note: Some commands may be unavailable, depending on the time you have selected in the navigation tree.

Table 2-11 Shortcut Menu Commands

Menu Command	Description
Insert	Inserts a new group in the palette. When you select this command, a window displays asking you to supply the name of the new group.
Rename	Displays a window asking you to supply the new name of the group.
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	Promotes the selected item to the next highest level in the navigation tree. For example, Field1 is the child object of Group1. Selecting Field1 and clicking the Promote tool makes it a sibling of Group1.

Table 2-11 Shortcut Menu Commands

Menu Command	Description
Demote	Demotes the selected item to the next lower level in the navigation tree. For example, group1 is the sibling of Field1. Field1 immediately follows Group1 in the navigation tree. Selecting Field1 and clicking the Demote tool makes it a child of Group1.

Adding Items to the Palette

To add items to the palette:

1. Choose View→Show Palette to display the palette.

   Note: If the Palette window is already displayed, skip this step.

2. From the navigation tree, choose the item you want to add to the palette.
3. Drag the item into the palette window.
4. When the item is placed in the position you want it (as sibling of the selected item), release the mouse button. The item is copied to the palette window.

Notes: You cannot add any node that depends on the existence of another node to the palette. For example, you cannot add Field or Group References, and you cannot add items that have a Repeat Field specified.

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

Deleting Items From the Palette

To delete items from the palette:

1. Select the item in the palette to be deleted and click the right mouse button. The Shortcut Menu displays.
2. Choose Delete. A message displays asking you to confirm the deletion.
3. Click OK to delete the item.

Adding Palette Items to a Message Format

To copy items from the palette to a message format:

1. Choose View→Show Palette to display the palette.

   Note: If the Palette window is already displayed, skip this step.

2. From the palette window, choose the item you want to add to your message format.

3. Drag the item into the navigation tree.

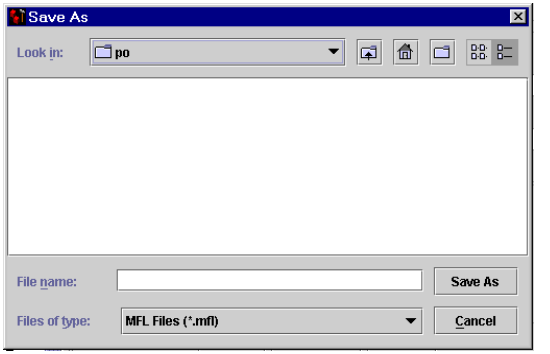
4. When the item is placed in the position you want it (as the sibling of the desired item), release the mouse button. The item is copied from the palette to the message format.

Saving a Message Format to a File

To save a message format file for the first time:

1. Choose File→Save As. The Save As dialog box displays.

Figure 2-9 Save As Dialog Box



2. Navigate to the directory where you want to save the file.

3. In the File Name text box, type the name you want to assign to the file.

4. Format Builder automatically assigns the .mfl extension to message format files by default if no extension is given.
5. Click Save As to save the file in the specified location with the specified name and extension.

To save a message format file using the same name, choose File→Save. The file is saved in the same location with the same name and extension.

To save a message format file using a different name, choose File→Save As and follow steps 1 through 5 above.

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 options.

See Also:

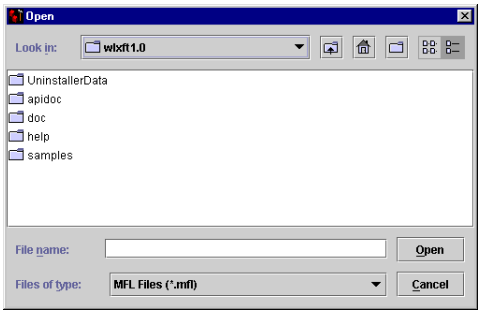
- Changing Options for a Message Format
- Setting Format Builder Options

Opening an Existing Message Format File

To open an existing message format file:

1. Choose File→Open. The Open dialog box displays.

Figure 2-10 Open Dialog Box



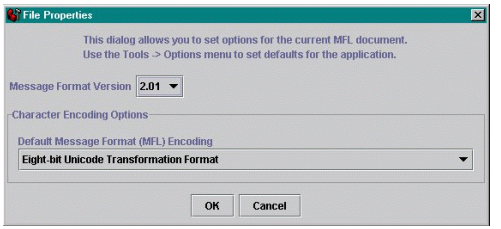
2. Navigate to the directory containing the desired file and select the file name.
3. Click Open. The file is loaded into Format Builder.

Changing Options for a Message Format

To change options for a message format file:

1. Select a Message Format from 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 2-11 File Properties Dialog Box



3. Select the character encoding for the Message Format Layout (MFL) from the list of encoding names and descriptions for this file. (To change the default settings, select Tools→Options.)
4. Click OK. The message format file will reflect your changes when you test it using Format Tester.

Working With the Repository

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

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

The repository allows the supported documents to be shared within WebLogic Integration. The repository provides access to these document types and provides manipulation of repository documents including access to the document text, description, notes, and removal of the document. The repository allows the supported documents to be shared between business process management, WebLogic Server, and B2B integration. The repository also includes a batch import utility that allows previously constructed MFL, DTD, XML Schema, and XSLT documents residing on a different file system to be easily migrated into the repository.

Retrieving Repository Documents

Perform the following steps to retrieve an MFL document from the repository:

1. Start Format Builder.
2. Choose Repository→Log In. The Repository Log In dialog box opens. Enter your user name, password, and the server where the repository resides.
3. Choose Repository→Retrieve. The Select-document-to-retrieve dialog box opens.
4. Select the document you want to retrieve from the document list.
5. Click Retrieve. The Select-document-to-retrieve dialog box is dismissed and you are returned to the Format Builder main window with your selected document listed in the navigation tree.

Once you have retrieved the specified document, you can edit it as you would any MFL document within Format Builder, store the document back into the repository, store the document back into the repository with a different name, or save as a local file.

Anytime you open a document that is stored in the repository, a read-only Document Repository Properties box displays in the Message Format detail panel when the message format node is selected. This properties box provides you with a document description and any notes that were attached to the document.

Storing Repository Documents

Perform the following steps to store a MFL document in the repository:

1. Start Format Builder.
2. Open the MFL document you want to store in the repository.

3. Log in to the repository.
4. Choose Repository→Store As. The Store As dialog box opens.
5. Enter the name you want to associate with this repository document in the Name field.
6. Enter a description of the repository document in the Description field.
7. Enter any notes you would like attached to the document in the Notes field.
8. Click Store. The Store As dialog box is dismissed and your MFL document displays in the navigation tree. A Document Repository Properties box with document Description and Notes information displays in the right pane.

If your Format Builder options specify generation of a DTD/XML Schema, these documents will also be stored in the repository using the supplied name.

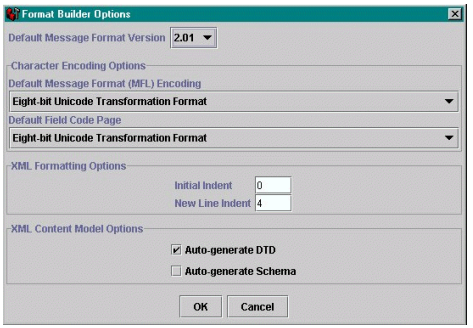
Setting Format Builder Options

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 displays.

Figure 2-12 Format Builder Options Dialog Box



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

Table 2-12 Format Builder Options Properties

Field	Definition
Default Message Format Version	Select the MFL version used when creating new documents. **Note:** Message formats contain their own format version specified on the Message Format pane.
Character Encoding Options	
Default Message Format (MFL) Encoding	Select the character encoding default for the Message Format Layout (MFL) from the list of encoding names and descriptions. This defines the format that your MFL document and XML output will take.
Default Field Code Page	Select the default field code page from the list of binary formats. This selection will be the default code page for each field that is created in your MFL document. It specifies the character encoding of the binary data for each field.
XML Formatting Options	
Initial Indent	Enter the number of spaces to indent the first line of the XML document created by WebLogic Integration.
New Line Indent	Enter the number of spaces to indent a new child line of the XML document created by WebLogic Integration.
XML Content Model Options	
Auto-generate DTD	Generates a DTD document when you save or store the MFL document. This document will be placed in the same directory as the message format when saving to a file and in the repository when storing.
Auto-generate Schema	Generates an Schema file when you save the MFL document. This document will be placed in the same directory as the message format when saving to a file and in the repository when storing.
Action Buttons	
OK	Saves your changes and closes this detail window.
Cancel	Discards your changes and closes this detail window.

Format Builder Menus

The following menus are available in Format Builder.

File Menu

The following commands are available from the File Menu.

Note: Some commands may be unavailable, depending on the actions you have taken.

Table 2-13 File Menu Commands

Menu 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 Properties dialog box for the currently loaded message format.
Exit	Exits Format Builder.

Edit Menu

The following commands are available from the Edit Menu.

Note: Some commands may be unavailable, depending on the actions you have taken and the state of the navigation tree and its items.

Table 2-14 Edit Menu Commands

Menu Command	Description
Undo	Reverses the previous action. The Undo command in the Edit Menu changes to indicate the action that can be undone. For example, changing the name of a field to Field1 and clicking Apply causes the Edit Menu to read “Undo Apply Field Field1”.
Redo	Reverses the effects of an Undo command. The Redo command in the Edit Menu changes to indicate the action that can be redone. For example, changing the name of a field to Field1 and then undoing that action causes the Edit Menu to read “Redo Apply Field Field1”.
Cut	Removes the item currently selected in the left-hand pane, and it’s child objects, from the navigation tree. This item is placed on the clipboard for pasting into another location. **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-hand pane for insertion elsewhere in the navigation tree. **Note:** This action is not available if the Message Format (root) item is selected.
Paste	Inserts the cut or copied item. An additional menu displays when you select Paste. You can choose to paste the item as a child or sibling of the selected item. In addition, you can choose to paste a reference as a sibling of the selected item.
Duplicate	Makes a copy of the item selected in the navigation tree. The duplicate item contains the same values as the original item. The name of the duplicate item is the same as the original item name, but the word “New” is inserted before the original name. For example, duplicating an item called “Field1” results in an item with the name “NewField1”. When you duplicate an item with a numeric value in its name, the new item name contains the next sequential number. For example, duplicating “NewGroup1” results in a group named “NewGroup2”.
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.
Promote	Promotes the selected item to the next highest level in the navigation tree. For example, Field1 is the child object of Group1. Selecting Field1 and clicking the Promote tool makes it a sibling of Group1.

Table 2-14 Edit Menu Commands

Menu Command	Description
Demote	Demotes the selected item to the next lower level in the navigation tree. For example, Group1 is the sibling of Field1. Field1 immediately follows Group1 in the navigation tree. Selecting Field1 and clicking the Demote tool makes it a child of Group1.

Insert Menu

The following commands are available from the Insert Menu.

Table 2-15 Insert Menu Commands

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

View Menu

The following commands are available from the View Menu.

Table 2-16 View menu Commands

Menu Command	Description
Show Palette	Displays the palette window.
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.

Table 2-17 Repository Menu Commands

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.

Tools Menu

The following commands are available from the Tools Menu.

