



# BEA WebLogic XML/Non-XML Translator User Guide

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

## Copyright

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

## Restricted Rights Legend

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

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

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

## Trademarks or Service Marks

BEA, WebLogic, Tuxedo, and Jolt are registered trademarks of BEA Systems, Inc. How Business Becomes E-Business, BEA WebLogic E-Business Platform, BEA Builder, BEA Manager, BEA eLink, BEA WebLogic Commerce Server, BEA WebLogic Personalization Server, BEA WebLogic Process Integrator, BEA WebLogic Collaborate, BEA WebLogic Enterprise, and BEA WebLogic Server are trademarks of BEA Systems, Inc.

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

BEA WebLogic XML/Non-XML Translator User Guide

---

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

---

---

# Contents

## About This Document

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

## 1. BEA WebLogic XML/Non-XML Translator Overview

Understanding XML Translation .....	1-2
What is XML Translator? .....	1-3
The Design-Time Component .....	1-4
The Run-Time Component .....	1-5
Binary to XML Translation .....	1-5
XML to Binary Translation .....	1-6
Post Translation Options and Considerations .....	1-7
Performing XML Transformation .....	1-7
Working with BEA WebLogic Process Integrator .....	1-8
Getting Started with the BEA WebLogic XML/Non-XML Translator .....	1-9

## 2. Building Format Definitions

Understanding the Data Formats Used with XML Translator .....	2-2
About Binary Data (Non-XML Data) .....	2-2
About XML Documents .....	2-3
About MFL Documents .....	2-6
Analyzing the Data to be Translated .....	2-7
Using the Format Builder .....	2-8

---

Starting Format Builder.....	2-8
Using the Format Builder Main Window.....	2-8
Using the Tree Pane.....	2-10
Using the Menu Bar.....	2-12
Using the Toolbar.....	2-12
Using the Shortcut Menus.....	2-15
Using Drag and Drop.....	2-16
Creating a Message Format.....	2-17
Valid Names.....	2-18
Creating a Group.....	2-18
Creating a Field.....	2-21
Creating a Comment.....	2-25
Creating References.....	2-26
Working with Pallets.....	2-29
Adding Items to the Pallet.....	2-29
Deleting Items From the Pallet.....	2-30
Adding Pallet Items to a Message Format.....	2-30
Saving a Message Format.....	2-30
Opening an Existing Message Format.....	2-31
Importing a COBOL Copybook.....	2-32
Setting Format Builder Options.....	2-34
Format Builder Menus.....	2-35
File Menu.....	2-35
Edit Menu.....	2-36
Insert Menu.....	2-38
View Menu.....	2-38
Tools Menu.....	2-39
Test Menu.....	2-39
Help Menu.....	2-39

### 3. Testing Format Definitions

Running the Tester.....	3-1
Debugging Formats.....	3-4

---

## 4. Using the Run-Time Component

Binary to XML .....	4-1
Generating XML with a Reference to a DTD .....	4-2
Passing in a Debug Writer.....	4-4
XML to Binary .....	4-5
Converting a Document object to Binary.....	4-6
Passing in a debug writer .....	4-8
Debug Output:.....	4-8
XML to XML Transformation .....	4-8
Initialization methods.....	4-10
init() method.....	4-10
Java API Documentation.....	4-12

## A. Supported Data Types

MFL Data Types.....	A-1
COBOL Copybook Importer Data Types.....	A-7

## Glossary



---

# About This Document

This document describes BEA WebLogic XML/Non-XML Translator, hereafter referred to as XML Translator, and provides instructions for using it to translate data from binary format to XML and from XML to binary format.

This document covers the following topics:

- BEA WebLogic XML/Non-XML Translator Overview
- Building Format Definitions
- Testing Format Definitions
- Using the Run-Time Component
- Supported Data Types
- Glossary

## 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 BEA WebLogic XML/Non-XML Translator 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 BEA WebLogic XML/Non-XML Translator 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:

- *BEA WebLogic XML/Non-XML Translator Installation Guide*
- *BEA WebLogic XML/Non-XML Translator Getting Started Guide*
- BEA Format Builder online help system



---

# Contact Us!

Your feedback on the BEA WebLogic XML/Non-XML Translator 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 BEA WebLogic XML/Non-XML Translator documentation.

In your e-mail message, please indicate that you are using the documentation for the BEA WebLogic XML/Non-XML Translator 1.0 release.

If you have any questions about this version of BEA WebLogic XML/Non-XML Translator, or if you have problems installing and running BEA WebLogic XML/Non-XML Translator, contact BEA Customer Support through BEA WebSupport at **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
<b>boldface text</b>	Indicates terms defined in the glossary.

---

---

<b>Convention</b>	<b>Item</b>
Ctrl+Tab	Indicates that you must press two or more keys simultaneously.
<i>italics</i>	Indicates emphasis or book titles.
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. <i>Examples:</i> <pre>#include &lt;iostream.h&gt; void main ( ) the pointer psz chmod u+w * \tux\data\ap .doc tux.doc BITMAP float</pre>
<b>monospace boldface text</b>	Identifies significant words in code. <i>Example:</i> <pre>void <b>commit</b> ( )</pre>
<i>monospace italic text</i>	Identifies variables in code. <i>Example:</i> <pre>String <i>expr</i></pre>
UPPERCASE TEXT	Indicates device names, environment variables, and logical operators. <i>Examples:</i> <pre>LPT1 SIGNON OR</pre>
{ }	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. <i>Example:</i> <pre>buildobjclient [-v] [-o name ] [-f <i>file-list</i>]... [-l <i>file-list</i>]...</pre>

---

---

Convention	Item
	Separates mutually exclusive choices in a syntax line. The symbol itself should never be typed.
...	Indicates one of the following in a command line: <ul style="list-style-type: none"> <li>■ That an argument can be repeated several times in a command line</li> <li>■ That the statement omits additional optional arguments</li> <li>■ That you can enter additional parameters, values, or other information</li> </ul> The ellipsis itself should never be typed. <i>Example:</i> <pre>buildobjclient [-v] [-o name ] [-f file-list]... [-l file-list]...</pre>
. . .	Indicates the omission of items from a code example or from a syntax line. The vertical ellipsis itself should never be typed.

---



# 1 BEA WebLogic XML/Non-XML Translator 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. BEA WebLogic XML/Non-XML Translator (hereafter referred to as XML Translator) 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 XML Translator?
  - The Design-Time Component
  - The Run-Time Component
- Post Translation Options and Considerations
  - Performing XML Transformation
  - Working with BEA WebLogic Process Integrator
- Getting Started with the BEA WebLogic XML/Non-XML Translator

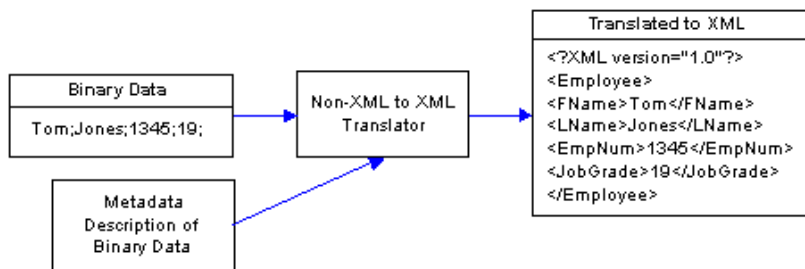
# Understanding XML Translation

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 XML Translator 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 XML Translator?

