BEA WebLogic Integration™

WebLogic Integration データ変換

リリース 7.0
マニュアルの日付： 2002 年 6 月

著作権

限定的権利条項

商標または登録商標

WebLogic Integration データ変換

パート番号	日付	ソフトウェアのバージョン
なし	2002 年 6 月	7.0

目次

3. フォーマット定義のビルド

5. リポジトリ ドキュメントの検索と保存

6. 実行時コンポーネントの使用法

A. サポートされるデータ型

B. カスタム データ型の作成

C. Purchase Order サンプルの実行

索引

このマニュアルの内容

このマニュアルでは、WebLogic Integration を使用して、バイナリ フォーマットと XML との間でデータを相互に変換する方法について説明します。具体的には、以下のトピックを取り上げます。

- 付録 1「データ変換の概要」
- 付録 3「フォーマット定義のビルド」
- 付録 2「フォーマット定義のテスト」
- 付録 4「メタデータのインポート」
- 付録 5「リポジトリ ドキュメントの検索と保存」
- 付録 6「実行時コンポーネントの使用法」
- 付録 A「サポートされるデータ型」
- 付録 B「カスタム データ型の作成」
- 付録 C「Purchase Order サンプルの実行」

対象読者

このマニュアルは、バイナリから XML へ、また XML からバイナリへのデータ変換を実行するアプリケーション プログラマと技術アナリストを主な対象としています。

e-docs Web サイト

BEA 製品のドキュメントは、BEA Systems, Inc. の Web サイトで入手できます。BEA のホーム ページで [製品のドキュメント] をクリックするか、または「e-docs」という製品ドキュメント ページ（http://edocs.beasys.co.jp/e-docs/index.html）を直接表示してください。

このマニュアルの印刷方法

Web ブラウザの [ファイル | 印刷] オプションを使用すると、Web ブラウザからこのマニュアルを一度に 1 ファイルずつ印刷できます。

このマニュアルの PDF 版は、Web サイトで入手できます。WebLogic IntegrationPDF を Adobe Acrobat Reader で開くと、マニュアルの全体（または一部分）を書籍の形式で印刷できます。PDF を表示するには、WebLogic Integration ドキュメントのホーム ページを開き、[PDF 版] ボタンをクリックして、印刷するマニュアルを選択します。

Adobe Acrobat Reader がない場合は、Adobe の Web サイト（http://www.adobe.co.jp/）で無料で入手できます。

関連情報

以下の BEA マニュアルも用意されています。

- *Data Integration プラグイン ユーザーズ ガイド*
- *Data Integration プラグイン オンライン ヘルプ*

サポート情報

WebLogic Integration のドキュメントに関するユーザからのフィードバックは弊社にとって非常に重要です。質問や意見などがあれば、電子メールで **docsupport-jp@bea.com** までお送りください。寄せられた意見については、WebLogic Integration のドキュメントを作成および改訂する BEA の専門の担当者が直に目を通します。

電子メールのメッセージには、WebLogic Integration リリース 7.0 のドキュメントをご使用の旨をお書き添えください。

本リリースの WebLogic Integration について不明な点がある場合、または WebLogic Integration のインストールおよび動作に問題がある場合は、BEA WebSupport (http://websupport.bea.com/custsupp) を通じて BEA カスタマ サポートまでお問い合わせください。カスタマ サポートへの連絡方法については、製品パッケージに同梱されているカスタマ サポート カードにも記載されています。

カスタマ サポートでは以下の情報をお尋ねしますので、お問い合わせの際はあらかじめご用意ください。

- お名前、電子メール アドレス、電話番号、ファクス番号
- 会社の名前と住所
- お使いの機種とコード番号
- 製品の名前とバージョン
- 問題の状況と表示されるエラー メッセージの内容

表記規則

このマニュアルでは、全体を通して以下の表記規則が使用されています。

表記法	適用
〔Ctrl〕+〔Tab〕	複数のキーを同時に押すことを示す。
斜体	強調または書籍のタイトルを示す。

等幅テキスト	コード サンプル、コマンドとそのオプション、データ構造体とそのメンバー、データ型、ディレクトリ、およびファイル名とその拡張子を示す。等幅テキストはキーボードから入力するテキストも示す。
	例
	`#include <iostream.h> void main ( ) the pointer psz`
	`chmod u+w *`
	`\tux\data\ap`
	`.doc`
	`tux.doc`
	`BITMAP`
	`float`

太字の等幅テキスト	コード内の重要な箇所を示す。 *例* `void **commit** ( )`
斜体の等幅テキスト	コード内の変数を示す。 *例* `String *expr*`
すべて大文字のテキスト	デバイス名、環境変数、および論理演算子を示す。 *例* LPT1 SIGNON OR
{ }	構文の中で複数の選択肢を示す。実際には、この括弧は入力しない。

表記法	適用

[ ]	構文の中で任意指定の項目を示す。実際には、この括弧は入力しない。
	例
	`buildobjclient [-v] [-o name ] [-f file-list]...`
	`[-l file-list]...`

|	構文の中で相互に排他的な選択肢を区切る。実際には、この記号は入力しない。

...	コマンドラインで以下のいずれかを示す。
	■ 引数を複数回繰り返すことができる
	■ 任意指定の引数が省略されている
	■ パラメータや値などの情報を追加入力できる
	実際には、この省略記号は入力しない。
	例
	`buildobjclient [-v] [-o name ] [-f file-list]...`
	`[-l file-list]...`

. . .	コード サンプルまたは構文で項目が省略されていることを示す。 実際には、この省略記号は入力しない。

1　データ変換の概要

ほとんどの EAI (Enterprise Application Integration : エンタープライズ アプリケーション統合) の問題領域で、データ変換は EAI ソリューションの固有の課題です。XML は急速にアプリケーション間の情報交換に関する標準となりつつあり、異なるアプリケーションを統合する上でかけがえのない手段となっています。しかし、ほとんどのデータ変換エンジンではバイナリ データ フォーマットと XML との間の変換がサポートされません。WebLogic Integration では従来のシステムで使用されるバイナリ フォーマットと XML との間のデータ変換がサポートされるので、各アプリケーションによる情報交換が可能になります。

この章の内容は以下のとおりです。

- XML 変換について
- データ統合について
- 変換後のオプションと考慮事項

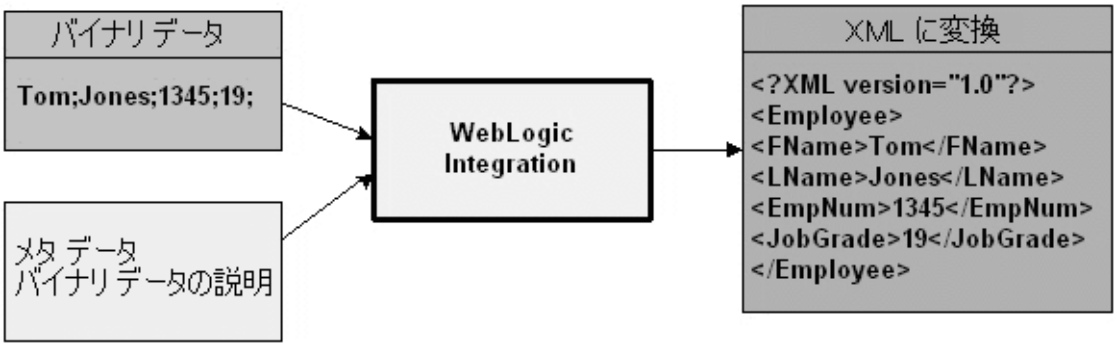
XML 変換について

従来のアプリケーションとの間で送受信するデータは、多くの場合、情報作成元のマシンに固有のバイナリ フォーマットで構成されたプラットフォーム固有の情報です。バイナリ データは自己記述型ではありません。このため従来のアプリケーションのバイナリ データを使用する各アプリケーションの中にデータのフォーマットに関する情報（メタデータ）を埋め込んで、各アプリケーションが理解できるようにする必要があります。

XML は、アプリケーション間で情報を交換するための標準になりつつあります。その理由として、XML ではデータの記述がデータ ストリームの中に埋め込まれ、アプリケーション間でデータを簡単に交換できるようにしていることが挙げられます。XML の表現するデータ構造は複雑ですが、解析は容易です。このため、アプリケーションを結合する場合に、各アプリケーションの中にメタデータを埋め込む必要がありません。

バイナリ データの XML への変換では、構造化されたバイナリ データを XML ドキュメントに変換するので、標準の XML 解析方法によってデータにアクセスできます。変換を行うときに使用するメタデータを作成する必要があります。変換プロセスの実行中、バイナリ データの各フィールドは、そのフィールドについて定義されたメタデータに従って、XML に変換されます。指定するメタデータには、フィールドの名前、データ型、サイズ、およびフィールドが省略可能かどうかの指定を記述しなければなりません。バイナリ データは、このバイナリ データの記述を使用して XML に変換されます。図 1-1 では、サンプルのバイナリ データを使用して XML への変換方法を示しています。

図 1-1 XML データ変換 (Tom;Jones;1345;19;)



WebLogic Server プラットフォームで開発されたアプリケーションの多くは、標準のデータ形式として XML を使用します。WebLogic Server プラットフォーム上の他のアプリケーションから従来のシステムのデータにアクセスする場合、WebLogic Integration を使用することで、システムのデータを、バイナリ フォーマットと XML フォーマットとの間で相互に変換できます。最終的な用途として特定の XML 固有言語に変換されたデータが必要な場合は、XML データ マッピング ツールを使用してデータを変換する必要があります。

データ統合について

WebLogic Integration では、従来のシステムのバイナリ データと XML を相互に変換できるようにすることにより、さまざまなエンタープライズ アプリケーションのデータ統合を容易なものにします。従来のデータが XML として使用で

きるようになると、XML アプリケーションによる直接使用、特定の XML 文法への変換、あるいは WebLogic Integration Studio でのワークフローの開始のための直接使用が可能になります。WebLogic Integration は、次の 3 つのデータ統合ツールを使用することにより、非 XML と XML との相互変換をサポートします。

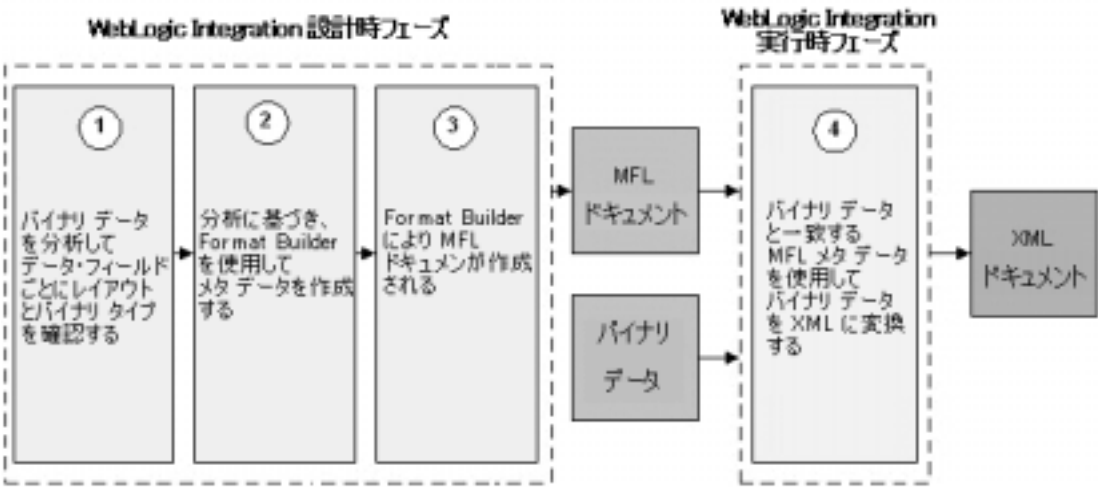
- 設計時 コンポーネント
- 実行時コンポーネント
- Business Process Management へのプラグイン

変換は、2 段階で行われます。まず、設計時ツールの Format Builder を使用してバイナリ データが記述されます。ここでは、Format Builder で作成するメタデータにバイナリ ファイルのレコード レイアウトが正確に反映されるようにバイナリ データを分析します。

次に、Format Builder でメタ データ（入力データの記述）を作成し、このメタデータを MFL (Message Format Language : メッセージ フォーマット言語) ドキュメントとして保存します。WebLogic Integration に付属の *インポータ*は、COBOL コピーブックなどバイナリ メタデータの共通ソースからメッセージ フォーマット定義を自動的に作成するユーティリティです。

次に、WebLogic Integration の実行時コンポーネントを使用してバイナリ データのインスタンスを XML に変換できます。図 1-2 は、非 XML から XML へのデータ変換に関するイベント フローを示しています。BPM 機能のプラグインを使用すると、変換のコンフィグレーションを容易に行うことができます。

図 1-2 非 XML から XML フォーマットへのデータ変換のイベント フロー



設計時 コンポーネント

WebLogic Integration の設計時データ統合コンポーネントは、Format Builder と呼ばれる Java アプリケーションです。Format Builder はバイナリ データ レコードの記述を作成するときに使用します。Format Builder を使用すると、XML データとバイナリ データを相互に変換できるようにバイナリ データのレイアウトと階層を記述できます。

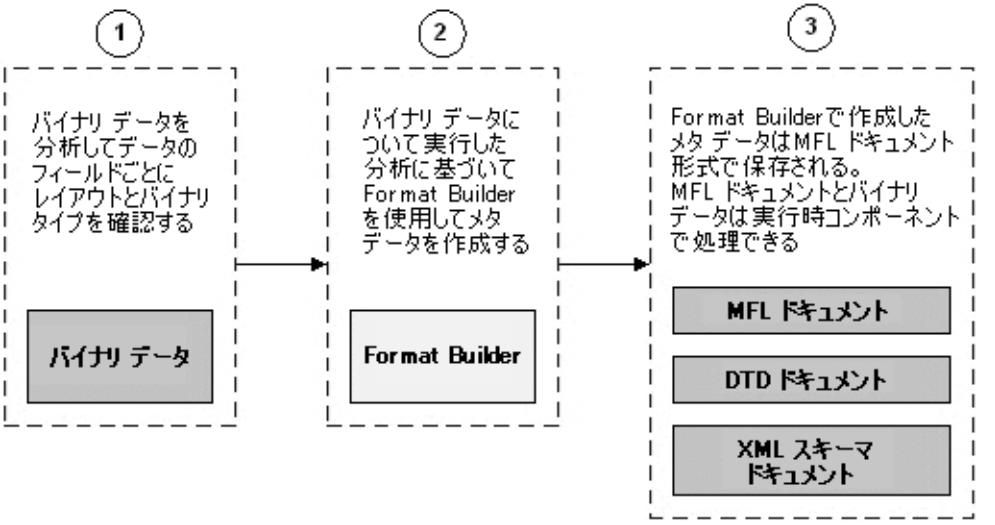
バイト シーケンスをフィールドとして記述し、各フィールドについて、データ型（浮動小数点、文字列など）、データのサイズ、フィールド名を指定できます。さらに、フィールドの集まり（グループ）、フィールドとグループの複数のインスタンス、および集約を定義できます。

作成する記述は、MFL と呼ばれる XML 文法で保存されます。MFL ドキュメントは、Data Integration の実行時コンポーネントと Business Process Management のプラグインが、バイナリ データ レコードのインスタンスと XML ドキュメントのインスタンスを相互に変換するのに使用するメタデータを含みます。

さらに、Format Builder は、変換によって作成される XML ドキュメントを記述する DTD または XML スキーマも作成します。

図 1-4 に、設計時フェーズでの、Format Builder を介したバイナリ データと XML データのプロセス フローを示します。

図 1-3　Format Builder を介した設計時プロセス フロー



Format Builder は格納された MFL メッセージの検索、検証および編集や、ユーザ自身のデータを使用したメッセージ フォーマット定義のテストに使用することもできます。MFL ドキュメントはファイル システムを使用して格納することも、リポジトリにアーカイブすることもできます。

テスト機能を使用すると、サンプル XML ファイルをバイナリ フォーマットに、サンプル バイナリ ファイルを XML フォーマットにそれぞれ変換することで、Format Builder で作成された XML ドキュメントを検証できます。変換されたデータをファイルに保存すれば、あとでテストできます。

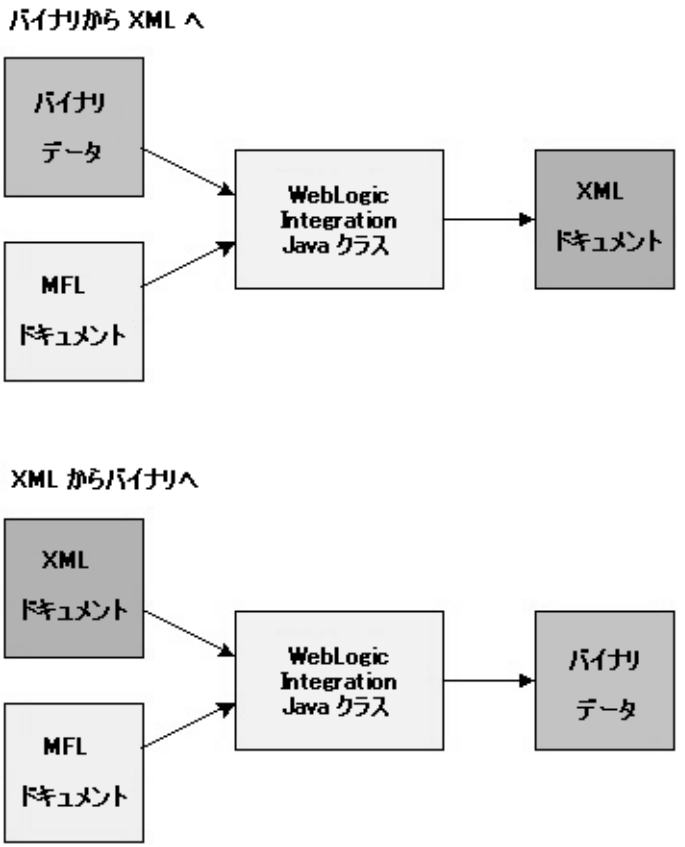
実行時コンポーネント

WebLogic Integration の実行時 Data Integration コンポーネントは Java クラスであり、このクラスには、バイナリ フォーマットと XML フォーマットの間でデータを変換するためのさまざまなメソッドが用意されています。この Java クラスは、さまざま方法で使用できます。具体的には、次のように使用されます。

- WebLogic Server を使用して EJB にデプロイされる。
- Studio のワークフローからビジネス オペレーションとして呼び出される。
- 任意の Java アプリケーションに統合される。

図 1-4 に、バイナリから XML への変換および XML からバイナリへの変換の両方について、実行時プロセス フローを示します。

図 1-4 実行時プロセス フロー



バイナリから XML への変換

コード リスト 1-1 のサンプル コードは、バイナリ データを含むファイルから XML ドキュメント オブジェクトへの解析を示します。MFL ファイル（mymfl.mfl）は、mybinaryfile のバイナリ データの記述を提供します。

コード リスト 1-1 バイナリから XML への変換のサンプル コード

```
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 からバイナリへの変換

コード リスト 1-2 のサンプル コードは myxml.xml に含まれる XML データから MFL ドキュメント mymfl.mfl で指定されるバイナリ フォーマットへの変換を示すサンプル コードです。バイナリ データはファイル mybinaryfile に書き込まれます。

コード リスト 1-2 XML からバイナリへの変換のサンプル コード