Table 2-18 Tools Menu Commands

Menu Command	Description
Import	Displays a list of the installed importers. Choose the importer from which you want to import a message.
Test	Opens 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 2-19 Help Menu Commands

Menu Command	Description
Help Topics	Displays the main Help screen.
How Do I	Provides step-by-step instructions for performing the basic tasks in Format Builder.
About	Displays version and copyright information about Format Builder and the JDK you are running.

3 Testing Format Definitions

Once you have built a format definition, you can test it using Format Tester. Format Tester parses and reformats data as a validation test and 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 Format Tester
- Using the Format Tester Main Window
- Testing Format Definitions
- Debugging Format Definitions

Starting Format Tester

To start Format Tester:

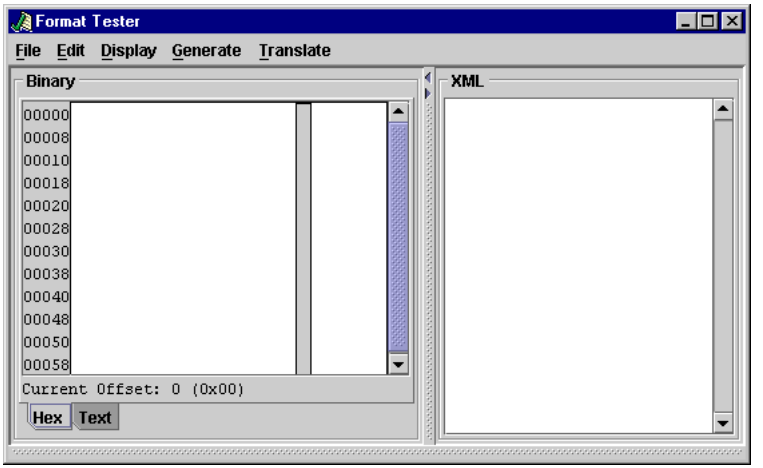
1. Start Format Builder by choosing Start→Programs→BEA WebLogic E-Business Platform→WebLogic Integration 2.1→Format Builder. The Format Builder main window displays.

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

2. Choose Tools→Test. The Format Tester dialog box displays.

   Note: The Tester works with the currently loaded message definition document.

Figure 3-1 Format Tester Dialog Box



Using the Format Tester Main Window

The following topics discuss the elements of the Format Tester main window and provide instructions for navigating and executing commands from the Format Tester main window.

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

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

Using the Menu Bar

The following menus are available in Format Tester. All Format Tester menus are expandable from your keyboard by pressing Alt + *mnemonic key*. Some menu commands are also executable using Ctrl + *letter* accelerator keys.

Figure 3-2 Menu Bar

File Menu

The following commands are available from the File menu.

Table 3-1 File Menu Commands

Menu Command	Description
Open Binary	Allows you to select a file to be displayed in the binary window. **Note:** The default file extension for binary files is .DATA.
Open XML	Allows you to select a file to be displayed in the XML section of the Format tester window. **Note:** The default file extension for XML files is XML.
Save Binary	Allows you to save the contents of the binary window.
Save XML	Allows you to save the contents of the XML window.
Debug Log	Allows the debug information to be saved in a text file.
Close	Closes the Format Tester window.

Edit Menu

The following commands are available from the Edit menu

Table 3-2 Edit Menu Commands

Menu Command	Description
Cut	Removes the currently selected text and places it on the clipboard for pasting into another location.
Copy	Copies the currently selected text and places it on the clipboard for pasting into another location.
Paste	Inserts the cut or copied text at the cursor location.
Find	Allows you to search for a hex or text value in the binary data.
Find Next	Continues your search to the next instance of the specified value.
Go To	Allows you to move the cursor in the binary editor to a specified byte offset.

Display Menu

The following commands are available from the Display menu

Table 3-3 Display Menu Commands

Menu Command	Description
XML	Allows the XML data panel to be hidden or shown. If hidden, the binary window expands to fill the width of the tester. The To XML button remains, but the splitter disappears.
Debug	Allows the Debug output window to be hidden or shown.
Clear→Binary	Resets the contents of the binary window to be empty.
Clear→XML	Resets the contents of the XML window to be empty.
Clear→Debug	Resets the contents of the debug window to be empty.
Hex→Offsets as Hexadecimal	Displays the offset values as hexadecimal. Selecting this option turns off the *Offsets as Decimal* display.
Hex→Offsets as Decimal	Displays the offset values as decimal. Mutually exclusive with the *Offset as Hexadecimal* selection.
Text→Values in ASCII	Changes the character set used when displaying the text portion of the hex editor display to ASCII. Mutually exclusive with the *Values in EBCDIC* menu selection.
Text→Values in EBCDIC	Changes the character set used when displaying the text portion of the hex editor display to EBCDIC. Mutually exclusive with the *Values in ASCII* menu selections.

Generate Menu

The following commands are available from the Generate menu.

Table 3-4 Generate Menu Commands

Menu Command	Description
Binary	Generates binary data to match the current format specification.
XML	Generates XML data to match the current format specification.
Prompt while generating data	If selected, you are prompted during the generation process to determine if optional fields or groups should be generated, determine which choice of children should be generated, and determine how many times a repeating group should repeat.

Translate Menu

The following commands are available from the Translate menu.

Table 3-5 Translate Menu Commands

Menu Command	Description
Binary to XML	Converts the contents of the binary window to XML.
XML to Binary	Converts the contents of the XML window to binary.

Using the Shortcut Menus

Instead of using the standard menus to find the command you need, use the right mouse button to click an item in the pop-up shortcut menu.

The following commands are available from the Shortcut menus.

Note: Some commands may be unavailable, depending on which display panel the mouse pointer is currently in.

Table 3-6 Shortcut Menu Commands

Menu Command	Description
Cut	Removes the currently selected text and places it on the clipboard for pasting into another location.
Copy	Copies the currently selected text and places it in the clipboard for pasting into another location.
Paste	Inserts the cut or copied text at the cursor location.
Clear	Resets the contents of the binary, XML, or Debug window to be empty.
Generate	Generates binary or XML data to match the current format specification.
To XML	Converts the contents of the binary window to XML.
To Binary	Converts the contents of the XML window to binary.
Text in ASCII	Changes the character set to ASCII when displaying text with the hex editor.
Text in EBCDIC	Changes the character set to EBCDIC when displaying text with the hex editor.

Using the Binary Window

The binary data display panel acts as hexadecimal editor, displaying data offsets, the hex value of individual bytes, and the corresponding text, which can be optionally displayed as ASCII/EBCDIC characters. The Binary window consists of the following three tabs used to select which mode to display the data:

- Hex Editor
- Text

The editor allows for editing of the hex byte or the 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 hexadecimal editor allows you to display your data offsets as Hexadecimal or Decimal.

To change your data offsets:

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

Using the Text Feature

To use the Text feature, select the Text tab from within the Binary window to view all printable characters. For example, carriage returns are shown as line breaks. If you have non-printable characters, the Text window displays them as small squares.

Using the XML Window

The XML data panel displays XML data that has been converted or translated from the contents of the Binary panel, or XML that is to be converted to Binary. The contents of the XML panel can be cleared or edited to suit your needs. When XML is generated, the XML Formatting Options specified in the Format Builder options dialog box are used. Refer to “Setting Format Builder Options” on page 2-38 for more information.

Using the Debug Window

The Debug window displays the actions that take place during the translation operation, any errors that are encountered, and field and group values along with delimiters. To determine the location of the error, determine the last field that parsed successfully and examine the specification of the next field on the navigation tree.

When you open the Format Tester, 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 feature allows for full debug information to be captured to a file. See "Using the Debug Log" on page 3-13 for more information.

Note: Using the Debug window or Log File increases the time required to translate from XML to Binary.

Using the Resize Bars

Resize bars are located between the Binary, XML, and Debug windows and enable each window to be resized to suit your needs. Each resize bar can selected and dragged up and down, or left or right, as appropriate, to enlarge one of the windows and reduce the other.

Each resize bar also contains two directional buttons that can be clicked to enlarge or diminish any of the three windows.

Testing Format Definitions

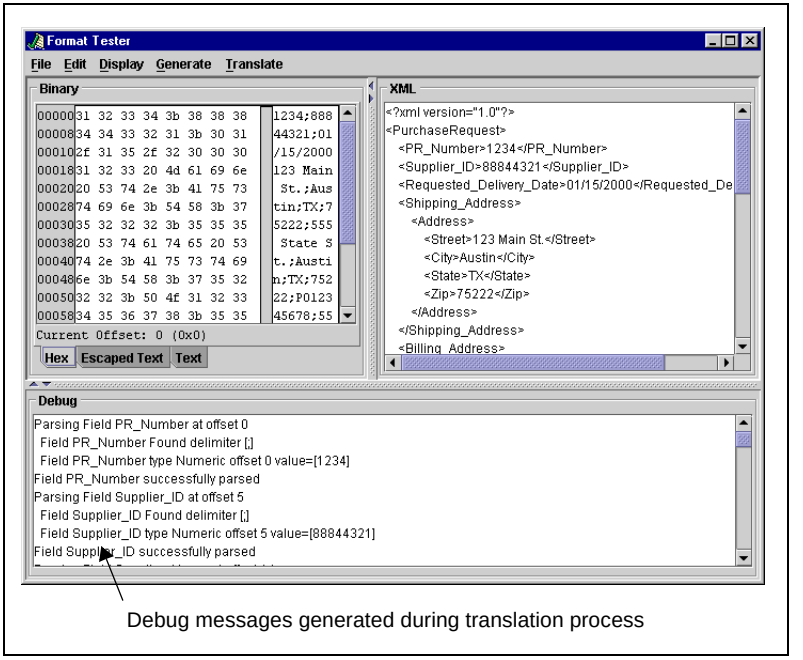
Perform the following steps to test a message format definition.

1. Start Format Builder.
2. Open a Message Format file.
3. Start Format Tester.
4. Click 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. Select Display→Debug if you want to view the actions that take place during the translation operation. This step is optional, but you must open the Debug window prior to the translation operation in order to view any debug information later.

6. Select Translate→Binary to XML, or Translate→XML to Binary to translate your data to the appropriate format. The translated data displays in the Binary or XML window.

Figure 3-3 Format Tester



7. Correct the errors, if present, and test the translation again.

8. Continue this process until the translation is successful.

Note: You can leave Format Tester open while you modify the Message Format from within Format Builder. Any changes to the message definition are automatically detected by Format Tester.

Debugging Format Definitions

The following topics discuss the various Format Tester features you can use to debug and correct your data.

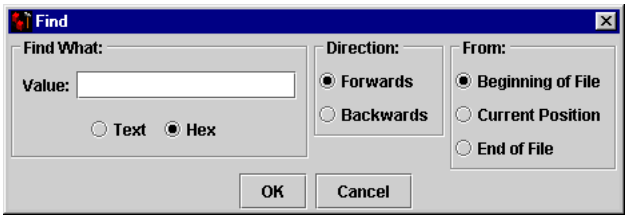
- Searching for Values
- Positioning to an Offset
- Using the Debug Log

Searching for Values

The Format Tester Find feature allows you to search for hex or text values in the binary data. Perform the following steps to perform a hex or text value search.