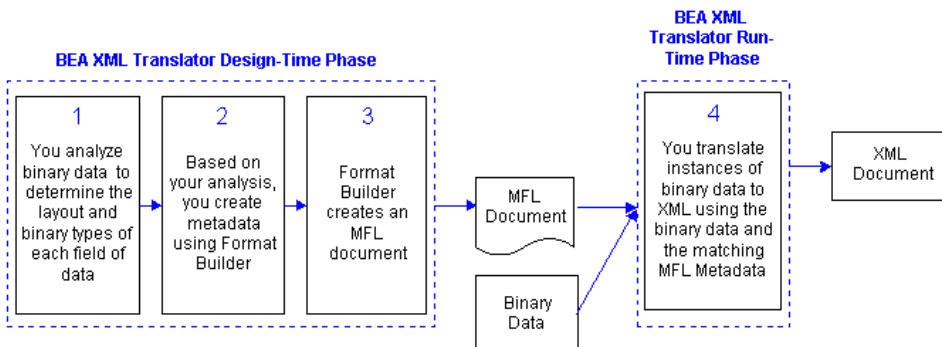
XML Translator facilitates the integration of data from diverse enterprise applications by supporting data translations between binary formats from legacy systems and XML. XML Translator normalizes legacy data into XML so it may be directly consumed by XML applications, transformed into a specific XML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

- The Design-Time Component
- The Run-Time Component.

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.

You can then use XML Translator's run-time component to translate instances of binary data to XML. Figure 1-2 shows the flow of events for non-XML to XML data translation.

**Figure 1-2 Flow of Events for Non-XML to XML Translation Using XML Translator**

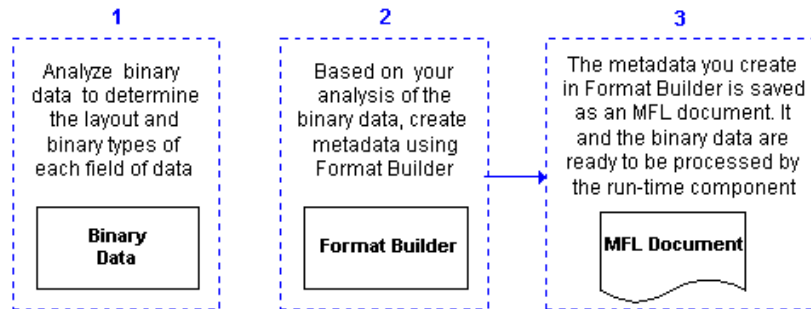


## The Design-Time Component

The design-time component is a Java application called Format Builder. Format Builder is used to create descriptions of binary data records. Format Builder allows you to describe the layout and hierarchy of the binary data so that it can be translated to or from XML. With Format Builder, 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. Format Builder allows you to further define groupings of fields (groups), repetition of fields and groups, and aggregation.

The description you create in Format Builder is saved in an XML grammar called Message Format Language (MFL). MFL documents are metadata used by the run-time component of XML Translator to translate an instance of a binary data record to an instance of an XML document (or vice-versa). Figure 1-3 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**



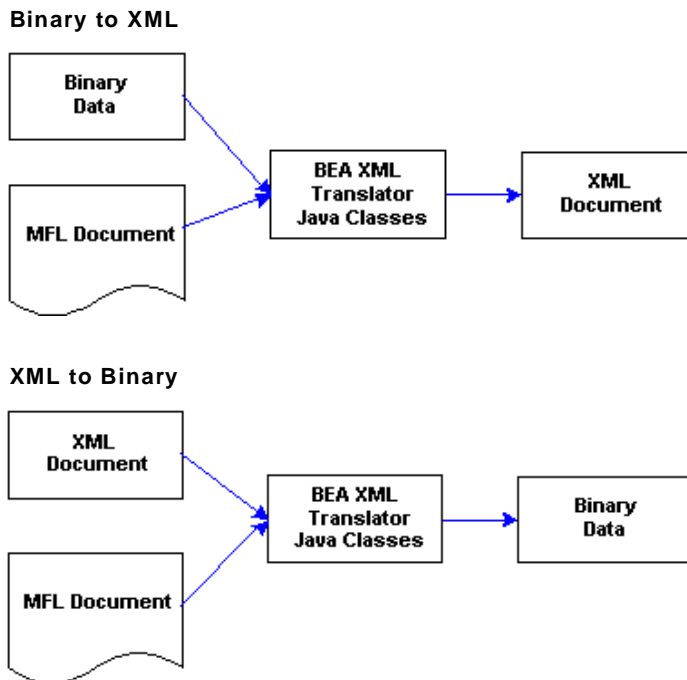
You can also use Format Builder to retrieve, validate, and edit stored MFL documents and to test message format definitions with your own data. The test feature allows you to select the option of testing the translation of XML data to binary format, or binary data to XML format. You may save the transformed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