```
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);
}
```

Business Process Management へのプラグイン

WebLogic Integration 付属の Business Process Management (BPM) ツールを使用すると、ワークフロー、B2B プロセス、およびエンタープライズ アプリケーション アセンブリを自動化できます。WebLogic Server で動作するように設計された、BPM ツールは、J2EE 準拠の堅牢なワークフローおよびプロセス インテグレーション ソリューションを実現します。

直感的なフローチャート パラダイムにより、ビジネス アナリストは BPM ツールの基礎となる WebLogic Integration プロセス エンジンを使用してアプリケーションにまたがるビジネス プロセスを定義したり、人間とアプリケーションの対話を自動化したりします。開発者は、プロセス エンジンを使用して、アプリケーション コンポーネントをアセンブルして、これらのアプリケーション コンポーネントを実行および管理できます。このために、新しいコードを書く必要はありません。

プロセス エンジンは新機能をプラグインとして組み込むことができる拡張可能なアーキテクチャを備えています。WebLogic Integration には、BPM アクションを操作して変換機能（XML からバイナリへ、バイナリから XML への両方）にアクセスできる Data Integration プラグインが含まれます。

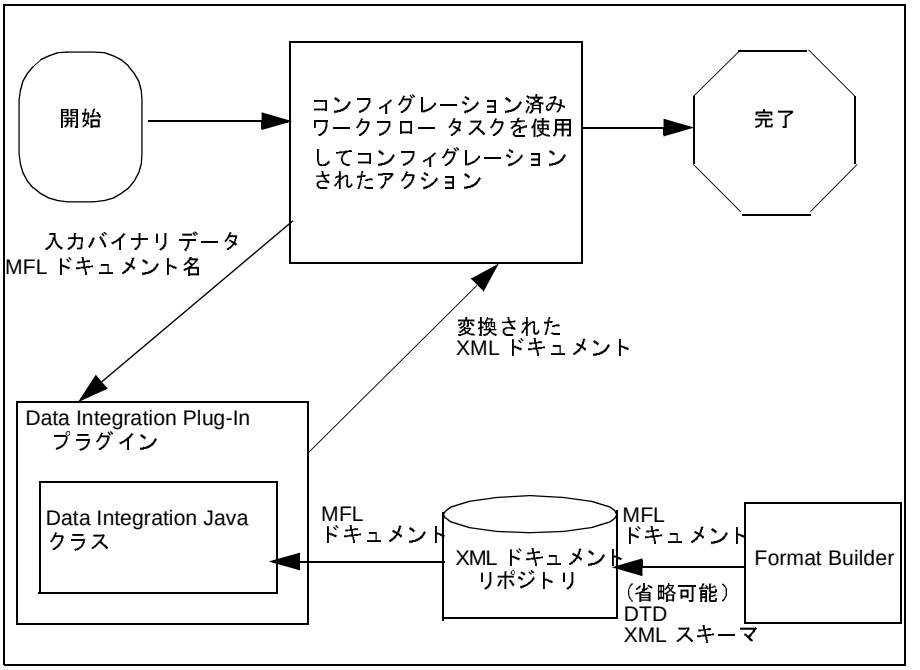
Business Process Management (BPM) 機能の Data Integration プラグインは、従来のシステムのバイナリ データの XML への変換を可能にすることにより、アプリケーション間の情報交換をサポートにします。Data Integration プラグインは、XML からバイナリへの変換と、バイナリから XML への変換を実行できる BPM アクションを提供します。

このデータ変換機能に加えて、Data Integration プラグインは、次の機能を提供します。

- バイナリ フォーマットでのイベント データ処理。
- パフォーマンスを向上させるための MFL ドキュメントのインメモリ キャッシュおよび変換オブジェクト プール
- バイナリ データを編集して表示するための、`BinaryData` 変数型
- クラスタ化された WebLogic Server 環境での実行

次の図は、Data Integration と BPM 機能の関係を示しています。

図 1-5 Data Integration と BPM の関係



変換後のオプションと考慮事項

バイナリ データの XML への変換が正常に完了したら、XML データを、Business Process Management (BPM) クライアント アプリケーション（Studio および Worklist）などの XML を使用する他のアプリケーションに送信できます。対象となるアプリケーションの要件により、生成された XML を、ここでさらに別の XML 文法、表示形式（HTML）、または別のバイナリ フォーマットに変換する必要があることがあります。このマニュアルでは、ある XML 文法から別の文法に XML ドキュメントを変換するプロセスを *XML 変換* と呼びます。

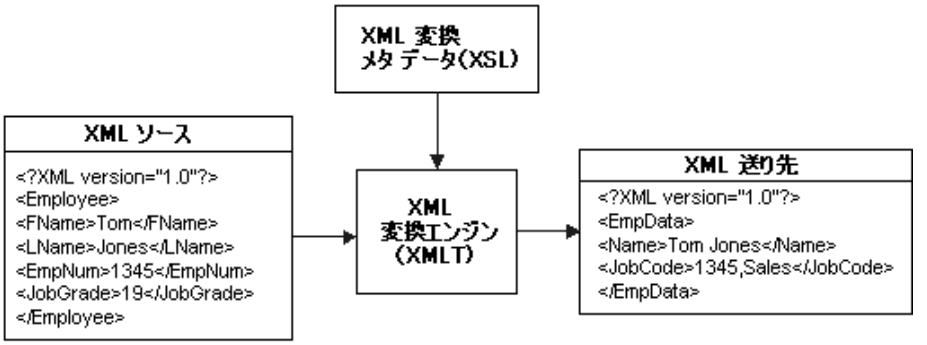
XML 変換は WebLogic Server の XML モジュールを使用して行うことができます。BPM の機能には、このモジュールを呼び出して XSL スタイル シートを使用して XML ドキュメントを 変換できるようにするアクションが含まれます。XML を次のいずれかのフォーマットに変換できます。

- 特定の XML 固有言語（RosettaNet、ebXML など）。
- 表示形式（HTML）。
- 別の MFL ドキュメントのフォーマットに対応し、WebLogic Integration の Data Integration コンポーネントにより、別のバイナリ フォーマットにコンバートできるフォーマット。

XSL (eXtensible Stylesheet Language) は XML ドキュメントのノードで実行できる一連の変換を記述する XML 言語です。XSL スタイルシートは XML ドキュメントを別の XML 固有言語や別のテキスト フォーマット（HTML、PDF など）にマッピングするために、XSLT (XSL Transformation) エンジンが使用できる XSL ドキュメントです。また、スタイルシートは WebLogic Integration のデータ変換実行時コンポーネントと併用して XML の変換に使用することもできます。

図 1-6 は､1 つの XML 文法が XSLT エンジンを使用して別の XML 文法に変換されるところを示します。この場合の変換メタデータは、1 つの XML 文法を別の XML 文法にマッピングする方法を記述する XSL スタイルシートです。

図 1-6　Tom;Jones;1345;19; の XML データ変換



2　フォーマット定義のテスト

フォーマット定義をビルドした後、Format Tester を使用してテストできます。Format Tester により検証テスト中にデータの解析と書式の再設定が行われ、サンプルのバイナリ データまたは XML データが生成されます。このサンプル データを編集、検索およびデバッグすれば、希望する結果を得ることができます。Format Tester では、データ変換実行時エンジンが変換テストに使用されます。

この章では、以下のトピックについて説明します。

- Format Tester の呼び出し
- [Format Tester] ダイアログ ボックスの使用法
- フォーマット定義のテスト
- フォーマット定義のデバッグ

Format Tester の呼び出し

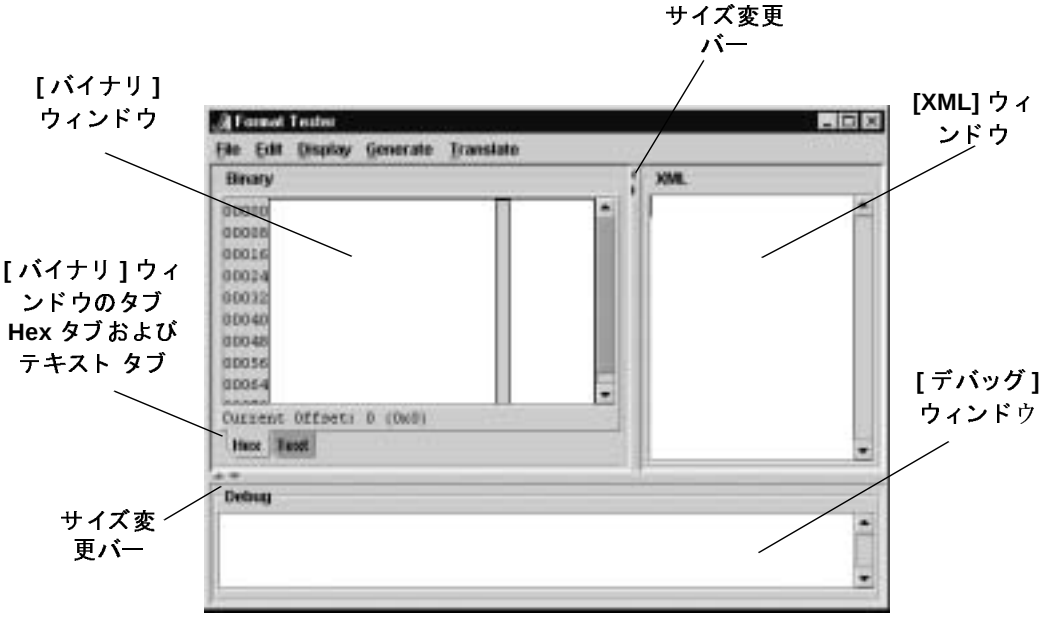
Format Tester を呼び出す手順は次のとおりです。

1. Format Builder が呼び出されていない場合は、[スタート ｜プログラム｜ BEA WebLogic Platform 7.0 ｜ WebLogic Integration 7.0 ｜ Format Builder] を選択して、Format Builder を呼び出し、テストする MFL ドキュメントを開きます。

   注意： Format Tester を実行するには、事前に、メッセージ フォーマット ドキュメントを Format Builder で開いておく必要があります。

2. 次の図に示すように、[Tools ｜ Test] を選択し、[Format Tester] ダイアログ ボックスを表示します。

図 2-1　[Format Tester] ダイアログ ボックス



[Format Tester] ダイアログ ボックスは、[Binary] ウィンドウ、[XML] ウィンドウ、および [Debug] ウィンドウの 3 つのウィンドウに分割されています。各ウィンドウはサイズ変更バーにより分割されます。サイズ変更バーをドラッグすると、ウィンドウのサイズを調整できます。また、バーの矢印をクリックして、ウィンドウの表示と非表示を切り替えることができます。たとえば、[Binary] ウィンドウと [XML] ウィンドウを分割するバーの左矢印をクリックすると、[Binary] ウィンドウが非表示になります。ウィンドウが非表示になっている場合、バーをドラッグするか該当する矢印をクリックすると、ウィンドウを再表示できます。

注意： セッションで初めて、Format Tester を開いたときは、[Binary] ウィンドウと [XML] ウィンドウのみが表示されます。[Debug] ウィンドウを表示するには、[Format Test] ダイアログ ボックス下部のサイズ変更バーを使用するか、[Display ｜ Debug] を選択して [Debug] ウィンドウの表示 / 非表示を切り替えます。

[Format Tester] ダイアログ ボックスの使用法

以下のトピックでは、[Format Tester] ダイアログ ボックスのさまざまなツールを使用して、ボックス内を移動しコマンドを実行する方法を説明します。

- [Binary] ウィンドウの使用
- [XML] ウィンドウの使用法
- [Debug] ウィンドウの使用
- サイズ変更バーの使用法
- メニュー バーの使い方
- ショートカット メニューの使用法

以下のトピックでは、これらの各機能をどのように使用すればタスクの実行に役立つかについて説明します。

[Binary] ウィンドウの使用

[Binary] ウィンドウには、次のようなサンプル データを表示できます。

- アクティブな MFL ドキュメントに基づいて生成された。
- [XML] ウィンドウのコンテンツから変換された。
- アクティブな MFL ドキュメントをテストするために特別に設計された。

既存のバイナリ データ ファイルを開いて、編集し、ウィンドウのコンテンツを保存できます。また、テストの状況に応じて、ウィンドウをクリアできます。詳細は、2-6 ページの「メニュー バーの使い方」および 2-10 ページの「ショートカット メニューの使用法」を参照してください。

[Format Tester] ダイアログ ボックスの [Binary] ウィンドウは、バイナリ ファイルのエディタとして使用できます。このウィンドウには、次のタブがあります。

- [Hex] ーデータ オフセット、個々のバイトの 16 進値、および対応するテキスト（ASCII または EBCDIC フォーマットで表示可能）を表示する。
- [Text] ーテキストのみを表示。

このエディタを使用すると、16 進バイトやテキスト値を編集できます。16 進データ値が変更されると、それに対応するテキスト値が更新され、テキスト値が変更された場合は 16 進データ値が更新されます。

データ オフセット機能の使用

[Hex] タブのデータ オフセット機能を使用すると、データ オフセットを 16 進または 10 進アドレスとして表示できます。

データ オフセットのフォーマットを変更する手順は次のとおりです。

1. [Display ｜ Hex] を選択します。次の 2 つのデータ オフセット オプションが表示されます。
   - [Offsets as Hexadecimal]
   - [Offsets as Decimal]
2. 必要に応じたオプションを選択します。[Binary] ウィンドウの [Data offset] 領域は、選択したオプションに合わせて自動的に変更されます。

テキスト機能の使用

[Binary] ウィンドウの [Text] タブは、印刷可能な文字（通常は、語および数字の形式で）および特定の制御文字（キャリッジ リターン、タブなど）を表示します。たとえば、キャリッジ リターンは行末中断として表示されます。印刷不能な文字は、小さな矩形で示されます。

[XML] ウィンドウの使用法

[XML] ウィンドウには、次のようなサンプル XML を表示できます。

- アクティブな MFL ドキュメントに基づいて生成された。
- [Binary] ウィンドウのコンテンツから変換された。

- アクティブな MFL ドキュメントをテストするために特別に設計された。

既存の XML ファイルを開いて、編集し、ウィンドウのコンテンツを保存できます。また、テストの状況に応じて、ウィンドウをクリアできます。詳細は、2-6 ページの「メニュー バーの使い方」および 2-10 ページの「ショートカット メニューの使用法」を参照してください。

XML が生成されるときは、Format Builder の [Options] ダイアログ ボックスで指定された XML フォーマット オプションが使用されます。詳細については、3-43 ページの「Format Builder のオプションの設定」を参照してください。

[Debug] ウィンドウの使用

[Debug] ウィンドウでは、変換処理中に発生する操作、発生したエラー、およびフィールドとグループの値と区切り記号が表示されます。エラーの原因を調べるには、正常に解析された最後のフィールドを確認して、ナビゲーション ツリーで次に配置されているフィールドのプロパティを調べます。

セッションで初めて、Format Tester を開いたときは、[Binary] ウィンドウと [XML] ウィンドウのみが表示されます。[Debug] ウィンドウを開くには、[Display | Debug] を選択して [Debug] ウィンドウに切り替えます。[Debug] ウィンドウは、[Binary] ウィンドウと [XML] ウィンドウの下に表示されます。

デバッグ出力は、メッセージのうちで最新の 64 KB までに制限されます。この制限により、大量のデバッグ出力によって JVM がメモリ不足になることが防止されます。

Debug Log 機能を使用すると、デバッグ情報をすべてファイルに保存できます。詳細については、2-15 ページの「デバッグ ログの使用」を参照してください。

注意： [Debug] ウィンドウまたはログ ファイルを使用すると、XML からバイナリへの変換に要する時間が長くなります。

サイズ変更バーの使用法

[Binary]、[XML]、[Debug] ウィンドウの間にあるサイズ変更バーを使用して、[Format Tester] の任意のウィンドウのサイズを変更できます。ウィンドウのサイズを変更するには、サイズ変更バーを選択して該当する方向（上方向または下方向、あるいは左方向または右方向）にドラッグし、いずれかのウィンドウを拡大し、もう一方のウィンドウを縮小します。

各サイズ変更バーには、方向を示す 2 つのボタンがあります。該当するボタンをクリックして、3 つのウィンドウのいずれかを表示 / 非表示にします。

メニュー バーの使い方

Format Tester の機能には、メイン ウィンドウ上部のメニュー バーにある 5 つのメニューからアクセスできます。

図 2-2　メニュー バー

Format Tester のメニューは、次のどちらかの方法で拡大できます。

- メニュー バーでメニュー名をクリックする。
- キーボードで、〔Alt〕+ 〔*key*〕 を押す。ここで、〔*key*〕 は、メニュー名の中の下線の付いた文字。

コマンドを実行するには、メニューから選択します。一部のコマンドは、メニューに示されているキーボード ショートカットを使用して実行することもできます（たとえば、〔Ctrl〕+ 〔*key*〕 シーケンス）。以下の各節では、各メニューで使用できるコマンドについて説明します。

[File] メニュー

[File] メニューには次のコマンドがあります。

表 2-1 [File] メニュー コマンド

コマンド	説明
[Open Binary]	[Open] ダイアログ ボックスを表示する。このダイアログ ボックスで、[Binary] ウィンドウのファイルを選択して開くことができる。 **注意：** バイナリ ファイルのデフォルトのファイル拡張子は、.DATA です。
[Open XML]	[Open] ダイアログ ボックスを表示する。このダイアログ ボックスで、[XML] ウィンドウのファイルを選択して開くことができる。 **注意：** XML ファイルのデフォルトのファイル拡張子は、.XML です。
[Save Binary]	[Save] ダイアログ ボックスを表示する。このダイアログ ボックスで、[Binary] ウィンドウのコンテンツを保存できる。
[Save XML]	[Save] ダイアログ ボックスを表示する。このダイアログ ボックスで、[XML] ウィンドウのコンテンツを保存できる。
[Debug Log]	[Save] ダイアログ ボックスを表示する。このダイアログ ボックスで、デバッグ情報をテキスト ファイルに保存できる。
[Close]	[Format Tester] ウィンドウを閉じる。

[Edit] メニュー

[Edit] メニューには次のコマンドがあります。

表 2-2 [Edit] メニューのコマンド

コマンド	説明
[Cut]	現在選択されているテキストを切り取り、クリップボードに格納して別の場所へ貼り付けることができる。
[Copy]	現在選択されているテキストをコピーし、クリップボードに格納して別の場所へ貼り付けることができる。

表 2-2 [Edit] メニューのコマンド （続き）

コマンド	説明
[Paste]	切り取ったテキストやコピーしたテキストをカーソル位置に挿入する。
[Find]	16 進値またはテキスト値を検索できる。このコマンドは、[Binary] ウィンドウのコンテンツにのみ適用される。 **注意：** テキスト検索には大文字 / 小文字の区別があります。
[Find Next]	現在のカーソル位置から最新の検索項目の検索を繰り返す。このコマンドは、[Binary] ウィンドウのコンテンツにのみ適用される。
[Go To]	[Binary] ウィンドウの指定のバイト オフセットにカーソルを移動する。

[Display] メニュー

[Display] メニューには次のコマンドがあります。

表 2-3 [Display] メニューのコマンド

コマンド	説明
[XML] チェックボックス	チェックすると、[XML] ウィンドウが表示され、チェックを外すと非表示になる。チェックを外すと、[Binary] ウィンドウが拡大され、[Format Tester] ダイアログ ボックス全体を占める。
[Debug] チェックボックス	チェックすると、[Debug] ウィンドウが表示され、チェックを外すと非表示になる。
[Clear ｜ Binary]	[Binary] ウィンドウのコンテンツをクリアする。
[Clear ｜ XML]	[XML] ウィンドウのコンテンツをクリアする。
[Clear ｜ Debug]	[Debug] ウィンドウのコンテンツをクリアする。
[Hex ｜ Offsets as Hexadecimal] オプション ボタン	オフセット値を 16 進数で表示する。[Hex ｜ Offsets as Decimal] オプションと同時に選択することはできない。

表 2-3 [Display] メニューのコマンド （続き）

コマンド	説明
[Hex ｜ Offsets as Decimal] オプション ボタン	オフセット値を 10 進数で表示する。[Hex ｜ Offsets as Hexadecimal] オプションと同時に選択することはできない。
[Text ｜ Values in ASCII] オプション ボタン	バイナリ ファイル エディタで表示されるテキストに使用する文字セットを ASCII に変更する。[Text ｜ Values in EBCDIC] オプションと同時に選択することはできない。
[Text ｜ Values in EBCDIC] オプション ボタン	バイナリ ファイル エディタで表示されるテキストに使用する文字セットを EBCDIC に変更する。[Text ｜ Values in ASCII] オプションと同時に選択することはできない。

[Generate] メニュー

[Generate] メニューには次のコマンドがあります。

表 2-4 [Generate] メニューのコマンド

コマンド	説明
[Binary]	MFL ドキュメントの指定に適合するバイナリ データを生成する。
[XML]	MFL ドキュメントの指定に適合する XML データを生成する。
[Prompt while generating data] チェックボックス	チェックすると、生成プロセスの実行中、次を指定することを求められる。 ■ 出力に省略可能なフィールドまたはグループを含むかどうか。 ■ 出力にどの「子の選択」を含むか。 ■ 出力に繰り返しフィールドまたは繰り返しグループを含む回数。

[Translate] メニュー

[Translate] メニューには次のコマンドがあります。

表 2-5 [Translate] メニューのコマンド

コマンド	説明
[Binary to XML]	MFL ドキュメントの指定に基づいて、[Binary] ウィンドウのコンテンツを XML に変換する。[XML] ウィンドウに XML 出力が表示される。
[XML to Binary]	MFL ドキュメントの指定に基づいて、[XML] ウィンドウのコンテンツをバイナリに変換する。[Binary] ウィンドウにバイナリ出力が表示される。

ショートカット メニューの使用法

[Binary]、[XML]、または [Debug] ウィンドウを右クリックすると、そのウィンドウで最も頻繁に使用されるコマンドのメニューが表示されます。次の表では、ショートカット メニューから使用できるコマンドについて説明します。

表 2-6 [Binary]、[XML]、[Debug] のショートカット メニュー コマンド

コマンド	説明
[Cut]	現在選択されているテキストを切り取り、クリップボードに格納して別の場所へ貼り付ける。
[Copy]	現在選択されているテキストをコピーし、クリップボードに格納して別の場所へ貼り付ける。
[Paste]	切り取ったテキストやコピーしたテキストをカーソル位置に挿入する。
[Clear]	[Binary]、[XML]、または [Debug] ウィンドウのコンテンツをクリアする。
[Generate]	MFL ドキュメントの指定に適合する バイナリ データまたは XML データを生成する。このコマンドは、[Binary] および [XML] ウィンドウのショート カット メニューでのみ使用できる。

表 2-6 [Binary]、[XML]、[Debug] のショートカット メニュー コマンド （続

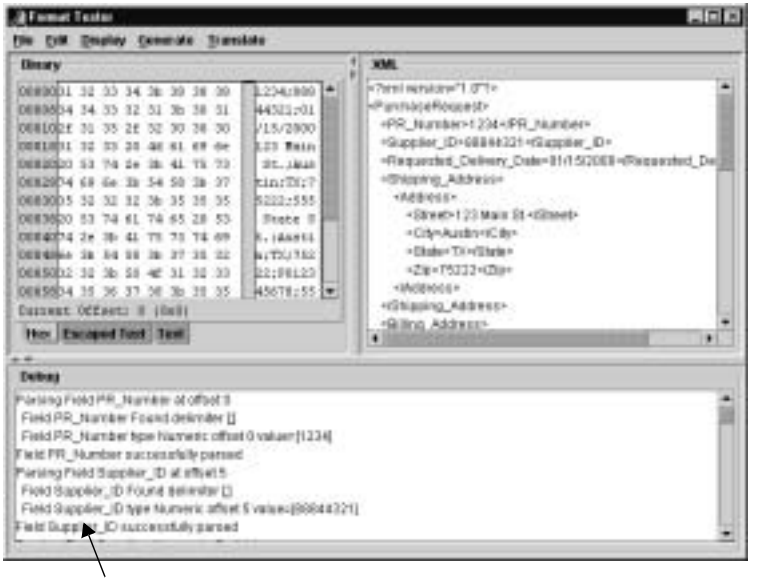
コマンド	説明
[To XML]	[Binary] ウィンドウのコンテンツを XML に変換する。このコマンドは、[Binary] ウィンドウのショート カット メニューでのみ使用できる。
[To Binary]	[XML] ウィンドウのコンテンツをバイナリに変換する。このコマンドは、[XML] ウィンドウのショート カット メニューでのみ使用できる。
[Text in ASCII]	[Hex] タブのテキスト領域で表示されるテキストに使用する文字セットを ASCII に変更する。
[Text in EBCDIC]	[Hex] タブのテキスト領域で表示されるテキストに使用する文字セットを EBCDIC に変更する。

フォーマット定義のテスト

メッセージ フォーマット定義をテストする手順は次のとおりです。

1. Format Builder を呼び出します。
2. Message Format ファイルを開きます。
3. Format Tester を呼び出します。
4. [File ｜ Open Binary] または [File ｜ Open XML] を選択して、変換して表示するファイルをロードするか、2 つのデータ ウィンドウのどちらかにデータを入力します。
5. 変換処理中に発生する操作を表示するには、[Display ｜ Debug] を選択します。この手順は省略可能です。デバッグ情報を後で表示できるようにする場合は、[Debug] ウィンドウを開いてから、変換処理を開始する必要があります。
6. データを希望のフォーマットに変換するには、[Translate ｜ Binary to XML]、または [Translate ｜ XML to Binary] を選択します。変換されたデータは [Binary] ウィンドウまたは [XML] ウィンドウに表示されます。

図 2-3 Format Tester



7. エラーが発生したら、修正して、変換のテストを再度実行します。
8. 変換が正常に終了するまで、手順 6 および 7 を繰り返します。

注意： Format Builder でメッセージ フォーマット ドキュメントを修正する間は Format Tester を開いたままにしておくことができます。ドキュメントに加えられた変更は、Format Tester により自動的に検出されます。

フォーマット定義のデバッグ

次のトピックでは、Format Tester の 3 つの機能を使用して、データをデバッグして修正する方法を説明します。

- 値の検索
- オフセットへの移動
- デバッグ ログの使用

値の検索

[Find] コマンドを使用すると、バイナリ データ内の 16 進値やテキスト値を検索できます。

16 進値またはテキスト値を検索する手順は次のとおりです。

1. [Format Tester] ダイアログ ボックスで [File ｜ Open Binary] を選択して検索対象のバイナリ データ ファイルを開きます。
2. [Edit ｜ Find] を選択し、[Find] ダイアログ ボックスを開きます。_

図 2-4 [Find] ダイアログ ボックス

3. 検索対象を [Value] フィールドに入力します。
4. [Text] または [Hex] オプション ボタンを選択して、値のタイプを指定します。
5. [Forwards] または [Backwards] オプション ボタンを選択して、検索方向を指定します。

6. [Beginning of File]、[Current Position]、または [End of File] オプション ボタンを選択して、検索の開始位置を指定します。
7. [OK] をクリックして [Find] ダイアログ ボックスを終了し、指定した検索を実行します。

   値が検出されたら、カーソルが値の位置に移動します。値が見つからない場合は、次のメッセージが表示されます。The specified search string was not found.
8. 現在のカーソル位置から検索を繰り返すには、[Edit ｜ Find Next] を選択します。

オフセットへの移動

[Go To] コマンドを使用すると、指定した 16 進アドレスまたは 10 進アドレス（オフセット）にカーソルを移動できます。

指定したオフセットに移動する手順は次のとおりです。

1. [Format Tester] ダイアログ ボックスで、[Edit ｜ Go To] を選択して、[Go To] ダイアログ ボックスを表示します。

図 2-5　[Go To] ダイアログ ボックス

2. [Offset] フィールドに対象となるオフセットを入力します。
3. [Dec] または [Hex] オプション ボタンを選択して、オフセットのタイプを指定します。
4. [Forwards] または [Backwards] オプション ボタンを選択して、方向を指定します。
5. [Beginning of File]、[Current Position]、または [End of File] オプション ボタンを選択して、開始位置を指定します。

6. [OK] をクリックして、ダイアログ ボックスを終了し、指定したオフセットにカーソルを移動します。

デバッグ ログの使用

デフォルトではデバッグ情報は保存されませんが、[Format Tester] ダイアログ ボックスを使用して、デバッグ ログ ファイルを指定できます。デバッグ ログ ファイルを指定すると、テスト中に生成されたすべてのデバッグ情報が指定したファイルに追加されます。

デバッグ ログ ファイルを指定する手順は次のとおりです。

1. [Format Tester] ダイアログ ボックスで、[File | Debug Log] を選択して、[Save] ダイアログ ボックスを表示します。

   注意： [File] メニューの [Debug Log] チェックボックスをチェックすると、チェックボックスが切り替わります。チェックボックスがチェックされている場合は、[File | Debug Log] を選択するとファイルへのログ記録がオフになります。

2. 対象となるディレクトリを選択して、次のいずれかを実行します。
   - 新しいログ ファイルを作成するには、[File name] フィールドに名前を入力して [Save] をクリックします。
   - 既存のログ ファイルを使用するには、ファイルを選択して [Save] をクリックします。

   既存ファイルを選択した場合は、新しいデバッグ情報は、既存のファイルの最後に追加されます。

3　フォーマット定義のビルド

フォーマット定義はバイナリ データの解析や作成に使用されるメタデータです。XML との間で相互に変換されるバイナリ データのフォーマット定義をビルドできます。

この項では、Format Builder を使用したフォーマット定義のビルドについて説明します。Format Builder は、データ統合のための、WebLogic Integration の設計時コンポーネントです。この章の内容は以下のとおりです。

- データ フォーマットについて
- 変換するデータの分析
- Format Builder の使用法

データ フォーマットについて

次のフォーマットとドキュメント タイプについて理解しておくと、Format Builder の使用法を把握しやすくなります。

- バイナリ（非 XML）データ
- XML ドキュメント
- DTD と XML スキーマ
- MFL ドキュメント

バイナリ（非 XML）データ

コンピュータは 2 進法に基づくため、アプリケーションでは、多くの場合、バイナリ フォーマットを使用してデータを表現します。バイナリ フォーマットで格納されるファイルは、コンピュータには読めますが、ユーザにとっては、必ずし

も読むことできるデータではありません。バイナリ フォーマットは実行可能プログラムと数値データに使用され、テキスト フォーマットはテキストデータに使用されます。多くのファイルでは、バイナリとテキストのフォーマットが組み合わされます。通常、そのようなファイルは、テキスト フォーマットのデータが一部含まれていても、バイナリ ファイルと考えられます。

バイナリ データは、XML と異なり、自己記述型ではありません。言い換えると、バイナリ データには、データのグループ分け、フィールドへの分割、またはその他の形での配置に関する記述はありません。バイナリ データは、バイトのシーケンスを生成するアプリケーションの目的に応じて、整数、文字列、または図として解釈できるバイトのシーケンスです。

たとえば、次のバイナリ データ文字列があるとします。

```
2231987
```

この文字列についてはさまざまな解釈ができます。たとえば、

- 日付として ー 2/23/1987
- 電話番号として ー 223-1987

このデータ文字列の目的をはっきりと理解しなければ、アプリケーションは文字列を正しく解釈できません。

バイナリ データがアプリケーションによって理解されるようにするには、アプリケーション自体にデータのレイアウトを埋め込むことが必要です。バイナリ ファイルに含まれる文字データのコード化に使用される文字セットもさまざまです。たとえば、IBM メインフレーム上の文字データは、通常、EBCDIC 文字セットを使用してコード化されるのに対し、デスクトップ コンピュータ上のデータは ASCII または Unicode のいずれかでコード化されます。

Format Builder を使用すると、バイナリ データのレイアウトを記述する MFL (Message Format Language : メッセージ フォーマット言語) ファイルを作成できます。MFL は、データの各フィールドの他、フィールドのグループ分け（グループ）、反復および集約を記述する要素を含む XML 言語です。バイナリ レコードの階層、フィールドのレイアウト、そしてフィールドとグループのグループ分けは MFL ドキュメントで表現されます。この MFL ドキュメントは、実行時に XML ドキュメントとの間でデータを変換するために使用されます。

コード リスト 3-1 バイナリ データのサンプル

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

XML ドキュメント

XML (eXtensible Markup Language) は、急速に、Web 上での構造化ドキュメントおよび構造化データの汎用フォーマットになりつつあります。バイナリ データと異なり、XML は自己記述型です。つまり、XML は、データ ブロックの開始と終了を示すタグ（'<' および '>' で囲まれた語）を利用します。これらのタグは、互いに関連する各データ コンポーネントの階層を定義します。これらのコンポーネントは、構造化ドキュメントの*各要素*に当たります。

XML にはこのような特性があるため、プラットフォームに依存しないで、データを表示し構造化するのに適しています。XML は、構造をはっきりと示すことができるため、XML を使用するとアプリケーション間でのデータ交換が簡単になります。データは標準形式で表示されるため、異なるシステムにあるアプリケーションでも XML 解析ツールでデータを解釈できるので、を専用のバイナリ フォーマットを使用して解釈する必要がありません。

コード リスト 3-2 に、XML ドキュメントのサンプルを示します。

コード リスト 3-2 XML ドキュメントのサンプル

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

```
</Shipping_Address>
</PurchaseRequest>
```

DTD と XML スキーマ

最初の XML 勧告では、XML ドキュメント インスタンス で許可される要素、属性、およびデータ型を記述する方法は XML DTD (XML Document Type Definition : XML 文書型定義) だけでした。その後、コンテンツ モデルを記述するより柔軟で便利な方法が求められていることが明らかになり、XML スキーマ定義言語についての検討作業が開始されました。この XML スキーマ定義言語は、2001 年 5 月の最終勧告により使用できるようになりました。

XML ドキュメントは、関連付けられた DTD または XML スキーマで記述されたコンテンツ モデルに準拠する場合に、*有効* とされます。XML ドキュメントの検証に XML パーサが必要とするメタデータは、DTD または XML スキーマのいずれかで伝達することができますが、XML スキーマ定義のほうが DTD と比較してより具体的です。すなわち、DTD よりも XML スキーマのほうが、コンテンツをより精密に制御できます。

コード リスト 3-3 に、DTD のサンプルを示します。

コード リスト 3-3 DTD のサンプル

```
<!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">
```

コード リスト 3-4 に、XML スキーマによる定義のサンプルを示します。

コード リスト 3-4 XML スキーマのサンプル

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

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

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

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

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

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

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

```
</xsd:complexType>
</xsd:element>

</xsd:schema>
```

MFL ドキュメント

MFL ドキュメント（単に、メッセージ フォーマット ドキュメントともいう）は、バイナリ データのレイアウトを記述するための特別な XML ドキュメントです。MFL ドキュメントは、データの各フィールドの他、フィールド（グループ）のグループ分け、反復および集約を記述する要素と属性を含む、`mfl.dtd` に準拠します。Format Builder を使用して、バイナリ レコードの階層、フィールドのレイアウト、フィールドとグループのグループ分けを定義すると、この情報は、MFL ドキュメントとして保存され、実行時変換に使用できます。MFL ドキュメントに取り込まれている情報を使用して、DTD または XML スキーマを生成できます。これらは、MFL ドキュメントにより生成される出力のコンテンツ モデルを記述します。

メッセージ フォーマット ドキュメントのトップレベル要素は、`MessageFormat` 要素で、メッセージ フォーマット名とバージョンを定義します。たとえば、WebLogic Integration と共にインストールされるサンプル `po.mfl` ドキュメントの ルート要素は、次のように記述されます。

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

WebLogic Integration は、現在、Message Format Language バージョン 2.02 をサポートしています。このバージョンでは、埋め込み、切り捨て、および削除に関する新機能をサポートします。Message Format Language バージョン 2.01 も引き続きサポートされます。

メッセージ フォーマット ドキュメントに割り当てられる名前は、MFL ドキュメントに基づいて生成される XML インスタンスのルート要素になります。たとえば、サンプル `po.mfl` ドキュメントを基に生成された XML ドキュメントでは、次が ルート要素になります。