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

Figure 3-4 Find Dialog Box



3. Enter the value you want to find.
4. Select Text or Hex to specify what type of value you are searching.
5. Specify the direction you want to search.
6. Specify the position in the file from which you want your search to begin.

7. Click OK. The Find dialog box disappears and your cursor displays next to the value for which you are searching.
8. Choose Edit→Find Next to search to the next instance of the specified value.

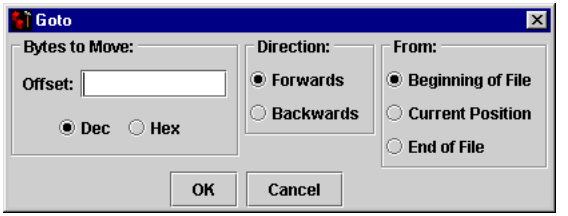
Positioning to an Offset

The Format Tester GoTo feature allows you to move the cursor in the binary editor to a byte offset you specify.

To move to a specified offset:

1. From within Format Tester, choose Edit→Go To. The Go To dialog box displays.

Figure 3-5 Goto Dialog Box



2. Enter the offset value you want to go to.
3. Select either Dec or Hex to specify the type of offset you want to go to.
4. Select either Forwards or Backwards to specify the direction you want your search to proceed within the file.
5. Select either Beginning of File, Current Position, or End of File to specify the starting position of your search within the file.
6. Click OK to have your cursor placed next to the offset you are looking for. The Go To dialog box disappears after you click the OK button.

Using the Debug Log

The Format Tester debug log feature allows you to save your debug information in a text file.

To use the Format Tester debug log, choose File→Debug Log. You will be presented with a standard Save As dialog box from which you can select the destination directory and name for this log file. If you select an existing file, the new debug information will be appended onto the end of the existing file.

Format Tester saves debug logs to the currently selected directory or the last directory selected.

4 Importing Meta Data

WebLogic Integration provides three utilities that allow you to import COBOL copybooks, convert C structure definitions, and convert FML Field table Classes into MFL files. The following topics provide information on how to perform these types of imports.

- 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 creating a message definition to translate the COBOL data. When importing a copybook, comments are used 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 displays.

Figure 4-1 COBOL Copybook Importer

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

Table 4-1 COBOL Copybook Importer Field Descriptions

Field	Description
File Name	Type the path and name of the file you want to import.
Browse	Click to navigate to the location of the file you want to import.
Byte Order	
Big Endian	Select this option to set the byte order to Big Endian. **Note:** This option is used for IBM 370, Motorola, and most RISC designs (IBM mainframes and most Unix platforms).
Little Endian	Select this option to set the byte order to Little Endian. **Note:** This option is used for Intel, VAX, and Unisys processors (Windows, VMS, Digital, Unix, and Unisys).
Character Set	
EBCDIC	Select this option to set the character set to EBCDIC. **Note:** These values are attributes of the originating host machine.
ASCII	Select this option to set the character set to ASCII. **Note:** These values are attributes of the originating host machine.

Table 4-1 COBOL Copybook Importer Field Descriptions

Field	Description
Other	The character encoding of the field data is selected using a list of code pages.
Action Buttons	
OK	Imports the COBOL Copybook using the settings you defined.
Cancel	Closes the dialog box and returns to Format Builder without importing.
About	Displays information about the COBOL Copybook importer including version and supported copybook features.

Once you have imported a copybook, you may work with it as you would any message format definition. If an error or unsupported data type is encountered in the copybook, a message is displayed informing you of the error. You can choose to display the error or save the error to a log file for future reference.

The following table provides a listing and description of the sample files installed for the COBOL Copybook Importer. All directory names are relative to the WebLogic Integration installation directories in the `samples\di\` subdirectory.

Table 4-2 COBOL Copybook Sample 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 two types of output data:

- MFL document
- C Code

Both the MFL document and C code output methods require a .c or .h input file to be specified, parsed, and the desired structure selected before choosing whether to generate MFL (default) or C code.

In addition to the requirement that all input to the parser consist of valid C code, all outside references, such as #include(s), #define(s), and typedef(s) must be resolved prior to use. This may require hand editing or use of the compiler's preprocessor.

There are platform considerations that affect the description of data for C code. For example, the length of a `long` on a platform will affect the binary data that conforms to a particular structure definition.

There are two methods 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, by choosing to generate C code source, you may compile the C code on the desired machine. The compiler on that machine accounts for the necessary platform dependent information. This will produce an executable file that when run will produce an MFL document and binary data that conforms to that MFL in separate files. 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 newly generated 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\di\cgf subdirectory.

C Struct Importer Sample Files

The following table provides a listing and description of the sample files installed for the C Struct Importer. All directory names are relative to the WebLogic Integration installation directory in the `samples\di\` subdirectory.

Table 4-3 C Struct Importer Sample Files

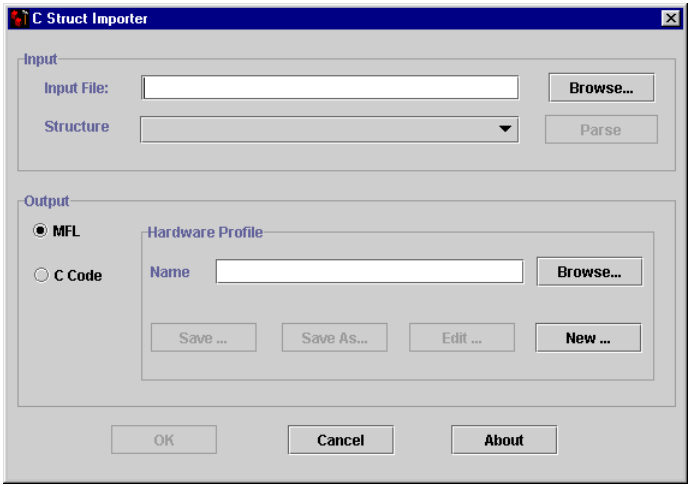
Directory	File	Description
C	emprec5.h	A C version of the emprec5.cpy sample Copybook file, with some typedefs.
C	emprec5n.h	A variant of the emprec5.h file using a nested struct definition, but no typedefs.
C	emprec5s.h	A simple version of the emprec5.h file.
C	ntfsez.h	A small sample extracted from the ntfs.h file. Designed to test recursive typedefs.
Cfg	cprofile.c	The source code for the cprofile.c utility. This utility is designed to generate profiles on various platforms.
The following .cfg files were all generated by the cprofile program on various platforms. Each .cfg file contains the 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 E-Business Platform→WebLogic Integration 2.1→Format Builder. The Format Builder main window displays.
2. Choose Tools→Import→C Struct Importer. The C Struct Importer dialog box displays.

Figure 4-2 C Struct Importer Dialog Box



The C Struct Importer dialog box opens with MFL specified as the default output and contains the following fields.

Table 4-4 C Struct Importer Field Descriptions

Field	Description
Input	
Input File	Enter the path and name of the file you want to import.
Browse	Click Browse to navigate to the directory containing the file you want to use.
Structure	This list box is populated with the list of structures found in the input file after it has been successfully parsed.
Parse	Click Parse to parse the input file. If successful, the Structure list box is populated with the list of structures found in the input file.
Output	
MFL	If you select this option button, you can generates MFL from a structure definition and a hardware configuration file. You will see the Hardware Profile group box.
C Code	If you select this option button, you can generate C source code to compile on the target machine and execute to produce MFL. You will see the C Code File Names group box.
Hardware Profile	
Name	Specify an existing profile either by entering the file name or using the Browse button. The prebuilt hardware profiles may be found in the `samples\di\cfg` directory.
Browse	Click Browse to navigate to the directory containing the file you want to use.
Save	Saves the current hardware profile.
Save As	Saves the current hardware profile under another name.
Edit	Click Edit to edit the current hardware profile listed in the Hardware Profile Name field.
New	Click New to create a new hardware profile.

Figure 4-3 C Struct Importer Dialog Box

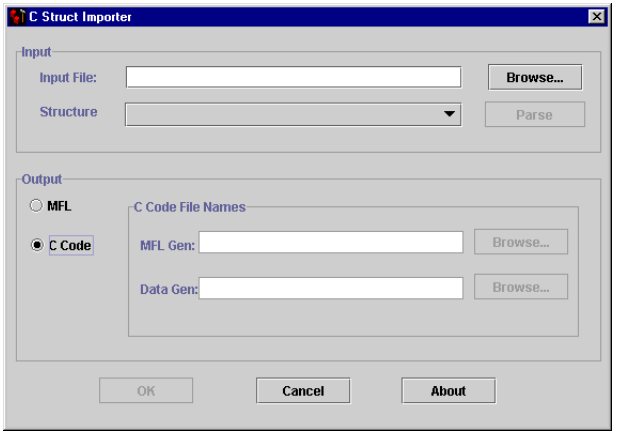


Table 4-5 C Struct Importer Dialog Box Fields

Field	Description
C Code File Names	
MFL Gen	Specifies the C source code file name that must be compiled on the target machine to generate MFL.
Browse	Click Browse 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.
Browse	Click Browse to navigate to the directory where you want the file to reside.
Action Buttons	
OK	Click OK to save your hardware profile changes.
Cancel	Click Cancel to dismiss your hardware profile changes.
About	Click About to view C Struct Importer version and date information.

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 located in the 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 complier normally used and should compile and execute 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 following commands for each of the platforms listed.

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

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

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

- On Unix:

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

Running the Hardware Profile Utility

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

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

The optional description is placed in the configuration file as the `DESCRIPTION` value. If the description contains embedded blanks, enclose it in quotes.

Generating MFL

Perform the following steps to generate MFL.

1. Enter a file name in the Input File field, or click Browse to select a file.
2. Click Parse to parse the file.

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

   Note: If your file does not parse correctly, it is recommended that you proceed in one of two ways:

   - Run your `.h` or `.c` source code through the compilers preprocessor and run that output through the parser.
   - Comment out the character creating the parse failure and attempt to parse again. Please note that the parser fails at the first failure it encounters.

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

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

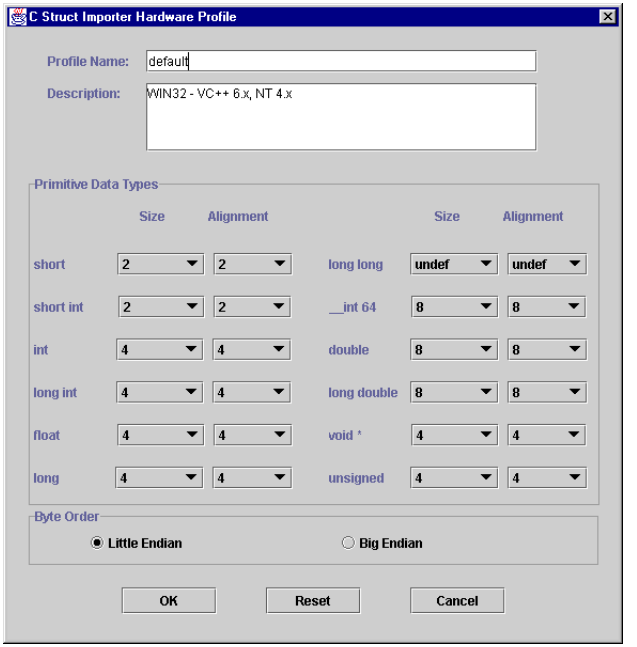
4. Specify an existing profile or create a new one by performing one of the following procedures.

