WebSphere Business Integration Express Plus

データ・ハンドラー・ガイド

WebSphere Business Integration Express Plus

データ・ハンドラー・ガイド

お願い

本書および本書で紹介する製品をご使用になる前に、 213 ページの『特記事項』に記載されている情報をお読みください。

本書は、IBM WebSphere Business Integration Server Express バージョン 4.3、IBM WebSphere Business Integration Server Express Plus バージョン 4.3、および新しい版で明記されていない限り、以降のすべてのリリースおよびモディフィケーションに適用されます。

本マニュアルに関するご意見やご感想は、次の URL からお送りください。今後の参考にさせていただきます。

http://www.ibm.com/jp/manuals/main/mail.html

なお、日本 IBM 発行のマニュアルはインターネット経由でもご購入いただけます。詳しくは

http://www.ibm.com/jp/manuals/ の「ご注文について」をご覧ください。

(URL は、変更になる場合があります)

お客様の環境によっては、資料中の円記号がバックスラッシュと表示されたり、バックスラッシュが円記号と表示されたりする場合があります。

原　典： WebSphere Business Integration Express Plus
Data Handler Guide

発　行： 日本アイ・ビー・エム株式会社

担　当： ナショナル・ランゲージ・サポート

第1刷 2004.8

この文書では、平成明朝体™W3、平成明朝体™W9、平成角ゴシック体™W3、平成角ゴシック体™W5、および平成角ゴシック体™W7を使用しています。この(書体*)は、(財）日本規格協会と使用契約を締結し使用しているものです。フォントとして無断複製することは禁止されています。

注* 平成明朝体™W3、平成明朝体™W9、平成角ゴシック体™W3、
平成角ゴシック体™W5、平成角ゴシック体™W7

目次

本書について v
対象読者 v
関連資料 v
表記上の規則 vi

本リリースの新機能 vii

第 1 部 始めに 1

第 1 章 データ・ハンドラーの概要 3
データ・ハンドラーとは 3
データ・ハンドラーのインスタンス化 14
データ・ハンドラーの呼び出し 19
メタデータ主導型ハンドラーの設計 22

第 2 章 データ・ハンドラーのインストールと構成 25
データ・ハンドラーのインストール 25
データ・ハンドラーの構成 25
データ・ハンドラーを使用するようにコネクターを構成する方法 30

第 3 章 XML データ・ハンドラー 33
概要 33
ビジネス・オブジェクト定義の要件 37
XML データ・ハンドラーの構成 43
DTD を使用する XML 文書 47
スキーマ文書を使用する XML 文書 61
ビジネス・オブジェクト定義の作成 89
ビジネス・オブジェクトの XML 文書への変換 92
XML 文書のビジネス・オブジェクトへの変換 94
XML データ・ハンドラーのカスタマイズ 96

第 4 章 Request-Response データ・ハンドラー 99
概要 99
ビジネス・オブジェクト定義の要件 107
Request-Response データ・ハンドラーの構成 111
要求データ・ハンドラーによるビジネス・オブジェクトの変換 115
応答データ・ハンドラーによるビジネス・オブジェクトの変換 116
エラー処理 117
Request-Response データ・ハンドラーのカスタマイズ 117

第 5 章 FixedWidth データ・ハンドラー 119
概要 119
FixedWidth データ・ハンドラーの構成 120
FixedWidth 文書へのビジネス・オブジェクトの変換 124
ビジネス・オブジェクトへの FixedWidth 文書の変換 125

第 6 章 Delimited データ・ハンドラー 129
概要 129
Delimited データ・ハンドラーの構成 130
ビジネス・オブジェクトの区切りデータへの変換 134

区切りデータのビジネス・オブジェクトへの変換 135

第 7 章 NameValue データ・ハンドラー 139
概要 139
NameValue データ・ハンドラーの構成 140
ビジネス・オブジェクトの NameValue データへの変換 144
NameValue データのビジネス・オブジェクトへの変換 145

第 2 部 カスタム・データ・ハンドラー 149

第 8 章 カスタム・データ・ハンドラーの作成 151
データ・ハンドラー開発過程の概要 151
データ・ハンドラー開発用ツール 153
データ・ハンドラーの設計 155
データ・ハンドラー基本クラスの拡張 156
メソッドのインプリメント 156
カスタム・ネーム・ハンドラーの作成 171
JAR ファイルへのデータ・ハンドラーの追加 173
データ・ハンドラー・メタオブジェクトの作成 174
その他のビジネス・オブジェクトのセットアップ 176
コネクターの構成 177
国際化データ・ハンドラー 177

第 9 章 データ・ハンドラーの基本クラス・メソッド 181
createHandler() 182
getBO() - 抽象 183
getBO() - パブリック 185
getBOName() 186
getBooleanOption() 187
getByteArrayFromBO() 188
getEncoding() 189
getLocale() 189
getOption() 190
getStreamFromBO() 191
getStringFromBO() 192
setConfigMOName() 193
setEncoding() 193
setLocale() 194
setOption() 195
traceWrite() 195

付録. XML ODA の使用 197
インストールと使用法 197
Business Object Designer Express での XML ODA の使用 200
生成されるビジネス・オブジェクト定義の内容 210
ビジネス・オブジェクト定義の情報の変更 211

特記事項 213
プログラミング・インターフェース情報 214
商標 215

索引 217

本書について

製品 IBM(R)WebSphere Business Integration Server Express および IBM(R) WebSphere Business Integration Server Express Plus は、InterChange Server Express、関連する Toolset Express、CollaborationFoundation、およびソフトウェア統合アダプターのセットで構成されています。Toolset Express に含まれるツールは、ビジネス・オブジェクトの作成、変更、および管理に役立ちます。プリパッケージされている各種アダプターは、お客様の複数アプリケーションにまたがるビジネス・プロセスに応じて、いずれかを選べるようになっています。標準的な処理のテンプレートである CollaborationFoundation は、カスタマイズされたプロセスを簡単に作成できるようにするためのものです。

本書では、提供されたデータ・ハンドラーとカスタム・データ・ハンドラー機能について説明します。

特に明記されていない限り、本書の情報は、いずれも、IBM WebSphere Business Integration Server Express と IBM WebSphere Business Integration Server Express Plus の両方に当てはまります。WebSphere Business Integration Server Express という用語と、これを言い換えた用語は、これらの 2 つの製品の両方を指します。

対象読者

本書は、コンサルタントおよびお客様用です。ビジネス・オブジェクトとメタデータ・オブジェクトについて理解している必要があります。XML データ・ハンドラーを使用するには、XML 文書、現在の XML 標準、および SAX (Simple API for XML) を理解している必要があります。また、データ・ハンドラー・ライブラリーを拡張する場合には、Java プログラム言語を習熟している必要があります。

関連資料

本書の対象製品の一連の関連文書には、WebSphere Business Integration Server Express のどのインストールにも共通する機能とコンポーネントの解説のほか、特定のコンポーネントに関する参考資料が含まれています。

関連文書は、http://www.ibm.com/websphere/wbiserverexpress/infocenter でダウンロード、インストール、および表示することができます。

注: 本書の発行後に公開されたテクニカル・サポートの技術情報や速報に、本書の対象製品に関する重要な情報が記載されている場合があります。これらの技術情報や速報は、WebSphere Business Integration のサポート Web サイト (http://www.ibm.com/software/integration/websphere/support/) で参照できます。適切なコンポーネント領域を選択し、「Technotes (技術情報)」セクションと「Flashes (速報)」セクションを参照してください。

表記上の規則

本書では、以下のような規則を使用しています。

courier フォント	コマンド名、ファイル名、入力情報、システムが画面に出力した情報などのリテラル値を示します。
新規の用語	初めて表示される新しい用語を示します。
イタリック、*イタリック*	変数名または相互参照を示します。
青のアウトライン	オンラインで表示したときにのみ見られる青のアウトラインは、相互参照用のハイパーリンクです。アウトラインの内側をクリックすると、参照先オブジェクトにジャンプします。
{ }	構文の記述行の場合、中括弧 {} で囲まれた部分は、選択対象のオプションです。1 つのオプションのみを選択する必要があります。
[ ]	構文の記述行の場合、大括弧 [] で囲まれた部分は、オプションのパラメーターです。
...	構文の記述行の場合、省略符号 ... は直前のパラメーターが繰り返されることを示します。例えば、option[,...] は、複数のオプションをコンマで区切って入力できることを示します。
< >	命名規則では、不等号括弧は名前の個々の要素を囲み、各要素を区別します。 (例: <server_name><*connector_name*>tmp.log)
¥	本書では、ディレクトリー・パスに円記号 (¥) を使用します。IBM WebSphere 製品のすべてのパス名は、IBM WebSphere 製品がシステムにインストールされているディレクトリーに対する相対パス名です。
ProductDir	製品のインストール先ディレクトリーを表します。IBM Business Integration Server Express 環境では、「IBM¥WebSphereICS」がデフォルト製品ディレクトリーです。IBM WebSphere Business Integration Adapters 環境では、「WebSphereAdapters」がデフォルト製品ディレクトリーです。
%text%	パーセント記号 (%) 内のテキストは、Windows text システム変数またはユーザー変数の値を示します。

本リリースの新機能

本書の最初のリリースです。

第 1 部 始めに

第 1 章 データ・ハンドラーの概要

この章では、WebSphere Business Integration システムのデータ・ハンドラーの概要を説明します。データ・ハンドラーは、ビジネス・オブジェクトから直列化データへの変換と、直列化データからビジネス・オブジェクトへの変換を行います。この直列化データは、アプリケーションで読み込み可能なフォーマット (ストリングまたは入力ストリーム) です。この章を構成するセクションは次のとおりです。

- 『データ・ハンドラーとは』
- 14 ページの『データ・ハンドラーのインスタンス化』
- 19 ページの『データ・ハンドラーの呼び出し』
- 22 ページの『メタデータ主導型ハンドラーの設計』

データ・ハンドラーとは

データ・ハンドラー は、Java クラス・インスタンスで、特定の直列化フォーマットとビジネス・オブジェクトとの変換を行います。データ・ハンドラーは、WebSphere Business Integration ブローカーと何らかの外部プロセスとの間で情報を転送するビジネス・インテグレーション・システムのコンポーネントによって使用されます。表 1 に、WebSphere Business Integration ブローカーと外部プロセス間の情報の転送を処理するコンポーネントを示します。

表 1. WebSphere Business Integration システムの内部で情報を転送するコンポーネント

コンポーネント	目的	詳細
アダプター	WebSphere Business Integration ブローカーと外部プロセス (アプリケーションやテクノロジーなど) との間で情報を転送する役割を果たします。 **注:** アダプターはコネクター と呼ばれるランタイム・コンポーネントを使用して、統合ブローカーとアプリケーション (またはテクノロジー) 間の情報転送を実際に処理します。 これらの外部プロセスでは、イベント・レコードをイベント・ストアに送信することにより、プロセス内部で発生するイベントが特定されます。アダプターは、このイベント・ストアの内部でイベントを検出します。外部プロセスがトリガー側のイベントを検出すると、アダプターは、このイベントを表しているビジネス・オブジェクトを作成して、このイベントを*非同期* の状態でビジネス統合ブローカーへ送信します。このビジネス・オブジェクトには、イベントのタイプ (Create や Update など) を示すために、データおよび動詞が格納されています。	IBM 提供のアダプターについては、個々のアダプター・ガイドを参照してください。 カスタム・アダプターの場合、カスタム・コネクターの詳細については、(コネクターのインプリメントに使用した言語に応じて) 「*コネクター開発ガイド (Java 用)*」または「*コネクター開発ガイド (C++ 用)*」を参照してください。

表 1. WebSphere Business Integration システムの内部で情報を転送するコンポーネント (続き)

コンポーネント	目的	詳細
アクセス・クライアント	InterChange Server 統合ブローカーと任意の外部プロセス (例: Web サーバー内部のサーブレットなど) との間で情報を転送する役割を果たします。 アクセス・クライアントとは、Server Access Interface を使用して InterChange Server と直接通信する外部プロセスです。このコンポーネントは、転送が必要な何らかの情報を受信すると、イベントを表すビジネス・オブジェクトを作成して、このイベントを*同期* 状態で InterChange Server 内部のコラボレーションに送信します。アダプターの場合と同様に、このビジネス・オブジェクトには、イベントのタイプ (Create や Update など) を示すためにデータおよび動詞が格納されています。	「アクセス開発ガイド」