## The Run-Time Component

The run-time component of XML Translator 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 BEA WebLogic Process Integrator, 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 a 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);  
}
```

---

## 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 WebLogic Process Integrator. Once your binary data has been put in a self-describing format such as XML, this data is available for use in other applications.

### Performing XML Transformation

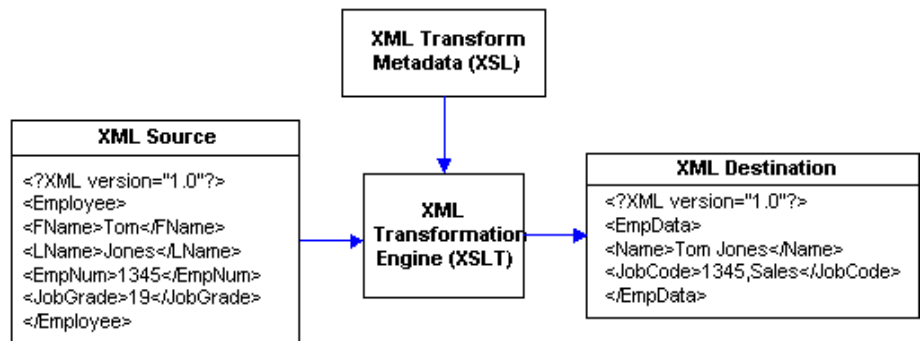
Once you have translated binary data into XML, you may need to transform the XML data to a different XML grammar, to a display format (HTML), or to another binary format. The process of transforming XML to another XML grammar is referred to in this document as *XML transformation*. XML transformation can be accomplished via third-party tools such as Apache's Xalan XSLT engine. 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 XML Translator

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 run-time component of XML Translator to transform XML.

Figure 1-5 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 1-5 XML Data Transformation of: Tom;Jones,1345;19**



## Working with BEA WebLogic Process Integrator

BEA WebLogic Process Integrator is a powerful workflow engine that automates workflow, business-to-business processes, and enterprise application assembly. WebLogic Process Integrator runs on BEA WebLogic Server and is a robust, J2EE standards-based workflow and process integration solution.

Using an intuitive flowchart paradigm, business analysts use the WebLogic Process Integrator Studio to define business processes that span applications or to automate human interaction with applications. Developers can use WebLogic Process Integrator to assemble application components quickly without programming. The assembled applications are executed and managed by the WebLogic Process Integrator engine.

A sample application is included with XML Translator that demonstrates how you can integrate it with WebLogic Process Integrator. The sample application is a workflow that simulates integration with legacy HR and payroll systems that do not accept XML. In the sample application, an XML document from a JMS topic or a manual form entry is used to initiate the creation of paychecks for employees. The document contains the employee number, hours worked, and pay period ending date. The employee name and

pay rate must be extracted from the HR system. The pay is calculated and then sent to the payroll system. Neither the HR nor payroll systems accept XML data. Included with the sample are the following:

- EJB wrapper for XML Translator Java classes
- EJBs that simulate HR and payroll legacy applications
- Pre-built WebLogic Process Integrator workflow

For detailed information about using the sample application, see *BEA WebLogic XML/Non-XML Translator Samples Guide*.

## Getting Started with the BEA WebLogic XML/Non-XML Translator

The steps outlined in Table 1-1 provide you with a high-level guideline to all of the tasks and processes that you must perform to install, configure, and work with the XML Translator. Think of these steps as a road map to guide you through the process and to point you to the resources available to help you.

**Table 1-1 Steps for Working with the XML Translator**

<b>Task</b>	<b>Resource</b>
1. Read the BEA WebLogic XML/Non-XML Translator Release Notes.	<i>BEA WebLogic XML/Non-XML Translator Release Notes.</i>
2. Make sure that all of the platform/environment prerequisites listed in the Installation and Configuration Guide and in the Release Notes have been met.	<i>BEA WebLogic XML/Non-XML Translator Release Notes and BEA WebLogic XML/Non-XML Translator Installation and Configuration Guide.</i>
3. Install the BEA WebLogic XML/Non-XML Translator.	<i>BEA WebLogic XML/Non-XML Translator Installation and Configuration Guide.</i>

# 1 *BEA WebLogic XML/Non-XML Translator Overview*

---

<b>Task</b>	<b>Resource</b>
4. Define the data format using Format Builder and generate translation metadata.	<i>BEA WebLogic XML/Non-XML Translator User Guide.</i>
5. Test the translation	<i>BEA WebLogic XML/Non-XML Translator User Guide.</i>

# 2 Building Format Definitions

The following sections provide information on building format definitions using the Format Builder included with BEA WebLogic XML/Non-XML Translator (hereafter referred to as XML Translator):

- Understanding the Data Formats Used with XML Translator
- Analyzing the Data to be Translated
- Using the Format Builder

The Format Builder included with XML Translator allows users to 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 the Data Formats Used with XML Translator

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

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 fields, and the grouping of fields and groups are expressed in an MFL document. This MFL document is used by XML Translator 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

Extended Markup Language, or XML, is a text format for exchanging data between different systems. It allows data to be described in a simple, standard, text-only format. In contrast to binary data, XML data embeds a description of the data within the data stream. Applications can share data more easily, since they are not dependent on the layout of the data being embedded within each application. Since the data is presented in a standard form, applications on disparate systems can interpret 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 tags 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's Sprockets</Supplier_Name>
  <Requested_Delivery_Date>2000-01-15T00:00:00</Requested_Delivery_Date>
  <Shipping_Address>
    <Address>
      <Street>123 Main St.</Street>
      <City>Austin</City>
      <State>TX</State>
      <Zip>75222</Zip>
    </Address>
  </Shipping_Address>
</PurchaseRequest>
```