   - Specify an existing profile either by entering the file name in the Hardware Profile Name field, or click Browse to select a file. Click Edit to open the hardware profile editor if you need to view or edit the profile parameters.

   ```
   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. This opens the Hardware Profile editor loaded with the default parameters. Specify a Profile Name, a description, and modify the primitive data types and byte order to suit you needs.

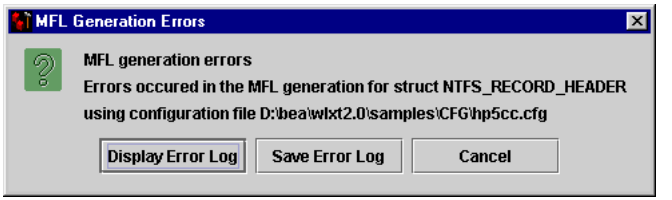
Figure 4-4 C Struct Importer Hardware Profile Dialog Box



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 displays providing you the opportunity to view or file the error log.

Figure 4-5 MFL Generation Errors Dialog Box



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

Once you have determined what errors were generated, you can return to the C Struct Importer and repeat the prior steps.

Generating C Code

Perform the following steps to generate C code.

1. Enter a file name in the Input File field, or click Browse to select a file.
2. Click Parse to parse the file.

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

   Note: If your file does not parse correctly, it is recommended that you proceed in one of two ways:

   - Run your `.h` or `.c` source code through the compilers preprocessor and run that output through the parser.
   - Comment out the character creating the parse failure and attempt to parse again. Please note that the parser fails at the first failure it encounters.
3. Select the desired structure from the Structure drop-down list box.
4. Select the C Code option button.
5. Enter a file name in either the MFL Gen or Data Gen fields, or click Browse to select a file.
6. Click OK. You will be warned about overwriting existing files and notified about the success or failure of the code generation.
7. Copy the generated source code to the target platform and compile and execute it.

   Note: You must copy the input file containing the struct declarations as well. Both programs, when compiled, take an argument of the output file name.
8. Copy the generated MFL or data back to the platform running Format Builder.

Importing an FML Field Table Class

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

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

FML Field Table Class Importer Prerequisites

Perform the following steps prior to 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. Please refer to the WebLogic Server documentation for information on FML Field Table Administration.
3. Compile the source code. The resulting class files are called `fldtbl` classes because they implement the `FldTbl` interface. These `fldtbl` classes must be moved to a location specified in the Format Builder `CLASSPATH`.

The `samples/fml` directory contains several `fldtbl` class fields that you can use as samples. These samples allow you to work through the following steps without having completed the previous three steps.

Note: These steps are normally done when configuring WebLogic Tuxedo Connector, so these class files may already exist.

FML Field Table Class Sample Files

The following table provides a listing and description of the sample files installed for the FML Field Table Class Importer. All directory names are relative to the WebLogic Integration installation directory in the `samples\di\` subdirectory.

Table 4-6 FML Field Table Class Sample Files

Directory	File	Description
`fml\`	`bankflds.class`	A compiled source file that serves as input to the FML Field Table Class Importer.
`fml\`	`bankflds.java`	A `fldtbl` source file generated by the `mkfldclass` utility.
`fml\`	`crdtflds.class`	A compiled source file that serves as input to the FML Field Table Class Importer.
`fml\`	`crdtflds.java`	A `fldtbl` source file generated by the `mkfldclass` utility.
`fml\`	`tBtest1flds32.class`	A compiled source file that serves as input to the FML Field Table Class Importer.
`fml\`	`tBtest1flds32.java`	A `fldtbl` source file generated by the `mkfldclass` utility.

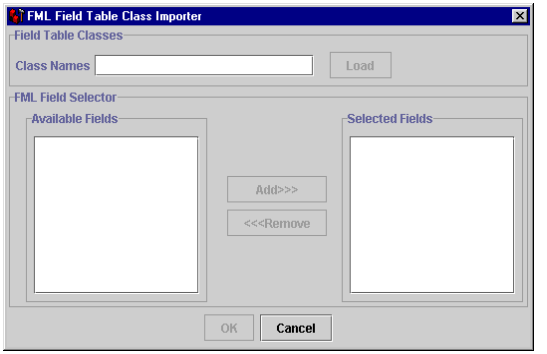
Creating XML with the FML Field Table Class Importer

Perform the following steps 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. Once you place them in the \ext directory you can populate the Available Fields list automatically from the FML Field Table Class Importer dialog box.

1. Start Format Builder. The Format Builder main window displays.

2. Choose Tools Import FML Field Table Class Importer. The FML Field Table Class Importer dialog box displays.



3. Enter the name of the `fldtbl` class file that is to be processed into the Class Names field.

   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. The list should be comma separated and each name does not have 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 are not in Format Builder's CLASSPATH, then an error dialog box is displayed. However, the valid `fldtbl` classes in the list will still be processed.

4. Click Load. The names of the fields from the field tables appear in the Available Fields list. The Available Fields list does not allow for duplicate names. If the name of a field appears in different field tables, it will only appear once on the list.

5. Select the desired fields from the Available Fields list and click Add. The selected fields will appear in the Selected Fields list. To remove a field from the Selected Fields list, select the desired field and click Remove.

6. Click OK after you have successfully selected all the necessary field names. The FML Field Table Class Importer dialog box will closes and the generated MFL will appear in the Format Builder navigation tree. The selected fields will be listed in the order they appear in the selected Fields list.

7. Edit the created MFL document to specify the order and occurrences of the fields that will be in the XML document which is passed to the WebLogic Tuxedo Connector FML/XML Translator from business process management.
8. Choose Tools→Test to open Format Tester.
9. From the Format Tester menu bar, choose Generate→XML.
10. Format Tester now creates an XML document that conforms to the MFL document in Format Builder. Edit the data content of the fields as desired.
11. Choose File→Save XML to save the XML document to the name and location of your choice.

    The created XML may be imported into business process management by using the XML instance editor. Refer to the business process management documentation for information on importing XML.

5 Retrieving and Storing Repository Documents

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

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

The repository allows the supported documents to be shared in WebLogic Integration. The repository provides access to these document types and provides manipulation of repository documents including access to the document text, description, notes, and removal of the document. The repository allows the supported documents to be shared between data integration, business process management (BPM), WebLogic Server, and B2B integration. The repository also includes a batch import utility that allows previously constructed MFL, DTD, XML Schema, and data integration documents residing on a different file system to be easily migrated into the repository.

This section discusses the following topics:

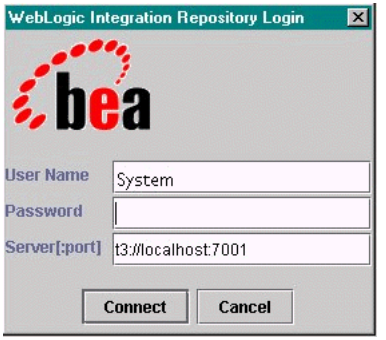
- Accessing the Repository
- Retrieving Repository Documents
- Storing Repository Documents
- Importing Documents into the Repository
- Using the Repository Document Chooser

Accessing the Repository

To access the repository:

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

Figure 5-1 WebLogic Integration Repository Login Dialog Box



3. Enter the userid specified for the connection in the User Name field.
4. Enter the password specified for the connection in the Password field.
5. Enter the server name and Port number in the Server[:port] field.

   Note: The WebLogic Integration Repository Login window allows up to three unsuccessful login attempts, after which, a login failure message is displayed. If you experience three login failures, choose Repository→Log In to repeat the login procedure.

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

The following commands are available from the Repository Menu.

Table 5-1 Repository Menu Commands

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 Repository Documents

Perform the following steps to retrieve repository documents:

1. Start Format Builder.
2. Log in to the WebLogic Integration Repository.
3. Choose Repository→Retrieve. The Select-document-to-retrieve dialog box opens.
4. Select the document you want to retrieve from the document list.

   Note: Only MFL file types are available in the Format Builder repository.
5. Click Retrieve. The Select-document-to-retrieve dialog box is dismissed and you are returned to the Format Builder main window with your selected document listed in the navigation tree.

Once you have retrieved the specified document, you can edit it as you would any MFL document within Format Builder, store the document back into the repository, store the document back into the repository with a different name, or save as a local file.

Anytime you open a document that is stored in the repository, a read-only Document Repository Properties box displays in the Message Format detail panel when the message format node is selected. This properties box provides you with a document description and any notes that were attached to the document.

Storing Repository Documents

Perform the following steps to store a MFL document in the repository:

1. Start Format Builder.
2. Open the MFL document you want to store in the repository.
3. Log in to the repository.
4. Choose Repository→Store As. The Store As dialog box opens.

   Note: Please do not add the MFL file extension to your entry. Only MFL file types are available in the Format Builder repository.

5. Enter the name you want to associate with this repository document in the Name field.
6. Enter a description of the repository document in the Description field.
7. Enter any notes you would like attached to the document in the Notes field.
8. Click Store. The Store As dialog box is dismissed and your MFL document displays in the Format Builder navigation tree. A Document Repository Properties box with document Description and Notes information displays in the Detail panel of the right pane of Format Builder.

If your Format Builder options specify generation of a DTD/XML Schema, these documents will also be stored in the repository using the supplied name.

Importing Documents into the Repository

The WebLogic Integration repository batch import utility provides a command line interface to the repository. It provides a mechanism for easily importing previously built MFL documents into the repository. The batch importer is capable of importing MFL, DTD, class, XSLT, and XML schema documents in any combination. The Batch Importer also works with any plug-in repository.

Use of the repository importer with the business process management repository requires the placement of a `wlxt-repository.properties` file in a `CLASSPATH` directory. The contents of this file identify the WebLogic Server hosting the repository. For example:

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

At the console command prompt, invoke the Batch Import Utility using the following command.

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

The following table describes the options.

Table 5-2 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 that follow. 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 that follow. Verbose mode is disabled by default.
-f	Optional switch 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 path names from the repository root folder. Folder names within a path should be separated by a forward slash.
-t	Optional switch specifying the default type of all the following files. 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 5-2 Options for the Import Command

Option	Description
files	specifies one or more filenames to be imported. Wildcards may be used based on the current command line shell.

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

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

The document type of imported documents is derived from the file extension as follows:

Table 5-3 Supported Document Types and Extensions

File Extension	Document Type Assigned
.dtd	DTD
.xsd	XML Schema
.mfl	MFL
.class	Java Class
.xsl	Extensible Stylesheet Language
anything else	Default type (defaults to MFL)

Using the Repository Document Chooser

The Repository Document Chooser provides a user interface exposing the contents of the repository. The user interface consists of six different dialog and message boxes that allow you to store, retrieve, and modify repository documents.

Using the Open Document Dialog Box

The following is an image of the Open Document dialog box used for retrieving repository documents.

Figure 5-2 Select Document To Retrieve Dialog Box

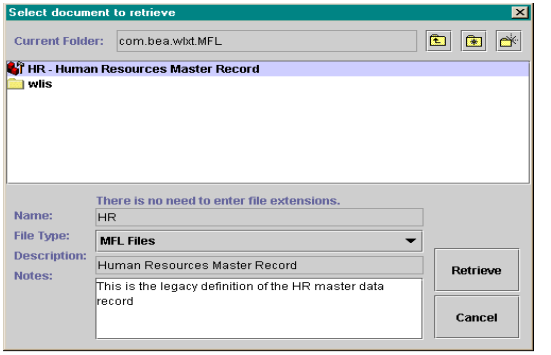


Table 5-4 Select Document To Retrieve Dialog Box Fields

Field	Definition
Current Folder	This field specifies the name of the current repository folder.
Up Folder icon	The button is used 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	This field is used to make the root the current folder if the current folder is not the root folder of the repository.
New Folder icon	This button is used to create a new child folder in the current folder. This icon is disabled if the repository does not support folders.
Document	The list field contains all the MFL documents in the current folder. Each entry in the list is prefixed by an icon indicating the type of object: Folder or MFL. Selecting an entry in the list causes its information to be displayed. Double-clicking an entry in the list causes it to be retrieved or become the current folder.
Retrieve	This button opens the selected document. If a folder is currently selected, pressing this button causes it to become the current folder.

Table 5-4 Select Document To Retrieve Dialog Box Fields

Field	Definition
Cancel	This button closes the dialog box without applying changes.
Name	This field specifies the name of the currently selected document folder.
Description	This field describes of the currently selected document folder.
Notes	The provides the notes attached to the currently selected document folder.

Using the Store Document Dialog Box

The Store Document dialog box differs from the Open Document dialog box only in its ability to enter a new document name, description, or notes.

Using the Shortcut Menus

Both the Open Document and Store Document dialog boxes provide the capability to update repository objects, rename existing objects, and remove objects. These update features are accessed by right-clicking on a repository object. The following is an image of the shortcut menus.

Figure 5-3 Store Document Dialog Box

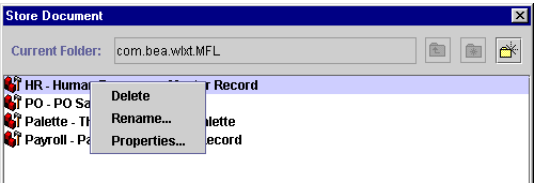


Table 5-5 Store Document Dialog Box Fields

Menu Command	Description
Delete	Opens a dialog box that warns you that an object is about to be permanently removed from the repository.
Rename	Opens a dialog box where you can rename the document in the repository.
Properties	Opens a dialog box where you can update the description and notes of the selected object.

6 Using the Run-Time Component

The data translation run-time component of WebLogic Integration consists of a Java class named WLXT (referred to as "the Java class" in this chapter). This class has various methods used to translate data between binary and XML formats. This Java class can be deployed in an EJB using WebLogic Server, invoked from a workflow in business process management, 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 include code samples that illustrate how to use data translation run-time component to parse binary data into XML, and serialize XML into binary:

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

Binary to XML

The following code listing uses the `parse()` method of the data translation run-time component to parse a file containing binary data 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}
```