表 1 に示すように、これら 2 つのコンポーネント (コネクターおよびアクセス・クライアント) の目的は、次に示す方法でブローカーと外部プロセスとの間で情報を転送することです。

- 統合ブローカーへ情報を転送する場合、これらのコンポーネントは情報をビジネス・オブジェクトのフォーマットに設定します。
- 外部プロセスへ情報を転送する場合、これらのコンポーネントは情報を固有の直列化フォーマットに設定します。

多くの場合、外部プロセスは、その固有の直列化データに対して XML など何らかの共通フォーマットを使用します。これらの共通フォーマットとビジネス・オブジェクト間の変換を各アダプター (またはアクセス・クライアント) が実行する代わりに、WebSphere Business Integration システムには、IBM が開発した数種類のデータ・ハンドラーが用意されています。さらに、カスタム・データ・ハンドラーを作成すると、独自の固有フォーマット間の変換を処理できます。こうすれば、アダプター (またはアクセス・クライアント) は、適切なデータ・ハンドラーを呼び出して、直列化データの Multipurpose Internet Mail Extensions (MIME) タイプに基づいてデータ変換を実行できます。

注: データ・ハンドラーは、`DataHandler` という名前の Java クラスにインプリメントされています。このクラスは抽象クラスであり、データ・ハンドラー開発者がこれを拡張してデータ・ハンドラー・インスタンスをインプリメントします。詳細については、156 ページの『データ・ハンドラー基本クラスの拡張』を参照してください。

このセクションでは、データ・ハンドラーに関する以下の情報について説明します。

- 5 ページの『IBM 提供のデータ・ハンドラー』
- 6 ページの『データ・ハンドラー・メタオブジェクト』
- 7 ページの『データ・ハンドラーの呼び出し用のコンテキスト』

IBM 提供のデータ・ハンドラー

IBM では、表 2 に示す Java アーカイブ (jar) ファイルに格納してデータ・ハンドラーを提供しています。これらの jar ファイルは、製品ディレクトリーの下の DataHandlers サブディレクトリーにあります。

表 2. IBM 提供のデータ・ハンドラーの JAR ファイル

内容	説明	データ・ハンドラーの JAR ファイル
基本データ・ハンドラー	テキスト・ベースのデータ・ハンドラーおよびある IBM 提供アダプターに固有なデータ・ハンドラー	CwDataHandler.jar
特殊なデータ・ハンドラー	XML データ・ハンドラー	CwXMLDataHandler.jar
カスタム・データ・ハンドラー	作成者がインプリメントするデータ・ハンドラー	CustDataHandler.jar

基本データ・ハンドラー

基本データ・ハンドラー・ファイルである CwDataHandler.jar には、大半の IBM 提供データ・ハンドラーが格納されています。このファイルは、製品ディレクトリーの DataHandlers サブディレクトリーに存在します。表 3 に、この基本データ・ハンドラー・ファイルに格納されている基本データ・ハンドラーを示します。

表 3. 基本データ・ハンドラー・ファイルに格納されている基本データ・ハンドラー

データ・ハンドラー	MIME タイプ	詳細
Request-Response データ・ハンドラー	text/requestresponse	99 ページの『第 4 章 Request-Response データ・ハンドラー』
FixedWidth データ・ハンドラー	text/fixedwidth	119 ページの『第 5 章 FixedWidth データ・ハンドラー』
Delimited データ・ハンドラー	text/delimited	129 ページの『第 6 章 Delimited データ・ハンドラー』
NameValue データ・ハンドラー	text/namevalue	139 ページの『第 7 章 NameValue データ・ハンドラー』

注: 本書では、表 3 に示すテキスト・データ・ハンドラーについて説明します。基本データ・ハンドラー・ファイルには、ある特定の IBM 提供アダプターに固有なデータ・ハンドラーも数種類含まれています。ある IBM アダプターが特殊なデータ・ハンドラーを使用する場合、このアダプターのアダプター・ガイドには、このアダプターのデータ・ハンドラーのインストール、構成、および使用に関する情報が記載されています。

特殊なデータ・ハンドラー

IBM では、いくつかのデータ・ハンドラーについて、個別のインストーラーを提供しています。これらの特殊なデータ・ハンドラーをインストールするには、「*WebSphere Business Integration Adapters インストール・ガイド*」で示す手順に従う必要があります。

データ・ハンドラーを基本データ・ハンドラー・ファイルから分離すると、多くのアダプターは、基本データ・ハンドラー・ファイルに存在するほかのデータ・ハンドラーを格納する際のオーバーヘッドを発生させずにデータ・ハンドラーを使用で

きるようになります。表 4 に、IBM から個別のインストーラーと個別の JAR ファイルが提供されているデータ・ハンドラーを示します。

表 4. 個別の JAR ファイルを持つ IBM 提供のデータ・ハンドラー

データ・ハンドラー	データ・ハンドラーの JAR ファイル	MIME タイプ	詳細
XML データ・ハンドラー	CwXMLDataHandler.jar	text/xml	33 ページの『第 3 章 XML データ・ハンドラー』

カスタム・データ・ハンドラー

IBM 提供のデータ・ハンドラーが直列化データからビジネス・オブジェクトへの変換を*処理しない* 場合は、独自のカスタム・データ・ハンドラーを作成できます。CustDataHandler.jar ファイルは、作成する任意のカスタム・データ・ハンドラーを保持することを目的としています。このファイルは、製品ディレクトリーの DataHandlers サブディレクトリーに存在します。カスタム・データ・ハンドラーの作成方法の詳細については、151 ページの『第 8 章 カスタム・データ・ハンドラーの作成』を参照してください。

注: カスタム・データ・ハンドラーの開発を促進するため、IBM では、FixedWidth、Delimited、NameValue の各データ・ハンドラーのソース・コードについてもサンプル・コードとして提供しています。詳細については、153 ページの『サンプル・データ・ハンドラー』を参照してください。

データ・ハンドラー・メタオブジェクト

コネクターまたは Server Access Interface プロセスは、入力ファイルの MIME タイプまたはビジネス・オブジェクト要求で指定された MIME タイプに基づいて、データ・ハンドラーをインスタンス化します。

データ・ハンドラー・メタオブジェクトは、階層構造のビジネス・オブジェクトで、任意の数の子オブジェクトを組み込むことができます。データ・ハンドラー構成情報は、次の階層に整列されています。

- *トップレベル* のメタオブジェクトには、MIME タイプについての情報が格納され、異なるデータ・ハンドラーをサポートできます。トップレベルの各属性は、データ・ハンドラー・インスタンスに関して子メタオブジェクトを参照するカーディナリティー 1 属性です。各属性は 1 つの MIME タイプを表し、操作できるデータ・ハンドラーを示します。
- *子* メタオブジェクトには、特定のデータ・ハンドラーの実際の構成情報が格納されます。各属性は構成プロパティーを表しており、そのデフォルト値やタイプなどの情報を提供しています。

注: データ・ハンドラーは、構成情報を保持するのにメタオブジェクトを使用する必要はありません。ただし、IBM から提供されたすべての データ・ハンドラーは、その構成情報に関してメタオブジェクトを使用することを想定して設計されています。

データ・ハンドラー・メタオブジェクトにより、コネクターや Server Access Interface プロセスが入力ファイルの MIME タイプまたはビジネス・オブジェクト要求で指定された MIME タイプに基づいてデータ・ハンドラーをインスタンス化でき

ます。データ・ハンドラーを構成するには、そのメタオブジェクトが正しく初期化され、呼び出し元 (コネクターまたはアクセス・クライアント) で使用できることを確認する必要があります。

注: IBM 提供の各データ・ハンドラーは、データ・ハンドラー・メタオブジェクトに定義される構成プロパティーを使用します。ただし、カスタム・データ・ハンドラーはその構成プロパティーに関するメタオブジェクトを使用することも使用しないこともあります。詳細については、155 ページの『データ・ハンドラー・メタオブジェクトの使用』を参照してください。

データ・ハンドラーの呼び出し用のコンテキスト

3 ページの表 1 に示すように、WebSphere Business Integration システム内部のデータを転送する必要があるコンポーネントは、データ・ハンドラーを呼び出すことができます。表 5 には、データ・ハンドラーを呼び出すことのできるコンポーネントの追加情報が記載されています。

表 5. データ・ハンドラーの呼び出し用のコンテキスト

コンポーネント	イベント通信のタイプ	フローのタイプ	データ・ハンドラーを起動するソフトウェア
アダプター	非同期	イベント・トリガー・フロー	コネクター
アクセス・クライアント	同期	呼び出しトリガー・フロー	Server Access Interface

表 5 に示すように、イベント・トリガー・フローでは、アダプターがデータ・ハンドラーを直接呼び出します。呼び出しトリガー・フローでは、Server Access Interface (アクセス・クライアントと呼ばれる) を使用する外部プロセスがデータ・ハンドラーの呼び出しを開始します。データ・ハンドラーは、アダプターから直接呼び出された場合でもアクセス・クライアントから間接的に呼び出された場合でも同じように動作します。これらのコンテキストについては、以降のセクションで説明します。

コネクター・コンテキストでのデータ・ハンドラー

イベント・トリガー・フローでは、コネクター と呼ばれるアダプターのランタイム・コンポーネントがデータ・ハンドラーと直接やり取りしてデータを変換します。

注: IBM 提供のアダプターについては、個々のアダプター・ガイドを参照してください。カスタム・アダプターについては、アダプターのインプリメントに使用した言語に応じて、「*コネクター開発ガイド (Java 用)*」または「*コネクター開発ガイド (C++ 用)*」を参照してください。これらのガイドは、WebSphere Business Integration Adapters の資料セットの一部として含まれています。

コネクターがデータ・ハンドラーを呼び出すと、データ・ハンドラーがコネクター・プロセスの一部として実行されます。図 1 に、コネクター・コンテキストでのデータ・ハンドラーを示します。

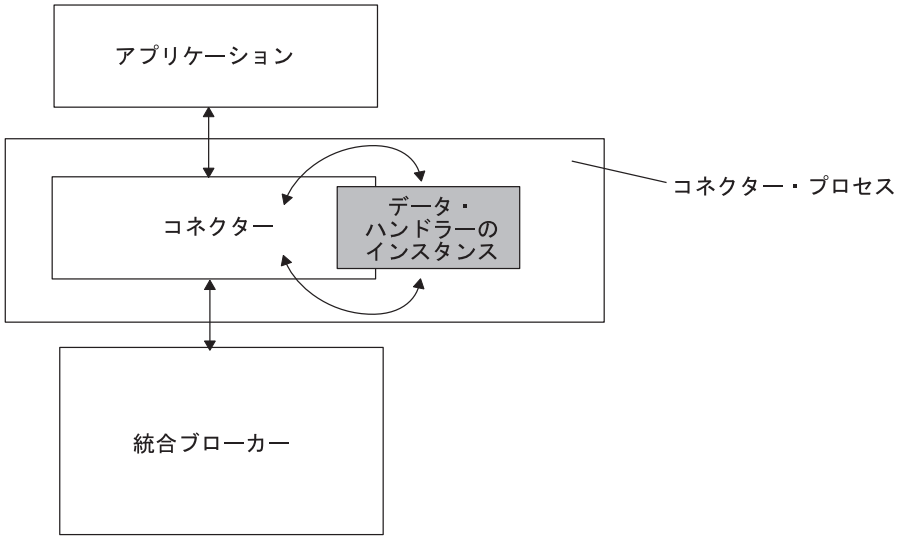

図 1. コネクターのコンテキストでのデータ・ハンドラー

データ変換には、ビジネス・オブジェクトの要件とフローの方向が反映されます。

- コネクターは、ビジネス・オブジェクト要求を処理する場合、ビジネス・オブジェクトからストリングへ変換するためにデータ・ハンドラーを呼び出します。
- コネクターは、イベント通知を処理する場合、ストリングからビジネス・オブジェクトへ変換するためにデータ・ハンドラーを呼び出します。

コネクターでのビジネス・オブジェクトからストリングへの変換: ビジネス・オブジェクトからストリングへの変換の場合、コネクターはデータ・ハンドラーを呼び出して、このデータ・ハンドラーにビジネス・オブジェクトを渡します。データ・ハンドラーは、ビジネス・オブジェクトおよびビジネス・オブジェクト定義の情報を使用して、データのストリームまたはストリングを作成します。このデータのストリームまたはストリングは、データ・ハンドラーと関連したフォーマット (通常は特定の MIME タイプ) で作成されます。ビジネス・オブジェクトからストリングへの変換が便利なのは、コネクターが統合ブローカーからビジネス・オブジェクトの形式で情報を受信するときです。コネクターは、情報を受信後、ビジネス・オブジェクトの情報をアプリケーション (またはテクノロジー) の直列化データとして送信する必要があります。