## 2 Building Format Definitions

---

```
</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 2-4 shows an example of an XML Schema.

### 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 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.
- 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:
  - 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.
  - 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.

## 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 included on the XML Translator CD-ROM. The sample purchase order illustrates how XML Translator translates data from one format to another. For more information on this sample data, refer to the *BEA WebLogic XML/Non-XML Translator Samples Guide*.

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

The Format Builder (the design-time portion of XML Translator) is used to build the format definitions that are used for data translations. For details on the steps you need to perform to thoroughly analyze your data, refer to the *BEA WebLogic XML/Non-XML Translator Samples Guide*.

# Using the 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.

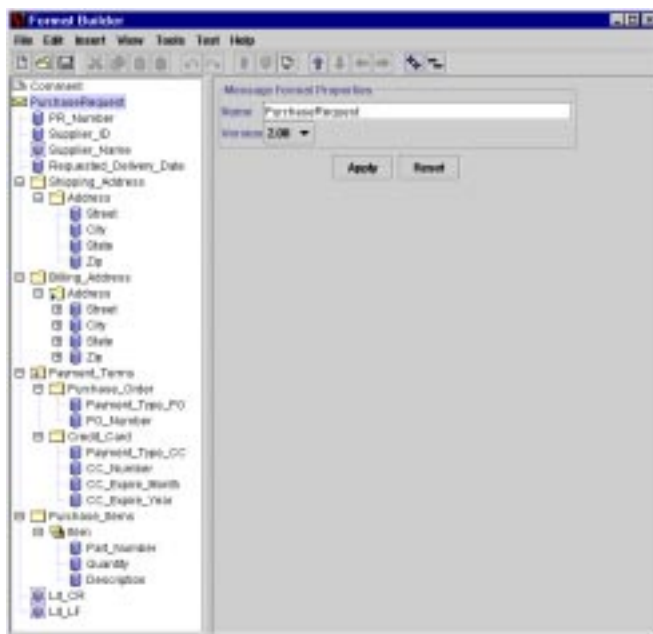
## Starting Format Builder

To start Format Builder, choose Start→Programs→BEA WebLogic E-Business Platform→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 Tree Pane) shows the structural information for the data format. The right pane (the Properties Pane) shows the detail for the item selected in the tree pane.

Figure 2-1 Format Builder Main Window



The structure of the binary data is defined in the tree pane 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 Tree Pane
- Using the Menu Bar
- Using the Toolbar
- Using Drag and Drop
- Using the Shortcut Menus

### Using the Tree Pane

The Tree Pane represents hierarchical/structural information about the format of the binary data in a tree. The root node of the 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 tree. Groups contain fields or other groups and are represented by non-leaf nodes in the 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 tree. This is done through the menus or the toolbar (see [Using the Menu Bar](#) and [Using the Toolbar](#)).

The icons that appear in the Tree Pane are described in the following table.

**Table 2-1 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 <code>PAYDATE</code> , <code>HOURS</code> , and <code>RATE</code> could be part of the <code>PAYINFO</code> group). Defines the formatting for all items contained in the group.
	Optional Group	A group that has 0 or more occurrences.
	Repeating Group	A group that has one or more occurrences.
	Optional Repeating Group	May or may not be included, but if included, more than one will occur.



Table 2-1 Tree Icon Descriptions

Tree Icon	Icon Name	Description
	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	Choose Choice of Children if 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 present.
	Repeating Field	A field that has one or more occurrences.
	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.
	Optional Repeating Field	A field that has 0 or more occurrences.
	Comment	Contains notes about the message format or the data translated by the message format.

**Table 2-1 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 to 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 tree pane and the state of the tree. Click a menu heading to open the menu and choose a command.

**Figure 2-2 Format Builder Menu Bar**

**File Edit Insert View Tools Test Help**

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

For a complete description of the menu commands, see Format Builder Menus.









### 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:

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 tree. The item can be pasted elsewhere in the tree. <b>Note:</b> 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 tree. <b>Note:</b> 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.

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 support multi-level undoing and redoing.
	Insert Field	Inserts a field as a sibling of the item selected in the tree pane.
	Insert Group	Inserts a group as a sibling of the item selected in the tree pane.
	Insert Comment	Inserts a comment as a sibling of the item selected in the tree pane.
	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 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 tree. For example, Group1 is the sibling of Field1. Field1 immediately follows Group1 in the tree. Selecting Field1 and clicking the Demote tool makes it a child of Group1.
	Expand	Expands all items in the tree pane to show child items.

Toolbar Button	Name	Description
	Collapse	Collapses the tree pane to show first level items only.

## 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 tree pane. 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 tree, or the state of the tree at the time.

**Table 2-2 Shortcut Menus**

Menu Command	Description
Cut	Removes the item currently selected in the left-hand pane, and it's child objects, from the tree.
Copy	Makes a copy of the item currently selected in the left-hand pane for insertion elsewhere in the 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.

Menu Command	Description
Duplicate	<p>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”.</p> <p>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”.</p>
Delete	Deletes the selected item.

### Using Drag and Drop

You can use the drag and drop feature of XML Translator to copy and/or move the items in the tree pane.

**Note:** The node being copied or moved is always inserted as a sibling of the selected node during the drag and drop process.

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 Pane displays in the detail window.

**Figure 2-4 Message Format Properties**

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

**Note:** The previous example and the examples that follow are based on MFL version 2.0 documents.

**Table 2-3 Message Format Properties**

Field	Description
<b>Message Format Properties</b>	
Name	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.
Version	The version of MFL you are using. Version 1.0 of MFL was introduced with BEA eLink Information Integrator. Format Builder adds new features, and its default version is 2.0.
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.

### Valid Names

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 XML Translator. 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 tree pane.
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.

**Note:** The following example applies to MFL version 2.0.

Field	Description
<b>Group Description</b>	
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.
<b>Group Occurrence</b>	

Field	Description
Occurrence	<p>Choose one of the following to indicate how often this group appears in the message format:</p> <ul style="list-style-type: none"><li>■ <b>Once</b> — Indicates the group appears only once.</li><li>■ <b>Repeat Delimiter</b> — Indicates the group will repeat until the specified delimiter is encountered.</li><li>■ <b>Repeat Field</b> — Indicates the group will repeat the number of times specified in the field denoted as the repeat field.</li><li>■ <b>Repeat Number</b> — Indicates the group will repeat the specified number of times.</li><li>■ <b>Unlimited</b> — Indicates the group will repeat an unlimited number of times.</li></ul> <p><b>Note:</b> Unless a group is defined as Optional, all groups occur at least once.</p>