In the prior listing, a new instance of the Java class is instantiated at line 12. A Uniform Resource Locator (URL) is created for a MFL file that was previously created with Format Builder. A FileInputStream is created for some binary data that exists in the file `mybinaryfile`. The URL for the MFL document, and the stream of binary data, are then passed into the parse method at line 16. The parse method converts the binary

data into an instance of a W3C Document object. This object can be converted to XML text via the `getXMLText()` method (as shown on line 17), or manipulated directly via the W3C DOM API.

Generating XML with a Reference to a DTD

The data translation includes `parse()` methods that allow a reference to a Document Type Definition (DTD) or an XML Schema to be output in the resulting XML document. The following listing illustrates this generation.

Listing 6-2 Sample XML Generation with a DTD Reference Code Example

```
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 4-2 and Listing 4-1 occurs in line 16. On line 16, a different parse method is invoked that allows a DTD file to be specified (`mydtd.dtd`), so that it is referenced in the resulting XML document.

Thus, the resulting XML has a DOCTYPE statement that refers to the DTD `mydtd.dtd` (see the following example).

```
<?xml version=”1.0”?>
<!DOCTYPE someRootNode SYSTEM ‘mydtd.dtd’>
```

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

Passing in a Debug Writer

All of the `parse()` methods allow a PrintWriter to be passed in as the last parameter of the `parse()` method. If this parameter is not null, WebLogic Integration will print debug messages to this PrintWriter. This allows you to debug the translation if the MFL document and the binary data do not agree. If debug messages are not desired, pass in null for this parameter as shown in the previous listings.

Listing 6-3 Passing in a Debug Writer Sample

```
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 }
```

At line 17, as a last parameter to the `parse()` method, a PrintWriter object is created from the System.out PrintStream. This will cause debug messages such as the ones shown below to be written to 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

The following code listing illustrates using WebLogic Integration 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 }
```

In the code example above, a new instance of the Java class is created at line 12. Then a URL is created for an MFL file, and a `FileInputStream` is created for a file containing XML text. A `FileOutputStream` is also instantiated to store the binary data that will result from the XML to binary translation. On line 17, the `serialize()` method is invoked, to translate the XML data contained in the `FileInputStream` 'in' (`myxml.xml`), to the binary format described in 'mymfl.mfl'. This binary data is written to the FileOutputStream 'out' (which is the file '`mybinaryfile`').

Converting a Document object to Binary

The listing below illustrates converting a W3C Document object to a binary format.

Listing 6-6 Converting a Document Object to Binary

```
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 illustrates passing in a Document object to the `serialize()` method of the Java class. This is useful when your application already has XML in the form of a Document object, or 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 on line 33, to convert it to the binary format specified by the MFL file '`mymfl.mfl`'.

Passing in a Debug Writer

The serialize methods also support passing in a PrintWriter parameter for the logging of debug messages. An example invocation of the serialize method with a PrintWriter object is given below.

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

This will cause debug messages such as the ones shown below to be written to the console.

The following code represents debug output.

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 data translation also provides methods to transform XML via XSLT. XSLT is a language for transforming XML documents. A XSLT stylesheet is an XML document that describes transformations that are to be performed on the nodes of an XML document. The Java class provides `transform()` methods that apply an XSLT stylesheet to an XML document. Using a stylesheet, an XML document can be transformed into HTML, PDF, or another XML dialect.

The listing below illustrates transforming an XML document using one of the transform 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  }
```

On line 15, an instance of WLXT is created. On the following line a URL is created for a previously created XSLT stylesheet. A `FileInputStream` is then created for a file containing XML text. A `FileOutputStream` is also created for the text that results from the XSLT transformation. On line 21, a `transform()` method of the Java class is invoked to transform the XML in the file '`myxml.xml`', according to the XSLT stylesheet '`mystylesheet.xsl`'. The output of the transformation is written to the file '`myoutputfile`'.

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 an `parse()`, `serialize()`, or `transform()` method. This greatly improves the performance of these methods, since the MFL document or XSLT stylesheet has already been processed and cached. This is particularly useful when data translation is used in an EJB or servlet, where 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 file name of a Properties file as a parameter. This `init()` method will retrieve the '`WLXT.stylesheets`' and '`WLXT.MFLDocuments`' properties from the Properties object. Each property is expected to contain a comma-delimited list of documents that are to be preprocessed and cached. When these documents are later referenced in a `parse()`, `serialize()`, or `transform()` method, the preprocessed version will be retrieved from the cache. The listing below demonstrates using an `init()` method to initialize an instance of the Java class.

Listing 6-9 Properties file myconfig.cfg:

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

Listing 6-10 Source code example of init() method using file 'myconfig.cfg'

```
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  }
```

The `init()` method on line 20 of the listing above, causes the object to preprocess the documents listed in the file 'myconfig.cfg'. When an MFL document is specified in the `parse()` method of line 34, this MFL document has already been processed an cached inside the object. The same is true of the stylesheet that is referenced in the invocation of the `transform()` method on line 46.

Java API Documentation

For the complete reference to using the Java class, see the Java API Documentation located in the `doc\apidoc` subdirectory of your WebLogic Integration installation.

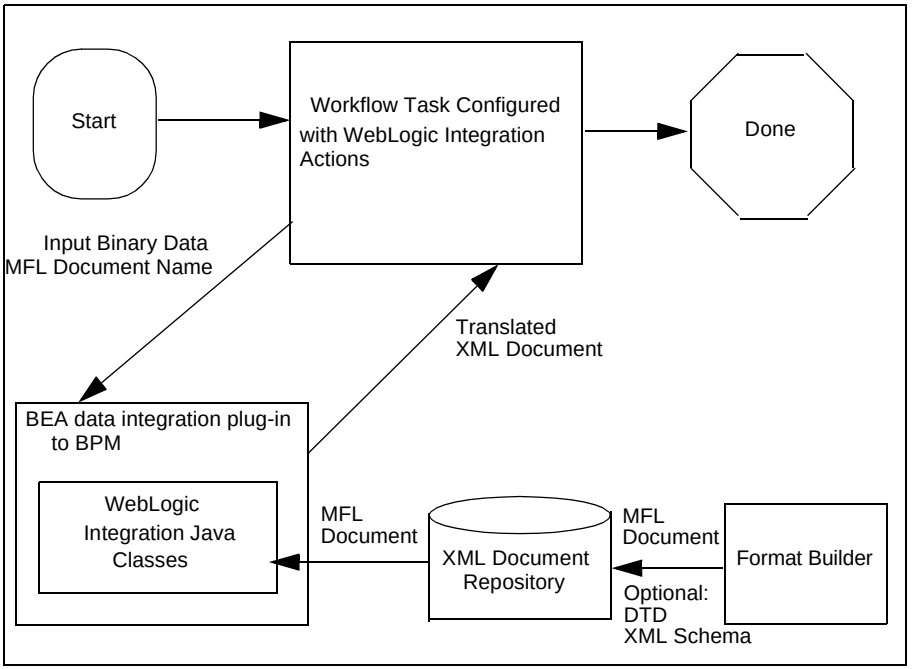
Run-Time Plug-In to Business Process Management

data integration plug-in for business process management (BPM) provides for an exchange of information between applications by supporting data translations between binary formats from legacy systems and XML. The data integration plug-in provides BPM actions that allow you to access XML to binary and binary to XML translations.

In addition to this data translation capability, the data integration plug-in provides event data processing in binary format, in-memory caching of MFL documents and translation object pooling to boost performance, a `BinaryData` variable type to edit and display binary data, exporting of entirely self-contained workflow definition packages, and execution within a WebLogic Server clustered environment.

The following illustration describes the relationship between the data integration component of WebLogic Integration and BPM.

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



A Supported Data Types

This section lists the following data types supported by WebLogic Integration.

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

MFL Data Types

Table A-1 lists the MFL data types that data integration supports. These types are specified in the `type` attribute of a FieldFormat element.

Table A-1 Supported MFL Data Types

Data Type	Description
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.
Binary (Hex 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-16.
DateTime: MM/DD/YY hh:mm	A string defining a date and time, i.e. 01/22/00 12:24.
DateTime: MM/DD/YY hh:mi AM	A string defining a date and time, i.e. 01/22/00 12:24 AM.