図 2 に、データ・ハンドラーがビジネス・オブジェクトからストリングへの変換を行う際の、コネクター・コンテキストでのデータ・ハンドラーを示します。

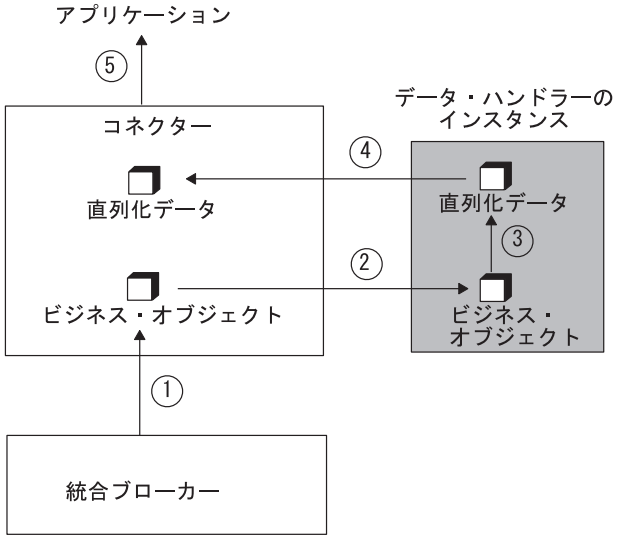

図 2. コネクターのコンテキストでのビジネス・オブジェクトからストリングへの変換

1. コネクターは、統合ブローカーからビジネス・オブジェクトを受け取ります。
2. コネクターは、データ・ハンドラーのインスタンスを作成してビジネス・オブジェクトを処理します (`DataHandler` 基本クラスの `createHandler()` 静的メソッドを使用)。

   コネクターがデータ・ハンドラーをインスタンス化する方法の詳細については、19 ページの『コネクター・コンテキストでのインスタンス化』を参照してください。
3. コネクターは、以下のデータ・ハンドラー・メソッドのいずれかを呼び出すことにより、ビジネス・オブジェクトからストリングへの変換を要求します。
   - `getStreamFromBO()`
   - `getStringFromBO()`
   - `getByteArrayFromBO()`

   コネクターは、ビジネス・オブジェクトを引き数としてこのメソッドに送信します。データ・ハンドラーは、ビジネス・オブジェクトを要求されたデータ・フォーマットに直列化します。
4. データ・ハンドラーは、直列化データをコネクターに戻します。
5. コネクターは直列化データを、電子メール、ファイル、または HTTP 接続などの宛先に書き込みます。

コネクターでのストリングからビジネス・オブジェクトへの変換: ストリングからビジネス・オブジェクトへの変換の場合、コネクターはデータ・ハンドラーを呼び出して、このデータ・ハンドラーに直列化データおよび関連の MIME タイプ・オブジェクトを渡します。データ・ハンドラーは、データのストリームまたはストリングを受け取ります。データ・ハンドラーは、データ・ストリーム内の情報を使用して、指定されたタイプのビジネス・オブジェクト・インスタンスを作成、命名、および設定します。ストリングからビジネス・オブジェクトへの変換は、コネクターが統合ブローカーにイベントを送信する必要がある場合に役に立ちます。アプリケーションは、このイベントを特定の MIME タイプを持つ直列化データとしてコネクターに送信します。

図 3 に、データ・ハンドラーがストリングからビジネス・オブジェクトへの変換を行う際の、コネクター・コンテキストでのデータ・ハンドラーを示します。

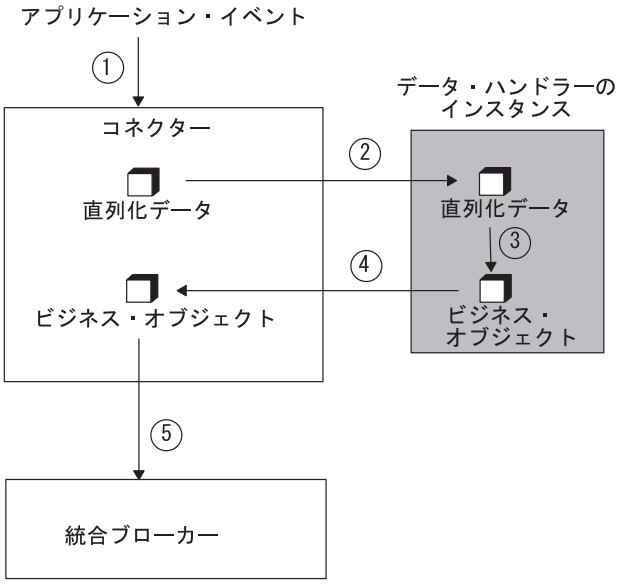

図 3. コネクターのコンテキストでのストリングからビジネス・オブジェクトへの変換

1. コネクターがアプリケーション・イベントを検出します。イベントの形式は、電子メール、テキスト・ファイル、XML 文書、またはデータ・ハンドラーが存在するそのほかの任意の共通フォーマットのいずれでも構いません。
2. コネクターは、データ・ハンドラーのインスタンスを作成してイベントを処理します (`DataHandler` 基本クラスの `createHandler()` 静的メソッドを使用)。

   コネクターがデータ・ハンドラーをインスタンス化する方法の詳細については、19 ページの『コネクター・コンテキストでのインスタンス化』を参照してください。
3. コネクターは、データ・ハンドラー・インスタンスの `getBO()` メソッドの引き数として直列化データを引き渡します。データ・ハンドラーはビジネス・オブジェクトのインスタンスを作成します。

   コネクターが `getBO()` メソッドのデータ変換先のビジネス・オブジェクトを指定する場合もあります。ビジネス・オブジェクトのタイプを指定するコネクターもあり、またデータ・ハンドラーが直列化テキストからビジネス・オブジェクト・タイプを抽出できることを前提とするコネクターもあります。データ・ハンドラーはデータを構文解析し、直列化データを基にビジネス・オブジェクトの属性値を取り込みます。
4. データ・ハンドラーは、ビジネス・オブジェクトをコネクターに戻します。
5. コネクターはビジネス・オブジェクトを統合ブローカーに送信します。

Server Access Interface のコンテキストでのデータ・ハンドラー

呼び出しトリガー・フローでは、アクセス・クライアントがデータ・ハンドラーとやり取りしてデータを変換します。アクセス・クライアント は外部プロセスで、Server Access Interface を使用して InterChange Server Express とやり取りします。

アクセス・クライアントは、Server Access Interface API の `ItoExternalForm()` メソッドまたは `IcreateBusinessObjectFrom()` メソッドを呼び出すと、データ・ハンドラーの呼び出しを開始します。 Server Access Interface は InterChange Server Express プロセスの一部として動作し、実際にデータ・ハンドラーを起動します。

InterChange Server

Server Access Interface は API で、アクセス・クライアントが InterChange Server 内のコラボレーションを実行できるようになります。このインターフェースは、InterChange Server が統合ブローカーの場合にのみ 使用できます。アクセス・クライアントにおけるこのインターフェースの詳細については、IBM WebSphere InterChange Server 資料セットの「*アクセス開発ガイド*」を参照してください。

アクセス・クライアントは、クライアント・ブラウザーからの要求を処理するサーブレットの場合があります。要求は、データ要求、注文要求、または別のタイプの企業間トランザクションの場合があります。別の例として、アクセス・クライアントは、Server Access Interface を使用して InterChange Server Express にアクセスし、別のアプリケーションとデータを交換する C++ または Java プログラムの場合もあります。

アクセス・クライアントがデータ・ハンドラーを必要とする呼び出しを開始すると、データ・ハンドラーが InterChange Server プロセスの一部として実行されます。図 4 に、Server Access Interface コンテキストでのデータ・ハンドラーを示します。この例では、アクセス・クライアントは Web サーバーとサーブレットです。

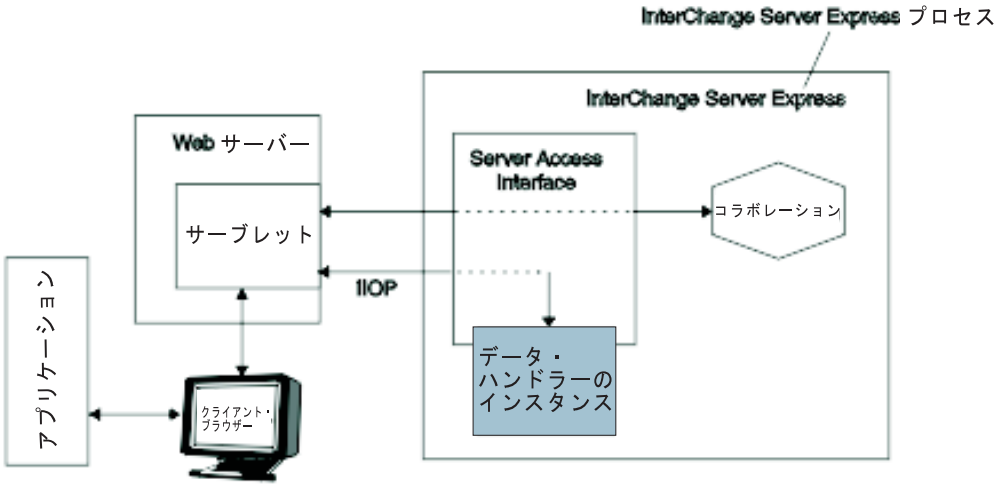

図 4. Server Access Interface のコンテキストでのデータ・ハンドラー

データ変換には、ビジネス・オブジェクトの要件とフローの方向が反映されます。

- InterChange Server Express からビジネス・オブジェクトを受け取るには、アクセス・クライアントがデータ・ハンドラーにビジネス・オブジェクトからストリングへの変換を要求する必要があります。

- InterChange Server Express にデータを送信するには、アクセス・クライアントがデータ・ハンドラーにストリングからビジネス・オブジェクトへの変換を要求する必要があります。

Server Access Interface でのビジネス・オブジェクトからストリングへの変換:
ビジネス・オブジェクトからストリングへの変換では、データ・ハンドラーはコラボレーションの実行結果としてビジネス・オブジェクトを受け取ります。データ・ハンドラーは、ビジネス・オブジェクト内の情報を使用してデータのストリームまたはストリングを作成します。このデータはデータ・ハンドラーと関連したフォーマットで、通常は特定の MIME タイプです。アクセス・クライアントはほとんどの場合、作成されたビジネス・オブジェクトを直列化データとしてアプリケーションに送信します。

図 5 に、データ・ハンドラーがアクセス・クライアントについてのビジネス・オブジェクトからストリングへの変換を行う際の、Server Access Interface コンテキストでのデータ・ハンドラーを示します。

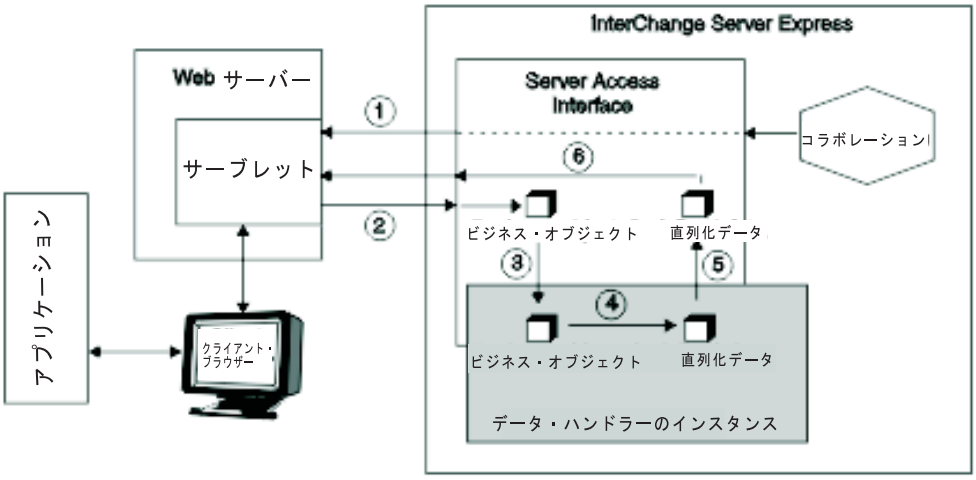

図 5. Server Access Interface コンテキストでのビジネス・オブジェクトからストリングへの変換

1. コラボレーションは、要求されたデータまたは要求されたアクションの結果をアクセス・クライアントに戻します。
2. ビジネス・オブジェクトを要求されたフォーマットに変換するために、アクセス・クライアントは Server Access Interface の `ItoExternalForm()` メソッドへの引き数としてビジネス・オブジェクトを送信します。
3. Server Access Interface は次のアクションを行います。
   - データ・ハンドラーのインスタンスを作成して変換を行います (`DataHandler` 基本クラスの `createHandler()` 静的メソッドを使用)。
   - 以下のいずれかのデータ・ハンドラー・メソッドの引き数として直列化データを送信します。
     - `getStreamFromBO()`
     - `getStringFromBO()`
     - `getByteArrayFromBO()`