<hr/> <b>Group Delimiter</b> <hr/>	
Delimited	<p>Select this option if the end of the group is marked with a delimiter. A delimiter identifies the end of a group of fields.</p> <p><b>Note:</b> Normally, groups are not delimited. They are usually parsed by content (the group ends when all child objects have been parsed).</p>
Delimiter is Shared	<p>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.</p>
Delimiter Tab	<p>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.</p> <p><b>Value</b> — Enter the delimiter that marks the end of the group of fields.</p>

Field	Description
Delimiter Field Tab	<p>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.</p> <ul style="list-style-type: none"> <li>■ <b>Field</b> — Select the field that contains the delimiter character string. A list of valid fields will be presented in a drop-down list.</li> <li>■ <b>Default</b> — Enter the default delimiter character that will be used if the above field is not present in the data. This value is required.</li> </ul>
<b>Group Update Buttons</b>	
Apply	Saves your changes to the message format document.
Duplicate	<p>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”.</p> <p>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”.</p>
Reset	Discards your changes to the detail window and resets all fields to the last saved values.

## 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 tree pane.
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**

**Field Description**

Name   Optional

Tagged  Tag

Type

**Field Occurrence**

Once

Repeat Delimiter

Repeat Field

Repeat Number

Unlimited

**Field Data Options**

**Termination**

Code Page

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

**Note:** The following example applies to MFL version 2.0.

Field	Description
<b>Field Description</b>	
Name	The name of the field. This name must comply with XML element naming conventions. Refer to <a href="#">Valid Names</a> for more information.
Optional	Select this option if this is an optional field. Optional means that the data for the field may or may not be present.
Tagged	Select this option if this is a tagged field. Being tagged means that a literal precedes 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 the tag. ACME INC is the field data.
Tag	If you selected the Field is Tagged option, enter the tag here.
Type	Select the data type of the field from the drop down list. The default is String.  <b>Note:</b> The Field Type you select dictates the Field Data Options that appear on the dialog.  Refer to Appendix A, "Supported Data Types," for a list of data types supported by XML Translator.
<b>Field Occurrence</b>	

Field	Description
Occurrence	<p>Choose one of the following to indicate how often this field appears in the message format:</p> <ul style="list-style-type: none"> <li>■ <b>Once</b> — Indicates the field appears only once.</li> <li>■ <b>Repeat Delimiter</b> — Indicates the field will repeat until the specified delimiter is encountered.</li> <li>■ <b>Repeat Field</b> — Indicates the field will repeat the number of times specified in the field denoted as the repeat field.</li> <li>■ <b>Repeat Number</b> — Indicates the field will repeat the specified number of times.</li> <li>■ <b>Unlimited</b> — Indicates the field will repeat an unlimited number of times.</li> </ul> <p><b>Note:</b> Unless a field is defined as optional, the field will occur at least one time.</p>
<p><b>Note:</b> The fields that display in the following sections of the dialog depend on the Field Type selected.</p>	
<p><b>Field Data Options</b></p>	
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.
Value	The value that appears in a literal field.
<p><b>Field Delimiter (Termination)</b></p>	
Length Tab	<p>Variable-size data types can have their length specified by a specific length value.</p> <p><b>Value</b> — Enter the length of the field data.</p>

Field	Description
Delimiter Tab	<p>Variable-sized data types can have their termination point specified by a delimiter. A delimiter is a character that marks the end of the field. The field data continues until the delimiter character is encountered.</p> <p><b>Value</b> — Enter the delimiter that marks the end of the field data.</p>
Length Field Tab	<p>Variable-sized data types can have their termination point specified by a length field. A length field precedes the data field and indicates how many bytes the data contains.</p> <ul style="list-style-type: none"> <li>■ <b>Type</b> — Select the type of the length field.</li> <li>■ <b>Length</b> — Select this option if the length field is a variable length; then, enter the number of bytes in the length field.</li> <li>■ <b>Delimiter</b> — Select this option if the end of the length field is marked by a delimiter; then, enter the delimiter character.</li> <li>■ <b>Length Occurs Before Tagged Field</b> — Select this option if the length field should appear before the tag in the data. The default order is tag before length.</li> </ul> <p><b>Note:</b> The Length and Delimiter selection will be disabled if the type of the length field determines its length (see <a href="#">Supported Data Types</a> for more information on field types).</p>
Delimiter Field Tab	<p>Variable-sized data types can have their termination point specified by a field that contains a delimiter character. A delimiter is a character that marks the end of the field. The field data continues until the field containing the delimiter character is encountered.</p> <ul style="list-style-type: none"> <li>■ <b>Field</b> — Select the field that contains the delimiter character.</li> <li>■ <b>Default</b> — Enter the delimiter character. You must supply a default value. The default is used when the delimiter field is not present.</li> </ul>
<b>Field Update Buttons</b>	
Apply	Saves your changes to the message format file.

Field	Description
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.

## 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 tree pane.
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.



**Figure 2-7 Comment Details**

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

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 all fields to the last saved values.

## 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 tree pane.
2. Choose Edit→Copy.
3. Choose the proper sibling in the tree.
4. Choose Edit→Paste→As Reference.

**Figure 2-8 Reference Details**

The screenshot shows a dialog box titled "Reference Details". It has two main sections. The first section, "Group Reference Description", contains a "Name" field with the text "Address" and an "Optional" checkbox. The second section, "Group Reference Occurrence", contains five radio button options: "Once" (selected), "Repeat Delimiter" (with an empty text field), "Repeat Field" (with a dropdown menu), "Repeat Number" (with an empty text field), and "Unlimited". At the bottom, there are three buttons: "Apply", "Edit Reference", and "Reset".