```
<PurchaseRequest>
```

MFL ドキュメントには、次について定義する要素および属性もあります。

- フィールドとフィールドのフォーマット－フィールドとは、アプリケーションのコンテキスト内で意味を持つバイト シーケンスです。このバイト シーケンスは、フィールドのフォーマットを定義します（たとえば、フィールド EMPNAME には、従業員名が入力される）。定義可能なフォーマット パラメータは次のとおりです。
  - Tagged －リテラルがデータ フィールドの前にあることを示し、フィールドの先頭を表します。
  - Length －数値がデータ フィールドの前にあることを示し、そのフィールドの長さを表します。
  - Occurrence －メッセージ フォーマットでフィールドが表示される回数を示します。フィールドを表示する回数を設定したり、区切り記号を定義して繰り返しフィールドの最後を指定したりできます。
  - Optional －そのフィールドが、指定されたメッセージ フォーマットに含まれない場合もあることを示します。
  - Code Page －フィールドのデータに対して使用される文字エンコーディングの種類を示します。
- グループとグループのフォーマット－グループとは、なんらかの関連があるフィールド、コメントおよびその他のグループまたは参照の集合（たとえば、フィールド PAYDATE、HOURS、および RATE は PAYINFO グループに属する）です。定義可能なグループ パラメータは、次のとおりです。
  - Tagged －グループの他のコンテンツの前にリテラルが付加されているということで、このリテラルは他のグループまたはフィールドでも付加することができます。
  - Occurrence －メッセージ フォーマット内でグループを繰り返す回数を指定するか、または繰り返しグループの最後をマークする区切り記号を指定します。区切り記号の詳細については、3-24 ページの「区切り記号の指定」を参照してください。
  - Choice of Children － グループ内の 1 つの項目だけがメッセージ フォーマットに現れることを意味します。
  - Optional － この構造内のデータが、指定されたメッセージ フォーマットに含まれない場合もあることを示します。
- 参照と参照のフォーマット － 参照は、フィールドまたはグループ フォーマットの別のインスタンスがデータ内に存在することを示します。参照フィールドや参照グループのフォーマットは、元のフィールドまたはグループのものと同じですが、参照フィールドや参照グループの Optional 設定と

Occurrence 設定は変更できます。たとえば、「*請求先*」住所と「*届け先*」住所がデータに含まれ、両方の住所で同じフォーマットを使用する場合、住所フォーマットは、1 度だけ作成して、そのフォーマットを参照することができます。すなわち、「*請求先*」住所の住所定義を作成し、「*届け先*」住所ではその定義を参照できます。

- コメント ー メッセージ フォーマットに関する追加情報を含むメモです。

変換するデータの分析

メッセージ フォーマットを作成する前に、バイナリ データのレイアウトを理解する必要があります。従来のシステムの発注書についてのサンプル データ、および発注書レコードに関する MFL および XML ドキュメントが WebLogic Integration と共にインストールされます。Purchase Order サンプルでは、WebLogic Integration によってデータが 1 つのフォーマットから別のフォーマットに変換される方法を示します。このサンプル データの詳細については、付録 C「Purchase Order サンプルの実行」を参照してください。

XML との間でバイナリ データを変換するには、バイナリ データの正確な記述を作成することが必要です。バイナリ データ（自己記述型でないデータ）の場合、以下の要素を指定する必要があります。

- 階層グループ
- Name、Optional、Repeating、Delimited などのグループ属性
- データ フィールド
- 名前、データ型、長さ / 終了、省略可能、繰り返しなどのデータ フィールド属性

Format Builder を使用して、これらの要素をデータ変換に使用するフォーマット定義に組み込みます。

Format Builder の使用法

Format Builder を使うとバイナリ データに関するフォーマット記述を作成して、MFL ドキュメントに格納できます。記述には、データの詳細な解析から得られた階層情報と構造情報を含む必要があります。これらのフォーマット記述は MFL ドキュメントに格納されます。Format Builder を使用すると、データにフォーマット記述を適用する前にテストすることもできます。

Format Builder の呼び出し

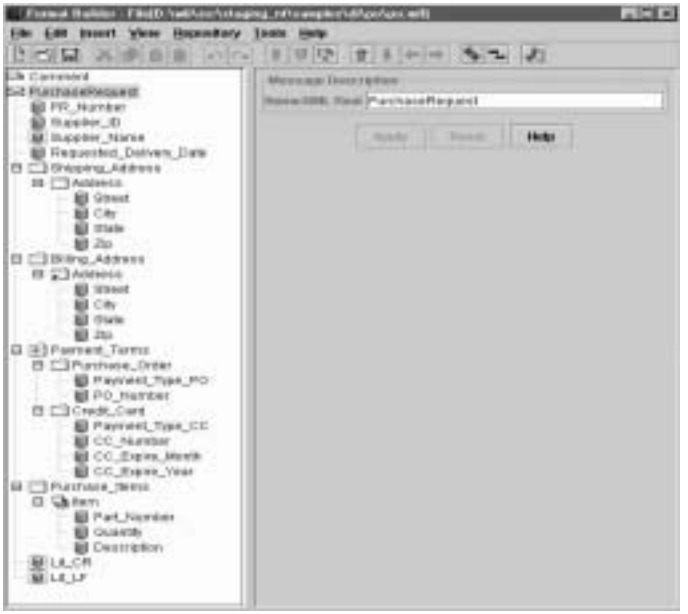
Format Builder を呼び出すには、[スタート | プログラム | BEA WebLogic Platform 7．0 | WebLogic Integration 7．0 | Format Builder] を選択します。Format Builder のウィンドウが表示されます。

Format Builder のウィンドウの使用法

[Format Builder] ウィンドウは縦方向の 2 つのペインに分かれています。左ペイン（ナビゲーション ツリー）には、アクティブな MFL ドキュメントで定義されているグループとフィールドの構造的な関係が表示されます。右ペインには各項目を定義するプロパティが表示されます。

編集中のファイルの情報は [Format Builder] ウィンドウのタイトル バーに表示されます。

図 3-1　[Format Builder] ウィンドウ



バイナリ データの構造は対象データに対応するフィールドとグループの組み合わせを使用してナビゲーション ツリーで定義されます。

以下のトピックでは、[Format Builder] ウィンドウのさまざまなツールを使用して、ウィンドウ内を移動しコマンドを実行する方法を説明します。

- ナビゲーション ツリーの使い方
- Format Builder のメニュー バーの使い方
- ツールバーの使い方
- ドラッグ アンド ドロップの使い方
- ショートカット メニューの使い方

ナビゲーション ツリーの使い方

ナビゲーション ツリーは、バイナリ データの構造を階層レイアウトで表示します。ナビゲーション ツリーのルート ノード（*Message node*）は作成または編集される MFL ドキュメントに対応します。子ノードには、グループ名またはフィールド名が付けられます。フィールドはナビゲーション ツリーでリーフ ノードによって表されます。グループにはフィールドまたはその他のグループが含まれ、ナビゲーション ツリーでは非リーフ ノードで表されます。

各ノードのアイコンには、そのノードに関する情報がカプセル化されています。このアイコンを見ると、ノードがメッセージ、グループ、フィールド、コメント、または参照のいずれなのか、グループやフィールドが繰り返されるのか、グループが*子の選択*なのかどうか、グループやフィールドが省略可能なのかどうかがわかります。

ナビゲーション ツリーでは、メニューやツールバーを使用して、ノードの追加、削除、移動、コピー、または名前の変更を行うことができます（詳細については、3-13 ページの「Format Builder のメニュー バーの使い方」および 3-13 ページの「ツールバーの使い方」を参照）。

次の表では、ナビゲーション ツリーに表示される各アイコンについて説明します。

表 3-1 ナビゲーション ツリーのアイコン

ツリー アイコン	アイコン名	説明
	メッセージ フォーマット	トップ レベルの要素。
	グループ	なんらかの関連があるフィールド、コメントおよびその他のグループまたは参照の集合（たとえば、フィールド PAYDATE、HOURS、および RATE は、PAYINFO グループに組み込まれる）。そのグループのすべての項目のフォーマットを定義する。
	グループ（省略可能）	メッセージ フォーマットに任意で含めることができるグループ。

表 3-1 ナビゲーション ツリーのアイコン （続き）

ツリー アイコン	アイコン名	説明
	Repeating Group	1 回または複数回含まれるグループ。
	Optional Repeating Group	任意で使用できるが、使用する場合はオカレンスが複数回であるグループ。
	Group Reference	データ内に、該当するグループの別のインスタンスが存在することを示す。参照グループのフォーマットは元のグループと同じだが、参照グループの Optional 設定と Occurrence 設定は変更できる。
	Group Choice	当該グループ内の 1 項目のみを、メッセージ フォーマットに取り込むことを示す。
	Field	アプリケーションのコンテキストで意味を持つバイトのシーケンス。このバイト シーケンスがフィールドのフォーマットを定義する（たとえば、フィールド `EMPNAME` には、従業員名が入力される）。
	Optional Field	該当するメッセージ フォーマットに任意で使用できるフィールド。
	Repeating Field	1 回または複数回含まれるフィールド。
	Optional Repeating Field	該当するメッセージ フォーマットに任意で使用できるが、使用する場合はオカレンスが複数回であるフィールド。
	Field Reference	データ内に、該当するフィールドの別のインスタンスが存在することを示す。参照フィールドはのフォーマットは、元のフィールドと同じだが、参照フィールドの Optional 設定と Occurrence 設定は変更できる。
	Comment	メッセージ フォーマット、またはメッセージ フォーマットによって変換されたデータについての注意が入力されている。

表 3-1 ナビゲーション ツリーのアイコン （続き）

ツリー アイコン	アイコン名	説明
⊟	Collapse	項目の横の負符号は、指定したその項目を折りたためることを示す。
⊞	Expand	項目の横の正符号は、指定したその項目を展開し、子項目を表示できることを示す。

Format Builder のメニュー バーの使い方

メニュー バーを使用すると、Format Builder の機能に迅速にアクセスできます。

図 3-2 Format Builder のメニュー バー

使用できるメニュー項目は、ユーザがその時点までに実行したアクションとナビゲーション ツリーで現在選択しているノードによって異なります。使用できないメニュー項目は、メニュー内ではグレー表示されます。

メニューは、次のどちらかの方法で表示できます。

- メニュー バーでメニュー名をクリックする。
- キーボードで、〔Alt〕+ 〔*key*〕を押す。ここで、〔*key*〕は、メニュー名の中の下線の付いた文字です。

コマンドを実行するには、メニューから選択します。一部のコマンドは、メニューに示されているキーボード ショートカットを使用して実行することもできます（たとえば、〔Ctrl〕+ 〔*key*〕シーケンス）。3-45 ページの「Format Builder のメニュー」では、各メニューで使用できるコマンドについて説明します。

ツールバーの使い方

また、よく使用されるコマンドには、ツールバーの各アイコンを使用してアクセスすることもできます。

図 3-3　Format Builder のツールバー

コマンドを実行するには、ツールバーの該当するアイコンをクリックします。使用できないコマンドのアイコンは、グレー表示されます。

次の表では、Format Builder のツールバーの各アイコンについて説明します。

表 3-2　Format Builder のツールバーのアイコン

ツールバーのアイコン	名前	説明
	[New]	新しいメッセージ フォーマットを作成する。
	[Open]	既存のメッセージ フォーマットを開く。
	[Save]	現在のメッセージ フォーマットを保存する。
	[Cut]	左ペインで現在選択されている項目とその子オブジェクトをナビゲーション ツリーから切り取る。切り取った項目はナビゲーション ツリーの他の場所に貼り付けることができる。 **注意：** メッセージ フォーマット（ルート）項目が選択されている場合、このコマンドは使用できません。
	[Copy]	左のペインで現在選択されている項目をナビゲーション ツリーの他の場所に挿入するために、コピーを作成する。 **注意：** メッセージ フォーマット（ルート）項目が選択されている場合、このコマンドは使用できません。

表 3-2 Format Builder のツールバーのアイコン （続き）

ツールバーのアイコン	名前	説明
	[Paste as Sibling]	切り取った、またはコピーした項目を、選択した項目の兄弟オブジェクトとして挿入する。
	[Paste as Reference]	切り取った、またはコピーした項目の参照を、選択した項目の兄弟オブジェクトとして挿入する。
	[Undo]	直前の操作を取り消す。ツール チップが取り消し可能な操作を示す。たとえば、フィールド名を「Address」に変更して [Apply] をクリックすると、ツールチップには「Undo Apply Field Address」と表示される。 Format Builder では、以前の操作を複数回にわたって取り消すことができる。
	[Redo]	[Undo] コマンドの結果を取り消す。ツール チップがやり直し可能な操作を示す。たとえば、フィールド名を「Address」に変更した後、その変更を取り消すと、[Redo] ツールチップには「Redo Apply Field Address」と表示される。 Format Builder では、以前の操作を複数回にわたってやり直しできる。
	[Insert Field]	フィールドを、ナビゲーション ツリーで選択した項目の兄弟として挿入する。
	[Insert Group]	グループを、ナビゲーション ツリーで選択した項目の兄弟として挿入する。
	[Insert Comment]	コメントを、ナビゲーション ツリーで選択した項目の兄弟として挿入する。
	[Move Up]	選択した項目を、その親の中でポジションを 1 つ上に移動する。

表 3-2 Format Builder のツールバーのアイコン （続き）

ツールバーのアイコン	名前	説明
	[Move Down]	選択した項目を、その親の中でポジションを 1 つ下に移動する。
	[Promote item]	選択した項目を、ナビゲーション ツリー内のすぐ上のレベルに割り当てる。たとえば、「Field1」が「Group1」の子オブジェクトである場合、「Field1」を選択して [Promote] ツールをクリックすると、「Field1」が「Group1」の兄弟になる。
	[Demote item]	選択した項目を、ナビゲーション ツリー内のすぐ下のレベルに割り当てる。たとえば、「Group1」と「Field1」が兄弟で、「Field1」がナビゲーション ツリー内で「Group1」のすぐ後に表示されている場合、「Field1」を選択して [Demote] ツールをクリックすると、「Field1」は「Group1」の子になる。
	[Expand All]	ナビゲーション ツリー内のすべての項目を展開し、子項目を表示する。
	[Collapse All]	ナビゲーション ツリーを折りたたんで、最初のレベルの項目のみを表示する。
	[Format Tester]	[Format Tester] ウィンドウを開く。

ショートカット メニューの使い方

ナビゲーション ツリーの項目を右クリックすると、その項目で最も頻繁に使用されるコマンドのメニューが表示されます。次の表では、ショートカット メニューから使用できるコマンドについて説明します。

注意： コマンドが使用できるかどうかは、ユーザが選択している項目およびその時点までに実行したアクションにより異なります。

コマンド	説明
[Cut]	左ペインで現在選択されている項目とその子オブジェクトをナビゲーション ツリーから切り取る。
[Copy]	左のペインで現在選択されている項目をナビゲーション ツリーの他の場所に挿入するために、コピーを作成する。
[Paste]	切り取った項目やコピーした項目を挿入する。[Paste] を選択すると、追加メニューが表示される。項目は、選択した項目の子または兄弟として貼り付けることができる。さらに、切り取った項目やコピーした項目の参照を、選択した項目の兄弟として貼り付けることもできる。
[Insert Group]	新しいグループを、ユーザの指定により、選択した項目の子または兄弟として挿入する。
[Insert Field]	新しいフィールドを、ユーザの指定により、選択した項目の子または兄弟として挿入する。
[Insert Comment]	コメントを、ユーザの指定により、選択した項目の子または兄弟として挿入する。
[Duplicate]	現在選択している項目のコピーを作成し、兄弟として貼り付ける。複製した項目は、複製元の項目と同じ値および子オブジェクトを持つ。複製された項目の名前は元の項目と同じだが、元の名前の前に「*New*」が追加される。たとえば、「MyGroup1」を複製すると、複製されたグループの名前は「NewMyGroup1」となる。
[Delete]	選択した項目を削除する。

ドラッグ アンド ドロップの使い方

ドラッグ アンド ドロップ機能を使って、ナビゲーションツリー内で項目のコピーや貼り付け、あるいは移動を行うことができます。

注意： ドラッグ アンド ドロップでは、コピーまたは移動の対象となるノードは、常に選択されたノードの兄弟として挿入されます。ノードをメッセージ フォーマット ノードにドラッグ アンド ドロップすると、そのノードは最後の子として挿入されます。

項目を移動する手順は次のとおりです。

1. 移動したい項目を選択します。
2. マウスの左ボタンを押したままで項目を目的のノードにドラッグします。
3. 項目を適切な場所まで移動したら、マウスの左ボタンを放します。項目が目的の場所に移動します。

項目をコピーして貼り付ける手順は次のとおりです。

1. コピーしたい項目を選択します。
2. 〔Ctrl〕を押したままにします。
3. 〔Ctrl〕を押したままでマウスの左ボタンを押し、そのまま選択した項目を目的の場所までドラッグします。
4. 任意の兄弟オブジェクトを選択したら、マウスの左ボタンと〔Ctrl〕を離します。項目のコピーが新しい場所に貼り付けられます。

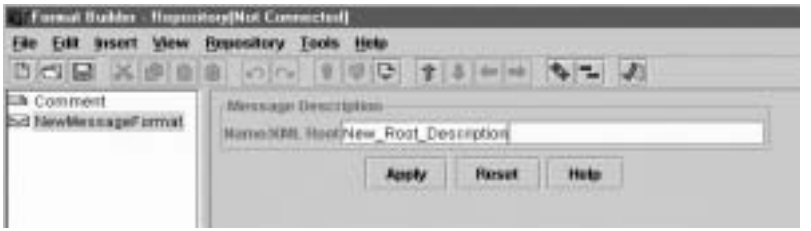
メッセージ フォーマットの作成

メッセージ フォーマット定義ファイルを作成するには、まず、メッセージ フォーマット（メッセージ フォーマット ファイルのルート ノード）を作成します。

メッセージ フォーマットを作成する手順は次のとおりです。

1. [File | New] を選択します。右ペインに、メッセージ フォーマットの詳細ウィンドウが表示されます。

図 3-4　メッセージ フォーマットの詳細ウィンドウ



2. [Name/XML Root] フィールドにメッセージ フォーマットの名前を入力します。

   注意： [Name/XML Root] フィールドのエントリは、このメッセージ フォーマット ドキュメントに基づいて生成される各 XML インスタンスのルート要素名になります。このため、このエントリは、次の「XML 要素の命名規則」で説明する規則に従う必要があります。

3. 以下のいずれか 1 つをクリックします。

   - [Apply] －メッセージ フォーマットのプロパティを更新する。
   - [Reset] －詳細ウィンドウで行った変更を廃棄して、すべてのフィールドを前回に適用した値にリセットする。
   - [Help] －メッセージ フォーマットの詳細ウィンドウについてのオンライン ヘルプ情報を表示する。

   注意： [Apply] および [Reset] オプションは、詳細ウィンドウで変更を行った後にのみ、有効になります。

XML 要素の命名規則

メッセージ フォーマット ドキュメントのルート ノード、フィールド、グループ、および参照に割り当てる名前は、メッセージ フォーマット ドキュメントに基づいて生成される XML インスタンスでは、XML 要素名に変換されます。したがってこれらの名前は、次に示す XML の命名規則に準拠させる必要があります。

- 名前の先頭は、文字またはアンダースコアでなければならない。
- 文字、数字、ピリオド記号、ハイフン記号、またはアンダースコア記号を使用できる。

以下の文字列は有効な名前の例です。

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

以下の文字列は無効な名前の例です。

- `1MyField` －先頭が数字ではいけない

- My>Field ー右向き不等号（>）は無効
- My Field ースペースは使用不可。

グループの作成

グループは、なんらかの関連があるフィールド、コメント、参照、およびその他のグループの集合です。たとえば、フィールド PAYDATE、HOURS、および RATE は、すべて PAYINFO グループを構成します。グループを、メッセージ フォーマット項目の子項目、別のグループの子、あるいはグループまたはフィールドの兄弟として作成できます。

グループを作成する手順は次のとおりです。

1. ナビゲーション ツリーから項目を選択します。
2. 以下のいずれか 1 つを選択します。
   - 選択した項目が、ルート ノードまたは他のグループの場合でしかも選択した項目の子としてグループを作成したい場合は、[Insert ｜ Group ｜ As Child] を選択します。
   - 選択した項目の兄弟としてグループを作成したい場合は、[Insert ｜ Group ｜ As Sibling] を選択します。

   右ペインに、グループの詳細ウィンドウが表示されます。

図 3-5 グループの詳細ウィンドウ

3. 次の表の説明に従ってグループのプロパティを定義します。

表 3-3 グループのプロパティ

カテゴリ	プロパティ	説明
[Group Description_]	[Name]	グループの名前。このエントリは、19 ページの「XML 要素の命名規則」で説明する規則に準拠する必要がある。
	[Optional]	省略可能なグループであれば [Optional] を選択する。
	[Choice of Children]	当該グループ内の 1 項目のみをメッセージ フォーマットに取り込む場合に [Choice of Children] を選択する。

表 3-3 グループのプロパティ （続き）

カテゴリ	プロパティ	説明
[Group Occurrence]（[Group Description] で [Optional] に定義されている場合を除いて、すべてのグループが少なくとも 1 回現れる）	[Once]	グループが 1 回だけ現れることを示すには、このオプションを選択する。
	[Repeat Delimiter]	ここで指定した区切り記号が見つかるまでグループが繰り返し現れることを示すには、このオプションを選択する。
	[Repeating Field]	繰り返しフィールドとして選択されたフィールドで指定された回数だけグループが繰り返し現れることを示すには、このオプションを選択する。
	[Repeat Number]	このオプションを選択して、ここで指定された回数だけグループが繰り返し現れることを示す。
	[Unlimited]	グループの発生回数に制限がないことを示すには、このオプションを選択する。

表 3-3 グループのプロパティ （続き）

カテゴリ	プロパティ	説明	
[Group Attributes]	[Group is Tagged]	タグ付きグループの場合はこのオプションを選択する。タグ付きとは、あるグループの他のコンテンツの前にリテラルが付加されているということで、このリテラルは他のグループまたはフィールドでも付加することができる。	
	[Group Delimiter]	グループの終了点は、*区切り記号*（フィールドのグループの最後をマークする文字列）で指定できる。グループは、区切り記号が見つかるまで続く。 **注意：** 通常、グループには区切り記号を使用しません。グループは通常、内容に基づいて解析されます（グループはすべての子オブジェクトの解析が完了した時点で終了）。区切り記号の詳細については、3-24 ページの「区切り記号の指定」を参照してください。 グループの区切り記号属性を指定するには、次のオプションから選択する。	
		[None]	区切り記号を含まないグループの場合はこのオプションを選択する。
		[Delimited]	グループの終了点を区切り記号文字列でマークする場合は、このオプションを選択して、[Value] フィールドに区切り記号の文字を入力する。
		[Delimiter Field]	グループの終了点を区切り記号文字列を含むフィールドでマークする場合は、このオプションを選択する。このオプションを選択すると、次の入力を求められる。 [Field] －区切り記号文字列を含むフィールドを選択する。ドロップダウン リストに有効なフィールドの一覧が表示される。 [Default] －データの中に選択されたフィールドが存在しない場合、使用するデフォルトの区切り記号文字を入力する。この値は必須。
		[Delimiter is Shared]	区切り記号によって、データのグループの終わりおよびグループ内での最後のフィールドの終わりがマークされることを示す場合、このオプションを選択する。この区切り記号は、当該グループ内およびグループ内の最後のフィールドによって共有され、データの終わりを示す。

4. 以下のいずれか 1 つをクリックします。
   - [Apply] ー グループのプロパティを更新する。
   - [Duplicat] ー 現在表示されているグループのコピーを作成し、兄弟として貼り付ける。

     複製したグループは、複製元のグループと同じ値および子オブジェクトを持つ。複製された項目の名前は元の項目名と同じだが、元の名前の前に「*New*」が追加される。たとえば、「MyGroup1」を複製すると、複製されたグループの名前は「NewMyGroup1」となる。
   - [Reset] ー 詳細ウィンドウで行った変更を廃棄して、すべてのフィールドを前回に適用した値にリセットする。
   - [Help] ー 詳細ウィンドウについてのオンライン ヘルプ情報を表示する。

   注意： [Apply] および [Reset] オプションは、詳細ウィンドウで変更を行った後にのみ、有効になります。

区切り記号の指定

Format Builder で正しい構文を入力して、区切り記号を指定できます。たとえば、タブ文字を区切り記号として指定する場合（'\u009'）、構成子 `\t` を入力して一致させます。

区切り記号として使用できる文字については、構成子を使用してその文字を区切り記号として指定する必要があります。次の表では、これらの文字に対応する構成子を示します。

表 3-4 文字区切り記号

構成子	区切り記号として指定する文字
`x`	x
`\\`	\ (バックラッシュ)
`\0n`	8 進値 0n (<= n <= 7) を含む文字
`\0nn`	8 進値 0nn (0<= n <= 7) を含む文字
`\0mnn`	8 進値 0mnn (0 <= m <= 3、0 <= n <= 7) を含む文字

表 3-4 文字区切り記号 （続き）

構成子	区切り記号として指定する文字
\xhh	16 進値 0xhh を含む文字
\uhhhh	16 進値 0xhhhh を含む文字
\t	タブ文字（'\u0009'）
\n	新行（行送り）文字（'\u000A'）
\r	キャリッジ リターン文字（'\u000D'）
\f	改ページ文字（'\u000C'）
\a	アラート（ベル）文字（'\u0007'）
\e	エスケープ文字（'\u001B'）
\cx	x に対応する制御文字

詳細については、次の「URL」を参照してください。

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

フィールドの作成

フィールドはアプリケーションにとってなんらかの意味を持つバイトのシーケンスです（たとえば、フィールド `EMPNAME` には従業員名が入力される）。フィールドは、メッセージ フォーマット ノードの子項目、グループの子、あるいはグループまたは別のフィールドの兄弟として作成できます。フィールド名は XML 出力で要素名として使用されるので、3-19 ページの「XML 要素の命名規則」で説明する規則に準拠する必要があります。

フィールドを作成する手順は次のとおりです。

1. 項目をナビゲーション ツリーから選択します。
2. 以下のいずれか 1 つを選択します。
   - 選択した項目の子としてフィールドを作成したい場合は、[Insert | Field | As Child] を選択します。

- 選択した項目の兄弟オブジェクトとしてフィールドを作成したい場合は、[Insert ｜ Field ｜ As Sibling] を選択します。

右ペインに、フィールドの詳細ウィンドウが表示されます。

図 3-6　フィールドの詳細ウィンドウ

3. 次の表の説明に従ってフィールドのプロパティを定義します。

表 3-5 フィールドのプロパティ

カテゴリ	プロパティ	説明
[Field Description]	[Name]	フィールドの名前。このエントリは、19 ページの「XML 要素の命名規則」で説明する規則に準拠する必要がある。
	[Optional]	省略可能なフィールドの場合はこのオプションを選択する。省略可能とは、当該フィールドのデータが必須ではないことを意味する。
	[Type]	ドロップダウン リストから当該フィールドのデータ型を選択する。デフォルトは String である。 **注意：** 選択したフィールドの型によって、表示されるフィールド データのオプションは異なります。 WebLogic Integration によってサポートされるデータ型のリストについては、付録 A「サポートされるデータ型」を参照。
[Field Occurrence]（[Field Description] で [Optional] に定義されている場合を除いて、すべてのフィールドが少なくとも 1 回現れる）	[Once]	フィールドが 1 回だけ現れることを示すには、このオプションを選択する。
	[Repeat Delimiter]	ここで指定された区切り記号が見つかるまでフィールドが繰り返し現れることを示すには、このオプションを選択する。
	[Repeating Field]	繰り返しフィールドとして選択されたフィールドで指定された回数だけフィールドが繰り返し現れることを示すには、このオプションを選択する。
	[Repeat Number]	このオプションを選択して、ここで指定された回数だけフィールドが繰り返し現れることを示す。
	[Unlimited]	フィールドの発生回数に制限がないことを示すには、このオプションを選択する。

表 3-5 フィールドのプロパティ （続き）

カテゴリ	プロパティ	説明
[Field Attributes]（表示される [Field Attributes] のプロパティは、[Field Description] で指定した [Type] によって異なる）。	［Field is Tagged］	タグ付きのフィールドの場合はこのオプションを選択する。タグ付きとは、先頭にリテラルが付加されているデータのことで、そのデータが存在することを意味する。タグ フィールドのデータ型をドロップダウン リストから選択する必要もある。たとえば、 `SUP:ACME INC` の場合、 `SUP:` はタグで、`ACME INC` がフィールド データである。 [Field is Tagged] オプションを選択した場合は、チェックボックスの右にあるフィールドにタグを入力する。
	[Field Default Value]	フィールドが XML にない場合にバイナリ データに挿入されるフィールドのデータ値を指定するには、このオプションを選択する。 フィールドがバイナリ データになく、そのフィールドが省略可能ではないときは、たとえデフォルト値が与えられてもバイナリ データは解析に失敗する。
	[Data Base TypeData Base Type]	フィールドが日付または時刻のフィールドである場合、基本型（Base Type）はデータを表示する文字（ASCII、EBCDIC、または数値）タイプを示す。
	[Year Cutoff]	データ フィールドに年号が 2 桁で入力されている場合、年号カットオフ属性により、2 桁で表された年号を 4 桁の年号に変換できる。年号を表す 2 桁の数値が年号カットオフ値と等しいかそれを上回る場合は、年号の値に「19」というプレフィックスが付加される。それ以外の場合は、「20」というプレフィックスが付加される。
	[Code Page]	フィールド データの文字エンコーディング。デフォルトのコードページを設定するには [Tools ｜ Options] を選択し、[Default Field Code Page] ドロップダウン リストからデフォルトのエンコーディングを選択する。
	[Value]	リテラル フィールドに表示される値。

表 3-5 フィールドのプロパティ （続き）

カテゴリ	プロパティ	説明	
[Field Attributes] (続き)	[Termination]	グループの区切り記号属性を指定するには、次のオプションから選択する。	
		[Length]	可変長のデータ型を固定値に設定するには、このオプションを選択する。このオプションを選択すると、次の入力を求められる。 [Length] －バイト数をフィールドに入力する。 [Trim Leading/Trailing] －データの先頭または末尾から指定されたデータを削除する。 [Pad] － XML データが指定された長さより短い場合、正しい長さになるように、指定されたデータを埋め込む。以下の埋め込みオプションからいずれか 1 つを選択する。 ■ フィールドの末尾に埋め込むには、[Trailing] を選択する。 ■ フィールドの先頭に埋め込むには、[Leading] を選択する。 [Truncate] －フィールドから指定された数の文字を削除する。次の切り捨てオプションを任意に組み合わせて選択する。 ■ フィールドの先頭から数えて指定した数の文字を削除するには、[Truncate First] を選択する。 ■ フィールドの末尾から数えて指定した数の文字を削除するには、[Truncate After] を選択する。 両方の切り捨てオプションを選択すると、まず [Truncate First] オプションが実行され、次に残りの文字に対して [Truncate After] が呼び出される。

表 3-5 フィールドのプロパティ （続き）

カテゴリ	プロパティ	説明	
[Field Attributes] (続き)	[Termination] (続き)	[Embedded Length]	可変長のデータ型の終了点を埋め込み長さで指定することを示すには、このオプションを選択する。埋め込み長さは、データ フィールドに先行して、データのバイト数を示す。このオプションを選択すると、次の入力を求められる。 ■ [Type] ー データ型、および必要な場合、終了に使用する長さまたは区切り記号を示す。 ■ [Tag/Length Order] ー [Tag] フィールドと [Length] フィールドの両方が存在する場合、その順序を指定する。デフォルトでは、[Tag] が [Length] に先行する。 ■ [Trim Leading/Trailing] ーデータの先頭または末尾から指定されたデータを削除する。 ■ [Truncate] ーフィールドから指定された数の文字を削除する。詳細については、[Length] オプションの [Truncate] オプションの説明を参照。
		区切り記号	可変長のデータ型の終了点を*区切り記号*で指定することを示すには、このオプションを選択する。区切り記号とは、フィールドの終わりをマークする値である。フィールドは区切り記号が見つかるまで続く。このオプションを選択すると、次の入力を求められる。 ■ [Value] ーフィールド データの終わりをマークする区切り記号を入力する。 ■ [Trim Leading/Trailing] ーデータの先頭または末尾から指定されたデータを削除する。 ■ [Truncate] ーフィールドから指定された数の文字を削除する。詳細については、[Length] オプションの [Truncate] オプションの説明を参照。

表 3-5 フィールドのプロパティ （続き）

カテゴリ	プロパティ	説明	
[Field Attributes] (続き)	[Termination] (続き)	[Delimiter Field]	可変長のデータ型の終了点を、区切り記号値を含むフィールドで指定するには、このオプションを選択する。このオプションを選択すると、次の入力を求められる。 ■ [Field] －区切り記号を含むフィールドを選択する。 ■ [Default] －デフォルトの区切り記号を入力する。区切り記号フィールドがない場合はこのデフォルト値を使用できる。値の入力は必須。 ■ [Trim Leading/Trailing] －データの先頭または末尾から指定されたデータを削除する。 ■ [Truncate] －フィールドから指定された数の文字を削除する。詳細については、[Length] オプションの [Truncate] の説明を参照。 区切り記号の詳細については、3-24 ページの「区切り記号の指定」を参照。
		[Decimal Position]	小数点の左側の桁数（0 ～ 16）を指定する。

4. 以下のいずれか 1 つをクリックします。
   - [Apply] －フィールドのプロパティを更新する。
   - [Duplicate] －現在表示されているフィールドのコピーを作成し、兄弟として貼り付ける。

     複製フィールドには複製元のフィールドと同じ値が含まれます。複製されたフィールドの名前は元の名前と同じですが、元の名前の前に「*New*」が追加されます。たとえば、元のフィールド名が「MyField1」の場合、複製される名前は「NewMyField1」となります。
   - [Reset] －詳細ウィンドウで行った変更を廃棄して、すべてのフィールドを前回に適用した値にリセットする。
   - [Help] －フィールドの詳細ウィンドウについてのオンライン ヘルプ情報を表示する。

注意： [Apply] および [Reset] オプションは、詳細ウィンドウで変更を行った後にのみ、有効になります。

[Padding Mandatory] フィールド

WebLogic Integration の以前のバージョンでは、実行時に必須フィールドのデータが存在しない場合は、そのフィールドでの埋め込みは実行されませんでした。WebLogic Integration 7.0 では、デフォルト値が指定されている場合、XML - バイナリ変換中に、データを含まない必須フィールドにデフォルト値を使用してデータの埋め込みが行われます。デフォルト値が指定されておらず、しかも、変換時にフィールドにデータがない場合は、エラーが発生します。

注意： 必須フィールドの埋め込みは、バイナリ - XML 変換ではサポートされません。

あるグループが複数回指定されるが、データが提供されるのは 1 回の出現に対してだけの場合、この機能は役立ちます。デフォルト値が指定されている場合、必須フィールドの埋め込み機能が起動されると、データが提供されていないすべてのグループ出現を対象にして、デフォルト値を使用した埋め込みが行われます。

コメントの作成

コメントは、メッセージ フォーマットまたはメッセージ フォーマットによって変換されるデータについてのメモです。コメントはメッセージ フォーマット定義に含まれますが、ドキュメンテーションと参照用としてのみ使用されます。コメントに番号が付けられたり、コメントが XML やバイナリ データに変換されることはありません。コメントは、任意のメッセージ フォーマット、グループ、またはフィールドの子または兄弟として作成できます。

注意： 通常、コメントは説明対象のノードの前に置くことになっています。

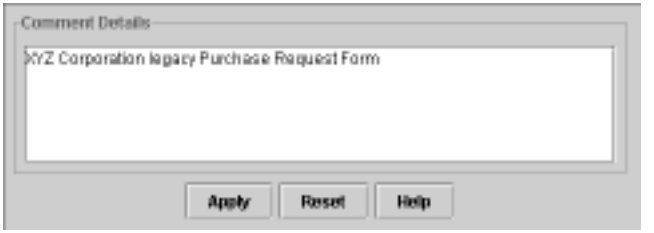
コメントを作成する手順は次のとおりです。

1. 項目をナビゲーション ツリーから選択します。
2. 以下のいずれか 1 つを選択します。
   - 選択した項目の子としてコメントを作成したい場合は、[Insert | Comment | As Child] を選択します。

- 選択した項目の兄弟としてコメントを作成したい場合は、[Insert | Comment | As Sibling] を選択します。

3. [Comment Details] フィールドにコメント テキストを入力します。

図 3-7 [Comment Detail] ウィンドウ



4. 以下のいずれか 1 つをクリックします。
   - [Apply] ーコメント テキストを更新する。
   - [Reset] ー詳細ウィンドウで行った変更を廃棄して、すべてのフィールドを前回に適用した値にリセットする。
   - [Help] ー [Comment Details] ウィンドウについてのオンライン ヘルプ情報を表示する。

   注意： [Apply] および [Reset] オプションは、詳細ウィンドウで変更を行った後にのみ、有効になります。

参照の作成

参照を使用すると、新しいコンテキストで既存のフィールドやグループのフォーマットを再利用できます。既存のフィールドまたはグループの参照を作成すると、同じフォーマットが使用されますが、参照フィールドまたは参照グループの [Optional] プロパティと [Occurrence] プロパティは変更できます。

たとえば、「*請求先*」住所と「*届け先*」住所がデータに含まれ、両方の住所で同じフォーマットを使用する場合、住所フォーマットは、1 度だけ作成して、そのフォーマットを参照することができます。すなわち、「*請求先*」住所の住所定義を作成し、「*届け先*」住所では、その定義を参照できます。

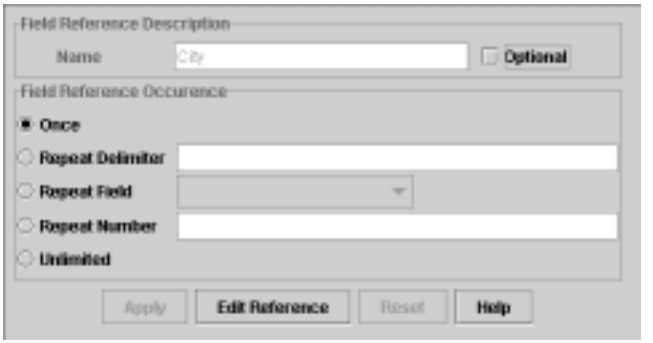
注意： 参照項目には元の項目とまったく同じ名前が与えられます。そこで、参照の対象となるフィールドやグループを作成するときは、「住所」など、汎用的な名前を使用する必要があります。たとえば、前の例では、「請求先」グループの子として「住所」グループを作成し、「届け先」グループから「住所」グループを参照できます。

参照を作成する手順は次のとおりです。

1. ナビゲーション ツリーから参照される項目を選択します。
2. [Edit ｜ Copy] を選択します。
3. 任意の位置で項目を選びます。この項目から参照を行います。次の手順で、参照として貼り付けられる項目は、選択された項目の兄弟として貼り付けられます。
4. [Edit ｜ Paste ｜ As Reference] を選択します。

   参照の詳細ウィンドウが表示されます。次の図では、例として、フィールド参照用の詳細ウィンドウを示します。

図 3-8　フィールド参照の詳細ウィンドウ



5. 次の表の説明に従って参照のプロパティを定義します。

表 3-6 参照のプロパティ

カテゴリ	プロパティ	説明
[Field Reference Description] または [Group Reference Description]	[Name]	この参照の対象となるフィールドまたはグループの名前。この値は変更できない。
	[Optional]	参照が省略可能であれば [Optional] を選択する。
[Field Reference Occurrence] または [Group Reference Occurrence] (省略可能として定義される場合を除いて、すべての参照項目は、少なくとも 1 回は発生する)	[Once]	参照項目が 1 回だけ現れることを示すには、このオプションを選択する。
	[Repeat Delimiter]	指定された区切り記号が見つかるまで参照項目が繰り返し現れることを示すには、このオプションを選択する。
	[Repeating Field]	繰り返しフィールドとして選択されたフィールドで指定された回数だけ参照項目が繰り返し現れることを示すには、このオプションを選択する。
	[Repeat Number]	このオプションを選択して、ここで指定された回数だけ参照項目が繰り返し現れることを示す。
	[Unlimited]	参照項目の発生回数に制限がないことを示すには、このオプションを選択する。

6. 以下のいずれか 1 つをクリックします。
   - [Apply] －参照のプロパティを更新する。
   - [Reset] －詳細ウィンドウで行った変更を廃棄して、すべてのフィールドを前回に適用した値にリセットする。
   - [Edit Reference] －項目を編集できるように、元の項目の詳細ウィンドウを表示する。
   - [Help] －参照の詳細ウィンドウについてのオンライン ヘルプ情報を表示する。

   注意： [Apply] および [Reset] オプションは、詳細ウィンドウで変更を行った後にのみ、有効になります。

パレットの操作

Format Builder のパレットを使用すれば、一般的に使用されるメッセージ フォーマット項目を格納して、必要に応じて、それらの項目をいつでもメッセージ フォーマット定義に挿入できます。

デフォルトのパレット（palette.xml）は、WebLogic Integration のインストール ディレクトリに保存されている MFL ドキュメントです。デフォルト パレットには、一般的なデータ フォーマット、リテラルおよび文字列があります。作成するメッセージ フォーマットでこれらの項目を使用することも、独自の項目をデフォルト パレットに追加することもできます。パレットで、独自の MFL ドキュメントを作成して使用したり、既存の MFL ドキュメントの項目を開いて使用したりすることもできます。

以下のトピックでは、パレットを使用するために必要な情報を提供します。

- パレットを開く
- パレットの [File] メニューの使い方
- パレットのショートカット メニューの使い方
- アクティブなメッセージ フォーマットの項目をパレットにコピーする
- 項目のパレットから削除する
- パレットの項目をパレットからアクティブなメッセージ フォーマットにコピーする

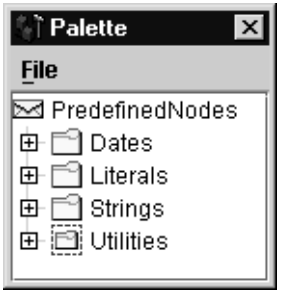
パレットを開く

パレットを開く手順は、次のとおりです。

1. Format Builder を呼び出します。
2. [View ｜ Show Palette] を選択します。

   [Palette] ウィンドウに、デフォルトのパレットが表示されます

図 3-9　パレット



ナビゲーション ツリーとパレットとの間では、項目をコピーできます。パレット内の項目は、ドラッグ アンド ドロップまたはショートカット メニューのコマンドを使用して整列できます。パレットの内容は、Format Builder を終了するときに自動的に保存されます。

注意： ナビゲーション ツリーとパレットとの間では、項目のコピーのみが許可されます。項目をウィンドウ間で移動することはできません。

パレットの [File] メニューの使い方

次の表で説明するコマンドは、パレットの [File] メニューから使用できます。

表 3-7 パレットの [File] メニューのコマンド

コマンド	説明
[Open]	[Open] ダイアログ ボックスを表示する。このダイアログ ボックスで、パレット内の既存の MFL ドキュメントを選択して開くことができる。
[Save]	パレットで現在開かれている MFL ドキュメントに加えた変更を保存する。
[Hide Palette]	[Plette] ウィンドウを閉じる。

パレットのショートカット メニューの使い方

パレットの項目またはフォルダを右クリックすると、ショートカット メニューが表示されます。次の表では、ショートカット メニューから使用できるコマンドについて説明します。

注意： 選択した項目によっては、一部のコマンドを使用できない場合があります。

表 3-8 パレットのショートカット メニュー コマンド

コマンド	説明
[Insert]	新しいフォルダを挿入する。このコマンドを選択すると、フォルダ名の入力を求められる。
[Rename]	フォルダの名前を変更する。このコマンドを選択すると、新しい名前の入力を求められる。
[Delete]	選択した項目を削除する。
[Move Up]	選択した項目を、その親の中でポジションを 1 つ上に移動する。
[Move Down]	選択した項目を、その親の中でポジションを 1 つ下に移動する。
[Promote]	選択した項目を、階層内のすぐ上のレベルに割り当てる。たとえば、「Field1」が「Group1」の子である場合、「Field1」を選択して [Promote] ツールをクリックすると、「Field1」は「Group1」の兄弟になる。
[Demote]	選択した項目を、階層内のすぐ下のレベルに割り当てる。項目を下のレベルに割り当てると、1 つ前にある兄弟の子になる。たとえば「Field1」が「Group1」の兄弟で、「Group1」の次にある場合、「Field1」を選択して [Demote] ツールをクリックすると、「Field1」は「Group1」の子となる。

アクティブなメッセージ フォーマットの項目をパレットにコピーする

Format Builder で現在開かれているドキュメントの項目をパレットにコピーする手順は次のとおりです。

1. パレットが表示されていない場合は、[View ｜ Show Palette] を選択してパレットを表示します。
2. ナビゲーション ツリーで、パレットに追加したい項目を選択します。
3. 項目を [Palette] ウィンドウにドラッグし、階層内の目的の位置にドロップします。

   項目は、選択した位置にコピーされます。

注意： 別の項目の有無に依存する項目をパレットに追加することはできません。たとえば、フィールド参照やグループ参照を追加することや、`Repeat Field` が指定された項目を追加することは許可されません。

コメントの追加は可能ですが、コメントはユニークな名前を持たないのでパレット上で区別が付きにくいため、追加しないことをお勧めします。

項目のパレットから削除する

項目をパレットから削除する手順は次のとおりです。

1. 削除する項目を右クリックし、ショートカット メニューを表示します。
2. [Delete] を選択します。

   削除の確認を求められます。
3. [OK] をクリックして、項目を削除します。

パレットの項目をパレットからアクティブなメッセージ フォーマットにコピーする

パレットの項目を、Format Builder で現在開かれているメッセージ フォーマット ドキュメントにコピーする手順は次のとおりです。

1. パレットが表示されていない場合は、[View ｜ Show Palette] を選択してパレットを表示します。
2. [Palette] ウィンドウで、メッセージ フォーマットに追加したい項目を選択します。
3. 項目を ナビゲーション ツリーにドラッグし、階層内の目的の位置にドロップします。

   項目がメッセージ フォーマットの目的の位置にコピーされます。

メッセージ フォーマットの保存と格納

この節の手順に従ってメッセージ フォーマット ドキュメントをファイル システムに保存したり、5-6 ページの「リポジトリへの MFL ドキュメントの保存」の説明に従って、ドキュメントをリポジトリに格納したりすることができます。

メッセージ フォーマット ファイルを初めて保存する手順は次のとおりです。

1. [File ｜ Save As] を選択して、[Save As] ダイアログ ボックスを表示します。

図 3-10　[Save As] ダイアログ ボックス

2. ファイルを保存するディレクトリに移動します。
3. [File Name] フィールドにこのファイルに割り当てる名前を入力します。

   注意： ファイル名に拡張子が指定されないときは、Format Builder によりデフォルトの拡張子 `..mfl` が自動的に割り当てられます。

4. [Save As] をクリックして、ファイルを指定した場所に指定した名前と拡張子で保存します。

既存のファイルの変更を保存するには、[File ｜ Save] を選択します。

既存のファイルを新しい名前で保存するには、[File ｜ Save As] を選択し、上記の手順 2 ～ 4 を実行します。

既存のメッセージ フォーマット ファイルを開くかまたは検索する

この節の手順に従ってメッセージ フォーマット ドキュメントをファイル システム上で開いたり、5-3 ページの「リポジトリからの MFL ドキュメントの検索」の説明に従って、リポジトリから検索したりすることができます。

既存のメッセージ フォーマット ファイルを開く手順は次のとおりです。

1. [File ｜ Open] を選択して、[Open] ダイアログ ボックスを表示します。

図 3-11 [Open] ダイアログ ボックス

2. 目的のファイルを見つけて、選択します。
3. [Open] をクリックして、Format Builder のファイルを開きます。

インターナショナライゼーション機能の使用

インターナショナライゼーション機能を含むように、個々のメッセージ ファイルのオプションを変更するか Format Builder のデフォルト オプションを設定することにより、Format Builder のインターナショナライゼーション機能を使用できます。詳細については、次のドキュメントを参照してください。

- 3-42 ページの「メッセージ フォーマットのオプションの変更」
- 3-43 ページの「Format Builder のオプションの設定」

メッセージ フォーマットのオプションの変更

メッセージ フォーマット ファイルのオプションを変更する手順は次のとおりです。

1. ナビゲーション ツリーでメッセージ フォーマットのルート ノードを選択します。
2. [File ｜ Properties] を選択します。

   [File Properties] ダイアログボックスに [Message Format Version] と [Default Message Format (MFL) Encoding] が表示されます。

図 3-12　[File Properties] ダイアログ ボックス

3. このファイルのエンコーディング名と説明のリストから MFL ドキュメントの文字エンコーディングの種類を選択します (新しいすべてのメッセージ フォーマット ドキュメントのデフォルト設定を変更するには、[Tools | Options] を選択)。
4. [OK] をクリックします。

   Format Tester でテストすると、MFL ドキュメントに変更が反映されます。

Format Builder のオプションの設定

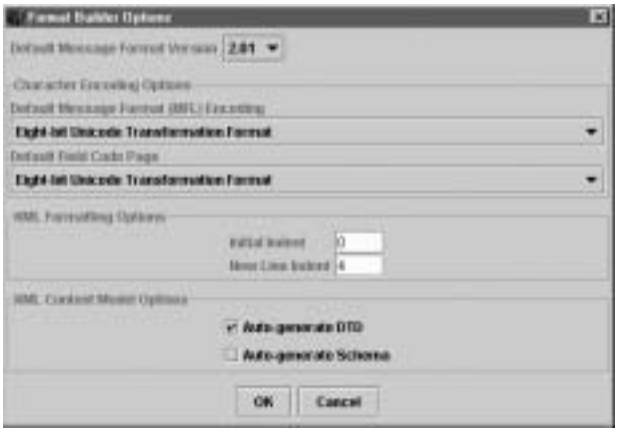
いくつかのオプションを設定すれば Format Builder の動作全体をコントロールできます。

Format Builder のオプションを設定する手順は次のとおりです。

1. [Tools | Options] を選択します。

   [Options] ダイアログ ボックスが表示されます。

図 3-13 [Format Builder Options] ダイアログ ボックス



2. 次の表に従って、フィールドにデータを入力します。

表 3-9 Format Builder オプション

カテゴリ	オプション	説明
なし	[Default Message Format Version]	新しい MFL ドキュメントに関連付けられるバージョンを選択する。 **注意：** 各メッセージ フォーマット ドキュメントは、独自のメッセージ フォーマット バージョンに関連付けられます。個々のメッセージ フォーマットに対して指定されるバージョンは、前の節 「メッセージ フォーマットのオプションの変更」で説明した [File Properties] ダイアログ ボックスを使用してデフォルト設定から変更できます。
[Character Encoding Options]	[Default Message Format (MFL) Encoding]	新しい MFL ドキュメントに関連付けられる文字エンコーディングを選択する。MFL ドキュメントに関連付けられた文字エンコーディングは、MFL ドキュメント自体および MFL ドキュメントが生成する XML 出力で使用するエンコーディングを指定する。
	[Default Field Code Page]	バイナリ フォーマットのリストからコード ページを選択する。このコード ページが、MFL ドキュメントで作成される各フィールドについて、デフォルト コード ページとして使用される。コード ページは、各フィールドのバイナリ データの文字エンコーディングを指定する。
[XML Formatting Options]	[Initial Indent]	XML 出力を生成する際に、ルート要素のインデントに使用するスペース数を入力する。
	[New Line Indent]	XML 出力を生成する際に、新しい子要素のインデントに使用するスペース数を入力する。

表 3-9 Format Builder オプション （続き）

カテゴリ	オプション	説明
[XML Content Model Options]	[Auto-generate DTD]	MFL ドキュメントで定義されたコンテンツ モデルを取りこむ DTD ドキュメントを生成する。[Auto-generate DTD] を指定すると、次の処理が行われる。 ■ MFL ドキュメントをファイル システムに保存すると、DTD は、同じディレクトリに保存される。 ■ MFL ドキュメントをリポジトリに格納すると、DTD も格納される。
	[Auto-generate Schema]	MFL ドキュメントで定義されたコンテンツ モデルを取りこむ XML スキーマ ドキュメントを生成する。[Auto-generate Schema] を指定すると、次の処理が行われる。 ■ MFL ドキュメントをファイル システムに保存すると、XML スキーマ ドキュメントは、同じディレクトリに保存される。 ■ MFL ドキュメントをリポジトリに格納すると、XML スキーマ も格納される。

3. 以下のいずれか 1 つをクリックします。
   - [OK] －変更を保存してダイアログ ボックスを閉じる。
   - [Cancel] －変更を取り消してダイアログ ボックスを閉じる。

Format Builder のメニュー

Format Builder では以下のメニューを使用できます。[File]、[Edit]、[Insert]、[View]、[Repository]、[Tools]、および、[Help]

以下の各節では、各メニューで使用できるコマンドについて説明します。

注意： ユーザがその時点までに実行したアクションおよびナビゲーション ツリーで選択している項目によっては、一部のコマンドは使用できないことがあります。

[File] メニュー

[File] メニューには次のコマンドがあります。

表 3-10 [File] メニュー コマンド

コマンド	説明
[New]	メッセージ フォーマット ドキュメントを新規作成する。
[Open]	既存のメッセージ フォーマット ドキュメントを開く。
[Close]	現在のメッセージ フォーマット ドキュメントを閉じる。
[Save]	現在のメッセージ フォーマット ドキュメントを保存する。
[Save As]	現在のメッセージ フォーマットを別の名前で保存する。
[Properties]	アクティブなメッセージ フォーマット ドキュメントの [File Properties] ダイアログボックスを開く。このダイアログ ボックスにより、アクティブな MFL ドキュメントのオプションを設定できる（3-42 ページの「メッセージ フォーマットのオプションの変更」を参照）。アプリケーションのデフォルトを設定するには、[Tools ｜ Options] を選択する（3-43 ページの「Format Builder のオプションの設定」を参照）。
[Exit]	Format Builder を終了する。

[Edit] メニュー

[Edit] メニューには次のコマンドがあります。

表 3-11 [Edit] メニューのコマンド

コマンド	説明
[Undo] (*操作の取り消し*)	直前の操作を取り消す。[Edit] メニューの [Undo] コマンドは、絶えず更新され、最も新しく実行され取り消しできる操作を示す。たとえば、フィールド名を「Field1」に変更して [Apply] をクリックすると、[Undo] コマンドのリストには、「Undo Apply Field Field1」というテキストが格納される。 Format Builder では、以前の操作を複数回にわたって取り消すことができる。
[Redo] (*操作のやり直し*)	[Undo] コマンドの結果を取り消す。[Edit] メニューの [Redo] コマンドは、絶えず更新され、やり直し可能な操作を示す。たとえば、フィールド名を「Field1」に変更して [Undo] をクリックすると、[Redo] コマンドのリストには、「Redo Apply Field Field1」というテキストが格納される。 Format Builder では、以前の取り消し操作を複数回にわたってやり直しできる。
[Cut]	項目をその子オブジェクトと共に削除する。削除された項目は、クリップボードに置かれ、新しい位置に貼り付けできる。 **注意：** [Message Format] (ルート) 項目が選択されている場合、このコマンドは使用できません。
[Copy]	選択した項目とその子オブジェクトのコピーを作成する。コピーは、クリップボードに置かれ、新しい位置に貼り付けできる。 **注意：** [Message Format] (ルート) 項目が選択されている場合、このコマンドは使用できません。

表 3-11 [Edit] メニューのコマンド （続き）

コマンド	説明
[Paste]	クリップボードの現在の内容を挿入する。[Paste] を選択すると、次の [Paste] メニュー オプションが表示される。 ■ [As Child] ■ [As Sibling] ■ [As Reference]
[Duplicate]	現在選択している項目のコピーを作成し、兄弟として貼り付ける。複製した項目は、複製元の項目と同じ値および子オブジェクトを持つ。複製された項目の名前は元の項目名と同じだが、元の名前の前に「*New*」が追加される。たとえば、元の項目の名前が「MyField1」の場合、複製の名前は「NewMyField1」となる。
[Delete]	ナビゲーション ツリー内で選択されている項目とその項目の子オブジェクトすべてを削除する。
[Move Up]	選択した項目を、その親の中でポジションを 1 つ上に移動する。
[Move Down]	選択した項目を、その親の中でポジションを 1 つ下に移動する。
[Promote]	選択した項目を、階層内のすぐ上のレベルに割り当てる。たとえば、「Field1」が「Group1」の子である場合、「Field1」を選択して [Promote] を選択すると、「Field1」は「Group1」の兄弟となり、「Group1」のすぐ後ろに挿入される。
[Demote]	選択した項目を、階層内のすぐ下のレベルに割り当てる。項目を下のレベルに割り当てると、1 つ前にあるグループの子になる。たとえば、「Field1」が「Group1」の兄弟で、「Group1」のすぐ後にあるとすると、「Field1」を選択して [Demote] を選択すると、「Field1」は「Group1」の子になる。

[Insert] メニュー

[Insert] メニューには次のコマンドがあります。

表 3-12 [Insert] メニュー コマンド

コマンド	説明
[Field]	新たにフィールドを挿入する。ナビゲーション ツリー内で選択した項目の子または兄弟として、フィールドを挿入することができる。
[Group]	新たにグループを挿入する。ナビゲーション ツリー内で選択した項目の子または兄弟として、グループを挿入することができる。
[Comment]	コメントを挿入する。ナビゲーション ツリー内で選択した項目の子または兄弟として、コメントを挿入することができる。

[View] メニュー

[View] メニューには次のコマンドがあります。

表 3-13 [View] メニュー コマンド

コマンド	説明
[Show Palette]	[Palette] ウィンドウを表示する。
[Expand All]	ナビゲーション ツリー全体を展開し、ナビゲーション ツリー内の全項目の子オブジェクトを表示する。
[Collapse All]	ナビゲーション ツリー全体を折りたたんで、ルートのメッセージ フォーマットのみを表示する。

[Repository] メニュー

[Repository] メニューには次のコマンドがあります。

注意： リポジトリの使い方については、第 5 章「リポジトリ ドキュメントの検索と保存」を参照してください。

表 3-14 [Repository] メニュー コマンド

コマンド	説明
[Log In]	WebLogic Integration の [Repository Login] ダイアログ ボックスが表示され、リポジトリに接続できる。
[Log Out]	リポジトリへの接続を切断する。
[Retrieve]	リポジトリのドキュメントを検索する。
[Store]	現在のドキュメントをリポジトリに格納する。
[Save As]	現在のドキュメントを別の名前でリポジトリに格納する。

[Tools] メニュー

[Tools] メニューには次のコマンドがあります。

表 3-15 [Tools] メニュー コマンド

コマンド	説明
[Import]	インストールされている Importer のリストを表示する。メッセージのインポートに使用する Importer を選択する。
[Test]	Format Tester を開く。
[User Defined Types]	[Add/Remove User Defined Types] ダイアログ ボックスを表示する。
[Options]	Format Builder の [Options] ダイアログ ボックスが表示される。

[Help] メニュー

[Help] メニューには次のコマンドがあります。

表 3-16 [Help] メニュー コマンド

コマンド	説明
[Help Topics]	デフォルト ブラウザでオンライン ヘルプを表示する。
[How Do I]	Format Builder の一般的なタスクのリストを表示する。各タスクをクリックすると、手順がステップごとに表示される。
[About]	実行中の Format Builder および JDK のバージョンと著作権情報が表示される。

4　メタデータのインポート

WebLogic Integration には、複数のユーティリティが含まれます。これらのユーティリティを使用すると、COBOL コピーブックをインポートしたり、C 構造体定義を変換したり、FML Field Table Class を MFL ファイルに変換したりすることができます。以下の各項目では、これらのインポート操作の方法について説明します。

- COBOL コピーブックのインポート
- C 構造体のインポート
- FML Field Table Class のインポート

COBOL コピーブックのインポート

WebLogic Integration には、COBOL データを変換するためのメッセージ定義を作成することにより、COBOL コピーブックを Format Builder にインポートできるようにする機能があります。コピーブックのインポート時には、インポートされるコピーブックとそれに含まれるグループおよびフィールドを説明するコメントを使用できます。

COBOL コピーブックをインポートする手順は次のとおりです。

1. [Tools ｜ Import ｜ COBOL Copybook Importer] を選択します。[COBOL Copybook Importer] ダイアログ ボックスが表示されます。

図 4-1　COBOL Copybook Importer

2. 次の表の説明に従って、プロパティを指定します。

表 4-1 COBOL Copybook Importer のプロパティ

プロパティ	値	説明
[File Name]	[Text String]	インポートするファイルのフル パス名をタイプするかまたは [Browse] ボタンを使用して、ファイルの位置に移動する。
[Byte Order]	[BigEndian]	このオプションは、IBM 370、Motorola、および大半の RISC 設計（IBM メインフレームやほとんどの UNIX プラットフォーム）で選択する。
	[Little Endian]	このオプションは、Intel、VAX、および Unisys プロセッサ（Windows、VMS、Digital、UNIX、および Unisys）で選択する。
[Character Set] **注意：**文字セットは、インポート元ホスト マシンの属性です。	[EBCDIC]	文字セットを EBCDIC に設定する場合に、このオプションを選択する。
	[US-ASCII]	文字セットを US-ASCII に設定する場合に、このオプションを選択する。
	[Other]	コード ページのリストを使用して、フィールド データの文字エンコーディングを選択する。

3. 以下のいずれか 1 つをクリックします。
   - [OK] ー定義した設定に基づいて COBOL コピーブックをインポートする。

- [Cancel] －インポートは行わずに、ダイアログ ボックスを閉じて、Format Builder に戻る。
- [About] －使用されているバージョン、サポートされているコピーブックの機能など、COBOL Copybook Importer についての情報を表示する。

コピーブックをインポートしたら、メッセージ フォーマット定義と同じ手順で操作できます。コピーブックにエラーやサポートされていないデータ型が含まれている場合は、エラーを通知するメッセージが表示されます。エラーを表示するか、または今後の参考にエラーをログ ファイルに保存するかを選択できます。

次の表では、COBOL Copybook Importer のためにインストールされたサンプル ファイルのリストと説明を示します。ディレクトリ名はすべて相対パス名で表記しています。指定されているディレクトリは、*SAMPLES_HOME*\integration\samples\di の下にあります。ここで、*SAMPLES_HOME* は、WebLogic Platform をインストールした位置にあるサンプル ディレクトリを示します。

表 4-2 COBOL コピーブック ファイルのサンプル

ディレクトリ	ファイル	説明
COBOL\	emprec5.cpy	コピーブック ファイルのサンプル
COBOL\	emprec5.data	emprec5.cpy に対応するテスト データ

C 構造体のインポート

WebLogic Integration には C 構造体インポート ユーティリティがあります。このユーティリティでは、次に示す種類の出力データを生成することにより、C 構造体定義を MFL メッセージ定義に変換できます。

- MFL ドキュメント
- C コード

どちらを出力する場合でも、まず、.c または .h 入力ファイルを指定して解析し、構造体を選択する必要があります。その作業が終わったら、MFL (デフォルト) を出力するのか C コードを出力するのかを選択できます。

パーサへのすべての入力は、有効な C コードでなければなりません。また、外部参照（#include、#define、typedef などのステートメント）は、使用する前にすべて解決しておく必要があります。解決方法は、手動で編集するかコンパイラのプリプロセッサを使用するかのいずれかです。

プラットフォーム固有のさまざまなパラメータが C コードのデータ記述に影響する場合があります。たとえば、あるプラットフォームの `long` の長さが特定の構造体定義に準拠するバイナリ データに影響を与えます。

MFL を Format Builder に直接生成するかしないかによって、上記のプラットフォーム依存に対処する方法が 2 とおりあります。MFL を生成し、その MFL を直接 Format Builder で表示するときは、プラットフォームに依存するパラメータをコンフィグレーション ファイルで指定しなければなりません。

一方、C でソースを生成する方法を使用すれば、C コードを相手先マシンでコンパイルできます。相手先マシンがコンパイルを実行すると、必要なプラットフォーム依存情報が取り込まれます。この方法により、実行時に、2 つのファイル（MFL ドキュメントとその MFL に準拠するバイナリ データ）を生成する実行可能ファイルを生成できます。MFL ドキュメントは Format Builder で、バイナリ データ ファイルは Format Tester で開くことができます。

MFL を直接 Format Builder に生成するには、プラットフォーム コンフィグレーション パラメータが既存のコンフィグレーション ファイルに存在するか、ハードウェア プロファイル エディタを使用してコンフィグレーション ファイルを新しく作成する必要があります。ハードウェア プロファイル エディタを使用すれば、既存のプロファイルを指定して、ロード、更新および保存を行うことができます。

ハードウェア プロファイルをユーザの必要に応じて生成するユーティリティのソース コードは、*`SAMPLES_HOME`*`\integration\samples\di\cfg` ディレクトリにあります。

C Struct Importer ファイルのサンプル

次の表では、C Structure Importer 用にインストールされたサンプル ファイルのリストと説明を示します。ディレクトリ名はすべて相対パス名で表記します。指定されているディレクトリは、*`SAMPLES_HOME`*`\integration\samples\di` の下にあります。

表 4-3 C Struct Importer ファイルのサンプル

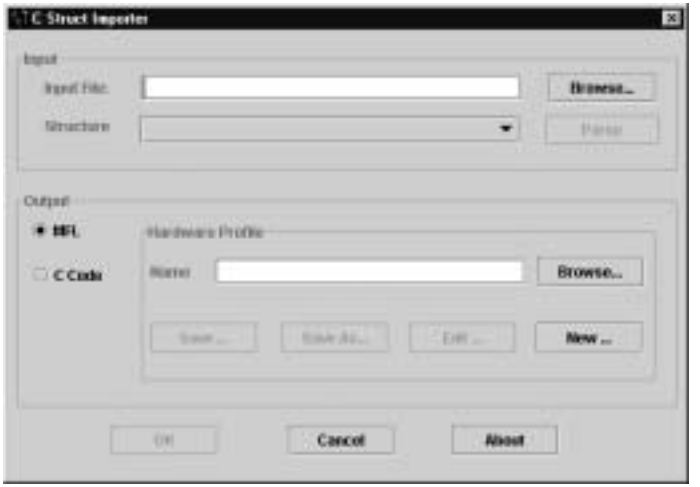
ディレクトリ	ファイル	説明
C	emprec5.h	複数の typedef が含まれるコピーブック ファイルのサンプル emprec5.cpy の C バージョン
C	emprec5n.h	ネストされる構造体定義を使用する emprec5.h ファイルのバリアント（typedef は使用しない）
C	emprec5s.h	emprec5.h ファイルのシンプル バージョン
C	ntfsez.h	ntfs.h ファイルから抽出されたスモール サンプル。再帰 typedef をテストすることが目的。
Cfg	cprofile.c	cprofile.c ユーティリティのソース コード。さまざまなプラットフォームのプロファイルを生成することが目的。
以下の .cfg ファイルは、cprofile プログラムによりさまざまなプラットフォームで生成されたものである。.cfg ファイルにはそれぞれ DESCRIPTION の値が含まれる。		
Cfg	dec8cc.cfg	DEC Alpha 1091、Digital Unix 4.0e、cc コンパイラ
Cfg	hp5cc.cfg	HP-UX B.11.00、cc コンパイラ
Cfg	nt4bcc5.cfg	Windows NT 4.0、Borland 5.x コンパイラ、デフォルト スイッチ
Cfg	nt4vc6.cfg	Windows NT 4.0、Visual C++ 6.x コンパイラ、デフォルト スイッチ
Cfg	sun7cc.cfg	SunOS 5.8、cc コンパイラ
Cfg	w95bcc5.cfg	Windows 95、Borland 5.x コンパイラ、デフォルト配置
Cfg	w95vc5.cfg	Windows 95、Visual C++ 5.x コンパイラ、デフォルト配置

C Structure Importer の呼び出し

C Struct Importer を呼び出す手順は次のとおりです。

1. [スタート | プログラム | BEA WebLogic Platform 7.0 | WebLogic Integration 7.0 | Format Builder] を選択して Format Builder を呼び出します。Format Builder のメイン ウィンドウが表示されます。
2. [Tools | Import_ | C Struct Importer] を選択します。[C Struct Importer] ダイアログ ボックスが表示されます。

図 4-2　[C Structure Importer] ダイアログ ボックス



[C Struct Importer] ダイアログ ボックスでは、次の表の説明に従って、インポートのプロパティを指定できます。

注意： 最初は、MFL が、デフォルト出力タイプとして指定されます。

表 4-4 C Struct Importer のプロパティ

カテゴリ	プロパティ	説明
[Input]	[Input File]	インポートするファイルのフル パス名をタイプするかまたは [Browse] ボタンを使用して、ファイルの位置に移動する。
	[Structure]	解析が正常に行われた後、入力ファイルで検出された構造体のドロップダウン リスト。
	[Parse]	入力ファイルを解析するには、このオプションを選択する。解析に成功すると、入力ファイル内の構造体リストが [Structure] リスト ボックスに表示される。

表 4-4 C Struct Importer のプロパティ （続き）

カテゴリ	プロパティ	説明
[Output]	[MFL]	このオプションを選択すると、構造体定義とハードウェア コンフィグレーション ファイルから MFL を生成できる。表示される [Hardware Profile] ダイアログ ボックスには、次のオプションがある。 ♦ [Name] ーファイル名を入力するか、または [Browse] オプションを使用して、既存のプロファイルを指定する。あらかじめ作成されたハードウェア プロファイルは `samples\di\cfg` ディレクトリにある。 ♦ [Save] ー現在のハードウェア プロファイルを保存する。 ♦ [Save As] ー現在のハードウェア プロファイルを別名で保存できる。 ♦ [Edit] ー現在のハードウェア プロファイルを編集できる。 ♦ [New] ー新しいハードウェア プロファイルを作成できる。
	[C Code]	このオプションを選択すると、対象マシンでコンパイルして実行すれば MFL が作成される C ソース コードを生成できる。表示される [C Code File Names] ダイアログ ボックスには、次のオプションがある。 ♦ [MFL Gen] ー MFL を生成するために、対象マシン上でコンパイルが必要な C ソース コード ファイル名を指定する。ファイルを格納するディレクトリに移動するには、[Browse] を使用する。 ♦ [Data Gen] ーテスト データを生成するために、対象マシン上でコンパイルが必要な C ソース コード ファイル名を指定する。ファイルを格納するディレクトリに移動するには、[Browse] を使用する。

図 4-3 [C Structure Importer] ダイアログ ボックス

3. 以下のいずれか 1 つをクリックします。
   - [OK] ーハードウェア プロファイルの変更を保存する。
   - [Cancel] ーハードウェア プロファイルの変更を取り消す。
   - [About] ーバージョン番号やリリース日など C Struct Importer の情報を表示する。

ハードウェア プロファイルについて

C Structure Importer によって使用されるハードウェア プロファイルには、特定のハードウェアとコンパイラの組み合わせについてのデータ サイズと位置合わせ情報が含まれ、C 構造体のための MFL を生成するのに使用されます。ハードウェア プロファイルはコンフィグレーション ファイルに格納され、作成、ロード、更新および保存が可能です。

SAMPLES_HOME\integration\samples\di\cfg ディレクトリにある cprofile.c ソース ファイルは、これらのプロファイルを任意のプラットフォーム向けに生成するために使用されます。このコードは、一般的なコンパイラを使用してコンパイルして実行できるように設計されており、ANSI 標準 C コンパイラを搭載したプラットフォームでコンパイルして実行し、C Struct Importer にインポートできるプロファイル コンフィグレーション ファイルを生成できるはずです。

ハードウェア プロファイルユーティリティをビルドする

受け入れられるパーサ入力を生成し、使用しているプラットフォームに適合するコマンドを実行する手順は次のとおりです。

- Windows NT では、VC++ プリプロセッサを使用する

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

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

- UNIX の場合

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

ハードウェア プロファイル ユーティリティの実行

`cprofile` プログラムを実行し、ハードウェア プロファイル名を指定するには、コマンド プロンプトで次のテキストを入力します。

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

[DESCRIPTION] は省略可能です。省略しない場合は、コンフィグレーション ファイルに、`DESCRIPTION` の値として入力してください。説明に埋め込みの空白が含まれるときは、空白を引用符で囲みます。

MFL の生成

MFL を生成する手順は次のとおりです。

1. [Input File] フィールドにファイル名を入力するか、[Browse] をクリックして表示されるリストからファイルを選択します。
2. [Parse] をクリックしてファイルを解析します。

   解析が完了すると、入力ファイル内の構造体が [Structure] リストに表示されます。

   注意： ファイルが正しく解析されない場合は、以下の 2 つの方法のどれかを実行することをお勧めします。

- .h または .c のソースコードをコンパイラのプリプロセッサから実行し、プロセッサの出力をパーサを通じて実行する。
- 解析障害の原因となる文字列をコメントアウトして解析を再度試みる。パーサは最初に互換性のないデータを検出した時点で失敗する点に注意してください。このため、この手順を繰り返すことが必要です。

3. 目的の構造体を [Structure] ドロップダウン リストから選択します。

   MFL を直接生成するためには、この時点でプロファイル コンフィグレーション データを入力する必要があります。これらのデータ入力は、ハードウェア プロファイルを新しく作成するか、または既存のプロファイルを指定して行うことができます。

4. 既存のプロファイルを指定する、および新たにプロファイルを作成する手順は次のとおりです。

   - [Hardware Profile Name] フィールドにファイル名を入力するか、[Browse] をクリックして表示されるリストからファイルを選択することにより、既存のプロファイルを指定します。

     プロファイル パラメータの参照または編集が必要な場合は、[Edit] をクリックして Hardware Profile Editor を呼び出します。

     注意： 共通のコンフィグレーション用のハードウェア プロファイルはあらかじめビルドされていて、`samples\c\cfg` ディレクトリにあります。

   - 新たにハードウェア プロファイルを作成するには、[New] をクリックします。表示される [Hardware Profile] エディタには、デフォルトのパラメータがロードされます。新しいプロファイルの名前と説明を指定して、必要に応じて原始データ型とバイト オーダーを修正します。

図 4-4　[C Struct Importer Hardware Profile] ダイアログ ボックス

5. [OK] をクリックしてハードウェア プロファイルの変更を保存し、[C Struct Importer] ダイアログ ボックスに戻ります。

6. [OK] をクリックして MFL を生成します。生成が成功すると、ナビゲーション ツリーに MFL オブジェクトが表示された Format Builder に戻ります。生成された MFL オブジェクトには、解析で使用された入力ファイルと同じ名前が反映されます。

   生成プロセスでエラーが検出された場合は、[MFL Generation Errors] ダイアログ ボックスが表示されます。このダイアログ ボックスを使用して、エラー ログを表示するかまたはファイルとして保存することができます。

図 4-5 [MFL Generation Errors] ダイアログ ボックス

7. 以下のいずれか 1 つをクリックします。
   - [Display Error Log] ー検出されたエラーがあれば表示する。
   - [Save Error Log] ーエラー ログを任意の位置に保存する。または
   - [Cancel] ー [MFL Generation Errors] ダイアログ ボックスを終了する。

発生したエラーを確認したら、[C Struct Importer] に戻って該当する手順を繰り返すことができます。

C コードの生成

C コードを生成する手順は次のとおりです。

1. [Input File] フィールドにファイル名を入力するか、[Browse] をクリックして表示されるリストからファイルを選択します。
2. [Parse] をクリックしてファイルを解析します。

   解析が完了すると、入力ファイル内の構造体が [Structure] リストに表示されます。

   注意： ファイルが正しく解析されない場合は、以下の 2 つの方法のどれかを実行することをお勧めします。
   - .h または .c のソースコードをコンパイラのプリプロセッサから実行し、プロセッサの出力をパーサを通じて実行する。
   - 解析障害の原因となる文字列をコメントアウトして解析を再度試みる。パーサは最初に互換性のないデータを検出した時点で失敗する点に注意してください。このため、この手順を繰り返すことが必要です。
3. 目的の構造体を [Structure] ドロップダウン リストから選択します。

4. [C Code] オプション 選択します。
5. [MFL Gen] フィールドまたは [Data Gen] フィールドにファイル名を入力するか、[Browse] をクリックして表示されるリストからファイルを選択します。
6. [OK] をクリックします。

   既存のファイルを上書きしようとしている場合にはメッセージが表示されます。また、コード生成の成功 / 失敗を示すメッセージが表示されます。
7. 生成されたソース コードを対象のプラットフォームにコピーし、ソース コードをコンパイルして実行します。

   注意： 構造体の宣言を含む入力ファイルもコピーする必要があります。コンパイルが完了したら、どちらのプログラムでも、出力ファイルの名前が引数として取り込まれます。
8. 生成された MFL またはデータを、Format Builder を実行しているプラットフォームにコピーします。

FML Field Table Class のインポート

FML Field Table Class Importer を使うと、WebLogic Tuxedo Connector と Business Process Management (BPM) 機能を容易に統合できます。Tuxedo のアプリケーション バッファと XML 間の変換は、WebLogic Tuxedo Connector に付属している FML/XML Translator によって行われます。

Tuxedo を BPM 機能と統合するには、WebLogic Tuxedo Connector とプロセス エンジンとの間で渡される XML を作成する必要があります。必要な XML を作成するには、FML Field Table Class Importer および Format Tester の XML 生成機能を使用します。

FML Field Table Class Importer の前提条件

Format Builder を起動する前に次の手順を実行してください。

1. FML バッファと関連付けられているフィールド テーブルを Tuxedo システムから WebLogic Server/WebLogic Tuxedo Connector 環境に移動します。

2. weblogic/wtc/jatmi/mkfldclass ユーティリティを使用して、フィールド テーブルを表す Java ソース コードを構築します。FML Field Table Administration の詳細については、WebLogic Server のマニュアルを参照してください。
3. ソース コードをコンパイルします。生成されたクラス ファイルは、FldTbl インタフェースを実装するため、fldtbl クラスと呼ばれます。これらのクラスは、Format Builder の CLASSPATH で指定された場所に移動する必要があります。

SAMPLES_HOME\integration\samples\di\fml ディレクトリには、サンプルとして使用できる複数の fldtbl クラス フィールドがあります。これらのサンプルを使用すると、前の 3 つの手順を完了しなくても Format Builder を起動できます。

注意： WebLogic Tuxedo Connector のコンフィグレーションを行うとき、ほとんどのユーザがこれらの手順を実行するので、これらの クラス ファイルはすでに存在している場合があります。

サンプル FML Field Table Class ファイル

次の表では FML Field Table Class Importer のためにインストールされたサンプル ファイルのリストと説明を示します。すべてのファイルが、*SAMPLES_HOME*\integration\samples\di\fml ディレクトリにあります。

表 4-5 FML Field Table Class のサンプル ファイル

ファイル	説明
bankflds.class	FML Field Table Class Importer の入力となるコンパイル済みソース ファイル
bankflds.java	mkfldclass ユーティリティによって作成される fldtbl ソース ファイル
crdtflds.class	FML Field Table Class Importer の入力となるコンパイル済みソース ファイル
crdtflds.java	mkfldclass ユーティリティによって作成される fldtbl ソース ファイル

表 4-5 FML Field Table Class のサンプル ファイル （続き）

ファイル	説明
tBtest1flds32.class	FML Field Table Class Importer の入力となるコンパイル済みソース ファイル
tBtest1flds32.java	mkfldclass ユーティリティによって作成される fldtbl ソース ファイル

FML Field Table Class Importer による XML の作成

FML Field Table Class Importer を使用して XML を作成する手順は次のとおりです。

注意： WebLogic Tuxedo Connector を使用して Java クラスを作成するときは、.class ファイルを \ext ディレクトリに配置できます。そうすると、[FML Field Table Class Importer] ダイアログ ボックスの値が [Available Fields] リストに自動的に取り込まれます。

1. Format Builder を呼び出します。Format Builder のメイン ウィンドウが表示されます。
2. [Tools | Import | FML Field Table Class Importer] を選択します。[FML Field Table Class Importer] ダイアログ ボックスが表示されます。

3. [Class Names] フィールドに、処理する `fldtbl` クラス ファイルの名前を入力します。

   1 つの FML バッファには、複数のフィールド テーブルのフィールドが含まれていることがあるため、[Class Names] フィールドには任意の数の `fldtbl` クラスのファイル名を入力できます。リストの項目は、カンマで区切る必要がありますが、名前に `.class` 拡張子を付ける必要はありません。

   注意： リスト内のクラスのいずれかが `weblogic/wtc/jatmi/mkfldclass` ユーティリティで作成された `fldtbl` クラスでない場合、または Format Builder の CLASSPATH に含まれていない場合は、エラー ダイアログ ボックスが表示されます。ただし、エラーが発生しても、リスト内の有効な `fldtbl` クラスは処理されます。

4. [Load] をクリックします。フィールド テーブル内のフィールド名が [Available Fields] リストに表示されます。[Available Fields] リストでは、重複する名前を表示することはできません。あるフィールド名が別の複数のフィールド テーブルにあっても、リストには 1 つしか表示されません。

5. [Available Fields] リストから任意のフィールドを選択し、[Add] をクリックします。選択したフィールドが [Selected Fields] リストに表示されます（フィールドを [Selected Fields] リストから削除するには、対象のフィールドを選択して [Remove] をクリック）。

6. [Selected Fields] リストが完成したら、[OK] をクリックします。[FML Field Table Class Importer] ダイアログ ボックスが閉じて、生成された MFL の名前が Format Builder のナビゲーション ツリーに追加されます。選択したフィールドは、[Selected Fields] リストと同じ順序で表示されます。

7. 作成した MFL ドキュメントを編集し、Business Process Management (BPM) から WebLogic Tuxedo Connector FML/XML Translator に渡される XML ドキュメント内のフィールドの順序と出現回数を指定します。

8. [Tools ｜ Test] を選択して Format Tester を開きます。

9. Format Tester のメニュー バーで、[Generate ｜ XML] を選択します。

   Format Tester によって、Format Builder 内の MFL ドキュメントに準拠する XML ドキュメントが作成されます。

10. 必要に応じて、XML ドキュメント内のフィールドのデータ コンテンツを編集します。

11. [File ｜ Save XML] を選択して、XML ドキュメントを、名前と位置を指定したファイルに保存します。

    作成された XML は、XML Instance Editor を使用して Business Process Management にインポートでき、Business Process Management で使用できます。XML のインポートの詳細については、BPM のマニュアルを参照してください。

5 リポジトリ ドキュメントの検索と保存

リポジトリに、次のドキュメント タイプを格納して集中的に管理することができます。

- MFL (Message Format Language : メッセージ フォーマット言語)
- XML DTD (XML Document Type Definition : XML 文書型定義)
- XML スキーマ
- XSLT スタイルシート

リポジトリを使用すれば、これらの種類のドキュメントを WebLogic Integration 内で共有できます。リポジトリからこれらのドキュメントを呼び出すことができ、テキスト、説明、注意へのアクセスを含むリポジトリ ドキュメントの操作とドキュメントの削除が可能です。また、リポジトリを使用すれば、サポート対象のドキュメントを WebLogic Server、business process management、および B2B integration の各機能間で共有できます。また、リポジトリにはバッチ インポート ユーティリティが含まれ、これを使用して、すでに作成した MFL、DTD、XML スキーマおよび別のファイルにある XSLT ドキュメントを簡単にリポジトリに移行できます。

この章では、以下のトピックについて説明します。

- リポジトリへのログイン
- [Repository] メニュー コマンド
- リポジトリからの MFL ドキュメントの検索
- リポジトリへの MFL ドキュメントの保存
- ドキュメントのリポジトリへのインポート
- [Select document to retrieve] ダイアログ ボックスと [Store Document] ダイアログ ボックスの使用法

リポジトリへのログイン

ドキュメントをリポジトリから取り出したり、リポジトリに保存したりするには、事前に、ログインする必要があります。

リポジトリにログインする手順は次のとおりです。

1. Format Builder が呼び出されていない場合は、[スタート | プログラム | BEA WebLogic Platform 7.0 | WebLogic Integration 7.0 | Format Builder] を選択して、Format Builder を呼び出します。
2. [Repository | Log In] を選択して、[WebLogic Integration リポジトリへのログイン] ダイアログ ボックスを開きます。
3. ユーザ名、パスワード、リポジトリが位置するサーバの名前を入力します。

   たとえば、WebLogic Integration がローカル マシンで実行されている場合、次の図に示す値を入力して、デフォルトの `admin` ユーザとしてログインします。

図 5-1　[WebLogic Integration リポジトリへのログイン] ダイアログ ボックス

注意： デフォルトのユーザ名とパスワードについては、『*WebLogic Integration の起動、停止およびカスタマイズ*』の「はじめに」にある「WebLogic Integration ユーザおよびパスワード」を参照してください。

4. [接続] をクリックします。

   ログインに成功したら、ダイアログ ボックスが閉じ、Format Builder タイトル バーにサーバ名とポート番号が表示されます。ここで、アクティブなリポジトリ メニュー コマンドのいずれかを選択できます。

   注意： [WebLogic Integration リポジトリへのログイン] ダイアログ ボックスでは、ログインに 3 回失敗するとエラー メッセージが表示されます。ログインに 3 回失敗した場合は、[Repository ｜ Log In] を選択して、ログインの手順を繰り返します。

[Repository] メニュー コマンド

次の表では、[Repository] メニューから使用できるコマンドについて説明します。

コマンド	説明
[Log In]	[WebLogic Integration リポジトリへのログイン] ダイアログ ボックスが表示され、リポジトリに接続できる。
[Log Out]	リポジトリへの接続を切断する。
[Retrieve]	リポジトリのドキュメントを検索する。
[Store]	現在のドキュメントをリポジトリに格納する。
[Store As]	現在のドキュメントを別の名前でリポジトリに格納する。

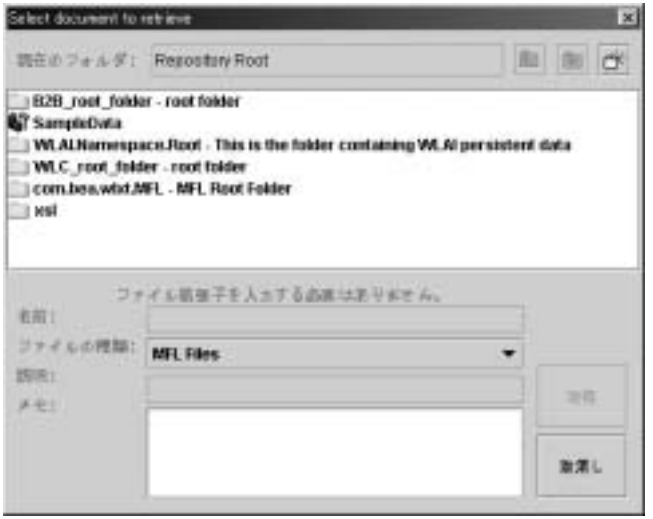
リポジトリからの MFL ドキュメントの検索

リポジトリから MFL ドキュメントを検索する手順は次のとおりです。

1. Format Builder が呼び出されていない場合は、[スタート ｜プログラム ｜ BEA WebLogic Platform 7.0 ｜ WebLogic Integration 7.0 ｜ Format Builder] を選択して、Format Builder を呼び出します。

2. 前の手順（「リポジトリへのログイン」）の説明に従って、リポジトリにログインします。
3. [Repository | Retrieve] を選択して、[Select document to retrieve] ダイアログ ボックスを表示します。

図 5-2 [Select document to retrieve] ダイアログ ボックス



[Select document to retrieve] ダイアログ ボックスで、コンテンツを表示したいフォルダをダブルクリックするか、[現在のフォルダ] テキスト ボックスの左にあるアイコンをクリックして、親フォルダに移動します。ドキュメント名および説明が表示されるので、選択に役立ちます。このダイアログ ボックスの使用法の詳細については、5-10 ページの「[Select document to retrieve] ダイアログ ボックスと [Store Document] ダイアログ ボックスの使用法」を参照してください。

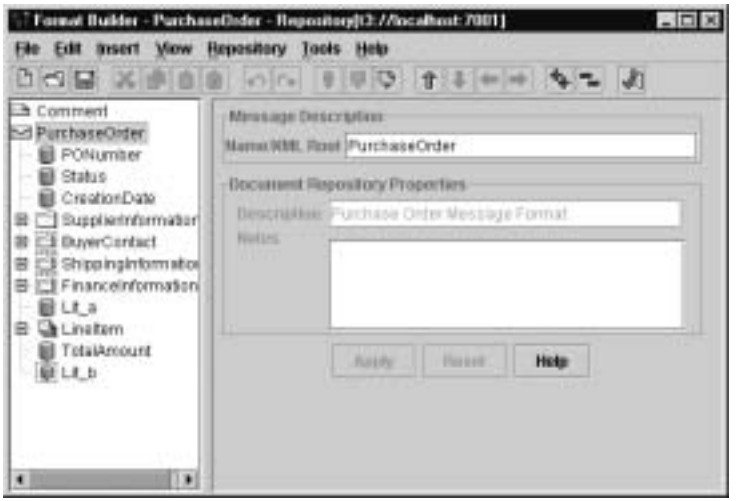
4. 取り出したい MFL ドキュメントを見つけて選択します。

   注意： Format Builder では、MFL ドキュメントだけを取り出すことができます。

5. [検索] をクリックします。

   ナビゲーション ツリーに、選択したドキュメントが表示されます。次の図に示すように、ルート ノードを選択して、ドキュメントの詳細ウィンドウを表示することができます。アクティブなドキュメントがリポジトリ ドキュメントの場合、詳細ウィンドウには、[Document Repository Properties] セクションが表示されます。

図 5-3 Format Builder のリポジトリ ドキュメント



リポジトリ ドキュメントを取り出したら、Format Builder で MFL ドキュメントを編集するのと同じ方法で、編集できます。変更が完了したら保存するために、次のいずれかの手順を実行してください。

- ドキュメントをリポジトリに保存するには、[Repository | Store] を選択する。
- ドキュメントをリポジトリに別名で保存するには、[Repository | Store As] を選択する。詳細については、次の 「リポジトリへの MFL ドキュメントの保存」の手順を参照。
- ドキュメントをローカル ファイルとして保存するには、[File | Save As] を選択する。詳細については、3-40 ページの「メッセージ フォーマットの保存と格納」を参照。

リポジトリへの MFL ドキュメントの保存

リポジトリに MFL ドキュメントを保存する手順は次のとおりです。

1. ドキュメントが開かれていない場合は、Format Builder でドキュメントを開きます。
2. 5-2 ページの「リポジトリへのログイン」の説明に従って、リポジトリにログインします。
3. [Repository | Store As] を選択します。

   [Store As] ダイアログ ボックスが表示されます。このダイアログ ボックスの使用法の詳細については、5-10 ページの「[Select document to retrieve] ダイアログ ボックスと [Store Document] ダイアログ ボックスの使用法」を参照してください。
4. [Name] フィールドに、このドキュメントに割り当てる名前を入力します。
5. [Description] フィールドに、ドキュメントの説明を入力します。
6. [Notes] フィールドにドキュメントに付けておきたいメモを入力します。
7. [Store] をクリックします。

   ドキュメントがリポジトリに格納されます。ルート ノードを選択すると、詳細ウィンドウにリポジトリ名、説明、およびメモが表示されます。

   注意： Format Builder のオプションで DTD/XML スキーマ ファイルの生成が指定されているときは、該当するファイル（複数の場合もある）も指定された名前でリポジトリに格納されます。

ドキュメントのリポジトリへのインポート

WebLogic Integration リポジトリのバッチ インポート ユーティリティは、リポジトリへのコマンド ライン インタフェースとなります。このユーティリティにより、以前にビルドされた MFL ドキュメントをリポジトリに簡単にインポートで

きます。Batch Importer を使用すると、MFL、DTD、クラス、XSLT、および XML スキーマといったドキュメントを任意の組み合わせでインポートできます。このユーティリティは、任意のプラグイン リポジトリを操作します。

リポジトリのインポート ユーティリティを Business Process Management リポジトリとともに使用するには、`wlxt-repository.properties` ファイルが `CLASSPATH` ディレクトリに存在しなければなりません。最初にリポジトリにログインしたとき、Format Builder がプロパティ ファイルを作成します。既存のプロパティ ファイルのディレクトリを、使用するクラスパスに追加するか、または次を含む `wlxt-wlpi-repository.properties` ファイルを作成してください。

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

ここで、*`port`* は、WebLogic Server のポート番号（デフォルト ポートは、`7001`）、*`host`* は、WebLogic Server のコンピュータ名または IP アドレスです。サーバがローカル マシンで実行されている場合は、`localhost` または `127.0.0.1` を指定します。

Windows システムでは、*`WLI_HOME`*`\bin` に、`LaunchImport.cmd` という名前のスクリプトがあります。このスクリプトを使用してインポート ユーティリティを起動する場合、`CLASSPATH` が正しく設定されてから、ユーティリティが起動します。インポート操作に適用できるリポジトリ パスおよびファイルリストを反映するように、このスクリプトを修正してください。

インポート ユーティリティを直接起動するには、コマンド プロンプトで次のコマンドを入力して、バッチ インポート ユーティリティを呼び出します。

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

次の表では、オプションについて説明します。

表 5-1 Import コマンドのオプション

オプション	説明
`-v`	冗長モードをオンにすることを示す。このスイッチはコマンド ライン内の任意の位置で指定でき、それ以降のすべての動作に影響する。冗長モードはデフォルトでは無効。
`-n`	冗長モードをオフにすることを示す。このスイッチはコマンド ライン内の任意の位置で指定でき、それ以降のすべての動作に影響する。冗長モードはデフォルトでは無効。

表 5-1 Import コマンドのオプション

オプション	説明
-f	以後の全ファイルの親フォルダを指定する省略可能なスイッチ。-f スイッチを複数指定するとインポート実行中にフォルダを変更できる。デフォルトでは、ドキュメントはリポジトリのルート フォルダにインポートされる。-f スイッチの特別な引数 @ により、ルート フォルダを指定できる。-f スイッチで指定されるフォルダ名は、常にリポジトリのルート フォルダからの絶対パス名である。パスの内部のフォルダ名はフォワード スラッシュ（/）で区切る必要がある。
-t	以後の全ファイルのデフォルト タイプを指定する省略可能なスイッチ。ドキュメントのタイプがファイル拡張子から判別できない場合、デフォルト タイプがドキュメントに割り当てられる。有効な値は .mfl、.dtd、.class、.xsl、および .xsd。
files	インポート対象の 1 つまたは複数のファイル名を示す。シェルやオペレーティング システムでサポートされるワイルド カード文字を使用できる。

オプションは、いずれもコマンド ラインのオプションの位置で有効となり、別のオプションによってオーバーライドされるまでそのまま有効です。たとえば、次のコマンド ラインを指定すると、カレント ディレクトリにある .dtd、.class、および .mfl ファイルがすべてインポートされますが、.class ファイルのインポート時にのみ、冗長モードが有効になります。

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

インポートされるドキュメントの文書型は、次の表に示すように、ドキュメントのファイル名の拡張子から抽出されます。

表 5-2 サポート対象の文書型と拡張子

ファイル拡張子	割り当てられる文書型
.dtd	DTD
.xsd	XML スキーマ
.mfl	MFL
.class	Java クラス
.xsl	Extensible Stylesheet Language

表 5-2 サポート対象の文書型と拡張子

ファイル拡張子	割り当てられる文書型
他の拡張子	デフォルト タイプ（MFL がデフォルト）

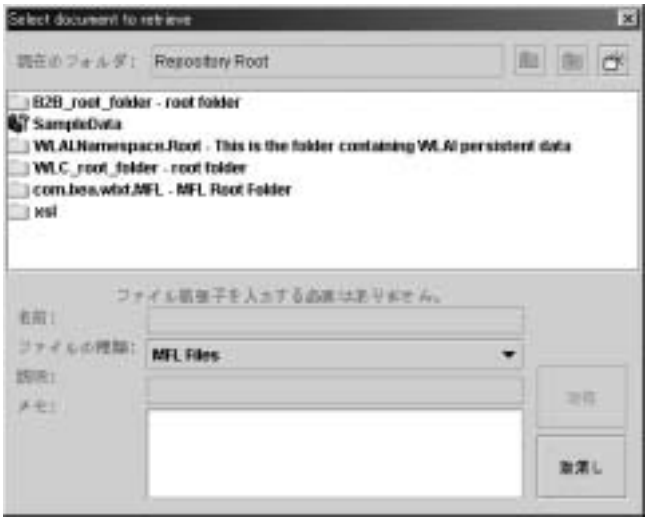
[Select document to retrieve] ダイアログ ボックスと [Store Document] ダイアログ ボックスの使用法

次のダイアログ ボックスを使用して、リポジトリ ドキュメントを取り出したり保存したりします。

- [Select document to retrieve] ダイアログ ボックス
- [Store Document] ダイアログ ボックス

次の図に、[Select document to retrieve] ダイアログ ボックスを示します。

図 5-4　[Select document to retrieve] ダイアログ ボックス



次の図に、[Store Document] ダイアログ ボックスを示します。このダイアログ ボックスは、[Select document to retrieve] ダイアログ ボックスとは、新しいドキュメント名、説明、およびメモを入力できるという点でのみ異なります。

図 5-5 [Store Document] ダイアログ ボックス

次の表に、[Select document to retrieve] ダイアログ ボックスと [Store Document] ダイアログ ボックスの各要素について説明します。

表 5-3 [Select document to retrieve] ダイアログ ボックスと [Store Document] ダイアログ ボックスの要素

フィールド、アイコン、ボタン	定義
[現在のフォルダ] フィールド	カレント リポジトリ フォルダの名前を表示する。
[親フォルダ] アイコン	カレント フォルダがリポジトリのルート フォルダでない場合に、カレント フォルダの親へと上に移動するときにクリックする。
[ルート フォルダ] アイコン	リポジトリのルート フォルダに戻るとき、クリックする。
[フォルダを作成] アイコン	カレント フォルダに新規フォルダを作成するとき、クリックする。リポジトリでフォルダがサポートされないときはこのアイコンは無効。

表 5-3 [Select document to retrieve] ダイアログ ボックスと [Store Document] ダイアログ ボックスの要素 （続き）

フィールド、アイコン、ボタン	定義
[Contents] フィールド	カレント フォルダの フォルダと MFL ドキュメントを一覧して表示。リストにある各エントリの前にはオブジェクトのタイプを示すアイコンが表示される。リストでエントリを選択すると、[名前]、[説明]、および [メモ] フィールドに、そのエントリの関連情報が表示される。フォルダをダブルクリックすると、カレント フォルダになる。[Select document to retrieve] ダイアログ ボックスで MFL ドキュメントをダブルクリックすると、そのドキュメントが Format Builder で開かれる。
[名前] フィールド	フォルダまたはドキュメントの名前。
[説明] フィールド	フォルダまたはドキュメントの説明
[メモ] フィールド	フォルダまたはドキュメントに付けられたメモ。
[検索] ボタン /[ストア] ボタン	[Select document to retrieve] ダイアログ ボックスで、[検索] をクリックすると、選択したドキュメントが Format Builder で開かれる。[Store Document] ダイアログ ボックスで、[ストア] をクリックすると、アクティブなドキュメントがリポジトリに格納される。フォルダが選択されているとき、[検索] または [ストア] ボタンをクリックすると、そのフォルダがカレント フォルダになる。
[取消し] ボタン	クリックすると、変更が適用されずにダイアログ ボックスが閉じる。

ショートカット メニューの使い方

[Select document to retrieve] ダイアログ ボックスまたは [Store Document] ダイアログ ボックスで、フォルダまたはドキュメントを右クリックすると、ショートカット メニューが表示され、名前変更、削除、または選択したオブジェクトのプロパティの表示および更新が可能になります。次の図に、ショートカット メニューを示します。

図 5-6 [Store Document] のショートカット メニュー

表 5-4 ショートカット メニュー コマンド

コマンド	説明
[削除]	[Confirm Delete] ダイアログ ボックスを表示する。[はい] をクリックするとオブジェクトが削除され、[いいえ] をクリックすると操作が取り消される。
[名前を変更]	[Rename Document] ダイアログ ボックスが表示される。新しい名前を入力して [了解] をクリックし、ドキュメントの名前を変更するか、または [取消し] をクリックして操作を取り消す。
[プロパティ]	[プロパティを変更] ダイアログ ボックスを表示する。必要に応じて、[説明] フィールドや [メモ] フィールドを修正し、[OK] をクリックして更新するか、[取消し] をクリックして操作を取り消す。

6　実行時コンポーネントの使用法

WebLogic Integration は、`com.bea.wlxt` クラスを使用して、実行時データ変換をサポートします。この章では、このクラスを *Java クラス*と呼びます。Java クラスでは、バイナリ フォーマットと XML フォーマットとの間のデータ変換にさまざまな方法が使用されます。この Java クラスは、BPM (Business Process Management) のワークフローから開始される WebLogic Server を使用して EJB にデプロイするか、任意の Java アプリケーションに統合できます。

データ変換 Java クラスには、バイナリ データを XML に変換する `parse()` メソッドがいくつかあります。また、データ変換実行時コンポーネントには、XML データをバイナリ フォーマットに変換する `serialize()` メソッドもいくつかあります。バイナリ データ フォーマットは MFL ドキュメントを通じて記述されます。WebLogic Integration では、MFL ドキュメントを使用してバイナリ データを XML から読み込み、XML に書き込みます。MFL ドキュメントは `parse()` メソッドまたは `serialize()` メソッドで URL を使用して指定します。

以下の節では、WebLogic Integration 実行時データ変換ツールを使用してバイナリ データを XML に解析し、XML をバイナリにシリアライズする方法を示すサンプル コードを示します。

- バイナリから XML へ
- XML からバイナリへの変換
- XML から XML への変換

バイナリから XML へ

次のコード リストでは、`parse()` メソッドを使用してバイナリ データ ファイルを XML に解析しています。

コード リスト 6-1 バイナリから XML への Parse() メソッドのサンプル