Table A-1 Supported MFL Data Types

Data Type	Description
DateTime: MM/DD/YY hh:mm:ss	A string defining a date and time, i.e. 01/22/00 12:24:00.
DateTime: MM/DD/YY hh:mm:ss AM	A string defining a date and time, i.e. 01/22/00 12:24:00 AM.
DateTime: DD/MM/YY hh:mm	A string defining a date and time, i.e. 22/01/00 12:24.
DateTime: DD/MM/YY hh:mm AM	A string defining a date and time, i.e. 22/01/00 12:24 AM.
DateTime: DD/MM/YY hh:mm:ss	A string defining a date and time, i.e. 22/01/00 12:24:00.
DateTime: DD/MM/YY hh:mm:ss AM	A string defining a date and time, i.e. 22/01/00 12:24:00 AM.
DateTime: MMDDYYhhmm	A string of numeric digits defining a date and time, i.e. 0122001224.
DateTime: YYYYMMDDhhmmss	A fourteen 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, i.e. 012200122400.
Date: DDMMMYY	A string defining a date, i.e. 22JAN00.
Date: DDMMMYYYY	A string defining a date, i.e. 22JAN2000.
Date: DD/MM/YY	A string defining a date, i.e. 22/01/00.
Date: DD/MM/YYYY	A string defining a date, i.e. 22/01/2000.
Date: DD-MMM-YY	A string defining a date, i.e. 22-JAN-00.
Date: DD-MMM-YYYY	A string defining a date, i.e. 22-JAN-2000.
Date: MMDDYY	A six digit numeric string defining a date, i.e. 012200.
Date: MMDDYYYY	An eight digit numeric string defining a date, i.e. 01222000.
Date: MM/DD/YY	A string defining a date, i.e. 01/22/00.

Table A-1 Supported MFL Data Types

Data Type	Description
Date: MM/DD/YYYY	A string defining a date, i.e. 01/22/2000.
Date: MMM-YY	A string defining a date, i.e. JAN-00.
Date: MMM-YYYY	A string defining a date, i.e. JAN-2000.
Date: MMMYY	A string defining a date, i.e. JAN00.
Date: MMMYYYY	A string defining a date, i.e. JAN2000.
Date: MMMDDYYYY	A string defining a date, i.e. JAN222000.
Date: YYYYMMDD	An eight byte numeric string of the format YYYYMMDD. A base data of String or EBCDIC may be specified to indicate the character encoding.
Date: Wed Nov 15 10:55:37 CST 2000	The default date format of the Java platform, i.e. '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 a delimiter field.
Filler	A sequence of bytes that is not translated to XML. This field of data is skipped over when translating binary data to XML. When translating XML to binary data, this field is written to the binary output stream as a sequence of spaces.
FloatingPoint: 4 bytes, Big-Endian	A four byte big endian floating point number that conforms to the IEEE Standard 754.
FloatingPoint, 4 bytes, Little-Endian	A four byte little endian floating point number that conforms to the IEEE Standard 754.
FloatingPoint: 8 bytes, Big-Endian	A eight byte big endian floating point number that conforms to the IEEE Standard 754.
FloatingPoint: 8 bytes, Little-Endian	A eight byte little endian floating point number that conforms to the IEEE Standard 754.
Floating Point IBM 4 byte IBM 8 byte	IBM Mainframe floating point.

Table A-1 Supported MFL Data Types

Data Type	Description
Integer: Signed, 1 byte	A one byte signed integer, i.e. '56' is 0x38.
Integer: Unsigned, 1 byte	A one byte unsigned integer, i.e. '128' is 0x80.
Integer: Signed, 2 byte, Big-Endian	A signed two-byte integer in big endian format, i.e. '4660' is 0x1234.
Integer: Signed, 4 byte, Big-Endian	A signed four-byte integer in big endian format, i.e. '4660' is 0x00001234.
Integer: Signed, 8 bytes, Big-Endian	A signed eight-byte integer in big endian format, i.e. '4660' is 0x0000000000001234.
Integer: Unsigned, 2 byte, Big-Endian	An unsigned two-byte integer in big endian format, i.e. '65000' is 0xFDE8.
Integer: Unsigned, 4 byte, Big-Endian	An unsigned four-byte integer in big endian format, i.e. '65000' is 0x0000FDE8.
Integer: Unsigned, 8 bytes, Big-Endian	A unsigned eight-byte integer in big endian format, i.e. '65000' is 0x000000000000FDE8.
Integer: Signed, 2 bytes, Little-Endian	A signed two-byte integer in little endian format, i.e. '4660' is 0x3412.
Integer: Signed, 4 bytes, Little-Endian	A signed four-byte integer in little endian format, i.e. '4660' is 0x34120000.
Integer: Signed, 8 bytes, Little-Endian	A signed eight-byte integer in little endian format, i.e. '4660' is 0x3412000000000000.
Integer: Unsigned, 2 bytes, Little-Endian	An unsigned two-byte integer in little endian format, i.e.'65000' is 0xE8FD.
Integer: Unsigned, 4 bytes, Little-Endian	An unsigned four-byte integer in little endian format, i.e. '65000' is 0xE8FD0000.
Integer: Unsigned, 8 bytes, Little-Endian	A unsigned eight-byte integer in little endian format, i.e. '65000' is 0xE8FD000000000000.

Table A-1 Supported MFL Data Types

Data Type	Description
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 is not translated to the XML data. When XML data is translated to a binary format, and a literal is defined as part of the binary format, WebLogic Integration writes the literal in the resulting binary byte stream.
Numeric	A string of characters containing only digits, i.e. '0' through '9'. Requires a length, length field, delimiter, or a delimiter field.
Packed Decimal: Signed	IBM signed packed format. Requires a length, length field, delimiter, or a 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 a delimiter field to be specified. The length or length field should specify the size of this field in bytes.
String	A string of characters. Requires a length, a length field, a delimiter, or a delimiter field. If no length, length field, or delimiter is defined for a data type String, a delimiter of "\x00" (a NUL character) will be assumed.
String: NUL terminated	A string of characters, optionally NUL (\x00) terminated, residing within a fixed length field. This field type requires a length attribute or length field which determines the amount of data read for the field. This data is then examined for a NUL delimiter. If a delimiter is found, data following the delimiter is discarded. If a NUL delimiter does not exist, the fixed length data is used as the value of the field.
Time: hhmmss	A string defining a time, i.e. 122400.
Time: hh:mm AM	A string defining a time, i.e. 12:24 AM.
Time: hh:mm	A string defining a time, i.e. 12:24.
Time: hh:mm:ss AM	A string defining a time, i.e. 12:24:00 AM.
Time: hh:mm:ss	A string defining a time, i.e. 12:24:00.

Table A-1 Supported MFL Data Types

Data Type	Description
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 a 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 a 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 a 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 where the sign indicator is in the last byte. The last byte only contains the sign indicator and is separated from the numeric value. Requires a length, length field, delimiter, or a delimiter field to be specified. The length or length field should specify the size of this field in bytes.
Zoned Decimal: Unsigned	IBM unsigned zoned decimal format. Requires a length, length field, delimiter, or a 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
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

Table A-2 COBOL Data Types

COBOL Type	Support
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. The following formats:

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

will be converted to an unsigned 4 byte integer type, while the following will generate errors:

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

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

The following values are defined as follows:

- Supported - the data type will be correctly parsed by the importer and converted to a message format field or group.

- Unsupported - this data type is not supported and the importer reports an error when the copybook is imported.
- Ignored - the 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, however, any copybook statement that conforms to ANSI standard COBOL will be parsed correctly by the Importer. The Importer's default data model, which is based on the IBM mainframe model, can be changed in Format Builder to compensate for character set and data "endianness".

When importing copybooks, the importer may identify fields generically that, upon visual inspection, could easily be identified by a more specific data type. For this reason, the copybook importer creates comments for each field found in the copybook. This information is useful in assisting you in editing the MFL data to better represent the original Copybook. For example:

original copybook entry:

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

results in:

A field of type `EBCDIC` with a length of 8

Closer inspection indicates that this is intended to be a date format and could be defined as

A field of type Date: MM/DD/YY

or

A field of type Data: 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 came from the pre-processor output of a `hello.c` file that contained an `#include <windows.h>`.

- 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;
```

- In-Line 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

The data integration component of WebLogic Integration uses a metadata language called Message Format Language (MFL), based on XML, to describe binary data structure. Format Builder creates and maintains this metadata as a data file, or an MFL document in the repository. MFL uses the following metadata information 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 on how to use User Defined Type are contained in the following topics:

- User Defined Types Sample Files
- 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

User Defined Types Sample Files

The following table provides a listing and description of the sample files installed for use with User Defined Types. All directory names are relative to the WebLogic Integration installation directory.

Table B-1 User Defined Types Sample Files

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 runtime engine.
`samples\di\userdef`	`readme.txt`	Explains how to compile and run the sample.

Table B-1 User Defined Types Sample Files

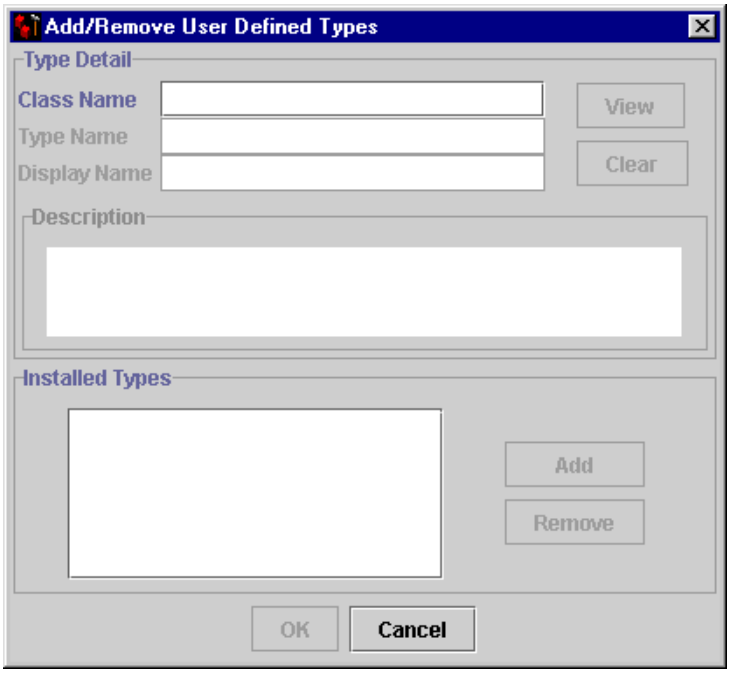
`samples\di\userdef`	`sample.data`	Data for the `ParseUserData` sample.
`samples\di\userdef`	`sample.mfl`	The Message Format Language (MFL) file for the `ParseUserData` sample.

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 E-Business Platform→WebLogic Integration 2.1→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

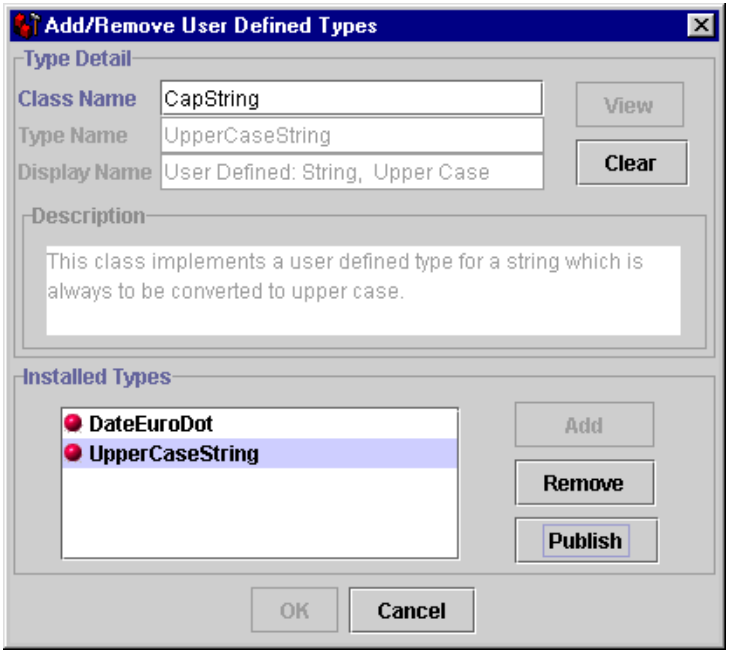


3. Enter the class name implementing the type in the Class Name field.

   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 Format Builder creates a `<wli home>/ext` directory specifically for containing User Defined Types. The `<wli home>` directory is where WebLogic Integration is installed.

4. Click View. Information about the requested class populates the following 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 the requested class cannot be loaded or does not conform to the requirements for a User Defined Type, an error message displays. Click OK to return to the Add/Remove User Defined Types dialog box. Refer to the User Defined Type Coding Requirements for required and optional interface methods, and available utility methods used for User Defined Types.

5. Once a valid User Defined Type is displayed, click Add to make it available for use within Format Builder.

Now that you have defined the new data type, the new Display Name appears in Format Builder along with the built-in data types. All User Defined Types displayed in the Data Type drop-down list box are prefixed with *User Defined:*, as in DisplayName text field.

Note: Because Format Builder cannot know the exact type of data represented by a User Defined Type, all generated XML Schema documents describing the XML generated by the data translation run-time component of WebLogic Integration for a particular transformation will use type of `xsd::string` to represent User Defined Type fields. This also affects Format Tester. When generating data for an MFL document containing User Defined Types, String data is generated for the corresponding fields. You must adjust the generated data so it will parse 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 static method of the `com.bea.wlxt.WLXT` class.

Installation of a User Defined Type into the data translation engine is not persistent. At termination of the current JVM process, 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
: specifies a reference to a new User Defined Type object.