4. データ・ハンドラーはビジネス・オブジェクトを構文解析して直列化データを作成します。
5. データ・ハンドラーは、直列化データを Server Access Interface に戻します。
6. Server Access Interface は直列化データをアクセス・クライアントに戻します。

Server Access Interface でのストリングからビジネス・オブジェクトへの変換:
ストリングからビジネス・オブジェクトへの変換では、データ・ハンドラーはデータのストリームまたはストリングを受け取ります。データ・ハンドラーは、データ・ストリーム内の情報を使用して、指定されたタイプのビジネス・オブジェクト・インスタンスを作成、命名、および設定します。ストリングからビジネス・オブジェクトへの変換は、アクセス・クライアントが InterChange Server InterChange Server Express のコラボレーションにビジネス・オブジェクトを送信する必要がある場合に役に立ちます。アクセス・クライアントがこの直列化データ (通常は、特定の MIME タイプ) をデータ・ハンドラーに送信します。

図 6 に、データ・ハンドラーがアクセス・クライアントについてのストリングからビジネス・オブジェクトへの変換を行う際の、Server Access Interface コンテキストでのデータ・ハンドラーを示します。

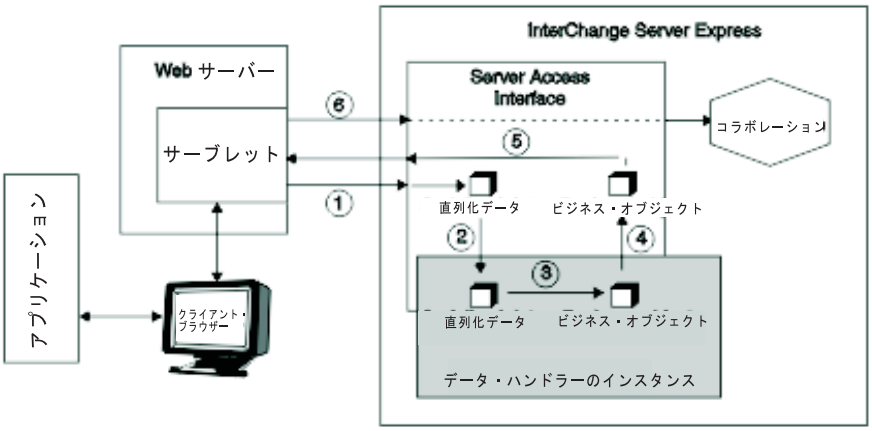

図 6. Server Access Interface コンテキストでのストリングからビジネス・オブジェクトへの変換

1. 直列化データをビジネス・オブジェクトに変換するために、アクセス・クライアントは Server Access Interface の `IcreateBusinessObjectFrom()` メソッドへの引き数として直列化データを送信します。
2. Server Access Interface は次のアクションを行います。
   - データ・ハンドラーのインスタンスを作成して変換を行います (`DataHandler` 基本クラスの `createHandler()` 静的メソッドを使用)。
   - データ・ハンドラー・インスタンスの `getBO()` メソッドの引き数として直列化データを引き渡します。
3. データ・ハンドラーはビジネス・オブジェクトのインスタンスを作成します。

   データ・ハンドラーはデータを構文解析し、直列化データを基にビジネス・オブジェクトの属性値を取り込みます。

4. データ・ハンドラーは、ビジネス・オブジェクトを Server Access Interface に戻します。
5. Server Access Interface はビジネス・オブジェクトをアクセス・クライアントに戻します。
6. アクセス・クライアントはビジネス・プロセス内のビジネス・オブジェクト・データを使用するコラボレーションを呼び出します。

データ・ハンドラーのインスタンス化

データ・ハンドラーは、コネクターまたは Server Access Interface (Business Integration Express 統合ブローカーと通信するアクセス・クライアントの場合) が使用できるクラス・ライブラリーとしてインプリメントされます。DataHandler 基本クラスは抽象クラスです。したがって、データ・ハンドラーをインスタンス化するには、DataHandler サブクラスの 1 つをインスタンス化する必要があります。各データ・ハンドラーは、IBM 提供のデータ・ハンドラーでもカスタム・データ・ハンドラーでも、DataHandler 基本クラスのサブクラスです。データ・ハンドラーをインスタンス化するメソッドは createHandler() です。

createHandler() メソッドは、データ・ハンドラー・メタオブジェクト の情報を使用して、インスタンス化の対象となるデータ・ハンドラーおよびこのデータ・ハンドラーを初期化する方法を調べます。データ・ハンドラー・メタオブジェクトは、階層構造のビジネス・オブジェクトで、任意の数の子オブジェクトを組み込むことができます。データ・ハンドラー構成情報は、次の階層に整列されています。

- *トップレベル* のメタオブジェクトには、MIME タイプについての情報が格納され、異なるデータ・ハンドラーをサポートできます。トップレベルの各属性は、データ・ハンドラー・インスタンスの子メタオブジェクトを参照するカーディナリティー 1 属性です。各属性は 1 種類の MIME タイプを表しており、その属性タイプは、この MIME タイプを操作できるデータ・ハンドラーの子メタオブジェクトを示しています。
- *子* メタオブジェクトには、特定のデータ・ハンドラーの実際の構成情報が格納されます。各属性は構成プロパティーを表しており、そのデフォルト値やタイプなどの情報を提供しています。

注: データ・ハンドラーは、構成情報を保持するのにメタオブジェクトを使用する必要はありません。ただし、IBM から提供されたすべての データ・ハンドラーは、その構成情報に関してメタオブジェクトを使用することを想定して設計されています。

createHandler() メソッドは、次の手順でデータ・ハンドラーをインスタンス化します。

- 『データ・ハンドラー・クラスの識別』
- 17 ページの『データ・ハンドラー構成プロパティーの設定』
- 18 ページの『ビジネス・オブジェクト・プレフィックスの設定』

データ・ハンドラー・クラスの識別

作成されるデータ・ハンドラーについて、DataHandler 基本クラス・インプリメンテーションをインスタンス化する必要があります。データ・ハンドラーのインスタ

ンス化メソッドは、createHandler() メソッドに引き数として渡される次の 2 つの値のうちの 1 つからこのデータ・ハンドラー・クラスの名前を派生させます。

- インスタンス化するデータ・ハンドラーのクラス名。
- 変換するデータの MIME タイプ。

クラス名の使用

データ・ハンドラーの呼び出し元からクラス名を引き数として渡されると、createHandler() はそのクラス名のデータ・ハンドラーをインスタンス化します。また、指定されたクラスを次の場所から探します。

1. CwDataHandler.jar ファイル
2. CwXMLDataHandler.jar ファイル
3. CustDataHandler.jar ファイル
4. CLASSPATH 中の他の部分

呼び出し元でデータ・ハンドラーのクラス名のみ を指定した場合、createHandler() はデータ・ハンドラー・メタオブジェクトの検索も、これらのオブジェクトからの構成プロパティーの設定も行いません。したがって、この方法でインスタンス化されたデータ・ハンドラーはメタオブジェクトを必要としません。カスタム・データ・ハンドラーがメタオブジェクトを使用するかどうかの詳細については、155 ページの『データ・ハンドラー・メタオブジェクトの使用』を参照してください。

MIME タイプの使用

データ・ハンドラーの呼び出し元がクラス名を引き数として渡さない 場合、createHandler() メソッドは MIME タイプの値を必要とします。呼び出し側のコンポーネント (コネクターまたはアクセス・クライアント) から MIME タイプを渡されると、createHandler() は、この MIME タイプに関連付けられている子データ・ハンドラー・メタオブジェクトを使用して、データ・ハンドラー・インスタンスのクラス名やそのほかの構成情報を派生させます。

注: メタオブジェクトの詳細については、25 ページの『データ・ハンドラーの構成』を参照してください。カスタム・データ・ハンドラーがメタオブジェクトを使用するかどうかの詳細については、155 ページの『データ・ハンドラー・メタオブジェクトの使用』を参照してください。

指定された MIME タイプからクラス名を派生させるために、createHandler() メソッドは次の手順を実行します。

1. MIME タイプを MIME タイプ・ストリングに変換します。

   createHandler() メソッドは、トップレベルのデータ・ハンドラー・メタオブジェクトを検索すると、ハイフン (-)、ピリオド (.)、スラッシュ (/) などすべての非英数字を下線 (_) に変換します。例えば、MIME タイプが text/html の場合、createHandler() はこの MIME タイプを解析して、text_html というストリングにします。

   createHandler() メソッドは、ピリオドが含まれる MIME タイプ名も検索できるように、この非英数字の変換を段階的に実行します。ただし、Business Object

Designer Express では、属性名にピリオド文字を使用できません。したがって、IBM では MIME タイプ名の中にピリオド文字を*使用しない* ことをお勧めします。

固有な MIME タイプ/サブタイプの組み合わせを作成して、特定の MIME タイプの変形を指定することができます。 MIME タイプ名では、MIME タイプとサブタイプを非英数字 (ハイフン (-) や下線 (_) など) で分離します。ただし、createHandler() は任意の非英数字を下線に置き換えるため、MIME タイプとサブタイプの分離には、下線のみを使用することをお勧めします。MIME タイプが text/xml-sgml の場合、このメソッドは、この MIME タイプを text_xml_sgml というストリングに変換します。

2. DataHandler 基本クラスの内の静的プロパティーからトップレベルのデータ・ハンドラー・メタオブジェクトの名前を取得します。このトップレベルのメタオブジェクトで、createHandler() は変換するデータの MIME タイプ・ストリングに一致する属性を検索します。
3. createHandler() がトップレベルのメタオブジェクト内で一致する MIME タイプ・ストリングを見つけると、createHandler() は次の手順を実行します。
   a. 呼び出し元でビジネス・オブジェクト・プレフィックスの値 (オプションの 3 番目の引き数) を createHandler() に指定すると、この値が MIME サブタイプの値として解釈されて MIME タイプに付加され、次の形式の MIME タイプ・ストリングが作成されます。

      MIMETypeString _BOPrefix

      この名前の属性が存在しない場合、createHandler() は、MIME タイプのみが一致する属性を検索します。例えば、呼び出し元から MIME タイプ edi が渡され、ビジネス・オブジェクト・プレフィックスが x12 の場合、createHandler() はトップレベルのメタオブジェクトから「edi_x12」という名前の属性を探します。この名前の属性が存在しない場合、createHandler() は「edi」という名前の属性を探します。
   b. 使用するデータ・ハンドラー・インスタンスの構成プロパティー (例えば、データ・ハンドラーの区切り文字に使用する文字を識別する可能性のある構成プロパティー) が格納された関連する子データ・ハンドラー・メタオブジェクトを取得します。
   c. 子メタオブジェクトから ClassName 属性の値を検索します。
      - ClassName 属性に値が格納されている場合、createHandler() はこのクラス名のデータ・ハンドラーをインスタンス化します。データ・ハンドラーのインプリメンテーション・プロセスの一部として、関連する子データ・ハンドラー・メタオブジェクトが作成されます。この時点で、データ・ハンドラーをインプリメントする開発者が、子メタオブジェクトの ClassName 属性にクラス名を追加できます。この ClassName 値は、4a で説明するように、クラス名がデフォルトのクラス名と異なる場合に必要です。
      - この ClassName 属性に、値が格納されていない 場合、createHandler() は、4a で説明するように、インスタンス化するデータ・ハンドラーのクラス名を作成します。

4. createHandler() メソッドがトップレベルのメタオブジェクト内で一致する MIME タイプ・ストリングを検出できなかった 場合は、次のようにして、インスタンス化するデータ・ハンドラーのクラス名を作成します。
   a. 基本データ・ハンドラー・パッケージに MIME タイプ・ストリングを付加します。