```
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}
```

このコードの次のシーケンスに注意してください。

1. 行 12 で、Java クラスの新しいインスタンスがインスタンス化されています。
2. 行 13 で、以前に Format Builder で作成された MFL ファイルの URL (Uniform Resource Locator) が作成されています。
3. 行 14 および 15 で、`mybinaryfile` のバイナリ データに対する `FileInputStream` が作成されています。
4. 行 16 で MFL ドキュメントの URL およびバイナリ データのストリームが、parse メソッドに渡されています。
5. 行 17 以降で、parse メソッドにより、バイナリ データが W3C Document オブジェクトのインスタンスにコンバートされます。このインスタンスは、さらに `getXMLText()` メソッドを使用して XML テキストにコンバートされます（別の方法として、parse メソッドにより、バイナリ データを W3C Document オブジェクトのインスタンスにコンバートした後、W3C DOM API を通じてそのオブジェクトを直接操作することもできる）。

DTD を参照する XML 生成

WebLogic Integration のデータ変換ツールの parse() メソッドを使用して、DTD (Document Type Definition : 文書型定義) または XML スキーマで生成された XML ドキュメント内で DTD または XML スキーマへの参照を含むことができます。次のリストは、この機能を示します。

コード リスト 6-2　DTD 参照による XML の生成のサンプル