```
public static void removeDataType (String name)
```

name
: specifies the name of the data type to remove.

The following example illustrates using theses 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

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 runtime, the Data Integration Plug-In loads User Defined Type classes from the repository as required. In addition, the Data Integration Plug-In will export the MFL and class files required to support the active template allowing a template to be imported on another BPM instance intact.

At runtime, the Data Integration Plug-In will retrieve both MFL documents and required User Defined Type classes from the repository. Class documents may be placed in the repository using one of the following methods:

- Publish User Defined Types to repository from Format Builder
- Use the Repository Import utility

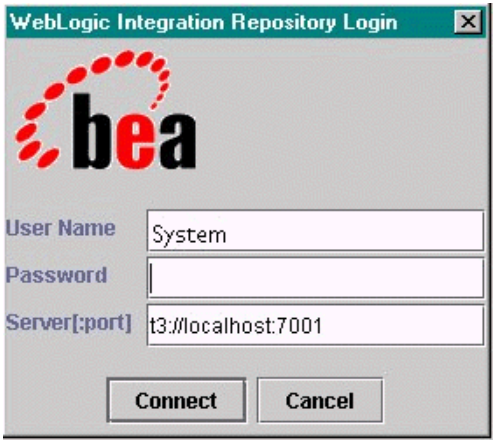
Publishing User Defined Types to the Repository from Format Builder

Perform the following steps to publish a User Defined Type to the repository.

1. Start Format Builder by choosing Start→Programs→BEA WebLogic E-Business Platform→WebLogic Integration 2.1→Format Builder. The Format Builder main window displays.

2. Choose Repository→Log In. The WebLogic Integration Repository Login window opens.

Figure B-3 WebLogic Integration Repository Login Window



3. Enter the userid specified for the connection in the User Name field.
4. Enter the password specified for the connection in the Password field.
5. Enter the server name and Port number in the Server[:port] field.

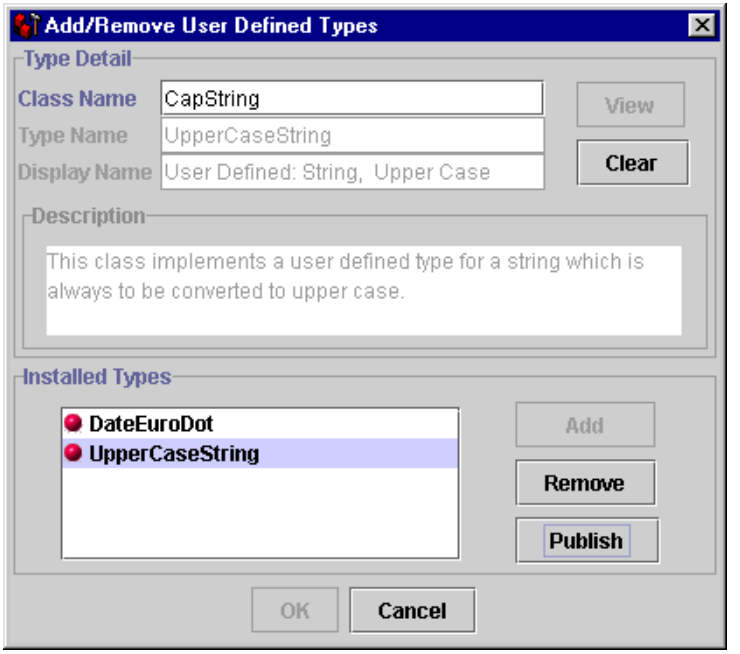
   Note: The WebLogic Integration Repository Login window allows up to three unsuccessful login attempts, after which, a login failure message is displayed. If you experience three login failures, choose Repository→Log In to repeat the login procedure.

6. Click Connect. If your login is successful, the Login window closes and the Format Builder Title bar displays the server name and port number entered on the WebLogic Integration Repository Login window. You may now choose any of the active repository menu items to access.
7. Choose Tools→User Defined Types. The Add/Remove User Defined Types dialog box opens.

   With a repository connection established, the Add/remove User Defined Types dialog box displays the status of each registered User Defined Type and allows for its publication to the repository. The User Defined Type repository status is

reflected by an icon of a ball preceding the type name of each installed User Defined Type.

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



The color of the icon associated with each User Defined Type indicates its status:

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

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

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. This is accomplished by passing the class file name on the Import command line. For example, to import all the class files in the current directory:

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

It must be noted that any Java class file can be imported to the repository. This is not the case for Format Builder User Defined Type class files. 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. Using the repository import utility allows these utility classes to be placed in the repository where they can be loaded 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 its derived classes. The `Bintype` class provides the basic framework the data integration component of WebLogic Integration uses to interface with a data type and also provides utility routines useful in 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 and their required and optional interface methods are used 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 successfully 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

The following interface routines are optional when using the data translation user defined data types utility.

`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. Returning *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: *flase*

`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 which may have 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 Types 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 class 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 Format Builder Types combo box.

Default: The class name of the class implementing the User Defined Type.

Utility Interface Routines

The following utility interface routines are available when using the data translation user defined data types utility.

`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) thorws 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`

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.

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 string parameter date according to format *fmt*. 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 string parameter date according to format *fmt*.

`protected static String writeDate(String date, SimpleDateFormat fmt)`
: Returns a date string suitable for output from the string parameter *date* using date format *fmt*.

`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 User Defined Type read and write methods. It encapsulates all the defined attributes for the field being read or written. `MFLField` supplies various methods that allows these attributes to be queried, and their respective values returned. Attributes that the User Defined Type does not support will never be present. For example, if the User Defined Type returned false to the `isValueOK()` method, it would never be passed to an MFLField object which returned 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 data type name of the field. This will be the name returned by the User Defined Type's `getTypeName()` method.

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

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

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

`public String getDelimRef()`
: Returns the field name of the field containing 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 this field has a default value.

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

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

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

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

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

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

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

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

`public boolean hasCutoff()`
Returns *true* if the field is defined as having a date cutoff value.

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

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

`public boolean hasCodepage()`
Returns *true* if a code page has been 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 present, occurs before the length field.

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

`public int getDecimalPLaces()`
Returns the number of decimal places defined for 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
- Understanding the Data Formats
- Performing Binary to XML Translation
- Performing XML to Binary Translation

What is Included in the Purchase Order Sample

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

Table 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`	Pre-built message format description of purchase order data.

Prerequisite Considerations

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

Understanding the Data Formats

To understand how the Format Builder is used, it helps to understand the data formats used by the data integration component of WebLogic Integration: binary data, XML, and MFL.

About Binary Data (Non-XML Data)

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

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