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

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

Field	Description
Occurrence	<p>Choose one of the following to indicate how often this reference field or group appears in the message format:</p> <ul style="list-style-type: none"> <li>■ <b>Once</b> — Indicates the reference appears only once.</li> <li>■ <b>Repeat Delimiter</b> — Indicates the reference will repeat until the specified delimiter is encountered.</li> <li>■ <b>Repeat Field</b> — Indicates the reference will repeat the number of times specified in the field denoted as the repeat field.</li> <li>■ <b>Repeat Number</b> — Indicates the reference will repeat the specified number of times.</li> <li>■ <b>Unlimited</b> — Indicates the reference will repeat an unlimited number of times.</li> </ul>
<b>Field Update Buttons</b>	
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.

## Working with Pallets

The pallet 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](#)) to copy items from the pallet into your message format definition.

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

**Note:** You can only copy items from the tree pane to the pallet and vice versa. You cannot move items between the windows.

The XML Translator pallet 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 pallet.

### Adding Items to the Pallet

To add items to the pallet:

1. Choose View→Show Pallet to display the pallet.  
**Note:** If the Pallet window is already displayed, skip this step.
2. From the tree pane of the XML Translator window, choose the item you want to add to the pallet.
3. Click and hold the left mouse button and drag the item into the pallet window.
4. When the item is placed in the position you want it (as sibling of the desired item), release the mouse button. The item is copied from the XML Translator window to the pallet window.

**Notes:** You cannot add any node that depends on the existence of another node to the pallet. 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 pallet.

### Deleting Items From the Pallet

To delete items from the pallet:

1. Select the item in the pallet 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 Pallet Items to a Message Format

To copy items from the pallet to a message format:

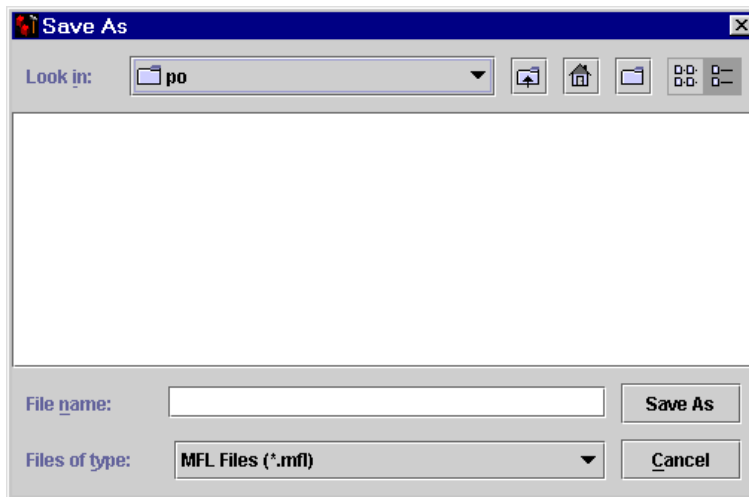
1. Choose View→Show Pallet to display the pallet.  
**Note:** If the Pallet window is already displayed, skip this step.
2. From the pallet window, choose the item you want to add to your message format.
3. Click and hold the left mouse button and drag the item into the tree pane of the Format Builder window.
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 pallet to the message format.

## Saving a Message Format

To save a message format file for the first time:

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

**Figure 2-9 Save As Dialog**



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.

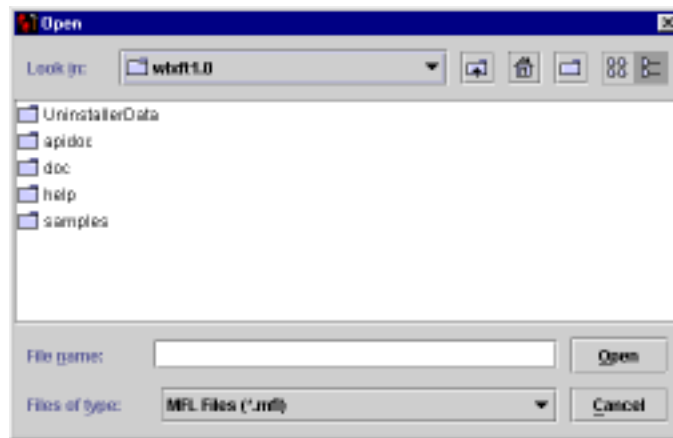
To save a message format file using a different name, choose File→Save As and follow steps 1 through 5 above.

## Opening an Existing Message Format

To open an existing message format file:

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

**Figure 2-10 Open Dialog**



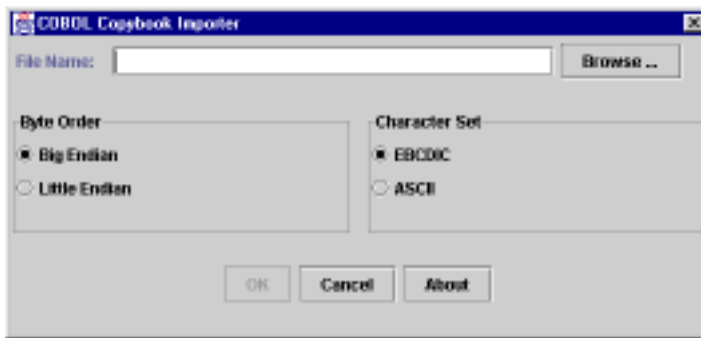
2. Navigate to the directory containing the desired file and select the file name.
3. Click Open. The file is loaded into Format Builder.

## Importing a COBOL Copybook

XML Translator 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 displays.



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

Field	Description
Filename	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.
<b>Byte Order</b>	
Big Endian	Select this option to set the byte order to Big Endian. <b>Note:</b> These values are attributes of the copybook data, not the copybook description file.
Little Endian	Select this option to set the byte order to Little Endian. <b>Note:</b> These values are attributes of the copybook data, not the copybook description file.

Field	Description
<b>Character Set</b>	
EBCDIC	Select this option to set the character set to EBCDIC. <b>Note:</b> These values are attributes of the copybook data, not the copybook description file.
ASCII	Select this option to set the character set to ASCII. <b>Note:</b> These values are attributes of the copybook data, not the copybook description file.
<b>Action Buttons</b>	
OK	Imports the COBOL Copybook using the settings you defined.
Cancel	Closes the dialog 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.

## 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 displays.



Figure 2-11 Format Builder Options Dialog



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

Field	Definition
Default Message Format Version	Select the MFL version used when creating new documents.  <b>Note:</b> Message formats contain their own format version specified on the Message Format pane.
<b>XML Formatting Options</b>	
Initial Indent	Enter the number of spaces to indent the first line of the Message Format document.
New Line Indent	Enter the number of spaces to indent a new child line of the Message Format document.
<b>XML Content Model Options</b>	
Auto-generate DTD	Generates a DTD document when you save the MFL document. This document will be placed in the same directory as the message format.
Auto-generate Schema	Generates an XML Schema file when you save the MFL document. This document will be placed in the same directory as the message format.
<b>Action Buttons</b>	

Field	Definition
OK	Saves your changes and closes the dialog.
Cancel	Discards your changes and closes the dialog.

## 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.

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 document
Exit	Exits the Format Builder program

### 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 tree pane and its items.

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 its child objects, from the tree. This item is placed on the clipboard for pasting into another location.  <b>Note:</b> 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 tree.  <b>Note:</b> 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 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 tree, as well as all child objects of that item.

## 2 Building Format Definitions

---

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

<b>Menu Command</b>	<b>Description</b>
Field	Inserts a new field. You can choose whether to insert the field as a child or sibling of the item selected in the 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 tree.
Comment	Inserts a comment. You can choose whether to insert the comment as a child or sibling of the item selected in the tree.

### View Menu

The following commands are available from the View Menu.

<b>Menu Command</b>	<b>Description</b>
Show Pallet	Displays the pallet window. For more information on the pallet, see <a href="#">Working with Pallets</a> .

<b>Menu Command</b>	<b>Description</b>
Expand All	Expands the entire tree pane to show the child objects of all items in the tree.
Collapse All	Collapses the entire tree pane to show only the root message format.

## Tools Menu

The following commands are available from the Tools Menu.