      ```
      com.crossworlds.DataHandlers
      ```

      例えば、MIME タイプ・ストリングが text_html の場合、作成されたストリングは次のようになります。

      ```
      com.crossworlds.DataHandlers.text.html
      ```

   b. メソッドが CLASSPATH 上で生成されたクラス名を見つけることができた場合、このクラスをインスタンス化します。次の場所から、このクラスを探します。
      - CwDataHandler.jar ファイル
      - CustDataHandler.jar ファイル
   - CLASSPATH 中の他の部分

クラス名と MIME タイプの使用

呼び出し側でクラス名と MIME タイプの*両方* を指定すると、createHandler() は次のアクションを行います。

- 指定されたクラスのデータ・ハンドラーを作成します。このクラスを見つけるために、データ・ハンドラーは 15 ページの『クラス名の使用』にリストされているディレクトリー内を検索します。
- MIME タイプに関連する子メタオブジェクトを使用して、このデータ・ハンドラーの構成オプションを初期化します。createHandler() が MIME タイプから子メタオブジェクトを判定する方法の詳細については、 15 ページの『MIME タイプの使用』を参照してください。

つまり、呼び出し側でクラス名を指定すると、このクラス名により子メタオブジェクトの ClassName 属性で指定されたクラス名がオーバーライド されます。

データ・ハンドラー構成プロパティーの設定

IBM から提供されているすべてのデータ・ハンドラー (表 3 および表 4) は、その構成情報に関してメタオブジェクトを使用することを想定して設計されています。データ・ハンドラー・メタオブジェクトは、次のような性質を持つ階層ビジネス・オブジェクトです。

- トップレベルのデータ・ハンドラー・メタオブジェクトでは、各属性は MIME タイプによって識別され、属性は子データ・ハンドラー・メタオブジェクトを表します。
- 子データ・ハンドラー・メタオブジェクトには、MIME タイプに関連付けられているデータ・ハンドラーの構成情報が含まれています。

IBM 提供のデータ・ハンドラーでは、その子データ・ハンドラー・メタオブジェクトに関連付けられている構成情報を使用して、データ・ハンドラーのプロパティーを初期化します。したがって、IBM は提供する各データ・ハンドラーに関する子メタオブジェクトを提供しています (表 8 を参照)。

注: データ・ハンドラー・メタオブジェクトの詳細については、 25 ページの『データ・ハンドラーの構成』を参照してください。

createHandler() メソッドはデータ・ハンドラーをインスタンス化した後、特殊な protected メソッド (setupOptions()) を呼び出して、該当する子データ・ハンドラー・メタオブジェクト内の値を使用してデータ・ハンドラーの構成を初期化します。

注: カスタム・データ・ハンドラーは、構成を初期化するのにメタオブジェクトを使用する必要はありません。データ・ハンドラーに関連する子メタオブジェクトがある場合、createHandler() メソッドはデータ・ハンドラーに関して setupOptions() を呼び出しません。詳細については、 176 ページの『その他のビジネス・オブジェクトのセットアップ』を参照してください。

メタオブジェクトの詳細については、 25 ページの『データ・ハンドラーの構成』を参照してください。

ビジネス・オブジェクト・プレフィックスの設定

createHandler() メソッドは、3 番目の引き数としてオプションのビジネス・オブジェクト・プレフィックスを受け取ることができます。この引き数を使用して MIME タイプ名を判別します (15 ページの『MIME タイプの使用』を参照)。createHandler() はデータ・ハンドラーをインスタンス化してその構成プロパティーを設定した後、最後のタスクとしてデータ・ハンドラーの BOPrefix 構成オプション (存在する場合) にこの 3 番目の引き数の値を設定します。

注: BOPrefix 構成オプションを使用するすべてのデータ・ハンドラー (XML データ・ハンドラーなど) は、このプレフィックスをビジネス・オブジェクト名の先頭に付加します。

データ・ハンドラーは、作成するどのビジネス・オブジェクト名にもこのプレフィックスを先頭に付加できます (ストリングからビジネス・オブジェクトへの変換時)。プレフィックスとビジネス・オブジェクト名の間に下線 (_) を設定します。例えば、コネクターが次の createHandler() 呼び出しで、XML データ・ハンドラーを起動する場合は、次のとおりです。

```
createHandler(null, "text/xml", "UserApp");
```

createHandler() メソッドは XML データ・ハンドラーをインスタンス化して、その BOPrefix 属性に「UserApp」を設定します。XML データ・ハンドラーが Customer ビジネス・オブジェクトを作成すると、このビジネス・オブジェクトは次のようになります。

- 直列化データからビジネス・オブジェクト名を取得します。
- 「UserApp」プレフィックスをビジネス・オブジェクト名の先頭に付加します。

作成されるビジネス・オブジェクト名は、「UserApp_Customer」です。

データ・ハンドラーの呼び出し

データ・ハンドラーが呼び出されるコンテキストに関係なく、データ・ハンドラーは createHandler() メソッドでインスタンス化されます。各コンテキストでのこのメソッドの呼び出し方法は、次のとおりです。

- コネクターは*明示的に* createHandler() を呼び出して、データ・ハンドラーをインスタンス化します。
- アクセス・クライアントが createHandler() を*暗黙的に* 呼び出します。Server Access Interface は、アクセス・クライアントが次の Server Access Interface メソッド、ItoExternalForm() または IcreateBusinessObjectFrom() のいずれかを使用してデータ・ハンドラー呼び出しを開始するときに、実際に createHandler() を呼び出します。

コネクター・コンテキストでのインスタンス化

図 7 に、データ・ハンドラーがコネクター・コンテキストで呼び出された場合の、データ・ハンドラーのインスタンス化の例を示します。

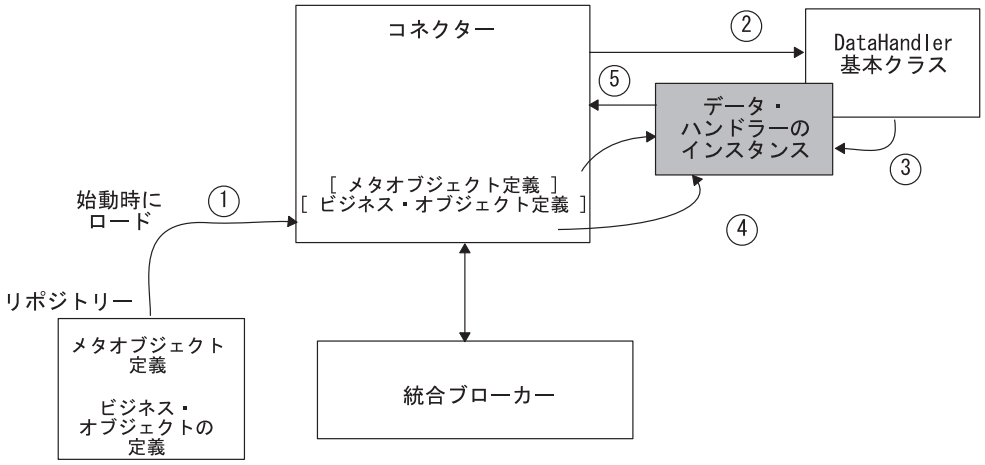

図 7. コネクターのコンテキストでのデータ・ハンドラーのインスタンス化

コネクター・コンテキストで呼び出されたデータ・ハンドラーをインスタンス化するために、コネクターは次の手順を実行します。

1. コネクターの始動時にメタオブジェクトがメモリーに読み込まれます。ただし、メタオブジェクトがコネクターのサポート・オブジェクトのリストに含まれている場合に限ります。

   メタオブジェクトは、ビジネス・オブジェクトとともにリポジトリーに格納されます。データ・ハンドラーがメタオブジェクト定義をアクセスできるためには、メタオブジェクト定義はビジネス・オブジェクト定義のように、メモリー内に存在する必要があります。これらのメタオブジェクトがコネクター・プロセスの一部としてメモリー内に存在する場合、メタオブジェクトはコネクター・コンテキストで呼び出されたデータ・ハンドラーからアクセスできます。
2. コネクターは DataHandler 基本クラス内の setConfigMOName() 静的メソッドを呼び出して、データ・ハンドラーの基本クラス内の静的プロパティーをデータ・ハンドラーのトップレベルのメタオブジェクトの名前に設定します。

このトップレベルのメタオブジェクトはコネクター・メタオブジェクトです (デフォルトでは MO_DataHandler_Default)。トップレベルのメタオブジェクトは、コネクターのサポート・オブジェクト・リストに存在する必要があります。

注: コネクターがデータ・ハンドラーのトップレベルのメタオブジェクトの名前を取得する方法は、コネクターの設計により決まります。詳細については、177 ページの『コネクターの構成』を参照してください。

3. コネクターは DataHandler 基本クラス内の createHandler() 静的メソッドを呼び出して、必要なデータ変換を実行する DataHandler 基本クラスのインスタンスを作成します。作成されるクラスの名前は、次の 2 つの方法のいずれかで決定されます。
   - コネクターからクラス名を引き数として渡されると、createHandler() メソッドはそのクラス名のデータ・ハンドラーをインスタンス化します。コネクターは、createHandler() を呼び出すときに、クラス名を明示的に指定できます。
   - クラス名の代わりに MIME タイプが渡されると、createHandler() は MIME タイプからクラス名を派生させます。

     createHandler() メソッドは、MIME タイプを MIME タイプ・ストリングに変換して、DataHandler 基本クラス内の静的プロパティーからデータ・ハンドラーのトップレベルのメタオブジェクトの名前を取得します。このトップレベルのメタオブジェクトから、createHandler() はデータ・ハンドラーの子メタオブジェクトの名前を取得します。この子メタオブジェクトには、インスタンス化するクラス名などの構成情報が格納されています。この派生方法の詳細については、14 ページの『データ・ハンドラー・クラスの識別』を参照してください。
4. データ・ハンドラーは必要なデータ変換を実行します。コネクター・エージェントは、該当する DataHandler メソッドを呼び出して必要な変換を実行します。
   - ストリングからビジネス・オブジェクトへの変換については、getBO() メソッド。
   - ビジネス・オブジェクトからストリングへの変換については getStringFromBO() メソッド、あるいはビジネス・オブジェクトからストリームへの変換については getStreamFromBO() メソッド。
5. データ・ハンドラーは、該当するフォーマットをコネクター・エージェントに戻します。

Server Access Interface のコンテキストでのインスタンス化

図 8 に、データ・ハンドラーが Server Access Interface コンテキストで呼び出された場合の、データ・ハンドラーのインスタンス化の例を示します。

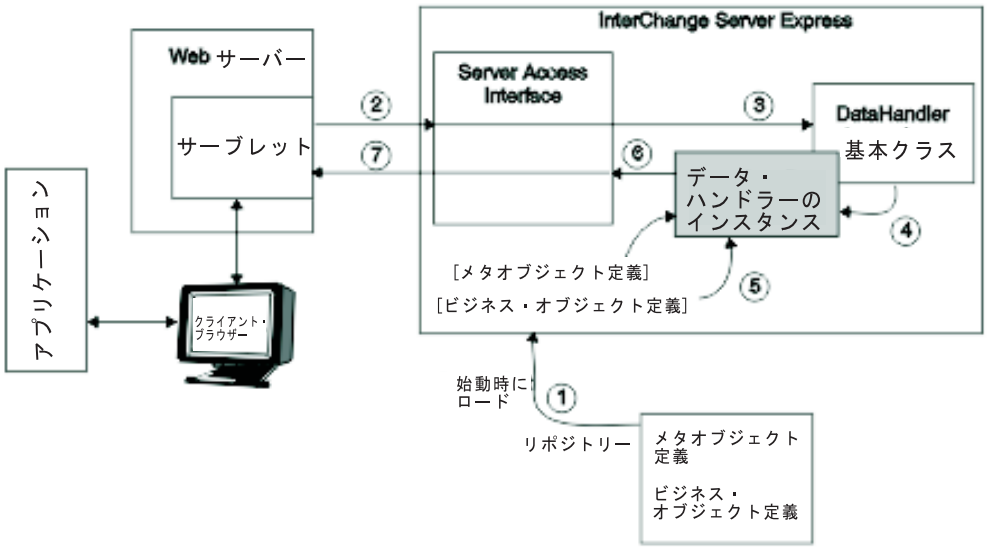

図 8. Server Access Interface コンテキストでのデータ・ハンドラーのインスタンス化

Server Access Interface コンテキストで呼び出されたデータ・ハンドラーをインスタンス化するために、Server Access Interface は次の手順を実行します。