```
2231987
```

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

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

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

About XML Documents

Extensible Markup Language, or XML, is a text format for exchanging data between different systems. It allows data to be described in a simple, standard, text-only format. In contrast to binary data, XML data embeds a description of the data within the data stream. Applications can share data more easily, since they are not dependent on the layout of the data being embedded within each application. Because the data is presented in a standard form, applications on disparate systems can interpret data in proprietary binary formats.

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

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

About MFL Documents

Message Format Language (MFL) is an XML language that describes the layout of binary data. This language includes elements to describe each field of data, as well as groupings of fields (groups), repetition, and aggregation. The hierarchy of a binary record, the layout of fields, and the grouping of fields and groups is expressed in an MFL document. MFL documents are created using Format Builder. The Format

Builder application allows you to define the structure of binary data and save that information in an MFL document. These MFL documents are then used to perform run-time translation.

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

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

"ship to" address in your data, you only need to define the address format once. You can create the "bill to" address definition and create a reference for the "ship to" address.

- Comments — Notes or additional information about the message format.

Performing Binary to XML Translation

The following sections provide information on building sample purchase order format definitions and testing the translation of binary data into XML format.

- Analyzing the Data to be Translated
- Testing the Translation

You can build format definitions for binary data that will be translated to or from XML. Format definitions are the metadata used to parse binary data.

Analyzing the Data to be Translated

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

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

Listing 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 example data is taken from a fictitious purchase order on a proprietary system that the XYZ Corporation uses. They would like to interchange this information with another system that accepts XML data.

Listing C-1 Sample Binary Purchase Order Data

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

Perform the following steps to analyze the purchase order data:

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

Table C-2 Purchase Order Master Record

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

Table C-2 Purchase Order Master Record

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

2. Identify hierarchical groups.

   Groups are collections of fields, comments, and other groups or references that are related in some way. In Table 6-4, notice that the sample data defines two distinct groups: Payment Terms and Purchase Items. In addition to these groups, notice that there are several fields related to shipping and billing addresses. You can define a group for Shipping Address and a group for Billing Address.

3. Identify group attributes.

   You 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, look at the Address group within the Shipping Address and Billing Address groups. These two groups contain the same fields with the same attributes. Therefore, you can define the Address group within the Shipping Address group and set up the Address group within the Billing Address group as a reference.

4. Identify data fields.

   Fields are a sequence of bytes that have some meaning to an application. In Table 6-4, some of the sample data fields are Purchase Request Number, Supplier ID, Supplier Name, etc.

5. Identify data field attributes.

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

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

Figure C-1 Analysis of Purchase Order Data

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

Testing the Translation

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

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

Step 1. Starting Format Builder and Creating the Message Format

To start Format Builder and create the message format:

1. Choose Start→Programs→BEA WebLogic E-Business Platform→WebLogic Integration 2.1→Format Builder. The Format Builder main window displays.
2. Choose File→New. A new message definition opens.
3. Enter **PurchaseRequest** as the message format name and select the appropriate MFL version from the drop-down list box.
4. Click Apply. The navigation tree changes to reflect the new message format name.

Step 2. Creating Fields

To create fields:

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

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

   These values were determined by our analysis of the raw purchase order data, as you can see in Table C-1.

3. Select the type of character encoding you want for the field data by selecting it from the Codepage drop-down list box.
4. Select the field data character coding you want to use by selecting from the Code Page drop-down list box.

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

   Note: Since the only difference between the PR_Number field and the Supplier_ID field is the name, we will use the Format Builder Duplicate feature to create the Supplier_ID field.

6. Select the PR_Number field you just created in the navigation tree and right-click to select Duplicate from the shortcut menu. A new field description is displayed in the right pane.

7. Enter Supplier_ID as the name and click Apply. The Supplier_ID field is created and stored in the message format file.

8. Select the Supplier_ID field you just created in the navigation tree and choose Insert→Field→As Sibling.

9. Enter the Supplier_Name field details as follows:

Field	Value
Name	Supplier_Name
Optional	Check box
Type	String
Field Occurrence	Once
■ Field is Tagged	Checkbox
■ Tag Text Field	SUP:
Imbedded Length	
■ Type	Numeric
■ Length	2

10. Click Apply. The Supplier_Name field is created and added to the navigation tree.

    Note: The dashed box around the field icon in the navigation tree indicates this is an optional field.

11. Select the Supplier_Name field you just created in the navigation tree and choose Insert→Field→As Sibling.

12. Enter the Requested_Delivery_Date field details as follows:

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

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

13. Click Apply. The Requested_Delivery_Date field is created and added to the navigation tree.

Step 3. Creating Groups

To create groups:

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

Field	Value
Name	Shipping_Address
Group Occurrence	Once

   These values were determined by our analysis of the raw purchase order data presented in Table C-1.

3. Click Apply. The Shipping_Address group is created and added to the navigation tree.

   We know from our initial data analysis that the Shipping Address and Billing Address groups contain the same fields with the same attributes. Therefore, we

can define the Address group within the Shipping Address group and set up the Address group within the Billing Address group as a reference.

4. Select the Shipping_Address group in the navigation tree and choose Insert→Group→As Child.
5. Use the data in Table C-1 to create the Address group and click Apply to save the data.
6. Follow the steps outlined in Step 2. Creating Fields, to create the Street, City, State, and Zip fields as children of the Address group.

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

Step 4. Creating a Group Reference

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

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

Field	Value
Name	Billing_Address
Field Occurrence	Once

   These values were determined by our analysis of the raw purchase order data presented in Table C-1.

3. Select the Address group under Shipping_Address in the navigation tree and choose Edit→Copy. This copies the Address group details (including child objects) and places them on the clipboard.

4. Select the Billing_Address group you created in step 2 in the navigation tree and choose Edit→Paste→As Reference. This pastes the copy of the Address group into the message definition as a sibling of the Billing Address group.

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

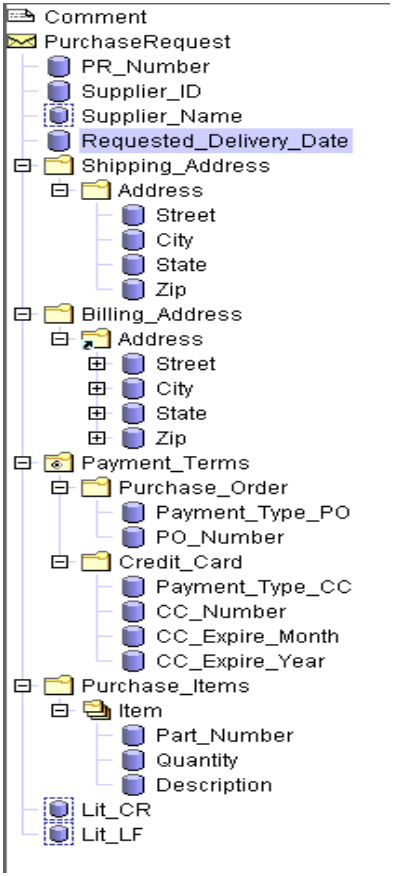
5. Change the Address reference group to be a child of the Billing_Address group by selecting the Address reference group in the navigation tree and choose Edit→Demote. The Address reference group moves *under* the Billing_Address group.

Step 5. Creating the Remaining Items

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

When you finish entering the items, your navigation tree should looks similar to Figure C-2.

Figure C-2 Completed Navigation Tree for Purchase Order Sample



Step 6. Saving the Message Format

To save a message format file:

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

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

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

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

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

Now, whenever you save message format documents, Format Builder will generate a DTD and/or a Schema for your message format.

Step 7. Testing the Message Format

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

1. Choose Tools→Test. The Tester opens. This allows you to test the translation of the binary purchase order data into XML.
2. Click Load and navigate to the `samples\di\po` directory.
3. Choose the file `PO_01.DATA` and click Open. The left pane displays the binary data.
4. Click Translate→Binary To XML >. The binary data is translated and the right pane displays the purchase order data in XML format.

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

5. If the translated data appears correct, choose File→ Save XML.
6. Navigate to the `samples\di\po` directory and enter `PO.XML` as a name for the XML data.

Performing XML to Binary Translation

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

1. Choose File→Open in Format Builder.
2. Navigate to the directory containing the desired file and select the file name.
3. Click open. The file is loaded into Format Builder.
4. Choose Tools→Test.
5. Under the XML panel, click Load, and navigate to the `samples\di\po` directory.
6. Choose the file `po.xml` and click Open. The right pane displays the XML data.
7. Click Translate→XML to Binary. The XML data is translated, and the right pane displays the purchase order data in Binary format.

   Note: You can see the messages output during the translation by selecting Display→ Debug.
8. If the translated data appears correct, choose File→ Save Binary.

Index

B

batch import utility 5-5
binary data, about 2-1

C

C struct importer
  hardware profiles 4-9
  starting 4-6
C structures 4-3
choice of children 2-18
COBOL copybook
  importer data types A-7
  importing 4-1
COBOL data types A-7
code page 2-24
com.bea.wlxt.bintype.Bintype class B-10
  optional interface routines B-11
  required interface routines B-11
  utility interface routines B-13
com.bea.wlxt.bintype.BintypeDate
  optional interface routines B-16
  required interface routines B-16
  utility interface routines B-16
com.bea.wlxt.bintype.BintypeDate class B-15
com.bea.wlxt.bintype.BinTypeString
  optional interface routines B-15
  required interface routines B-14
  utility interface routines B-15
com.bea.wlxt.bintype.BinTypeString class B-14
com.bea.wlxt.mfl.MFLField class B-17
comments 2-6, 2-26
customer support contact information viii

D

data base type 2-24
data fields 2-7
Data Gen 4-8
data offsets 3-8
data types
  COBOL A-7
  MFL A-1
  support A-1
debug writer 6-4
Delimiter 2-25
delimiter
  field 2-25
  group 2-19
delimiter shared 2-20
document type definition 6-3
documentation, where to find it vii

E

edit menu 2-40
  copy 2-41
  cut 2-41
  delete 2-41
  demote 2-42
  duplicate 2-41

move down 2-41
move up 2-41
paste 2-41
promote 2-41
redo 2-41
undo 2-41

F

field 2-5
creating 2-22
data type 2-23
delimiter 2-23, 2-24
name 2-23, C-12
occurrence 2-23
optional 2-23, C-12
parameters 2-5
field data options
code page 2-24
data base type 2-24
value 2-24
year cutoff 2-24
field delimiter 2-23, 2-24
delimiter 2-25
delimiter field 2-25
length 2-24
field occurence
once 2-23
repeat delimiter 2-23
repeat field 2-23
repeat number 2-23
unlimited 2-23
file menu 2-40
close 2-40
exit 2-40
new 2-40
open 2-40
save 2-40
save as 2-40
FML Field Table Class
importing 4-13
sample files 4-14
Format Builder
setting options 2-38
starting 2-8
using 2-7, 2-8
Format Tester
debug log 3-13
debug window 3-8
starting 3-1

G

group
attributes 2-7
creating 2-17
delimiter 2-19
description 2-18
occurence 2-19
group delimiter
delimited 2-19
delimiter is shared 2-20
group occurence
once 2-19
repeat delimiter 2-19
repeat field 2-19
repeat number 2-19
unlimited 2-19

H

hardware profiles 4-9
building 4-9
help menu 2-44
about 2-44
help topics 2-44
how do I 2-44
hierarchical groups 2-7

I

importing C structures 4-3

insert menu 2-42
    comment 2-42
    field 2-42
    group 2-42

L

length of field delimiter 2-24

M

menu bar 2-11
Message Format 2-5
    adding palette items 2-33
    default version 2-39
    opening 2-35, 2-36
    saving 2-33
message node 2-9
MFL data types A-1
MFL documents, about 2-5
MFL Gen 4-8

N

name
    field 2-23, C-12
    group 2-18

O

occurrence
    field 2-23
    group 2-19
offsets, positioning 3-12
once
    field occurence 2-23
    group occurence 2-19
optional
    field 2-23, C-12
    group 2-18

P

palette 2-29
    adding items 2-32
    deleting items 2-32
printing product documentation viii
provided 2-12

R

references
    creating 2-27
    description 2-6
related information viii
repeat delimiter
    field occurence 2-23
    group occurence 2-19
repeat field
    field occurence 2-23
    group occurence 2-19
repeat number
    field occurence 2-23
    group occurence 2-19
repository
    accessing 5-2
    importing documents 5-5
repository document chooser 5-6
repository documents
    retrieving 5-3
    storing 5-4
root node 2-9

S

shortcut menus 2-14
support, technical ix

T

toolbar 2-11
tools menu 2-43
    import 2-43

options 2-43
Tree Pane 2-8
using 2-9

U

unlimited
field occurence 2-23
group occurence 2-19
user defined types B-1
coding requirements B-10
registering B-3
sample files B-2

V

valid names 2-16
value, field data option 2-24
view menu 2-42
collapse all 2-42
expand all 2-42
show pallet 2-42

X

XML content model options 2-39
XML document type definition 5-1
XML documents, about 2-2
XML formatting options 2-39
XML Schema 5-1
XSLT Stylesheet 5-1