```
1 import com.bea.wlxt.*;
2 import org.w3c.dom.Document
3 import java.io.FileInputStream;
4 import java.net.URL;
5
6 public class Example2
7 {
8  public static void main(String[ ] args)
9  {
10   try
11   {
12     WLXT wlxt = new WLXT();
13     URL mflDocumentName = new URL(“file:mymfl.mfl”);
14     FileInputStream in = new FileInputStream(“mybinaryfile”);
15
16     Documentdoc=wlxt.parse(mflDocumentName,in,"mydtd.dtd",
17     null);String xml = wlxt.getXMLText(doc, 0, 2);
18     System.out.println(xml);
19   }
20     catch (Exception e)
21   {
22     e.printStackTrace(System.err);
23   }
24  }
25 }
```

コード リスト 6-1 と コード リスト 6-2 とでは、違いが見られるのは、行 16 だけです。つまり、この行 16 では、DTD ファイル（mydtd.dtd）を指定できる別の parse メソッドが呼び出されているため、生成される XML ドキュメントでは、DTD ファイルが参照されます。次の例では、生成される XML ドキュメントの DOCTYPE ステートメントを示します。ここでは、mydtd.dtd を参照しています。

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

このような parse メソッドにより、生成される XML に XML スキーマを参照させることができます。

Debug Writer の指定

WebLogic Integration でサポートされるすべての `parse()` メソッドでは、最後のパラメータとして、PrintWriter を渡すことができます。このパラメータが null でない場合、WebLogic Integration はデバッグ メッセージを指定した PrintWriter に印刷します。MFL ドキュメントとバイナリ データが一致しないとき変換をデバッグする必要がある場合に、この印刷が役立ちます。

デバッグ メッセージを印刷したくない場合、前のリストに示すように、このパラメータに null を渡します。

コード リスト 6-3 Debug Writer を渡す例