1. サーバーの始動時に、リポジトリー内のその他のビジネス・オブジェクト定義とともに、メタオブジェクトがメモリーに読み込まれます。

   メタオブジェクトは、ビジネス・オブジェクトとともにリポジトリーに格納されます。データ・ハンドラーがメタオブジェクト定義をアクセスできるためには、メタオブジェクト定義はビジネス・オブジェクト定義のように、メモリー内に存在する必要があります。これらのメタオブジェクトが Business Integration Express (および Server Access Interface) プロセスの一部としてメモリー内に存在する場合、メタオブジェクトは Server Access Interface コンテキストで呼び出されたデータ・ハンドラーからアクセスできます。

2. アクセス・クライアントは、次の Server Access Interface メソッド、`IcreateBusinessObjectFrom()` または `ItoExternalForm()` のいずれかでデータ・ハンドラーのインスタンスの作成を開始します。

   これらのメソッドは、変換するデータの MIME タイプを渡します。

   注: アクセス・クライアントは、データ・ハンドラーを呼び出すには Server Access Interface メソッドを使用する必要があります。これらのメソッドはデータ・ハンドラー・インターフェース・メソッドを間接的に呼び出しますが、これらのメソッドではデータ・ハンドラー・インターフェースの一部しか使用できません。 Server Access Interface メソッドの詳細については、「*アクセス開発ガイド*」を参照してください。データ・ハンドラー・インターフェースに用意されているメソッドの詳細については、181 ページの『第 9 章 データ・ハンドラーの基本クラス・メソッド』を参照してください。

3. Server Access Interface はデータ・ハンドラーのトップレベルのメタオブジェクトの名前を `MO_Server_DataHandler` に設定します。

4. Server Access Interface は DataHandler サブクラスのインスタンスを作成して、必要なデータ変換を実行します (DataHandler 基本クラスの createHandler() メソッドを使用)。

   Server Access Interface のコンテキストで呼び出された場合、createHandler() メソッドは、クラス名を*指定しません*。代わりに、createHandler() は MIME タイプを MIME タイプ・ストリングに変換し、データ・ハンドラーのトップレベルのメタオブジェクトの名前を取得します。このトップレベルのメタオブジェクトから、createHandler() はデータ・ハンドラーの子メタオブジェクトの名前を取得します。この子メタオブジェクトには、インスタンス化するクラス名などの構成情報が格納されています。この派生方法の詳細については、14 ページの『データ・ハンドラー・クラスの識別』を参照してください。
5. データ・ハンドラーは必要なデータ変換を実行します。Server Access Interface は該当する DataHandler メソッドを呼び出して必要な変換を実行します。
   - ストリングからビジネス・オブジェクトへの変換については、getBO() メソッド。
   - ビジネス・オブジェクトからストリングへの変換については getStringFromBO() メソッド、あるいはビジネス・オブジェクトからストリームへの変換については getStreamFromBO() メソッド。
6. データ・ハンドラーは、要求されたフォーマットを Server Access Interface に戻します。
7. Server Access Interface は要求されたフォーマットをアクセス・クライアントに戻します。

メタデータ主導型ハンドラーの設計

IBM 提供のデータ・ハンドラーはメタデータ主導型です。メタデータは、ビジネス・オブジェクト定義に格納されたビジネス・オブジェクトについてのデータです。ビジネス・オブジェクト定義内のメタデータに、ビジネス・オブジェクトのインスタンス内のデータを説明する情報が用意されています。一般的に、ビジネス・オブジェクト・メタデータには、ビジネス・オブジェクトの構造、属性プロパティーの設定値、およびアプリケーション固有情報の内容が格納されます。また、データの処理方法についての指示も格納されています。

コネクターは通常、ビジネス・オブジェクトを処理する際には、ビジネス・オブジェクト・メタデータを使用するように設計されています。同様に、データ・ハンドラーはビジネス・オブジェクト・メタデータを使用するように設計されています。例えば次のようになります。

- XML データ・ハンドラーは、各ビジネス・オブジェクト定義に各属性を説明するアプリケーション固有情報が格納されることを必要とします。このテキストにより、データ・ハンドラーは XML エレメント、XML 属性、処理命令、およびその他の XML マークアップ・タイプを識別できます。
- FixedWidth データ・ハンドラーは、MaxLength ビジネス・オブジェクト属性プロパティーの値を使用して、固定長ストリングを構文解析します。
- NameValue データ・ハンドラーは、ビジネス・オブジェクト属性の名前と値を使用します。

メタデータ主導型データ・ハンドラーは、データ・ハンドラーにハードコーディング直接記述された情報でなく、ビジネス・オブジェクト定義にエンコードされたメタデータに基づいて、サポートする各ビジネス・オブジェクトを処理します。したがって、データ・ハンドラーは新規や変更されたビジネス・オブジェクトを処理することができ、データ・ハンドラーのコードを変更する必要はありません。

第 2 章 データ・ハンドラーのインストールと構成

この章では、データ・ハンドラーのインストールと構成方法について説明します。また、コネクターを構成してデータ・ハンドラーをサポートする方法についても説明します。この章を構成するセクションは次のとおりです。

- 『データ・ハンドラーのインストール』
- 『データ・ハンドラーの構成』
- 30 ページの『データ・ハンドラーを使用するようにコネクターを構成する方法』

データ・ハンドラーのインストール

IBM WebSphere Business Integration Server Express 製品には、基本データ・ハンドラー・ファイル CwDataHandlers.jar および XML データ・ハンドラー CwXMLDataHandler.jar が含まれます。そのため、この製品には、5 ページの表 3 にリストしているデータ・ハンドラーが組み込まれています。 (InterChange Server Express 内の) Server Access Interface プロセスは、CwDataHandlers.jar ファイルに入っているどのデータ・ハンドラーにもアクセスすることができます。InterChange Server Express インストーラーは自動的に、このデータ・ハンドラー・ファイルをインストールします。InterChange Server Express インストーラーの使用については、「*WebSphere Business Integration Server Express インストール・ガイド*」を参照してください。

インストールが完了すると、表 6 のファイルがシステムの製品ディレクトリーにインストールされます。

表 6. インストールされるデータ・ハンドラーのファイル構造 (WebSphere Business Integration Express の場合)

ディレクトリー	説明
DataHandlers	CwDataHandler.jar ファイル (IBM 提供データ・ハンドラーのコンパイル・バージョン用) および CwXMLDataHandler.jar ファイル (別個の XML データ・ハンドラーを含む) が格納されます。
DataHandlers¥repository¥DataHandlers	Server Access Interface コンテキストで呼び出されるデータ・ハンドラーのメタオブジェクト定義のテキスト・ファイルが格納されます (InterChange Server Express で使用)。

データ・ハンドラーの構成

データ・ハンドラー・メタオブジェクトにより、コネクターや Server Access Interface プロセスが入力ファイルの MIME タイプまたはビジネス・オブジェクト要求で指定された MIME タイプに基づいてデータ・ハンドラーをインスタンス化できます。データ・ハンドラーを構成するには、そのメタオブジェクトが正しく初期化され、呼び出し元 (コネクターまたはアクセス・クライアント) で使用できることを確認する必要があります。

注: IBM 提供の各データ・ハンドラーは、データ・ハンドラー・メタオブジェクトに定義される構成プロパティーを使用します。

IBM 提供のデータ・ハンドラーをサポートするために、IBM は表 7 にリストするデータ・ハンドラー・メタオブジェクトを提供します。

表 7. IBM 提供のデータ・ハンドラーのメタオブジェクト

メタオブジェクト・レベル	数量	ロケーション
トップレベル		
InterChange Server Express の場合	1 つ	repository¥edk
コネクターの場合	1 つ	DataHandlers¥repository¥DataHandlers
子	各データ・ハンドラーにつき 1 つ	DataHandlers¥repository¥DataHandlers

1 つ以上のデータ・ハンドラーを呼び出し元で使用するように構成するには、次のようにする必要があります。

- トップレベルのメタオブジェクトでは、呼び出し元にサポートされる MIME タイプとその関連データ・ハンドラーを用意します。
- 子メタオブジェクトでは、呼び出し元にデータ・ハンドラーの必要な振る舞いに関する適切な構成情報を用意します。

トップレベルのメタオブジェクト

トップレベルのデータ・ハンドラー・メタオブジェクトにより、MIME タイプが子データ・ハンドラー・メタオブジェクトに関連付けられます。子メタオブジェクトにより構成情報が用意され、インスタンス化するデータ・ハンドラー・クラスの名前が常に格納されます。したがって、トップレベルのメタオブジェクトは MIME タイプとデータ・ハンドラーを関連付けます。特定のトップレベル・メタオブジェクトにアクセスできる呼び出し側のすべてのコンポーネントは、MIME タイプがこのメタオブジェクトに存在する任意のデータ・ハンドラーを呼び出すことができます。

特定のトップレベル・メタオブジェクトの内部で該当する MIME タイプ属性をグループ化し、使用が必要なデータ・ハンドラーを格納しているメタオブジェクトの名前を呼び出し側のコンポーネントから指定することにより、呼び出し側のコンポーネントがサポートできるデータ・ハンドラーを制御できます。IBM は次のトップレベルのデータ・ハンドラー・メタオブジェクトを提供します。

- データ・ハンドラーを起動するアクセス・クライアントが使用できるデータ・ハンドラーを識別する MO_Server_DataHandler。
- データ・ハンドラーを起動するコネクターが使用できるデータ・ハンドラーを識別する MO_DataHandler_Default。

MO_Server_DataHandler メタオブジェクト

Server Access Interface プロセスは MO_Server_DataHandler メタオブジェクトを使用して、使用できるデータ・ハンドラーを識別します。提供される MO_Server_DataHandler のバージョンは、MIME タイプをサポートするようには構

成されていません。単一のダミー属性のみが含まれています。ご使用の WebSphere Business Integration Express にインストールされているデータ・ハンドラーをサポートするように、このメタオブジェクトをカスタマイズすることができます。アクセス・クライアントが MIME タイプをサポートするように設定する場合は、トップレベルの MO_Server_DataHandler メタオブジェクトの dummy 属性を、サポートされる MIME タイプの名前に変更して、その MIME タイプに対して関連する子メタオブジェクトを指定します。

例えば、アクセス・クライアントで text_xml MIME タイプをサポートするには、dummy 属性を text_xml に名前変更し、その属性のタイプとして、その MIME タイプの関連した子メタオブジェクトの名前を指定します。この子メタオブジェクトが XML データ・ハンドラーを構成します。図 9 は、MO_Server_DataHandler メタオブジェクトに、MO_DataHandler_DefaultXMLConfig 子メタオブジェクトを表す属性 text_xml が含まれている場合を示しています。

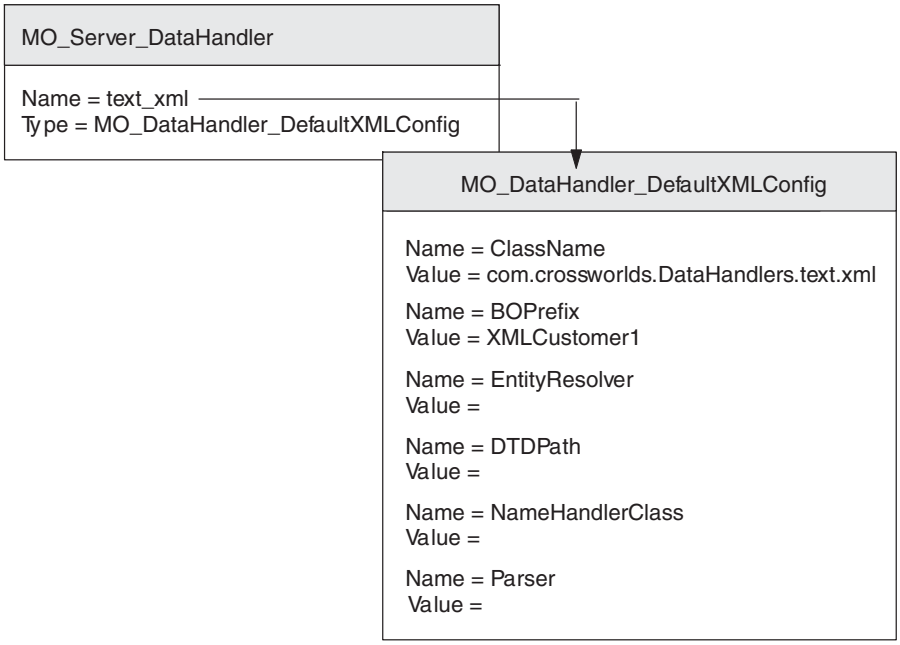

図 9. MO_Server_DataHandler メタオブジェクト

アクセス・クライアントが別の MIME タイプをサポートするようにする場合は、トップレベルの MO_Server_DataHandler メタオブジェクトの新しい属性を MIME タイプごとに定義して、その MIME タイプに対して関連する子メタオブジェクトを指定します。複数のデータ・ハンドラーを起動する場合は、データ・ハンドラー・インスタンスごとに子メタオブジェクトを定義する必要があります。追加の MIME タイプをサポートするには、以下のいずれかの方法があります。