<b>Menu Command</b>	<b>Description</b>
Import	Displays a list of the installed importers. Choose the importer from which you want to import a message.
Options	Displays the Format Builder Options dialog. For more information, see Setting Format Builder Options.

## Test Menu

The following command is available from the Test Menu.

<b>Menu Command</b>	<b>Description</b>
Message Format	Opens the Format Tester. Refer to “Testing Format Definitions” for more information.

## Help Menu

The following commands are available from the Help Menu.

<b>Menu Command</b>	<b>Description</b>
Help Topics	Displays the main Help screen.

## 2 *Building Format Definitions*

---

<b>Menu Command</b>	<b>Description</b>
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.

# 3 Testing Format Definitions

Once you have built a format definition, you can test it using the built-in test feature of Format Builder. The test feature parses and reformats data as a validation test and also generates sample binary or XML data. Using the Tester, you can make sure the message formats you build with Format Builder produce the expected results. The Tester uses the runtime WLXT Java class to perform the test translation.

This section discusses the following topics:

- Running the Tester
- Debugging Formats

## Running the Tester

To run the tester:

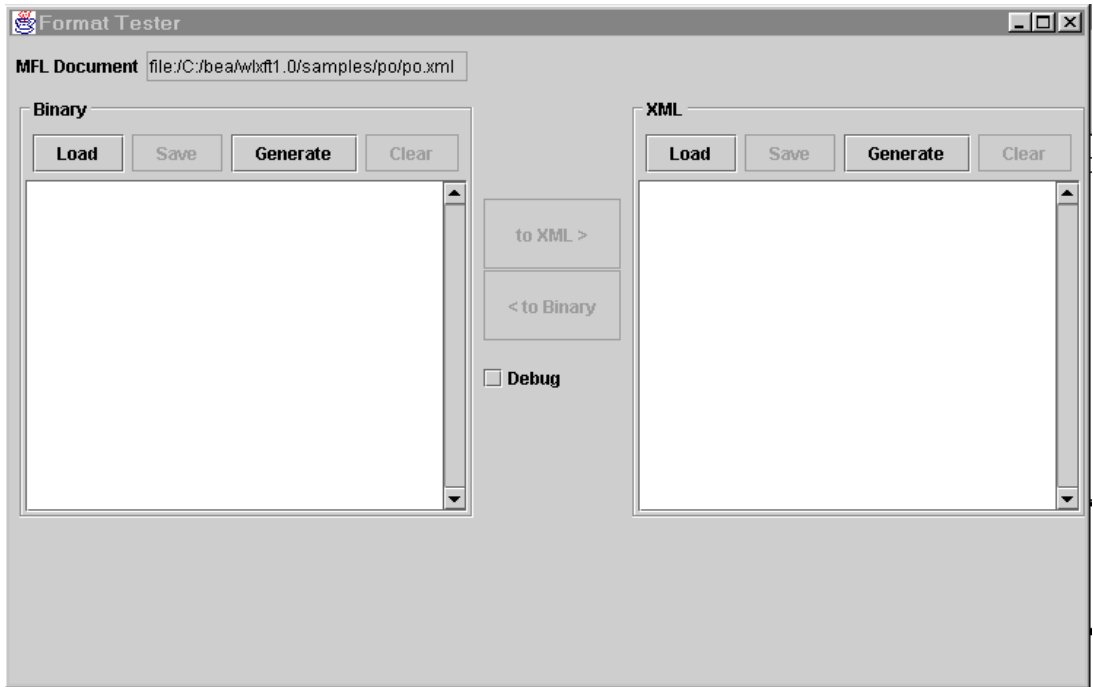
1. Start Format Builder.

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

2. Choose Test→Message Format. The Format Tester dialog displays.

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

**Figure 3-1 Format Tester Dialog**



3. Load a data file in the binary or XML edit control or enter your own data

Field	Description
MFL Document	Displays the name of the message format definition currently loaded in Format Builder.
<b>Binary</b>	
Load	Allows you to select the binary data file you want to work with.
Save	Allows you to save the binary data to a file.
Generate	Automatically generates binary test data based on the current message format definition.
Clear	Clears the data from the display area.



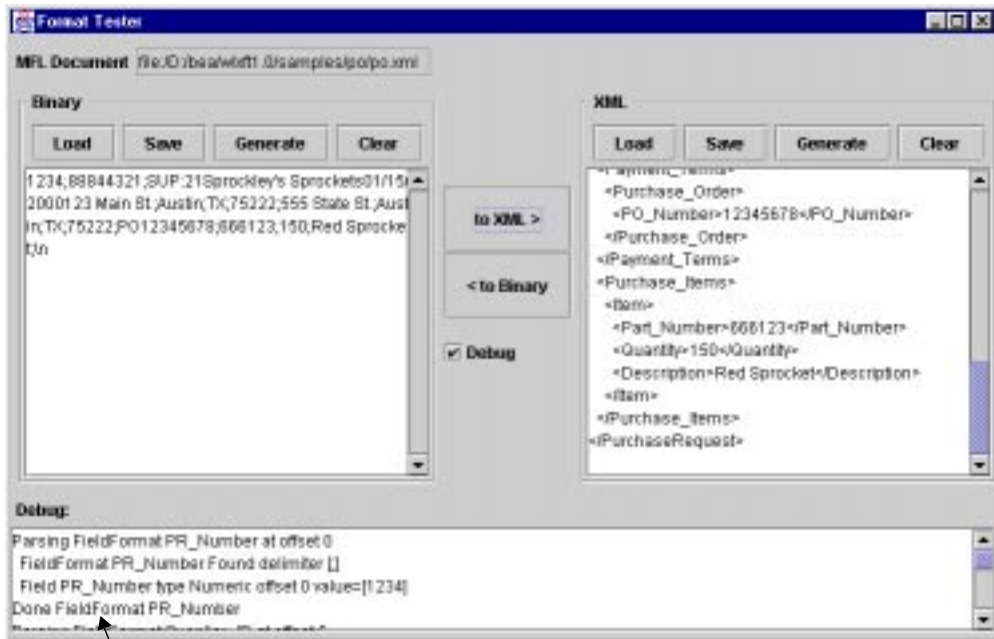
<b>Field</b>	<b>Description</b>
Display Area	<p>Displays the results of loading or translating a file. You can also enter your own data or modify data you have entered.</p> <p><b>Note:</b> Clicking the right mouse button in this display area opens a shortcut menu. From this menu, you can choose the Cut, Copy, or Paste commands. These commands allow you to cut or copy text displayed in this area and paste it in another location within this display area.</p>
<b>Action Buttons</b>	
To XML	Translates the binary data in the currently loaded file to XML format, based on the current message format definition.
To Binary	Translates the XML data in the currently loaded file to binary format, based on the current message format definition.
Debug	Select this option if you want to display the actions the Tester performs during a translation.
<b>XML</b>	
Load	Allows you to select the XML data file you want to work with.
Save	Allows you to save the XML data to a different filename.
Generate	Automatically generates XML test data based on the current message format definition.
Clear	Clears the data from the display area.
Display Area	<p>Displays the results of loading or translating a file. You can also enter your own data or modify data you have entered.</p> <p><b>Note:</b> Clicking the right mouse button in this display area opens a shortcut menu. From this menu, you can choose the Cut, Copy, or Paste commands. These commands allow you to cut or copy text displayed in this area and paste it in another location within this display area.</p>
<b>Debug</b>	

<b>Field</b>	<b>Description</b>
Debug Display Area	Displays the actions being performed by the Tester during a translation.

## Debugging Formats

The following steps illustrate using the Tester to debug a message definition that does not translate correctly.

1. Start Format Builder.
2. Open a Message Format.
  1. Open the Tester dialog box and load the file (either XML or binary) that you want to translate.
  2. Select Debug. The Debug text box opens at the bottom of the Format Tester dialog.
  3. Click the appropriate button (to XML or to Binary) to translate your data. The Debug window displays the actions that take place during the translation operation, including any errors that are encountered. The field and group values will be displayed 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 Tree Pane of Format Builder.



Errors encountered during translation process

4. Correct the errors and test the translation again.
5. Continue this process until the translation is successful.

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



---

# 4 Using the Run-Time Component

The run-time component of XML Translator consists of a Java class named WLXT. This class has 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 BEA WebLogic Process Integrator, or integrated into any Java application.

The XML Translator class provides several `parse()` methods that translate binary data into XML. XML Translator also provides several `serialize()` methods that translate XML data to a binary format. Binary data formats are described via MFL documents. XML Translator 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 code samples below illustrate how to use XML Translator to parse binary data into XML, and serialize XML into binary.

## Binary to XML

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

### Listing 4-1 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 XML Translator 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 of XML Translator 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 XML Translator `getXMLText()` method (as shown on line 17), or manipulated directly via the W3C DOM API.

## Generating XML with a Reference to a DTD

WLXT also 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 4-2 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             Document doc = 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 of XML Translator allow a `PrintWriter` to be passed in as the last parameter of the `parse()` method. If this parameter is not null, XML Translator 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 4-3 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
18                PrintWriter(System.out,true));
19            String xml = wlxt.getXMLText(doc, 0, 2);
20            System.out.println(xml);
21        }
22        catch (Exception e)
23        {
24            e.printStackTrace(System.err);
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 4-4 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 XML Translator to convert XML text to binary format.

---

**Listing 4-5 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 XML Translator 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 of XML Translator 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 4-6 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
32              FileOutputStream("mybinaryfile");
33
34          wlxt.serialize(mflDocumentName, doc, out, null);
35          out.close();
36      }
37      catch (Exception e)
38      {
39          e.printStackTrace(System.err);
40      }
41  }
```