```
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 Example2
8 {
9  public static void main(String[ ] args)
10 {
11   try
12   {
13     WLXT wlxt = new WLXT();
14     URL mflDocumentName = new URL(“file:mymfl.mfl”);
15     FileInputStream in = new FileInputStream
16     (“mybinaryfile”);
17     Document doc=wlxt.parse(mflDocumentName,in,new
       PrintWriter(System.out,true));
18     String xml = wlxt.getXMLText(doc, 0, 2);
19     System.out.println(xml);
20   }
21     catch (Exception e)
22   {
23     e.printStackTrace(System.err);
24   }
25  }
26 }
```

行 17 で、parse() メソッドの最後のパラメータとして、PrintWriter オブジェクトが System.out PrintStream から呼び出されます。このオブジェクトの呼び出しにより、次のようなデバッグ メッセージがコンソールに表示されます。

コード リスト 6-4 デバッグ出力

```
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 からバイナリへの変換

次のサンプル コード リストは、WebLogic Integration ツールを使用して XML テキストをバイナリ フォーマットにコンバートする方法を示します。

コード リスト 6-5 XML からバイナリへの変換のサンプル

```
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 }
```

このコードの次のシーケンスに注意してください。

1. 行 12 で、Java クラスの新しいインスタンスが作成されています。
2. 行 13 で、MFL ファイルの URL が作成され、XML テキストを含むファイルに対する `FileInputStream` が作成されています。
3. `FileOutputStream` もインスタンス化され、XML からバイナリへの変換によって生成される。バイナリ データが格納されます。
4. 行 17 では `serialize()` メソッドが呼び出され、`FileInputStream` 'in' (`myxml.xml`) に含まれる XML データが、`'mymfl.mfl'` に記述されるバイナリ フォーマットに変換されます。生成されるこのバイナリ データは FileOutputStream ‘out’ (ファイル `‘mybinaryfile’`) に書き込まれます。

ドキュメント オブジェクトのバイナリ フォーマットへの変換

次のサンプル リストは、W3C ドキュメント オブジェクトをバイナリ フォーマットに変換する方法を示します。

コード リスト 6-6 ドキュメント オブジェクトのバイナリ フォーマットへの変換

```
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       // XML をドキュメント オブジェクトに解析する
14       Document doc = null;
15       try
16       {
17         DOMParser parser = new DOMParser();
18         parser.parse("myxml.xml");
19         doc = parser.getDocument();
20       }
21       catch (Exception e)
22       {
23         e.printStackTrace(System.err);
24         System.exit(1);
25       }
26
27       try
28       {
29         WLXT wlxt = new WLXT();
30         URL mflDocumentName = new URL("file:mymfl.mfl");
31         FileOutputStream out = new
           FileOutputStream("mybinaryfile");
32
33         wlxt.serialize(mflDocumentName, doc, out, null);
34    out.close();
35       }
36       catch (Exception e)
37       {
38       e.printStackTrace(System.err);
39       }
40    }
41  }
```

このサンプルは、Java クラスの `serialize()` メソッドにドキュメント オブジェクトを 指定する方法を示します。アプリケーションにすでに Document オブジェクト形式の XML が含まれる場合や、DOM API を使用して Document オブジェクトが作成済みであればこの方法が便利です。行 14 ～ 25 では、ファイル `myxml.xml` に含まれる XML テキストが XML パーサを使用して Document オブジェクトに変換されます。このドキュメント オブジェクトは行 33 に渡され、MFL ファイル「`mymfl.mfl`」によって指定されたバイナリ フォーマットにコンバートされます。

Debug Writer の指定

serialize メソッドは、デバッグ メッセージのロギング用の PrintWriter パラメータ指定もサポートします。PrintWriter オブジェクトで serialize メソッドを呼び出すサンプルを次に示します。

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

この呼び出しにより、コンソールには、デバッグ メッセージが、次のように表示されます。

コード リスト 6-7 デバッグ出力

```
Processing xml and mfl nodes tcp1
Processing xml node NAME
Checking MFL node NAME
Found matching MFL node NAME
Writing field NAME value John Doe
Processing xml node PAYINFO
Checking MFL node PAYINFO
```

XML から XML への変換

WebLogic Integration データ変換ツールには、XSLT を介して XML を変換するメソッドもあります。XSLT は、このような変換のために設計された言語です。Java クラスは、*XSLT スタイルシート*を XML ドキュメントに適用する `transform()` メソッドを提供します。XSLT スタイルシートは、XML ドキュメントのノードで実行される変換を記述する XML ドキュメントです。スタイルシートを使用すれば、XML ドキュメントを HTML、PDF、または別の XML 固有言語に変換できます。

次のサンプル コード リストは、Java クラスによって提供されるメソッドの 1 つを使用した XML ドキュメントの変換方法を示します。

コード リスト 6-8 XML から XML への変換

```
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 }
```

このコードの次のシーケンスに注意してください。

1. 行 15 で WLXT のインスタンスが作成されます。
2. 行 16 で、すでに作成されている XSLT スタイルシートの URL が作成されます。
3. 行 17 で、、XML テキストを含むファイルに対して `FileInputStream` が作成されます。
4. XSLT 変換によって作成されるテキストに対する `FileOutputStream` も作成されます。

5. 行 21 で Java クラスの `transform()` メソッドが呼び出され、`myxml.xml` ファイル内の XML が、`mystylesheet.xsl` スタイルシートに従って変換されます。変換の出力は `myoutputfile` ファイルに書き込まれます。

初期化メソッド

Java クラスは、MFL ドキュメントと XSLT スタイルシートを*前処理*するためのメソッドをいくつか提供します。これらのドキュメントが前処理されると、内部的にキャッシュされ、`parse()`、`serialize()`、または `transform()` メソッドで参照されるときに再利用されます。MFL ドキュメントや XSLT スタイルシートが事前に処理されてキャッシュされるため、前処理により、これらのメソッドのパフォーマンスは大幅に改善されます。この方法は、同じ MFL ドキュメントや XSLT スタイルシートが繰り返し使用される EJB オブジェクト、またはサーブレットでデータ変換が使用されるときに特に便利です。

Java クラスでは、`java.util.Properties` オブジェクトまたは Properties ファイルのファイル名をパラメータに使用する `init()` メソッドが 2 つ提供されます。この `init()` メソッドは、`WLXT.stylesheets` および `WLXT.MFLDocuments` のプロパティを Properties オブジェクトから取り出します。各プロパティには前処理とキャッシュの対象となるドキュメントをカンマで区切ったリストが含まれることになっています。これらのドキュメントが後で `parse()`、`serialize()`、または `transform()` メソッドで参照される場合、前処理されたバージョンがキャッシュから取り出されます。

次のサンプル リストは、`init()` メソッドを使用して、Java クラスのインスタンスを初期化する方法を示します。

コード リスト 6-9　myconfig.cfg プロパティ ファイル

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

コード リスト 6-10　myconfig.cfg ファイルを使用した init() メソッドのサンプル ソース コード

```
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      // プロパティ ファイルを使用して WLXT を初期化する
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      // バイナリ データを XML に解析する
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  }
```

行 20 では、`init()` メソッドにより、`myconfig.cfg` ファイルでリストされたドキュメントをオブジェクトが前処理します。このため、2 つのドキュメント（MFL ドキュメントとスタイルシート）は、処理が済んでおりオブジェクト内にキャッシュされてから、次のように指定されます。

- MFL ドキュメントは前処理されてから、行 34 で、`parse()` メソッド内で指定される。
- スタイルシートは前処理されてから、行 46 で、`transform()` メソッドの呼び出し時に指定される。

Java API のマニュアル

Java クラスの使用法に関する詳しい説明については、WebLogic Integration Javadoc で `com.bea.wlxt` を参照してください。Javadoc は、WebLogic Integration をインストールしたディレクトリの下にある `docs\apidocs` ディレクトリにもあります。

Business Process Management への実行時プラグイン

Business Process Management (BPM) の Data Integration プラグインは、従来のシステムで使用されていたバイナリ フォーマットと XML との間でのデータ変換をサポートすることにより、アプリケーション間での情報交換を簡単にします。言い換えれば、Data Integration プラグインが提供する BPM アクションを使用して、XML とバイナリ間での変換機能にアクセスできます。

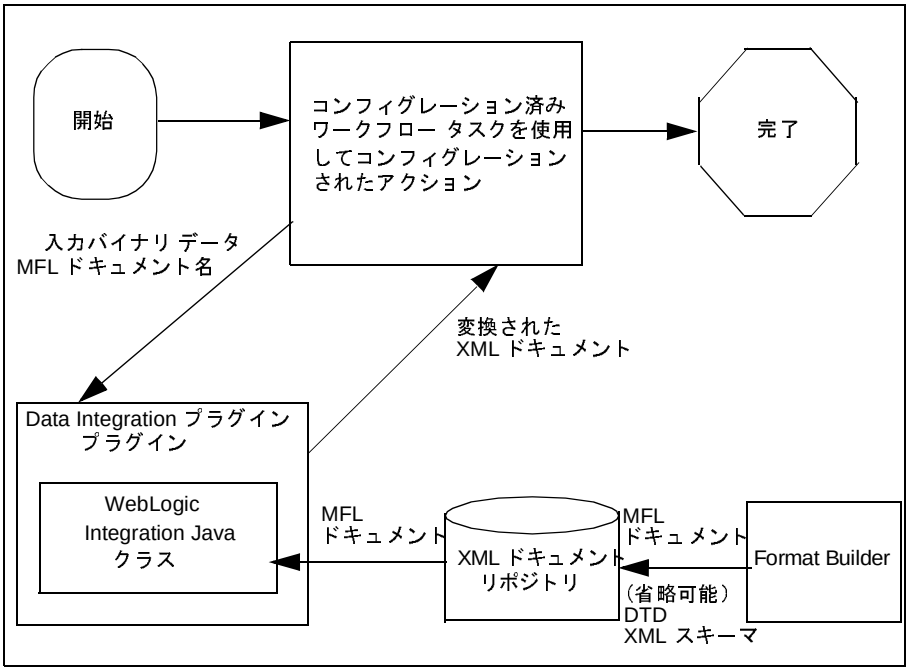
また、Data Integration プラグインは、次の機能も提供します。

- バイナリ フォーマットでのイベント データ処理

- MFL ドキュメントのインメモリ キャッシュおよび変換オブジェクト プール（両方の機能によりパフォーマンスが向上する）
- バイナリ データを編集して表示するための `BinaryData` 変数型
- 完全の独立したワークフロー定義パッケージのエクスポート
- WebLogic Server クラスタ環境での実行

次の図に、WebLogic Integration Data Integration ツールと BPM の関係を示します。

図 6-1　BPM への実行時プラグイン



A　サポートされるデータ型

WebLogic Integration は、次のデータ型をサポートします。

- MFL のデータ型
- COBOL Copybook Importer データ型
- メタデータのインポートからの C Structure Importer

この節では、これらのデータ型について説明します。

MFL のデータ型

表 A-1 では WebLogic Integration によってサポートされる MFL データ型を一覧で表記しています。これらの型は、FieldFormat 要素の `type` 属性に指定されます。

表 A-1 サポートされる MFL データ型

データ型	説明	例
バイナリ（Base64 エンコーディング）	任意の文字の値を入力できる。長さ、長さフィールド、区切り記号、または区切り記号フィールドが必要。このフィールドの結果の XML データは、Base64 を使用してエンコードされる。	なし
バイナリ（16 進エンコーディング）	任意の文字の値を入力できる。長さ、長さフィールド、区切り記号、または区切り記号フィールドが必要。このフィールドの結果の XML データは、Base16 を使用してエンコードされる。	なし

表 A-1 サポートされる MFL データ型 （続き）

データ型	説明	例
DateTime: *MM/DD/YY hh:mm*	日付と時刻を定義する文字列。	01/22/00 12:24
DateTime: *MM/DD/YY hh:mi* AM	日付と時刻を定義する文字列。	01/22/00 12:24 AM
DateTime: *MM/DD/YY hh:mm:ss*	日付と時刻を定義する文字列	01/22/00 12:24:00
DateTime: *MM/DD/YY hh:mm:ss* AM	日付と時刻を定義する文字列。	01/22/00 12:24:00 AM
DateTime: *DD/MM/YY hh:mm*	日付と時刻を定義する文字列。	22/01/00 12:24
DateTime: *DD/MM/YY hh:mm* AM	日付と時刻を定義する文字列。	22/01/00 12:24 AM
DateTime: *DD/MM/YY hh:mm:ss*	日付と時刻を定義する文字列。	22/01/00 12:24:00
DateTime: *DD/MM/YY hh:mm:ss* AM	日付と時刻を定義する文字列。	22/01/00 12:24:00 AM
DateTime: *MMDDYYhhmm*	日付と時刻を定義する数字列。	0122001224
DateTime: *YYYYMMDDhhmmss*	*YYYYMMDDHHMISS* フォーマットの 14 バイトの数字列。基本 データ型を指定可能。	
DateTime: *MMDDYYhhmmss*	日付と時刻を定義する数字列。	012200122400
Date: *DDMMMYY*	日付を定義する文字列。	22JAN00
Date: *DDMMMYYYY*	日付を定義する文字列。	22JAN2000
Date: *DD/MM/YY*	日付を定義する文字列。	22/01/00
Date: *DD/MM/YYYY*	日付を定義する文字列。	22/01/2000
Date: *DD-MMM-YY*	日付を定義する文字列。	22-JAN-00
Date: *DD-MMM-YYYY*	日付を定義する文字列。	22-JAN-2000
Date: *MMDDYY*	日付を定義する 6 桁の数字列。	012200
Date: *MMDDYYYY*	日付を定義する 8 桁の数字列。	01222000

表 A-1 サポートされる MFL データ型 （続き）

データ型	説明	例
Date: *MM/DD/YY*	日付を定義する文字列。	01/22/00
Date: *MM/DD/YYYY*	日付を定義する文字列。	01/22/2000
Date: *MMM-YY*	日付を定義する文字列。	JAN-00
Date: *MMM-YYYY*	日付を定義する文字列。	JAN-2000
Date: *MMMYY*	日付を定義する文字列。	JAN00
Date: *MMMYYYY*	日付を定義する文字列。	JAN2000
Date: *MMMDDYYYY*	日付を定義する文字列。	JAN222000
Date: *YYYYMMDD*	*YYYYMMDD* フォーマットの 8 バイトの数字列。文字のエンコーディングを示すために、文字列または EBCDIC タイプの基本データを指定できる。	
Date: Wed Nov 15 10:55:37 CST 2000	Java プラットフォームのデフォルト日付フォーマット	'WED NOV 15 10:55:37 CST 2000'
EBCDIC	IBM の EBCDIC (Extended Binary Coded Decimal Interchange Code) の文字列。長さ、長さフィールド、区切り記号、または区切り記号フィールドが必要。	
フィラー	XML に変換されないバイト シーケンス。バイナリ データを XML に変換するときは、このデータ フィールドは省略される。XML をバイナリ データに変換するとき、このフィールドには一連のスペースでバイナリ出力ストリームに書き込まれる。	
浮動小数点：4 バイト、ビッグ エンディアン	IEEE 標準 754 に準拠する 4 バイト ビッグ エンディアンの浮動小数点数。	

表 A-1 サポートされる MFL データ型 （続き）

データ型	説明	例
浮動小数点 : 4 バイト、リトル エンディアン	IEEE 標準 754 に準拠する 4 バイト リトル エンディアンの浮動小数点数。	
浮動小数点 : 8 バイト、ビッグ エンディアン	IEEE 標準 754 に準拠する 8 バイト ビッグ エンディアンの浮動小数点数。	
浮動小数点 : 8 バイト、リトル エンディアン	IEEE 標準 754 に準拠する 8 バイト リトル エンディアンの浮動小数点数。	
浮動小数点 IBM 4 バイト IBM 8 バイト	IBM のメインフレーム用浮動小数点	
整数 : 符号付き、1 バイト	1 バイトの符号付き整数	56 は 0x38 になる。
整数 : 符号なし、1 バイト	1 バイトの符号なし整数	128 は 0x80 になる。
整数 : 符号付き、2 バイト、ビッグ エンディアン	ビッグ エンディアン フォーマットの符号付き 2 バイト整数	4660 は 0x1234 になる。
整数 : 符号付き、4 バイト、ビッグ エンディアン	ビッグ エンディアン フォーマットの符号付き 4 バイト整数	4660 は 0x00001234 になる。
整数 : 符号付き、8 バイト、ビッグ エンディアン	ビッグ エンディアン フォーマットの符号付き 8 バイト整数	4660 は 0x0000000000001234 になる。
整数 : 符号なし、2 バイト、ビッグ エンディアン	ビッグ エンディアン フォーマットの符号なし 2 バイト整数	65000 は 0xFDE8 になる。
整数 : 符号なし、4 バイト、ビッグ エンディアン	ビッグ エンディアン フォーマットの符号なし 4 バイト整数	65000 は 0x0000FDE8 になる。
整数 : 符号なし、8 バイト、ビッグ エンディアン	ビッグ エンディアン フォーマットの符号なし 8 バイト整数	65000 は 0x000000000000FDE8 になる。