- IBM 提供のデータ・ハンドラー (5 ページの表 3 および 6 ページの表 4 を参照) がサポートしているいずれかの MIME タイプをトップレベルのデータ・ハンドラー・メタオブジェクトに追加する。
- 独自のカスタム MIME タイプおよび子メタオブジェクトを定義する。ただし、この MIME タイプをサポートするデータ・ハンドラーが存在する場合に限ります。

注: データ・ハンドラーのトップレベルのサーバー・メタオブジェクト名は、MO_Server_DataHandler のデフォルト名でなければなりません が、トップレベルのメタオブジェクトに任意の数の子メタオブジェクトを組み込むように構成できます。

MO_DataHandler_Default メタオブジェクト

デフォルトでは、コネクターは MO_DataHandler_Default メタオブジェクトを使用して、使用できるデータ・ハンドラーを識別します。提供される MO_DataHandler_Default は、IBM が提供するすべてのデータ・ハンドラー (この資料には記載されていない一部のアダプター固有データ・ハンドラーを含む) の MIME タイプをサポートするように構成されています。

使用するコネクターが別の MIME タイプをサポートするようにする場合は、コネクターがサポートする各 MIME タイプについて属性が MO_DataHandler_Default メタオブジェクトに存在することを確認する必要があります。この属性は、該当する MIME タイプを指定して、その MIME タイプの関連する子メタオブジェクトを表す必要があります。追加の MIME タイプをサポートするには、独自のカスタム MIME タイプおよび子メタオブジェクトを定義します。ただし、この MIME タイプをサポートするデータ・ハンドラーが存在する場合に限ります。

注: コネクターのトップレベルのメタオブジェクトの名前は、特定のコネクター、さらにはコネクターが処理する必要のある特定のビジネス・オブジェクトまたは特定のファイル・タイプに対応して変更することができます。しかし、どのオブジェクトを使用する場合でも、コネクター定義でサポートされる必要があります。したがって、別のトップレベル・オブジェクトを使用する場合は、それをサポートするようにコネクター定義を構成する必要があります。詳細については、 30 ページの『データ・ハンドラーを使用するようにコネクターを構成する方法』を参照してください。

図 10 に、2 つのデータ・ハンドラー (XML と NameValue) を定義する、コネクターのトップレベルのデータ・ハンドラー・メタオブジェクトを示します。

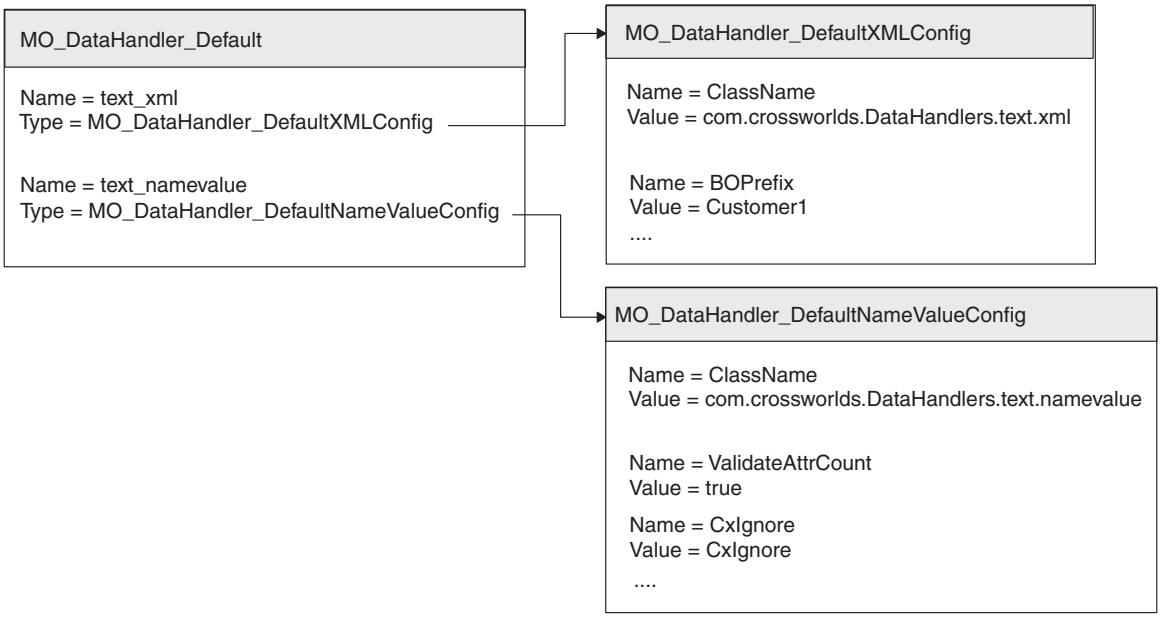

図 10. 2 つの異なるデータ・ハンドラーのメタオブジェクトの例

注: データ・ハンドラーをアクセスできるコネクターの場合、トップレベルのデータ・ハンドラー・メタオブジェクトは、コネクターのサポート・オブジェクトのリストに存在する必要があります。存在しない場合、コネクターは始動時にメタオブジェクトをロードできません。

子メタオブジェクト

子データ・ハンドラー・メタオブジェクトはフラットなビジネス・オブジェクトで、データ・ハンドラーを初期化する構成情報が格納されます。データ・ハンドラーのタイプが異なれば構成要件が異なるので、子メタオブジェクトはさまざまな属性を持ちます。この構成情報により、データ・ハンドラー・インスタンスの振る舞いをカスタマイズします。したがって、子メタオブジェクト内の一連の属性値により特定の構成が定義され、次にそれが特定のデータ・ハンドラーの振る舞いに関連付けられます。特定の子メタオブジェクトにアクセスするすべての呼び出し元は、構成情報によって定義されている関連のデータ・ハンドラーの振る舞いを呼び出します。

- 所定のトップレベルのメタオブジェクトにアクセスするすべての呼び出し元が、特定のデータ・ハンドラーのただ *1* つの 振る舞いのみを必要とする場合、子メタオブジェクト内の該当する構成情報を指定して、この子メタオブジェクトをトップレベルのメタオブジェクト内のデータ・ハンドラーの MIME タイプと関連付けます。

  例えば次のようになります。

  – 特定のデータ・ハンドラーの振る舞いにアクセスするすべてのコネクターの場合は、コネクターのトップレベルのメタオブジェクト内のデータ・ハンドラーの MIME タイプに、子メタオブジェクトが関連付けられていることを確認します (デフォルトでは `MO_DataHandler_Default`)。

– 特定のデータ・ハンドラーの振る舞いにアクセスするすべてのアクセス・クライアントで、サーバーのトップレベルのメタオブジェクト内のデータ・ハンドラーの MIME タイプに、子メタオブジェクトが関連付けられていることを確認します (MO_Server_DataHandler)。

- ある特定のトップレベル・メタオブジェクトにアクセスする呼び出し元が、特定のデータ・ハンドラーの複数の 振る舞いを必要とする場合は、データ・ハンドラーの振る舞いごとに該当の構成情報を持つ子メタオブジェクトを作成して、それぞれの子メタオブジェクトを (トップレベルのデータ・ハンドラー・メタオブジェクト) の固有な MIME タイプ名に関連付けます。

  IBM では、子メタオブジェクトの名前を MIME タイプ/サブタイプの固有の組み合わせにすることをお勧めします。

  text_*MIMEtype* _*subtype*

  ここで、

  – MIME タイプ (*MIMEtype*) は、データ・ハンドラーがサポートする MIME タイプを表します。
  – MIME サブタイプ (*subtype*) は、データ・ハンドラーの特定の振る舞いを表します。

  例えば、コネクターがすべてデフォルトの XML データ・ハンドラーと SGML バージョンの*両方* をサポートできる場合、text_xml と text_xml_sgml の MIME タイプを作成できます。MIME タイプ名に使用できる文字は、英数字とピリオド (.) および下線 (_) の 2 つの特殊文字のみです。

IBM は、表 8 に示すように、提供する各データ・ハンドラーに関する子データ・ハンドラー・メタオブジェクトを提供します。

表 8. 子データ・ハンドラー・メタオブジェクト

子メタオブジェクト	詳細
MO_DataHandler_DefaultXMLConfig	43 ページの『XML データ・ハンドラーの構成』
MO_DataHandler_DefaultFixedWidthConfig	120 ページの『FixedWidth データ・ハンドラーの構成』
MO_DataHandler_DefaultDelimitedConfig	130 ページの『Delimited データ・ハンドラーの構成』
MO_DataHandler_DefaultNameValueConfig	140 ページの『NameValue データ・ハンドラーの構成』
MO_DataHandler_DefaultRequestResponseConfig	111 ページの『Request-Response データ・ハンドラーの構成』

データ・ハンドラーを使用するようにコネクターを構成する方法

データ・ハンドラーをコネクター・コンテキストで実行する場合、データ・ハンドラーを使用するようにコネクターを構成する必要があります。

- コネクターは、データ・ハンドラーをインスタンス化するためにデータ・ハンドラー・メタオブジェクトにアクセスできる必要があります。

  データ・ハンドラーを初めて起動する前に、コネクターはデータ・ハンドラー基本クラス内の静的プロパティーにトップレベルのデータ・ハンドラー・メタオブ

ジェクトの名前を設定します。このトップレベルのメタオブジェクトから、データ・ハンドラーはその構成情報を取得します。その後データ・ハンドラーがインスタンス化されるたびに、そのデータ・ハンドラー・インスタンスの構成プロパティーが取得されます。データ・ハンドラーは、その作業を実行するためにこの構成情報にアクセスできる必要があります。

- データ・ハンドラーをインスタンス化するには、コネクターがそのデータ・ハンドラーのクラス名かまたはそのデータの MIME タイプを認識する必要があります。

  コネクターが createHandler() を呼び出してデータ・ハンドラーを起動する際に、クラス名またはそのデータの MIME タイプを渡します。

  – コネクターから MIME タイプが渡されると、createHandler() メソッドはトップレベルのデータ・ハンドラー・メタオブジェクトに名前が MIME タイプと一致する属性があるかを調べます。一致する属性が見つかると、createHandler() メソッドは MIME タイプに関連する子メタオブジェクト内の ClassName 属性の値を調べます。
  – コネクターからクラス名が渡されると、createHandler() メソッドはそのクラス名のデータ・ハンドラーをインスタンス化します。

  コネクターから正しいクラス名または MIME タイプが渡されない場合、インスタンス化プロセスは失敗します。詳細については、14 ページの『データ・ハンドラー・クラスの識別』を参照してください。

コネクターは、この構成情報を取得するためにさまざまな方法で構成されます。例えば次のようになります。

- Adapter for XML には、トップレベルのメタオブジェクトの名前を指定する構成プロパティー DataHandlerConfigMO があります。このプロパティーが設定されていないと、コネクターはメタオブジェクトを見つけることができません。さらに、XML コネクターのすべてのトップレベル・ビジネス・オブジェクトには、ビジネス・オブジェクト内のデータの MIME タイプを指定する MimeType 属性が必要です。コネクターは MimeType 属性値を使用して、該当するデータ・ハンドラーを起動します。
- Adapter for JText には、独自の構成メタオブジェクトがあります。このメタオブジェクトには、ClassName、DataHandlerConfigMO、および MimeType 属性があり、それぞれクラス名、データ・ハンドラー・メタオブジェクト、およびファイルの MIME タイプを指定します。

その他のコネクターでは、データ・ハンドラーを使用するための構成方法が異なっていることがあります。詳細については、コネクターのアダプター・ガイドを参照してください。

コネクターがトップレベルのデータ・ハンドラー・メタオブジェクトを見つけることができないか、またはクラス名か MIME タイプを判別できない場合は、データ・ハンドラーを作成できません。したがって、データ・ハンドラーを使用するようにコネクターを構成するには、次のことを行ってください。