---

This example illustrates passing in a Document object to the `serialize()` method of the XML Translator 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 to XML Translator 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.

#### Debug Output:

The following code represents debug output.

---

#### Listing 4-7 Debug Output

---

```
Processing xml and mfl nodes tcpl  
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

XML Translator 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 XML Translator 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 XML Translator class.

---

**Listing 4-8 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 XML Translator 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 20, a `transform()` method of the XML Translator 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 XML Translator 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 XML Translator is used in an EJB or servlet, where the same MFL documents or XSLT stylesheets are used repeatedly.

## init() method

The XML Translator 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 XML Translator class.

### Listing 4-9 Properties file `myconfig.cfg`:

---

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

---

### Listing 4-10 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 XML Translator 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 and cached inside the XML Translator 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 XML Translator class, see the Java API Documentation located in the `apidoc` subdirectory of your XML Translator installation.



# A Supported Data Types

This section lists the following data types supported by XML Translator.

- MFL Data Types
- COBOL Copybook Importer Data Types

## MFL Data Types

Table A-1 lists the MFL data types that XML Translator supports. These types are specified in the “type” attribute of a FieldFormat element.

**Table A-1 Supported MFL Data Types**

Data Type	Description
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.

<b>Data Type</b>	<b>Description</b>
Numeric	A string of characters containing only digits, i.e. '0' through '9'. Requires a length, length field, delimiter, or a delimiter field.
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.
EBCDIC	A string of characters in IBM Extended Binary Coded Decimal Interchange Code. 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.
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: 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: 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.

<b>Data Type</b>	<b>Description</b>
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.
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.

<b>Data Type</b>	<b>Description</b>
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.
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.
DateTime: YYYYMMDDhhmmss	A fourteen byte numeric string of the format YYYYMMDDHHMISS. A Base data type may be specified.
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.
DateTime: MMDDYYhhmm	A string of numeric digits defining a date and time, i.e. 0122001224.
DateTime: MMDDYYhhmmss	A string of numeric digits defining a date and time, i.e. 012200122400.
Date: MM/DD/YY	A string defining a date, i.e. 01/22/00.
Date: MM/DD/YYYY	A string defining a date, i.e. 01/22/2000.
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.
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.

<b>Data Type</b>	<b>Description</b>
Date: DDMMYY	A string defining a date, i.e. 22JAN00.
Date: DDMMYYYY	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.
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.
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: MMM-YY	A string defining a date, i.e. JAN-00.
Date: MMM-YYYY	A string defining a date, i.e. JAN-2000.
Date: MMYYY	A string defining a date, i.e. JAN00.
Date: MMYYYYY	A string defining a date, i.e. JAN2000.
Date: MMMDDYYYY	A string defining a date, i.e. JAN222000.
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.

<b>Data Type</b>	<b>Description</b>
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'
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 WLXT. 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, WLXT writes the literal in the resulting binary byte stream.
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.

# COBOL Copybook Importer Data Types

Table lists the COBOL data types and the support provided by the Importer.

**Table 4-1 COBOL Data Types**

<b>COBOL Type</b>	<b>Support</b>
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

## A Supported Data Types

---

<b>COBOL Type</b>	<b>Support</b>
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



---

# Glossary

## **Binary Data**

A file format for data encoded as a sequence of bits, but not necessarily consisting of a sequence of printable characters (text). The term is often used for executable machine code.

## **Big Endian**

Binary format where most significant byte has the lowest address. This format is used on IBM 370 and most RISC designs.

## **COBOL Copybook Importer**

Reads a COBOL Copybook and generates a message format reflecting the data structure of the COBOL Copybook.

## **Code Page**

In the context of this documentation, the character encoding of the field data.

## **Data Transformation**

In the context of this documentation, data transformation is the term used to describe the mapping of XML data to another XML format. An example would be mapping an instance of a RosettaNet document to an instance of an ebXML document.

## **Data Translation**

In the context of this application, data translation is the process of converting binary data to or from XML.

## **Delimiter**

A sequence of bytes that denote the end of a field or group of data.

## **Document Type Definition (DTD)**

Defines what content can exist in an XML document. DTDs are part of the W3C XML Specification 1.0.

---

### **eXtensible Stylesheet Language: Transformations (XSLT)**

An XML language designed for transforming one XML document into another. An XSLT document, or stylesheet, describes data transformations that are to be performed on nodes of an XML document. Using XSLT, an XML document can be transformed into a variety of text formats (XML, HTML, PDF, etc.). XSLT is a W3C recommendation.

### **Field**

A sequence of bytes that are interpreted by an application as a unit of data.

### **Group**

A set of fields and/or groups that are to be treated as having a unifying relationship.

### **Group Choice**

A group comprised of fields or other groups that are mutually exclusive in the actual binary data.

### **Java Message Service (JMS)**

A peer-to-peer messaging system for java programs to send and receive messages. A JMS application is capable of sending or receiving application defined messages (asynchronous requests, reports, or events) to other JMS applications so that these separate applications can collaborate or coordinate their efforts.

### **Little Endian**

Binary format in which bytes at lower address have lower significance. This format is used on Intel and VAX processors.

### **Message Format**

The description of a binary format produced by Format Builder.

### **Metadata**

Data that is used to describe other data. Message Formats created using Format Builder are the metadata used to parse binary data.

### **Message Format Language (MFL)**

An XML language created by BEA that describes the native representation and hierarchy of binary data. MFL is an XML description of binary data.

---

## **Reference**

A group or field that relies on a prior definition to determine its name, type, and termination attributes.

## **Schema**

An XML document that defines what can be in an XML document. A Schema definition is more specific than a DTD and provides much finer-grained control over the content that can exist in an XML document.

## **Servlet**

A server-side Java program that is usually executed in response to an HTTP request and produces its output in a browser.

## **Stylesheet**

An XSL document. A stylesheet describes data transformations (or mappings) that are to be performed on an XML document. A stylesheet describes which nodes of an XML document are to be manipulated (using XPath) and which manipulations are to be performed.

## **WebLogic Process Integrator**

Workflow engine for BEA WebLogic application servers that automates workflow, business-to-business processes, and application assembly.

## **WebLogic Server**

WebLogic's standards-based, pure-java application server, for assembling, deploying and managing distributed Java applications. WebLogic Server manages application components and DBMS connections to ensure security, scalability, performance, and transaction integrity.

## **XML - Extensible Markup Language**

Data format that is easily read and manipulated by both humans and computers; data and meta-data are both included in the data, to provide a standard self-describing syntax for representing information. XML is a World Wide Web Consortium (W3C) standard.

## **XPath**

Used within XSLT, XPath is an XML language that identifies parts of an XML document to be processed. XPath is used in XSLT to specify which nodes of an XML document are to be copied or manipulated during an XSLT transformation. XPath is a W3C recommendation.



---

# Index

## B

binary data, about 2-2

## C

choice of children 2-19

COBOL copybook

importing 2-32

COBOL copybook importer data types A-7

COBOL data types A-7

code page

field data option 2-24

comment, creating 2-25

comments 2-7

customer support contact information ix

## D

data base type

field data options 2-24

data fields 2-7

data types

COBOL A-7

MFL A-1

support A-1

debug writer 4-4

delimited

group delimiter 2-20

delimiter

field 2-24

group 2-20

delimiter field

field delimiter 2-25

delimiter is shared

group delimiter 2-20

document type definition 4-2

documentation, where to find it viii

## E

edit menu 2-36

copy 2-37

cut 2-37

delete 2-37

demote 2-38

duplicate 2-37

move down 2-37

move up 2-37

paste 2-37

promote 2-37

redo 2-36

undo 2-36

## F

field 2-6

creating 2-21

data type 2-23

delimiter 2-24

name 2-23

occurrence 2-23

optional 2-23

parameters 2-6

---

- tag 2-23
- field data options
  - data base type 2-24
  - value 2-24
  - year cutoff 2-24
- field data options, code page 2-24
- field delimiter 2-24
  - delimiter 2-24
  - delimiter field 2-25
  - length 2-24
  - length field 2-24
- field occurrence
  - once 2-23
  - repeat delimiter 2-23
  - repeat field 2-23
  - repeat number 2-23
  - unlimited 2-23
- file menu 2-35
  - close 2-36
  - exit 2-36
  - new 2-36
  - open 2-36
  - save 2-36
  - save as 2-36
- Format Builder
  - setting options 2-34
  - starting 2-8
  - using 2-8

## G

- group
  - creating 2-18
  - delimiter 2-20
  - description 2-19
  - occurrence 2-19
- group attributes 2-7
- group delimiter
  - delimited 2-20
  - delimiter 2-20
  - delimiter is shared 2-20

- group occurrence
  - once 2-20
  - repeat delimiter 2-20
  - repeat field 2-20
  - repeat number 2-20
  - unlimited 2-20

## H

- help menu 2-39
  - about 2-39
  - help topics 2-39
  - how do I 2-39
- hierarchical groups 2-7

## I

- insert menu 2-38
  - comment 2-38
  - field 2-38
  - group 2-38

## L

- length
  - field delimiter 2-24
- length field
  - field delimiter 2-24

## M

- menu bar 2-12
- Message Format 2-6
  - adding pallet items 2-30
  - default version 2-35
  - opening 2-31
  - saving 2-30
- message node 2-10
- MFL data types A-1
- MFL documents, about 2-6



---

## N

name

- field 2-23
- group 2-19

## O

occurrence

- group 2-20

occurrence

- field 2-23

once

- field occurrence 2-23
- group occurrence 2-20

optional

- field 2-23
- group 2-19

## P

pallet

- adding items 2-29
- deleting items 2-30

pallets 2-29

printing product documentation viii

Properties Pane 2-8

## R

references 2-7

- creating 2-26

related information viii

repeat delimiter

- field occurrence 2-23
- group occurrence 2-20

repeat field

- field occurrence 2-23
- group occurrence 2-20

repeat number

- field occurrence 2-23
- group occurrence 2-20

root node 2-10

## S

shortcut menus 2-15

support

technical ix

## T

test menu 2-39

message format 2-39

toolbar 2-12

buttons 2-12

tools menu 2-39

import 2-39

options 2-39

Tree Pane 2-8

using 2-10

## U

unlimited

field occurrence 2-23

group occurrence 2-20

## V

valid names 2-18

value

field data option 2-24

view menu 2-38

collapse all 2-38

expand all 2-38

show pallet 2-38

## X

XML content model options 2-35

XML documents, about 2-3

XML formatting options 2-35