表 A-1 サポートされる MFL データ型 （続き）

データ型	説明	例
整数：符号付き、2 バイト、リトル エンディアン	リトル エンディアン フォーマットの符号付き 2 バイト整数	4660 は 0x3412 になる。
整数：符号付き、4 バイト、リトル エンディアン	リトル エンディアン フォーマットの符号付き 4 バイト整数	4660 は 0x34120000 になる。
整数：符号付き、8 バイト、リトル エンディアン	リトル エンディアン フォーマットの符号付き 8 バイト整数	4660 は 0x3412000000000000 になる。
整数：符号なし、2 バイト、リトル エンディアン	リトル エンディアン フォーマットの符号なし 2 バイト整数	65000 は 0xE8FD になる。
整数：符号なし、4 バイト、リトル エンディアン	リトル エンディアン フォーマットの符号なし 4 バイト整数	65000 は 0xE8FD0000 になる。
整数：符号なし、8 バイト、リトル エンディアン	リトル エンディアン フォーマットの符号なし 8 バイト整数	65000 は 0xE8FD000000000000 になる。
リテラル	値の属性の内容によって決定されるリテラル値。バイナリ データを XML に変換するとき、バイナリ データ内の指定されたリテラルの有無が WebLogic Integration で確認される。リテラルは読み込まれるが、XML データには変換されない。XML データをバイナリ フォーマットに変換する際に、リテラルがそのフォーマットの一部として定義されている場合、WebLogic Integration により、そのリテラルは最終的なバイナリ バイト ストリームに書き込まれる。	
数値	数字（0 ～ 9）のみを含む文字列。長さ、長さフィールド、区切り記号、または区切り記号フィールドが必要。	

表 A-1 サポートされる MFL データ型 （続き）

データ型	説明	例
パック 10 進数 : 符号付き	符号付きの IBM パック形式。長さ、長さフィールド、区切り記号、または区切り記号フィールドを指定する必要がある。長さまたは長さフィールドには、このフィールドのサイズをバイト数で指定する。	
パック 10 進数 : 符号なし	符号なしの IBM パック形式。長さ、長さフィールド、区切り記号、または区切り記号フィールドを指定する必要がある。長さまたは長さフィールドには、このフィールドのサイズをバイト数で指定する。	
文字列	文字セット。長さ、長さフィールド、区切り記号、または区切り記号フィールドが必要。「文字列」のデータ型に対して長さを指定しない場合、長さフィールドまたは区切り記号が定義され、区切り記号として `"\x00"` (ヌル文字) を指定したものと見なされる。	
文字列 : ヌル文字で終了	固定長フィールド内の文字列、オプションでヌル文字（`\x00`）で終了。このフィールド型では、フィールドで読み出すデータのサイズを決定する長さ属性または長さフィールドが必要。その後、このデータのヌル区切り記号の有無が調べられる。区切り記号が見つかった場合、その区切り記号の後のデータは破棄される。そうでない場合は、フィールドの値として、固定長のデータが使用される。	
Time: *`hhmmss`*	時刻を定義する文字列。	122400
Time: *`hh:mm`* AM	時刻を定義する文字列。	12:24 AM
Time: *`hh:mm`*	時刻を定義する文字列。	12:24

表 A-1 サポートされる MFL データ型 （続き）

データ型	説明	例
Time: *hh:mm:ss* AM	時刻を定義する文字列。	12:24:00 AM
Time: *hh:mm:ss*	時刻を定義する文字列。	12:24:00
ゾーン 10 進数： 先行符号	符号インジケータが最初のニブルに含まれる符号付きの IBM ゾーン 10 進数形式。長さ、長さフィールド、区切り記号、または区切り記号フィールドを指定する必要がある。長さまたは長さフィールドには、このフィールドのサイズをバイト数で指定する。	
ゾーン 10 進数： Leading separate sign	符号インジケータが最初のバイトに含まれる符号付きの IBM ゾーン 10 進数形式。第 1 バイトには符号インジケータのみが含まれ、数値と分離される。長さ、長さフィールド、区切り記号、または区切り記号フィールドを指定する必要がある。長さまたは長さフィールドには、このフィールドのサイズをバイト数で指定する。	
ゾーン 10 進数： 符号付き	符号付きの IBM ゾーン 10 進数形式。長さ、長さフィールド、区切り記号、または区切り記号フィールドを指定する必要がある。長さまたは長さフィールドには、このフィールドのサイズをバイト数で指定する。	

表 A-1 サポートされる MFL データ型 （続き）

データ型	説明	例
ゾーン 10 進数 : Trailing separate sign	符号インジケータが最終バイトに含まれる符号付きの IBM ゾーン 10 進数形式。最終バイトには符号インジケータのみが含まれ、数値と分離される。長さ、長さフィールド、区切り記号、または区切り記号フィールドを指定する必要がある。長さまたは長さフィールドには、このフィールドのサイズをバイト数で指定する。	
ゾーン 10 進数 : 符号なし	符号なしの IBM ゾーン 10 進数形式。長さ、長さフィールド、区切り記号、または区切り記号フィールドを指定する必要がある。長さまたは長さフィールドには、このフィールドのサイズをバイト数で指定する。	

COBOL Copybook Importer データ型

COBOL データ型と Importer によるサポートの有無を次の表で示します。

表 A-2 COBOL のデータ型

COBOL の型	サポートの有無
BLANK WHEN ZERO (ゾーン 10 進数)	サポートする
COMP-1、COMP-2 (浮動小数点)	サポートする
COMP-3、パック 10 進数	サポートする

表 A-2 COBOL のデータ型 （続き）

COBOL の型	サポートの有無
COMP、COMP-4、バイナリ（整数）	サポートする
COMP、COMP-4、バイナリ（固定長）	サポートする
COMP-5、COMP-X	サポートする
DISPLAY (英数字)	サポートする
DISPLAY 数値（ゾーン 10 進数）	サポートする
編集済み英数字	サポートする
編集済み浮動小数点数値	サポートする
編集済み数値	サポートする
グループ レコード	サポートする
INDEX	サポートする
JUSTIFIED RIGHT	無視される
OCCURS (固定配列)	サポートする
OCCURS DEPENDING (可変長)	サポートする
OCCURS INDEXED BY	無視される
OCCURS KEY IS	無視される
POINTER	サポートする
PROCEDURE-POINTER	サポートする
REDEFINES	サポートする
SIGN IS LEADING SEPARATE (ゾーン 10 進数)	サポートする
SIGN IS TRAILING (ゾーン 10 進数)	サポートする
SIGN IS TRAILING SEPARATE (ゾーン 10 進数)	サポートする
SIGN IS LEADING (ゾーン 10 進数)	サポートする

表 A-2 COBOL のデータ型 （続き）

COBOL の型	サポートの有無
SYNCHRONIZED	無視される
66 RENAMES	無視される
66 RENAMES THRU	無視される
77 レベル	サポートする
88 レベル (条件付き)	無視される

これらのデータ型のサポートは、限定されています。次のいずれかのフォーマットで提供されたデータは、符号なし 4 バイト整数型にコンバートされます。

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

次のフォーマットで提供されるデータはエラーを発生させます。

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

上記のサンプルでは、`pic9(5)` を `pic x(5)` の代わりに使用できます。

次の表に、これらのデータ型サポートについての 3 つのレベルを定義します。

表 A-3 データ型サポートのレベル

サポートのレベル	定義
サポートする	このデータ型は Importer によって正しく解析され、メッセージ フォーマット フィールドまたはグループにコンバートされる。
サポートしない	このデータ型はサポートされず、Importer によってコピーブックのインポート時にエラーが報告される。
無視される	このデータ型は解析され、メッセージ フォーマットにコメントが追加される。対応するフィールドやグループは作成されない。

ベンダ固有の一部の拡張子は、Importer で認識されません。ただし、ANSI 標準 COBOL に準拠するコピーブック ステートメントは正しく解析されます。IBM のメインフレーム モデルを基にした Importer のデフォルト データ モデルを Format Builder で変更することで、文字セットやデータのエンディアン特性を補正できます。

これらのフィールドは特定のデータ型として容易に識別できますが、コピーブックをインポートする際、Importer がこれらのフィールドを一般的に識別することがあります。このためにコピーブックの Importer は、コピーブック内の各フィールドに対してコメントを作成します。MFL データを編集する際、この情報は有用で、元のコピーブックのフォーマットの改善に役立ちます。たとえば、元のコピーブックに次のエントリがあるとします。

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

コピーブックがインポートされた後、このエントリは、8 バイト長の EBCDIC 型のフィールドに表示されます。よく調べると、このエントリが、次のいずれかの方法でフォーマットできる日付であることがわかります。

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

メタデータのインポートからの C Structure Importer

C Structure Importer では、匿名ユニオン、ビットフィールド、またはインライン アセンブラ コードを含むファイルは解析されません。以下のサポートされない構造体のサンプルは #include <windows.h> ステートメントを含む hello.c ファイルのプリプロセッサ出力からのものです。

- 匿名ユニオン

```
#line 353 “e:\\program files\\microsoft visual
studio\\vc98\\include\\winnt.h”
typedef union_LARGE_INTEGER{
  struct {
    DWORD LowPart;
    LONG HighPart;
  };
  struct {
```

```
    DWORD LowPart;
    LONG HighPart;
  } u;
#line 363 “e:\\program files\\microsoft visual
studio\\vc98\\include\\winnt.h”
    LONGLONG QuadPart;
} LARGE_INTEGER
```

- ビット フィールド

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

- インライン アセンブラ コード

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

```
  }
}
```

B　カスタム データ型の作成

WebLogic Integration は、データ統合を行う際、XML に基づく MFL (Message Format Language : メッセージ フォーマット言語) と呼ばれるメタデータ言語を使用して、バイナリ データの構造について記述します。Format Builder はメタデータをデータ ファイルとして、すなわち MFL ドキュメントとして作成し、リポジトリ内に維持します。

MFL では、以下のメタデータによりバイナリ フィールドが記述されます。

- データ型
- 長さ / 区切り記号
- 省略可能 / 必須
- デフォルト値
- コード ページ エンコーディング

この情報のうち、データ型が最も重要です。選択されたデータ型により、有効なメタデータ属性とその解釈方法が決まります。

WebLogic Integration の Data Integration コンポーネントには、自分だけに必要なカスタム データ型を作成できるユーザ定義型機能が含まれます。ユーザ定義型機能により、これらのカスタム データ型をデータ変換実行時エンジンにプラグインできます。いったんプラグインされたユーザ定義のデータ型は、特徴や機能の面で組み込みデータ型と区別できません。

以下の各節では、ユーザ定義型の使用法について説明します。

- ユーザ定義型のサンプル
- ユーザ定義型の登録
- ユーザ定義型の作成
- Data Integration プラグイン用ユーザ定義型のコンフィグレーション
- ユーザ定義型コーディング要件
- com.bea.wlxt.mfl.MFLField クラス

ユーザ定義型のサンプル

次の表では、ユーザ定義型を使用してインストールされたサンプル ファイルについて説明します。すべてのディレクトリ名は、WebLogic Integration サンプル ディレクトリ（*SAMPLES_HOME*\integration）からの相対パスで示します。ここで、*SAMPLES_HOME* は、WebLogic Platform をインストールした位置にあるサンプル ディレクトリを示します。

表 B-1 ユーザ定義型のサンプル

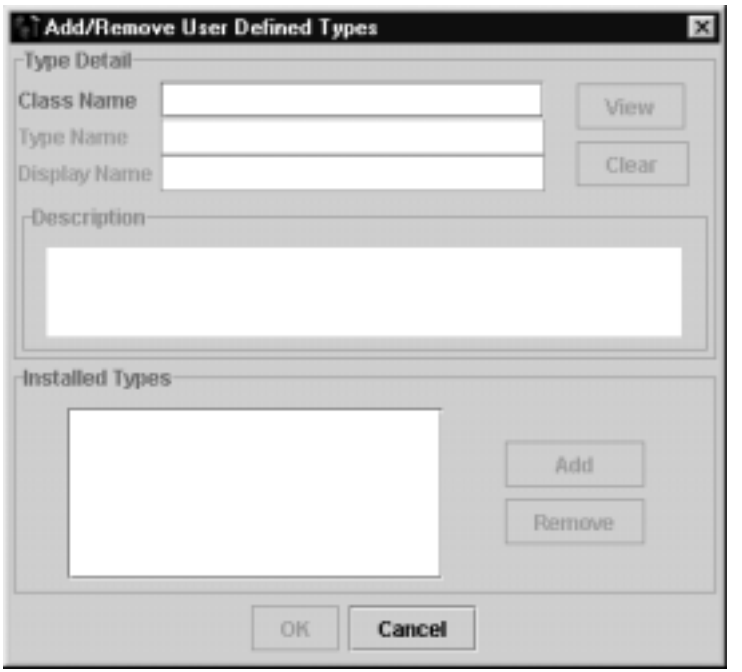
ディレクトリ	ファイル	説明
samples\di\userdef	CapString.java	文字列を大文字に変換するユーザ データ型のソース
samples\di\userdef	Dddmmyy.java	欧州の日付書式をサポートするユーザ定義型のソース
samples\di\userdef	Makefile	サンプル ソースをビルドするためのメイク ファイル
samples\di\userdef	ParseUserDef.java	サンプル ユーザ定義型のインストールと実行時エンジンでの使用を示すソース
samples\di\userdef	readme.txt	サンプルをコンパイルして実行する方法に関する説明
samples\di\userdef	sample.data	ParseUserData サンプルのデータ
samples\di\userdef	sample.mfl	ParseUserData サンプルに関するメッセージ フォーマット言語（MFL）ファイル

ユーザ定義型の登録

新しいユーザ定義型を登録する手順は次のとおりです。

1. [スタート | プログラム | BEA WebLogic Platform 7.0 | WebLogic Integration 7.0 | Format Builder] を選択して Format Builder を呼び出します。Format Builder のメイン ウィンドウが表示されます。
2. [Tools | User Defined Types] を選択します。[Add/Remove User Defined Types] ダイアログ ボックスが表示されます。

図 B-1 [Add/Remove User Defined Types] ダイアログ ボックス



3. [Class Name] フィールドに、指定した型を実装するクラスのクラス名を入力します。

   注意： 入力するクラス名には、必ずモジュールの定義にあるパッケージ名が含まれます。また、命名されたクラスは、Format Builder の CLASSPATH に存在する必要があります。WebLogic Integration により

Format Builder がインストールされると、特にユーザ定義型を格納するための *WLI_HOME*/ext ディレクトリが作成されます。ここで、*WLI_HOME* ディレクトリは、WebLogic Integration がインストールされているディレクトリです。

4. [View] をクリックします。指定されたクラスに関する情報が、次のように複数の表示専用フィールドに取り込まれます。

図 B-2　[Add/Remove User Defined Types] ダイアログ ボックス

- [Type Name] － `getTypeName()` への呼び出しによって返される。
- [Display Name] －リテラル `User Defined:` でプレフィックスされた `getDisplayName()` の戻り値。
- [Description] － `getDescriptionText()` によって戻される。

指定されたクラスがロードできない場合、あるいはユーザ定義型の要件に準拠しない場合には、エラー メッセージが表示されます。[OK] をクリックし [Add/Remove User Defined Types] ダイアログ ボックスに戻ります。

ユーザ定義型で使用できる必須のインタフェース メソッドと省略可能なインタフェース メソッド、およびユーティリティ メソッドの一覧については、「ユーザ定義型コーディング要件」を参照してください。

5. 有効なユーザ定義型が表示されたら、[Add] をクリックして Format Builder 内で使用できるようにします。

新しいデータ型を定義すると、フィールドの詳細ウィンドウの [Type] ドロップダウン リストに新しい表示名が表れます。[Type] ドロップダウン リスト ボックスに表示されるユーザ定義型にはすべて、[DisplayName] テキスト フィールドと同じように [*User Defined*] のプレフィックスが付きます。

Format Builder は、ユーザ定義型で示される正確な型を認識できないので、XML 出力のコンテンツ モデルの記述のために生成される XML スキーマでは、`xsd::string` 型を使用してユーザ定義型を表します。

このデータ型を使用することは、Format Tester にも影響を及ぼします。ユーザ定義型を含む MFL ドキュメントに対するデータを生成する場合、文字列データが対応するフィールドに対して生成されます。生成されたデータは、ユーザ定義型に応じて解析できるように調整する必要があります。

ユーザ定義型の作成

データ変換エンジンへのインタフェースは Java プログラムによって呼び出される API です。プロセス エンジンの新しいユーザ定義型の作成は、`com.bea.wlxt.WLXT` クラスの静的メソッドによって実行されます。

データ変換エンジンでのユーザ定義型のインストールは永続的ではありません。現在の JVM プロセスが終了すると、ユーザ定義型のコンフィグレーション情報は破棄されます。そのため、スタンドアロン エンジンを使用するクライアントでは、各プログラムの開始時にユーザ定義型をすべてインストールする必要があります。

ユーザ定義型には、以下のパブリック機能が定義されています。

- `public static void addNewDataType(String` *`name`*`, Bintype` *`binType`*`)`
  - *`name`* は、MFL における新しいデータ型の名前を指定する。
  - *`binType`* は、新しいユーザ定義型オブジェクトを参照する。

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

  `name` は、削除されるデータ型の名前を指定する。

以下の例では、これらの API を使用してユーザ定義型 `CapString` をインストールして削除する方法を示します。

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

// データ型オブジェクトを作成して、インストールする。
Bintype udt = new CapString();
WLXT.addNewDataType(“UpperCaseString”, udt);
                        .
                        .
                        .

// 上でインストールされた udt を削除する。
WLXT.removeDataType(“UpperCaseString”);
```

Data Integration プラグイン用ユーザ定義型のコンフィグレーション

Business Process Management (BPM) の Data Integration プラグインで使用されるユーザ定義型は、WebLogic Integration リポジトリに CLASS ドキュメントとして格納されます。実行時に、Data Integration プラグインは必要に応じてリポジトリからユーザ定義データ型のクラスを読み込みます。さらに、Data Integration プラグインにより、アクティブなテンプレートをサポートするために必要になる MFL とクラス ファイルがエクスポートされるため、別の BPM インスタンスではテンプレートをそのままインポートできます。

Data Integration プラグインは MFL ドキュメントと必要なユーザ定義型クラスの両方を実行時にリポジトリから取得します。クラス ドキュメントは、次のどちらかの方法を使用してリポジトリに格納できます。

- Format Builder からユーザ定義型をリポジトリに発行する。
- リポジトリ インポート ユーティリティを使用する。

ユーザ定義型の Format Builder からリポジトリへの発行

ユーザ定義型をリポジトリに発行する手順は次のとおりです。

1. [スタート | プログラム | BEA WebLogic Platform 7.0 | WebLogic Integration 7.0 | Format Builder] を選択して Format Builder を呼び出します。Format Builder のメイン ウィンドウが表示されます。
2. [Repository | Log In] を選択します。[WebLogic Integration リポジトリへのログイン] ダイアログ ボックスが表示されます。

図 B-3 [WebLogic Integration リポジトリへのログイン] ダイアログ ボックス

3. [ユーザ名] フィールドに、接続用に指定された User ID を入力します。
4. [パスワード] フィールドに、接続用に指定されたパスワードを入力します。
5. [サーバ [: ポート]] フィールドに、サーバーの名前とポート番号を入力します。

注意： ログインを 3 回試みて失敗すると、[WebLogic Integration リポジトリへのログイン] ダイアログ ボックスが終了し、ログイン失敗を示すメッセージが表示されます。失敗した後再度ログインを試みるには、

[Repository ｜ Log In] を選択して、[WebLogic Integration リポジトリへのログイン] ダイアログ ボックスを再表示します。

6. [接続] をクリックします。ログインに成功すると、ログイン ウィンドウが閉じ、Format Builder のタイトル バーに [WebLogic Integration リポジトリへのログイン] ダイアログ ボックスに入力したサーバ名とポート番号が表示されます。ここで、アクティブなリポジトリ メニュー 項目のいずれかにアクセスできます。

7. [Tools ｜ User Defined Types] を選択します。[Add/Remove User Defined Types] ダイアログボックスが開きます。

   リポジトリとの接続が確立されていると、[Add/Remove User Defined Types] ダイアログボックスには登録済みの各ユーザ定義データ型のステータスが表示され、リポジトリに公開できます。ユーザ定義データ型のリポジトリ ステータスは、登録済みの各ユーザ定義データ型の型名の前にある球型のアイコンによって示されます。

図 B-4　[Add/Remove User Defined Types] ダイアログ ボックス

各ユーザ定義型のステータスは、アイコンの色で示されます。

表 B-2 アイコンの色とステータス

アイコンの色	ステータス
緑色	ユーザ定義データ型は、リポジトリに公開されている。
黄色	ユーザ定義データ型は、リポジトリに公開されているが、ローカルのクラスとリポジトリのクラスが一致していない。
赤色	ユーザ定義データ型は、リポジトリ内に存在しない。

8. 登録されている型の一覧から、公開するクラスを選択して、[Publish] をクリックします。選択したエントリのアイコンの色が緑になるはずです。これは、そのクラスがリポジトリの中に正常に格納されたことを意味します。

リポジトリ インポート ユーティリティを使用したユーザ定義型のリポジトリへの発行

リポジトリ インポート ユーティリティを使用して、データ変換ユーザ定義型を含む Java クラス ファイルをインポートできます。このためにインポート ユーティリティを使用する場合、クラス ファイルの名前を、Import コマンド ラインに渡してください。たとえばカレント ディレクトリのクラス ファイルをすべてインポートするときは、次のように指定します。

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

任意の Java クラス ファイルをリポジトリにインポートできることに注意してください。Format Builder で作成されたデータ変換ユーザ定義型クラス ファイルではそうはいきません。この機能は、たとえばユーティリティ定義型が `com.bea.wlxt.bintype.Bintype` クラスを拡張しない追加クラス ファイルに依存する場合に有効です。リポジトリ インポート ユーティリティを使用することにより、これらのユーティリティ クラスをリポジトリに格納できます。リポジトリでは、これらのクラスにはリポジトリ クラス ローダでアクセスできます。

ユーザ定義型コーディング要件

ユーザ定義型は `com.bea.wlxt.bintype.Bintype` 抽象クラス、またはその派生クラスの 1 つを拡張するために必要です。`Bintype` クラスは、WebLogic Integration の Data Integration コンポーネントがデータ型との通信に使用する基本的な枠組みを提供します。このクラスは、バイナリ データ型の処理に役立つユーティリティ ルーチンも提供します。さらに、`BintypeDate` および `BintypeString` という `Bintype` の 2 つのサブクラスにより、それぞれ日付と文字列型に関する追加のユーティリティ ルーチンが提供されます。

次のクラスは、そのクラスが提供する必須および省略可能なインタフェース メソッドとともに、ユーザ定義型で使用できます。

com.bea.wlxt.bintype.Bintype クラス

`com.bea.wlxt.bintype.Bintype` クラスは、以下の必須メソッド、省略可能メソッドおよびユーティリティ メソッドで構成されます。

必須インタフェース ルーチン

WebLogic Integration のユーザ定義データ型ユーティリティを使用するときは、以下のインタフェース メソッドが必要です。

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

このルーチンはデータ値をバイナリ データ ストリームから読み込むときに呼び出されます。読み込みメソッドによりデータ型に該当するバイト数が読み取られ、それが文字列表現に変換され、文字列値が返されます。***`mflField`*** パラメータは、データ フィールドの属性を示す `com.bea.wlxt.mfl.MFLField` オブジェクトへの参照です。

ユーザ定義型によって、指定されたデータ要素が読み取られない場合、`com.bea.wlxt.bintype.BintypeException` が発生したエラーを説明するテキスト文字列とともに返されます。

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

書き込みメソッドにより、文字列値がバイナリ データ型のデータ表現にコンバートされ、バイナリ出力ストリームに書き込まれます。

ユーザ定義型によって、指定されたデータ要素が正常に書き込まれない場合、`com.bea.wlxt.bintype.BintypeException` が発生したエラーを説明するテキスト文字列と一緒に返されます。

省略可能なインタフェース ルーチン

データ変換ユーザ定義データ型ユーティリティを使用するときは、以下のインタフェース ルーチンを省略できます。

`public boolean canBeFieldType()`

ユーザ定義データ型をデータ フィールドの型として使用できるときは、true が返されます。

デフォルト値 : true

`public boolean canBeLenFieldType()`

ユーザ定義型を [Length] フィールドのデータ型として使用できるときは、true が返されます。

デフォルト値 : true

`public boolean canBeTagFieldType()`

ユーザ定義型を [Tag] フィールドの型として使用できるときは、true が返されます。

デフォルト値 : true

`public boolean canBeDelimited()`

ユーザ定義型が区切られたデータ値をサポートする場合は、true が返されます。

デフォルト値 : false

`public boolean isFixedSize()`

ユーザ定義型が固定サイズ データ値を表わすとき、true が返されます。このメソッドから true が返されることは、データ型に固有の固定サイズがあることを表します。

デフォルト値 : true

`public boolean isDateType()`
ユーザ定義型が日付データ型を表すときは、true が返されます。

デフォルト値 : false

`public boolean isCutoffRequired()`
ユーザ定義型が日付データ型で、2 桁の年を調整するためにカットオフ値が必要なときは、true が返されます。このメソッドが使用されるのは、`isDateType()` によって true が返される場合だけです。

デフォルト値 : false

`public boolean isCodepageOK()`
ユーザ定義型がコード ページをサポートするときは、true が返されます。

デフォルト値 : false

`public boolean isValueOK()`
ユーザ定義型が初期値属性をサポートするときは、true が返されます。

デフォルト値 : false

`public boolean canHaveDecimalPlaces()`
ユーザ定義型が小数位のある数字を表すときは、true が返されます。

デフォルト値 : false

`public boolean canBePadded()`
ユーザ定義型が固定長データ値を埋め込むときは、true が返されます。

デフォルト値 : false

`public boolean canBeTrimmed()`
ユーザ定義型が先頭と末尾の削除をサポートするときは、true が返されます。

デフォルト値 : false

`public String getDescriptiontext()`
このデータ型の関数を説明するテキストを含む文字列が返されます。このメソッドはドキュメント用のユーザ定義型で実装する必要があります。

デフォルト値 : 空の文字列

`public String getTypeName()`
データ型名を含む文字列を返します。この関数は、ユーザ定義型とともに使用され、その戻り値はMFLドキュメントで使用される型名です。

デフォルト値:ユーザ定義型を実装するクラスのクラス名

`public String getDisplayName()`
データ型名に対するDisplay Nameを含む文字列を返します。この値は、フィールドの詳細ウィンドウの[Type]ドロップダウン リストに表されます。

デフォルト値:ユーザ定義型を実装するクラスのクラス名

ユーティリティ インタフェース ルーチン

データ変換ユーザ定義データ型ユーティリティを使用するときは、以下のユーティリティ インタフェース ルーチンを使用できます。

`public static byte[ ] getBinaryBytes (String str)`
各文字列の下位8ビットを採用することによりJava文字列をバイナリバイトの配列にコンバートします。コードページに基づく変換は実行されません。

`public static String makeString(byte[ ] b)`
バイナリ バイトの配列をJava文字列値に変換します。コードページに基づく変換は実行されません。

`protected void reverseBytes (byte[ ] data)`
バイト オーダーとバイナリ値の配列を反転します。

`protected String readTag(InputStream byteStream, MFLField fld) throws BinTypeexception`
データ フィールドと関連付けられたタグ値を読み込み、返します。このルーチンにより、読み取られたタグ値が`fld`オブジェクトの期待値と比較され、一致しなければ`Bintypeexception`が返されます。タグ値をサポートしない`fld`にこのメソッドを呼び出したときは、空の文字列が返されます。

`protected int readlength(InputStream byteStream, MFLField fld) thorws BintypeException`
データ フィールドと関連付けられた長さフィールドが読み込まれ、返されます。長さフィールドをサポートしないフィールドで呼び出されたときは、ゼロが返されるだけです。

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

fld と関連付けられたタグ フィールドを指定された ByteStream に書き込みます。***fld*** でタグ値がサポートされなければ、このメソッドでは何も書き込まれません。

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

fld と関連付けられた長さフィールドを指定された byteStream に書き込みます。***fld*** で長さフィールドがサポートされなければ、このメソッドでは何も書き込まれません。

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

mflField フィールドの区切り記号が 1 つ見つかるまで、指定された入力ストリームからデータを読み取ります。戻り値は、読み取られたバイナリ データから区切り記号値を除いたデータです。***mflField*** が区切り記号がないものとして定義されている場合、このメソッドではデータが読み取られずに null が返されます。

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

fld で定義された埋め込みと削除の機能を ***value*** として渡されるデータに適用します。***applyPad*** および ***applyTrim*** パラメータにより、実行するのが埋め込みなのか、削除なのか、それとも両方なのかがコントロールされます。戻り値は、指定された操作を実行したあとの結果データです。デフォルトでは、このメソッドはフィールドの切り捨てを実行しません。

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

fld で定義された埋め込み、削除、切り捨ての機能を ***value*** として渡されるデータに適用します。***applyPad***、***applyTrim*** および ***applyTruncate*** パラメータにより、実行するのが埋め込みなのか、削除なのか、それとも切り捨てなのかがコントロールされます。戻り値は、指定された操作を実行したあとの結果データです。

com.bea.wlxt.bintype.BintypeString クラス

com.bea.wlxt.bintype.BintypeString クラスは、以下の必須ルーチン、省略可能ルーチンおよびユーティリティ ルーチンで構成されます。

必須インタフェース ルーチン

com.bea.wlxt.bintype.Bintype クラスと同じです。

省略可能なインタフェース ルーチン

com.bea.wlxt.bintype.Bintype クラスと同じです。

ユーティリティ インタフェース ルーチン

com.bea.wlxt.bintype.Bintype クラスと同じで、以下のユーティリティ インタフェース ルーチンが加わります。

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

このメソッドでは、バイナリ データの配列が *mflField* で定義されたコードページに応じて Java String に変換されます。

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

このメソッドでは String データ型を表す値がバイナリ入力文字列から読み取られます。*mflField* で定義された長さ / 区切り記号の属性すべてが戻りデータ値の抽出に反映されます。

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

mflField で定義された属性と渡された *codepage* に応じて、指定された *byteStream* にデータ値が書き込まれます。*codepage* が null として渡されると、デフォルトのシステム コード ページが使用されます。

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

指定されたデータの両端から特定の文字が取り除かれ、それによってできた文字列が返されます。

`protected String trimLeading (String data, char trimChar)`
指定されたデータの先頭から特定の文字が取り除かれ、それによってできた文字列が返されます。

`protected String trimTrailing(String data, char trimChar)`
指定されたデータの末尾から特定の文字が取り除かれ、それによってできた文字列が返されます。

クラス com.bea.wlxt.bintype.BintypeDate

`com.bea.wlxt.bintype.BintypeDate` クラスは、以下の必須ルーチン、省略可能ルーチンおよびユーティリティ ルーチンで構成されます。

必須インタフェース ルーチン

`com.bea.wlxt.bintype.Bintype` クラスと同じです。

省略可能なインタフェース ルーチン

`com.bea.wlxt.bintype.Bintype` クラスと同じです。

ユーティリティ インタフェース ルーチン

`com.bea.wlxt.bintype.Bintype` クラスと同じで、以下のユーティリティ インタフェース ルーチンが加わります。

`protected static String readDate(String date, SimpleDateFormat fmt)`
fmt フォーマットに応じて文字列パラメータ日付から日付が解析されます。戻り値は文字列フォーマットの有効日付です。

`protected static String readDate(String date, SimpleDateFormat fmt, MFLField fld, int yearpos`
フォーマット *fmt* に応じて、文字列パラメータ日付から 2 桁の年を含む日付を解析します。

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

文字列パラメータ *`date`* からの出力に適する日付文字列を日付フォーマット *`fmt`* を使用して返します。

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

指定された基本データ型を使用して、未加工のバイナリ フォーマットの日付を Java String に変換します。

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

指定された基本データ型を使用して、Java String フォーマットの日付をバイナリ バイト配列に変換します。

com.bea.wlxt.mfl.MFLField クラス

`MFLField` クラスは、ユーザ定義型のすべての読み取りおよび書き込みメソッドに渡されます。MFLField クラスには、読み取りや書き込みが行われるフィールド用に定義された属性がすべて含まれます。`MFLField` はさまざまなメソッドを提供するため、これらの属性を照会し、各値を返すことができます。ユーザ定義型がサポートしていない属性が、含まれることはありません。たとえば、ユーザ定義型から false が `isValueOK()` メソッドに返される場合、そのユーザ定義型は `MFLField.hasValue()` メソッドに true を返した `MFLField` オブジェクトには渡されません。

`com.bea.wlxt.mfl.MFLField` クラスは、以下の MFLField メソッドをサポートします。

`public String getName()`

データ フィールドの名前を返します。

`public String getType()`

このフィールドのデータ型の名前を返します。この名前は、ユーザ定義型の `getTypeName()` メソッドにより返されます。

`public boolean hasBaseType()`

フィールドに基本型が定義されている場合、true が返されます。このメソッドでは日付データ型にのみ true が返されます。

`public String getBaseType`
　　このフィールドの基本データ型の名前を返します。

`public boolean hasDelimRef()`
　　このフィールドに区切り記号参照が定義されている場合、true を返します。

`public String getDelimRef()`
　　このフィールドに対する区切り記号値を含むフィールドのフィールド名を返します。

`public boolean hasdelimRefValue()`
　　区切り記号参照フィールドに値が含まれる場合、true を返します。

`public boolean hasDefaultValue()`
　　このフィールドにデフォルト値がある場合、true を返します。

`public String getDefaultValue()`
　　このフィールドのデフォルト値を返します。

`public boolean hasPad()`
　　このフィールドに埋め込み値が定義されている場合、true を返します。

`public String getPad()`
　　このフィールドの埋め込み値を返します。

`public boolean hasTrimLeading()`
　　このフィールドに先頭の削除値が定義されている場合、true を返します。

`public String getTrimLeading()`
　　このフィールドの先頭の削除値を返します。

`public boolean hasTrimTrailing()`
　　このフィールドに末尾の削除値が定義されている場合、true を返します。

`public String getTrimTrailing()`
　　このフィールドの末尾の削除値を返します。

`public boolean isOptional()`
　　このフィールドが省略可能と定義されている場合、true を返します

`public boolean hasCutoff()`
　　このフィールドに日付カットオフ値が定義されている場合、true を返します。

`public int getCutoff()`
　　このフィールドに定義された日付カットオフ値を返します。

`public boolean hasLength()`
このフィールドに正確な長さが定義されている場合、true を返します。

`public int getLength()`
このフィールドに定義された正確な長さを返します。

`public boolean hasDelim()`
このフィールドに区切り記号値が定義されている場合、true を返します。

`public String getDelim()`
このフィールドに定義された区切り記号値を返します。

`public boolean hasValue()`
このフィールドに初期値が定義されている場合、true を返します。

`public String getValue()`
このフィールドに定義された初期値を返します。

`public boolean hasCodepage()`
このフィールドにコード ページが定義されている場合、true を返します。

`public String getCodepage()`
このフィールドに定義されたコード ページの名前を返します。

`public boolean hasTagField()`
このフィールドにタグ フィールドが定義されている場合、true を返します。

`public boolean hasLenField()`
このフィールドに長さフィールドが定義されている場合、true を返します。

`public boolean isTagBeforeLength()`
フィールド タグ値がある場合にそれが長さフィールドの前に出現すると、true を返します。

`public boolean hasDecimalPlaces()`
このフィールドに小数位が定義される場合、true を返します。

`public int getDecimalPLaces()`
このフィールドに定義された小数位の桁数を返します。

`public String getPadType() { return(padType); }`
このフィールドに設定されている埋め込みの種類を返します (先頭または末尾)

`public boolean hasTruncateFirst() { return(0 != truncateFirst); }`
フィールドの先頭で、切り捨てられる文字の文字数 *n* が定義されている場合、true を返します。

`public int getTruncateFirst() { return(truncateFirst); }`
定義されている、このフィールドの先頭で切り捨てられる文字数を返します。

`public boolean hasTruncateAfter() { return(0 != truncateAfter); }`
フィールドの末尾で切り捨てられる文字の文字数 *n* が定義されている場合、true を返します。

`public int getTruncateAfter() { return(truncateAfter); }`
定義されている、このフィールドの末尾で切り捨てられる文字数を返します。

C　Purchase Order サンプルの実行

WebLogic Integration ソフトウェアには、Format Builder を使用してバイナリデータのメッセージ フォーマット定義を作成する基本テクニックを例示することを目的とする Purchase Order サンプルが含まれます。Purchase Order サンプルは DTD、MFL、および DATA ファイルから成ります。これらのサンプルは WebLogic Integration の Data Integration コンポーネントのインストールをテストするのに使用できます。

この章では、以下の内容について説明します。

- Purchase Order サンプルに含まれるもの
- 前提となる事項
- バイナリから XML への変換の実行
- XML からバイナリへの変換の実行

Purchase Order サンプルに含まれるもの

次の表では、Purchase Order サンプル アプリケーションに付属のファイルについて説明します。すべてのディレクトリ名は、WebLogic Integration サンプル ディレクトリ（*SAMPLES_HOME*\integration）からの相対パスで示します。ここで、*SAMPLES_HOME* は、WebLogic Platform をインストールした位置にあるサンプル ディレクトリを示します。

表 C-1 Purchase Order サンプル アプリケーション ファイルのリスト

ディレクトリ	ファイル	説明
samples\di\po	po_01.data	バイナリ フォーマットの発注書データ
	po_02.data	バイナリ フォーマットの追加発注書データ
	po.dtd	発注書文書型定義
	po.mfl	事前にビルドされた発注書データのメッセージ フォーマット記述

前提となる事項

Purchase Order サンプルを実行する前に、必ず、特定のソフトウェア アプリケーションをインストールし、特定のタスクを完了しておいてください。詳細については、『*WebLogic Integration リリース ノート*』を参照してください。

Format Builder の使い方を把握するには、WebLogic Integration に付属の Data Integration ツールで使用されるデータ フォーマット、すなわちバイナリ データ、XML、および MFL について理解しておくと役に立ちます。これらの項目についてまだよくわからない場合は、3-1 ページの「データ フォーマットについて」を参照してください。

バイナリから XML への変換の実行

以下の節では、サンプルの発注書のフォーマット定義をビルドして、バイナリデータの XML フォーマットへの変換をテストする方法について説明します。

- 変換するデータの分析
- フォーマット定義の作成と変換のテスト

バイナリ データと XML との間での変換に必要な情報を提供するフォーマット定義をビルドできます。フォーマット定義は、バイナリ データ ファイルの内容を解析するために使用されるメタデータです。

変換するデータの分析

XML との間でバイナリ データを変換するには、バイナリ データの正確な記述を作成することが必要です。バイナリ データ（自己記述型でないデータ）の場合、以下の要素を指定する必要があります。

- データ フィールド
- 名前、データ型、長さ / 終了、省略可能、繰り返しなどのデータ フィールド属性
- 関連するフィールドのグループ
- 名前、省略可能、繰り返し、区切りなどのグループ属性
- 階層構造（フィールドのグループや他のグループ）

コード リスト C-1 は、WebLogic Integration の CD-ROM (およびダウンロード) に収録されている `\samp;es\di\po\po_01.data` という名前のバイナリ データのサンプルを示します。この例では、サンプル データは XYZ Corporation が使用する専用システムにある架空の発注書から取得されます。この会社は、XML データを受け取る別のシステムと情報交換したいと考えています。

コード リスト C-1 バイナリ フォーマットのサンプル発注書

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

発注書 のデータを解析する手順は次のとおりです。

1. データの定義を生成します。データの定義を生成するには、印刷された仕様書または内部資料を使用しなければならない場合があります。このサンプルでは、発注書フォーマットを表 C-2 に記述してあります。

表 C-2 発注書マスタ レコード

カテゴリ	フィールド名	データ型	長さ	説明
Basic Information	Purchase Request Number	数値	セミコロンで区切る	購買部門によって割り当てられる発注依頼番号。この番号は、発注のステータスを購買依頼から納入、そして支払いまで追跡するために使用される。
	Supplier ID	数値	セミコロンで区切る	企業のサプライヤ データベースに定義されている指定サプライヤの ID。承認済みサプライヤの指定は、バイヤが購買依頼から発注依頼を作成するときに行う。
	Supplier Name	文字	リテラル “SUP” が前に付く。このフィールドの後ろに 2 桁の数字の長さフィールドが続く	企業のサプライヤ データベースに定義されている指定サプライヤの名前。このフィールドの前には、それが存在することを示すリテラルが付く。
	Requested Delivery Date	日付 *MM/DD/YYYY*	10 文字	購買依頼担当者が指定した納期

表 C-2 発注書マスタ レコード （続き）

カテゴリ	フィールド名	データ型	長さ	説明
Shipping Address	Street	文字	セミコロンで区切る	依頼された品目の配送に使用される町名
	City	文字	セミコロンで区切る	依頼された品目の配送に使用される市町村名
		文字	セミコロンで区切る	依頼された品目の配送に使用される州
	Zip	数値	セミコロンで区切る	依頼された品目の配送に使用される ZIP コード
Billing Address	Street	文字	セミコロンで区切る	請求に使用される住所（町名）
	City	文字	セミコロンで区切る	請求に使用される市町村名
	State	文字	セミコロンで区切る	請求に使用される州
	Zip	数値	セミコロンで区切る	請求に使用される ZIP コード
Payment Terms	サポートされる支払い条件は発注書かクレジット カード。支払い情報の前に付くリテラルがタイプを示す。			
	PO Type	文字	リテラル “PO”	PO の支払い条件を示す。
	PO Number	数値	セミコロンで区切る	発注書番号
	Credit Card Type	文字	リテラル “CC”	クレジット カードの支払い条件を示す。
	Credit Card Number	数値	セミコロンで区切る	クレジット カード番号
	Credit Card Expiration Month	数値	セミコロンで区切る	クレジット カードの有効期限の月
	Credit Card Expiration Year	数値	セミコロンで区切る	クレジット カードの有効期限の年

表 C-2 発注書マスタ レコード （続き）

カテゴリ	フィールド名	データ型	長さ	説明
Purchase Items	以下のフィールドでは、購入する品目を指定する。この情報は、同じ購買依頼に含まれる品目ごとに繰り返すことが可能。少なくとも 1 品目が必要。			
	Part Number	数値	セミコロンで区切る	依頼品目のサプライヤ部品番号
	Quantity	数値	セミコロンで区切る	依頼数量。ゼロより大きい数が必要。
	Description	文字	セミコロンで区切る	依頼された品目の説明

2. フィールドを確認します。

   フィールドとは、アプリケーションにとって意味のあるバイト シーケンスです。たとえば、図 C-2 の、Purchase Request Number、Supplier ID、Supplier Name などはすべてフィールドです。

3. フィールド属性を指定します。

   フィールド属性には、フィールド名、フィールドに格納されるデータの型、フィールドの長さ、またはフィールドの終了を示す区切り記号があります。たとえば、[Supplier ID] フィールドはセミコロン（;）で区切られ、フィールド データの終了を示しますが、[Requested Delivery Date] の長さは 10 桁に決まっています。

4. 階層グループを指定します。

   グループはなんらかの関連があるフィールド、コメントおよびその他のグループまたは参照の集合です。表 C-1 のサンプル データでは、はっきりと区別されるいくつかのグループ（Shipping Address、Billing Address、Payment Terms、および Purchase Items）が定義されています。

5. グループ属性を指定します。

   階層グループの属性を定義する必要があります。グループ属性には、グループの名前、グループが省略可能、繰り返し、区切りのいずれであるか、別のグループへの参照として定義されているかどうかが含まれます。たとえば、Shipping Address と Billing Address の両方で同じフィールド（`Street`、`City`, `State`、`Zip`）を定義する必要があるので、`Shipping_Address` グループの `Address` グループを定義して、`Billing_Address` グループから、`Address` グループへの参照を設定できます。

上記の手順を完了したら、図 C-1 のサンプルに示すように、データをスプレッドシートに入力できます。このスプレッドシートは、発注書のメッセージ定義を作成する際にガイドとして役立てることができます。

図 C-1　発注書データの分析

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

フォーマット定義の作成と変換のテスト

この節では、表 C-1 で説明したバイナリ データを XML に変換するために、メッセージ フォーマットを作成するプロセスを順を追って説明します。発注書のメッセージ フォーマットを正しく作成するために、作成したファイルを製品 CD-ROM に収録されている `\samples\di\po\po.mfl` ファイルと比較することができます。

手順 1. Format Builder の呼び出しとメッセージ フォーマットの作成

Format Builder を呼び出してメッセージ フォーマットを作成する手順は次のとおりです。

1. [スタート｜プログラム｜ BEA WebLogic Platform 7．0 ｜ WebLogic Integration 7．0 ｜ Format Builder] を選択します。Format Builder のメイン ウィンドウが表示されます。
2. [File ｜ New] を選択します。

   新しいメッセージ フォーマット ルート ノードが作成され、ナビゲーション ツリーに表示されます。
3. [Name/XML Root] フィールドに「`PurchaseRequest`」と入力します。
4. [Apply] をクリックします。

   メッセージ フォーマット ルート ノードの名前が更新されます。

手順 2. 基本情報フィールドの作成

発注書の基本識別情報を取り出すために必要なフィールドを作成する手順は次のとおりです。

1. [`PR_Number`] フィールドを作成するには、[Insert ｜ Field ｜ As Child] を選択します。

   ナビゲーション ツリーに新しいフィールドが追加されます。詳細ウィンドウに、デフォルトのフィールド プロパティが表示されます。
2. 次の表の説明に従ってフィールドのプロパティを定義します。

対象となるセクション	実行する手順
[Field Description]	[Name] フィールドに 「PR_Number」 と入力する。
	[Type] ドロップダウン リストから、[Numeric] を選択する。
Field Occurrence[Field Occurrence]	[Once] オプション ボタンが選択されていることを確認する。
[Field Attributes:][Termination]	[Delimiter] オプション ボタンを選択する。
	[Delimiter] タブの [Value] フィールドにセミコロン（;）を入力する。
[Field Attributes:][Code Page]	[Code Page] ドロップダウン リストから目的のコード ページを選択する。コード ページは、各フィールドのバイナリ データの文字エンコーディングを指定する。

注意： これらの値は、図 C-1 で示した発注書の生データを分析した結果決められました。

3. [Apply] をクリックします。

   フィールドのプロパティが更新されます。

   注意： [PR_Number] フィールドと [Supplier_ID] フィールドの違いは名前だけなので、[Supplier_ID] フィールドの作成には、Format Builder の複製機能を使用できます。

4. [PR_Number] フィールドを右クリックします。

5. ショートカット メニューから [Duplicate] を選択します。

   複製されたフィールド（NewPR_Number）が兄弟として追加され、ナビゲーション ツリーで選択されます。

6. [Name] フィールドに「Supplier_ID」と入力し、[Apply] をクリックして更新します。

7. メッセージ フォーマット ドキュメントに対して行った変更を保存する手順は次のとおりです。

   a. [File ｜ Save As] を選択して、[Save As] ダイアログ ボックスを表示します。

   b. `samples/di/po` ディレクトリに移動します。

   c. 任意のファイル名を入力します。たとえば、「`my_po.mfl`」と入力します。

      注意： 拡張子を指定しない場合、Format Builder により、自動的に `.mfl` 拡張子がメッセージ フォーマット ファイルに割り当てられます。

   d. [Save As] をクリックします。

8. [`Supplier_Name`] フィールドを追加するには、[Insert ｜ Field ｜ As Sibling] を選択します。

   ナビゲーション ツリーに新しいフィールドが追加されます。詳細ウィンドウに、デフォルトのフィールド プロパティが表示されます。

9. 次の表の説明に従って、[`Supplier_Name`] フィールドのプロパティを定義します。

対象となるセクション	実行する手順
[Field Description]	[Name] フィールドに「`Supplier_Name`」と入力する。
	[Optional] チェック ボックスをチェックする。
	[Type] ドロップダウン リストから、[`String`] を選択する。
[Field Occurrence]	[Once] オプション ボタンが選択されていることを確認する。
[Field Attributes:]	[Field is Tagged] チェックボックスをチェックする。
	[Field is Tagged] フィールドに次を入力する。 `SUP:`

対象となるセクション	実行する手順
[Field Attributes:][Termination]	[Embedded Length] オプション ボタンを選択する。
	[Description] タブの [Type] ドロップダウン リストで、[Numeric] を選択する。
	[Length] オプション ボタンが選択されていることを確認し、[Length] テキスト ボックスに「2」と入力する。

10. [Apply] をクリックします。

    フィールドのプロパティが更新されます。

    注意： ナビゲーション ツリーでフィールド アイコンの周囲が点線のボックスで表示される場合、そのフィールドは、省略可能です。

11. [Requested_Delivery_Date] フィールドを追加するには、[Insert | Field | As Sibling] を選択します。

    ナビゲーション ツリーに新しいフィールドが追加されます。詳細ウィンドウに、デフォルトのフィールド プロパティが表示されます。

12. 次の表の説明に従って、[Requested_Delivery_Date] フィールドのプロパティを定義します。

対象となるセクション	実行する手順
[Field Description]	[Name] フィールドに「Requested_Delivery_Date」と入力する。
	[Type] ドロップダウン リストから [Date:MM/DD/YYYY] を選択する。
[Field Occurrence]	[Once] オプション ボタンが選択されていることを確認する。
[Field Attributes:]	[Data Base Type] ドロップダウン リストで [String] が選択されていることを確認する。

注意： フィールドの詳細ウィンドウの内容は、[Type] 設定によって決まります。ドロップダウン リストからデータ型を選択すると、長さは指定しなくても決定されます。このため、[Field Attributes:] の [Termination] プロパティは表示されません。

13. [Apply] をクリックします。

    フィールドのプロパティが更新されます。

14. [File ｜ Save] を選択して、変更を保存します。

手順 3. Shipping Address グループと Billing Address グループの作成

Shipping Address グループと Billing Address グループを作成する手順は次のとおりです。

1. ナビゲーション ツリーで任意のフィールドを選択し、[Insert ｜ Group ｜ As Sibling] を選択します。

   ナビゲーション ツリーに新しいグループが追加されます。詳細ウィンドウに、デフォルトのグループ プロパティが表示されます。

2. 次の表の説明に従って、[Shipping_Address] グループのプロパティを定義します。

対象となるセクション	実行する手順
[Group Description]	[Name] フィールドに「Shipping_Address」と入力する。
[Group Occurrence]	[Once] オプション ボタンが選択されていることを確認する。
[Group Attributes]	[Group Delimiter] について [None] オプションが選択されていることを確認する。

注意： これらの値は 表 C-1 で示す発注書の生データを分析した結果決められました。

3. [Apply] をクリックします。

   グループのプロパティが更新されます。

   注意： [_Shipping_Address] グループと [Billing_Address] グループの違いは名前だけなので、[Billing_Address] のフィールドは、Format Builder の複製機能を使用して作成できます。

4. [Shipping_Address] グループを右クリックします。

5. ショートカット メニューから [Duplicate] を選択します。

   複製されたグループ（NewShipping_Address）が兄弟として追加され、ナビゲーション ツリーで選択されます。

6. [Name] フィールドに「Billing_Address」と入力し、[Apply] をクリックして更新します。

   [Shipping_Address] グループと [Billing_Address] グループは、同じ属性の同じフィールドを含むので、[Shipping_Address] グループで [Address] グループを定義して、[Billing_Address] グループでは [Address] グループを参照として設定できます。

7. ナビゲーション ツリーで [Shipping_Address] グループを選択し、[Insert | Group | As Child] を選択します。

8. 図 C-1 のデータを使用して、[Address] グループを作成し、[Apply] をクリックしてグループのプロパティを更新します。

9. 手順 2. 基本情報フィールドの作成、で説明する手順と図 C-1 のデータに従い、[Street]、[City]、[State_]、および [Zip] の各フィールドを [Address] グループの子として作成します。

   注意： [Street] フィールドを作成すれば、[Duplicate] ボタンを使用して [City] フィールドと [State] フィールドを作成できます。

10. [Address] グループへの参照を作成するには、[Shipping_Address] の下の [Address] グループを右クリックして、ショートカット メニューから [Copy] を選択します。

    [Address] グループのプロパティ（子フィールドを含む）がコピーされ、クリップボードに置かれます。

11. [Billing_Address] グループを右クリックして、ショートカット メニューから [Paste As Reference] を選択します。

    [Billing_Address] グループのすぐ後に、[Address] 参照が貼り付けられます。アイコンの矢印は、グループが参照グループであることを示します。

12. [Address] グループ参照を [Billing_Address] グループの子にするには、[Address] グループ参照を選択してから [Edit ｜ Demote] を選択します。

    [Address] 参照は、[Billing_Address] グループの子になります。

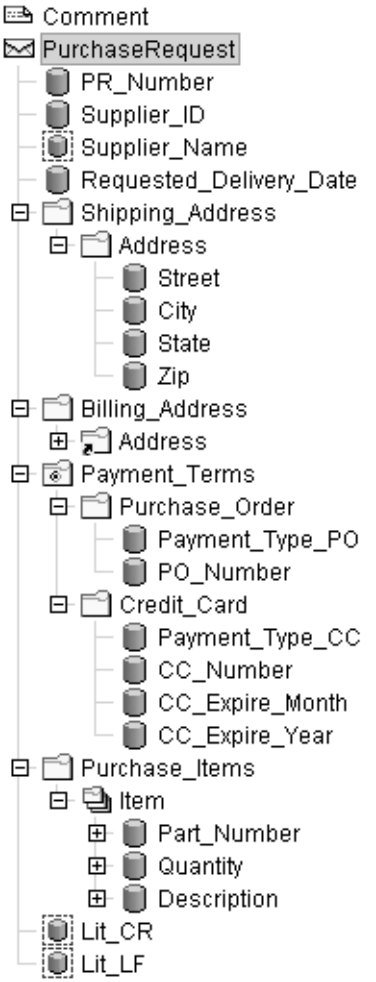
13. [File ｜ Save] を選択して、変更を保存します。

手順 4. 残りの項目の作成

必要に応じて、フィールドおよびグループの作成または複製のプロセスを繰り返し、発注書メッセージ フォーマット ドキュメントを完成させます。図 C-1 で示す発注書生データの分析を利用して各項目に入力する必要がある値を決定します。サポート情報として、\samples\di\po\po.mfl ファイルを参照してください。

必要な項目の入力を終えると、ナビゲーション ツリーは図 C-2 のようになるはずです。

図 C-2 完成した Purchase Order サンプルのナビゲーション ツリー



手順 5. 完成したメッセージ フォーマットの保存

[File | Save] を選択して、完成したメッセージ フォーマットを保存します。

完成した MFL ドキュメントが保存されます。現在設定されている Format Builder オプションによっては、MFL ドキュメントで取り込まれたコンテンツ モデルを記述する DTD や XML スキーマも保存されます。

Format Builder を以下の方法で設定すると、作成したメッセージ フォーマット ドキュメントの DTD や XML スキーマを自動的に生成できます。

1. [Tools | Options] を選択し、[Format Builder Options] ダイアログ ボックスを表示します。
2. ダイアログ ボックスの下部の [XML Content Model] セクションで、必要に応じて、[Auto-generate DTD] や [Auto-generate Schema] をチェックします。
3. [OK] をクリックするとオプションが更新されます。

これで、メッセージ フォーマット ドキュメントを保存すると必ず、Format Builder によって、各メッセージ フォーマット ドキュメントの指定したファイルが生成されます。

手順 6. メッセージ フォーマットのテスト

メッセージ フォーマットを使用してデータを変換する前に、メッセージ フォーマットをテストしてエラーがないかどうかを確認する手順は次のとおりです。

1. [Tools | Test] を選択し、[Format Tester] ダイアログ ボックスを表示します。
2. [File | Open Binary] を選択して、[開く] ダイアログ ボックスを表示します。
3. `samples/di/po` ディレクトリに移動します。
4. ファイルを選択し、[開く] をクリックします。

   [バイナリ] ウィンドウにバイナリ データが表示されます。
5. [Translate | Binary To XML] をクリックします。

   アクティブな MFL ドキュメントを基にして、`PO_01.DATA` ファイルの内容が XML に変換されます。[XML] ウィンドウに XML 出力が表示されます。

   注意： 各変換手順の説明を表示するには、[Display | Debug] を選択し、[Debug] ウィンドウを開きます。次に、[Translate | Binary To XML] を選択します。プロセスの各手順でメッセージが表示されます。

6. 変換されたデータが正しければ、[File | Save XML] を選択します。
7. [ファイル名] フィールドに「PO.XML」という名前をタイプし、次に、[保存] をクリックして XML 出力を保存します。

XML からバイナリへの変換の実行

Format Builder を使用してメッセージ定義を作成して、XML データのバイナリへの変換をテストすることもできます。この操作を実行するのに必要な手順は、バイナリ データを XML に変換する場合と基本的に同じです。XML データをバイナリに変換するためには、最初にバイナリ フォーマットの MFL 記述を作成します。Purchase Order サンプル ファイルを、プロセスのテストに使用する手順は次のとおりです

1. Format Builder メニューで [File | Open] を選択します。
2. 発注書のメッセージ フォーマット ドキュメントを選択します。
3. [開く] をクリックします。

   ナビゲーション ツリーに、メッセージ フォーマット ドキュメントが表示されます。
4. [Tools | Test] を選択し、[Format Tester] ダイアログ ボックスを表示します。
5. Format Tester メニューで [File | Open XML] を選択します。
6. samples\di\po ディレクトリに移動します。
7. po.xml ファイルを選択し、[開く] をクリックします。右ペインに XML データ ページが表示されます。
8. [Translate | XML to Binary] を選択します。

   XML データが変換され、右ペインに発注書データがバイナリ フォーマットで表示されます。

   注意： 各変換手順の説明を表示するには、[Display | Debug] を選択し、[Debug] ウィンドウを開きます。次に、[Translate | XML to Binary] を選択します。プロセスの各手順でメッセージが表示されます。

9. 変換されたデータが正しければ、[File ｜ Save Binary] を選択します。
10. [ファイル名] フィールドに名前（たとえば、`test_po.data`）をタイプし、次に、[保存] をクリックして バイナリ出力を保存します。

索引

G

H

I

M

N

O

R

T

U

ま

め

ゆ

り

る