1. コネクターのトップレベルのデータ・ハンドラー・メタオブジェクトの名前の構成方法を決定します。メタオブジェクト名のスペルが正しいことを確認します。

2. MIME タイプの構成方法を決定します。MIME タイプのスペルが正しいことを確認します。
3. トップレベルのデータ・ハンドラー・メタオブジェクトが、コネクターのサポート・オブジェクトのリストに存在することを確認します。
4. データ・ハンドラーの子メタオブジェクトに、データ・ハンドラー・クラス名の値が (ClassName 属性に) 正しく指定されていることを確認します。

第 3 章 XML データ・ハンドラー

IBM WebSphere Business Integration Data Handler for XML は、*XML データ・ハンドラー*と呼ばれており、ビジネス・オブジェクトを XML 文書に、また XML 文書をビジネス・オブジェクトに変換します。XML データ・ハンドラーのインストールの手順については、25 ページの『データ・ハンドラーのインストール』を参照してください。

注: XML データ・ハンドラーは XML バージョン 1.0 をサポートします。

この章では、XML データ・ハンドラーによる XML 文書の処理方法と、XML データ・ハンドラーで処理されるビジネス・オブジェクトの定義方法を説明します。また、XML データ・ハンドラーの構成方法についても説明します。この章を構成するセクションは次のとおりです。

- 『概要』
- 37 ページの『ビジネス・オブジェクト定義の要件』
- 43 ページの『XML データ・ハンドラーの構成』
- 47 ページの『DTD を使用する XML 文書』
- 61 ページの『スキーマ文書を使用する XML 文書』
- 89 ページの『ビジネス・オブジェクト定義の作成』
- 92 ページの『ビジネス・オブジェクトの XML 文書への変換』
- 94 ページの『XML 文書のビジネス・オブジェクトへの変換』
- 96 ページの『XML データ・ハンドラーのカスタマイズ』

概要

XML データ・ハンドラーはデータ変換モジュールで、その主な役割はビジネス・オブジェクトと XML 文書相互間の変換をすることです。XML 文書は、`text/xml` MIME タイプの直列化データです。 XML データ・ハンドラーは、コネクターおよびアクセス・クライアントで使用できます。

この概要のセクションでは、XML データ・ハンドラーの次の内容について説明します。

- 『XML 文書およびビジネス・オブジェクトの処理』
- 34 ページの『XML データ・ハンドラーのコンポーネント』

XML 文書およびビジネス・オブジェクトの処理

XML 文書では、スキーマと呼ばれるテンプレートを使用して、その構造を定義します。 34 ページの表 9 に、このスキーマを定義するための最も一般的なデータ・モデルを示します。

表 9. XML データ・モデル

XML データ・モデル	詳細
文書タイプ定義 (DTD) スキーマ文書	47 ページの『DTD を使用する XML 文書』 61 ページの『スキーマ文書を使用する XML 文書』

DTD またはスキーマ文書により XML 文書の構造が記述されるように、ビジネス・オブジェクトの構造はビジネス・オブジェクト定義により記述されます。XML データ・ハンドラーは、ビジネス・オブジェクトと XML 文書の間の変換を実行するとき、ビジネス・オブジェクト定義を使用します。XML データ・ハンドラーは、ビジネス・オブジェクト定義の構造およびアプリケーション固有情報を使用することにより、変換方法を決定します。ビジネス・オブジェクト定義を正しく作成することにより、データ・ハンドラーがビジネス・オブジェクトと XML 文書との相互の変換を正しく行うことができます。XML データ・ハンドラーが XML 文書とビジネス・オブジェクトの間の変換を実行するためには、関連するビジネス・オブジェクト定義を見つけることが必要です。

XML 文書からビジネス・オブジェクトへの変換、およびその逆の変換に XML データ・ハンドラーを使用するためには、次のステップを実行することが必要です。

表 10. XML データ・ハンドラーの使用

ステップ	詳細
1. XML 文書とビジネス・オブジェクトの構造を記述しているビジネス・オブジェクト定義が存在し、実行中の XML データ・ハンドラーから使用できること。	37 ページの『ビジネス・オブジェクト定義の要件』 60 ページの『DTD からのビジネス・オブジェクト定義の作成』
2. XML データ・ハンドラーが運用環境向けに構成されていること。	43 ページの『XML データ・ハンドラーの構成』
3. 適切なデータ操作を実行するために、XML データ・ハンドラーがコネクター (またはアクセス・クライアント) から呼び出されていること。	
a) データ操作: 呼び出し元からビジネス・オブジェクトを受け取り、これを XML 文書に変換し、変換結果を呼び出し元に渡します。	92 ページの『ビジネス・オブジェクトの XML 文書への変換』
b) データ操作: 呼び出し元から XML 文書を受け取り、ネーム・ハンドラーおよび SAX パーサーを使用して、ビジネス・オブジェクトを作成します。次にデータ・ハンドラーがこのビジネス・オブジェクトを呼び出し元に戻します。	94 ページの『XML 文書のビジネス・オブジェクトへの変換』

XML データ・ハンドラーのコンポーネント

XML データ・ハンドラーは次の 2 つのコンポーネントを使用して、XML データをビジネス・オブジェクトに変換します。

- ネーム・ハンドラー
- XML (SAX) パーサーの単純 API

- (オプション) XML 文書に DTD が存在し、エンティティー参照が設定されている場合には、XML データ・ハンドラーは、もう 1 つのコンポーネントであるエンティティー・リゾルバーを使用して参照を解決します。

図 11 に、XML データ・ハンドラーのコンポーネントとその相互の関係を示します。これらのコンポーネントは、この後のセクションで説明します。

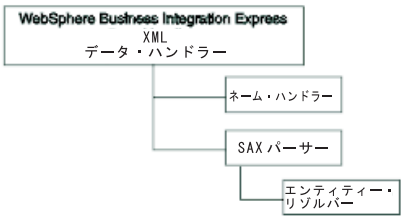

図 11. XML データ・ハンドラーのコンポーネント

ネーム・ハンドラー

XML データ・ハンドラーはネーム・ハンドラーを使用して、XML メッセージからビジネス・オブジェクトの名前を抽出します。データ・ハンドラーは、XML データ・ハンドラー子メタオブジェクト内の `NameHandlerClass` 属性値を基に、ネーム・ハンドラーのインスタンスを呼び出します。

- クラス名が `NameHandlerClass` 属性に指定されていると、XML データ・ハンドラーはネーム・ハンドラーを使用してビジネス・オブジェクト名を判別します。
- クラス名が指定されていないと、データ・ハンドラーはデフォルトのネーム・ハンドラーを使用してビジネス・オブジェクト名を判別します。デフォルトのネーム・ハンドラーは、XML 文書の root エレメントの名前および `BOPrefix` プロパティーを使用して、ビジネス・オブジェクト名を生成します。

  `BOPrefix_rootElement`

カスタム・ネーム・ハンドラーの作成方法については、96 ページの『カスタム XML ネーム・ハンドラーの作成』を参照してください。

SAX パーサー

XML 子メタオブジェクト内 `Parser` 属性の Default Value プロパティーにパーサーが*指定されていない* 場合には、データ・ハンドラーはデフォルトの SAX パーサーを使用します。

```
org.apache.xerces.parsers.SAXParser
```

検証パーサーを使用するには、次のステップのいずれかを実行します。

- XML 子メタオブジェクト内の `Validation` 属性の Default Value プロパティーを true に設定します。 IBM がこの属性に対して提供しているデフォルト値は false です。
- IBM から提供されている検証 SAX パーサーを使用するには、`Parser` 属性の Default Value プロパティー内にあるクラス名を下記に変更します。

  ```
  com.ibm.xml.parsers.ValidatingSaxParser
  ```

ビジネス・オブジェクト定義が DTD に基づいている場合には、ローカル・エンティティー・リゾルバーを使用し、DTDPath 属性内の文書タイプ定義 (DTD) またはスキーマ・パスに Default Value を指定します。すべての DTD または Schema ファイルを、DTDPath 属性に指定した場所に配置してください。

注: 検証パーサーを使用する場合、正しい EntityResolver を使用していて、DTDPath が正しく設定されていることを確認します。確認を行うための命令については、43 ページの『XML データ・ハンドラーの構成』を参照してください。

あるいは、IBM 提供の検証用でない SAX パーサーを使用する方法もあります。このパーサーを使用するには、XML 子メタオブジェクトの Parser 属性 の Default Value プロパティーを値 com.ibm.xml.parsers.SAXParser に設定します。

エンティティー・リゾルバー

エンティティー・リゾルバーにより、SAX パーサーによる XML データ内の外部参照 (DTD およびスキーマ文書の参照など) の解決方法を指定します。 XML 文書にエンティティー参照が含まれている場合、SAX パーサーは XML データ・ハンドラー構成メタオブジェクト内の EntityResolver 属性を使用してエンティティー・リゾルバーのインスタンスを起動します。

外部参照の処理は、EntityResolver で指定されるエンティティー･リゾルバーのクラスにより異なります。表 11 は、XML データ・ハンドラーが提供するエンティティー･リゾルバーのクラスを表しています。

表 11. XML データ・ハンドラーのエンティティー･リゾルバーのクラス

エンティティー･リゾルバーのクラス	説明
DefaultEntityResolver LocalEntityResolver	デフォルトのエンティティー・リゾルバーのクラスです。このエンティティー・リゾルバーが呼び出されると、すべての外部参照は無視されます。このローカル・エンティティー・リゾルバーは、外部参照をローカル・ファイル名として処理します。検証用に使用するデータ・モデルにより、振る舞いが異なります。 • 検証用に DTD を使用する場合、ローカル・エンティティー･リゾルバーは、systemID が file:// または http:// で始まり、DTDPath 属性が設定されていれば、systemID 内のパスを DTDPath メタオブジェクト属性の値で置き換えます。 systemID がパス名でないか、または DTDPath 属性が設定されていない場合、外部参照は無視されます。 • 検証用に スキーマ文書を使用する場合、ローカル・エンティティー･リゾルバーは、パスが file:// または http:// で始まるか、DOS ファイル名 (例えば、「D:¥xmlschemas¥test」) を含んでいれば、schemaLocation または noNamespaceSchemaLocation 属性が指定するパスを DTDPath メタオブジェクト属性の値に置き換えます。

表 11. XML データ・ハンドラーのエンティティー・リゾルバーのクラス (続き)

エンティティー・リゾルバーのクラス	説明
URIEntityResolver	このエンティティー・リゾルバーは、外部参照をローカル・ファイル名またはダウンロード可能 URL として処理します。外部参照は次のいずれかのケースで動的に解決されます。 • 検証用に DTD を使用する場合: DOCTYPE が http:// または file:// で始まる SYSTEM 値を含んでいるとき。 • 検証用にスキーマ文書を使用する場合: schemaLocation または noNamespaceSchemaLocation 属性が http:// または file:// で始まるとき。 次にエンティティー・リゾルバーは、HTTP 接続を開き、指定された Web サイトから DTD またはスキーマ文書をダウンロードします。 **重要:** XML データ・ハンドラーは DTD またはスキーマ文書をキャッシュに入れません。データ・ハンドラーがエンティティー・リゾルバーとして、URIEntityResolver クラスを使用する場合、XML 文書を構文解析するたびに HTTP 接続を開きます。したがって、ネットワーク・トラフィックにより XML データ・ハンドラーのパフォーマンスが影響されます。

注: 表 11 のエンティティー・リゾルバーのクラスには必ず、次のクラス・プレフィックスが付いています。

```
com.crossworlds.DataHandlers.xml
```

XML 文書でスキーマ文書を使用する場合、そのスキーマ文書に組み込まれているすべての外部スキーマは、外部エンティティーとしても取り扱われます。したがって、SAX パーサーは、これらの組み込まれたスキーマ文書を解決するために、エンティティー・リゾルバーを呼び出します。XML 文書がスキーマ・ロケーションを指定するために schemaLocation または noNamespaceSchemaLocation を使用する場合は、EntityResolver 属性を LocalEntityResolver または URIEntityResolver のいずれかに、(組み込み済みまたはインポート済みの) 外部スキーマ文書の検証用として、設定が可能です。

別の方法を使用して外部エンティティーを検出する必要がある場合は、カスタム・エンティティー・リゾルバーを作成する必要があります。カスタム・エンティティー・リゾルバーの作成の詳細については、 98 ページの『カスタム・エンティティー・リゾルバーの作成』を参照してください。

ビジネス・オブジェクト定義の要件

ビジネス・オブジェクト定義が XML データ・ハンドラーの要件に適合することを確認するには、このセクションのガイドラインを使用します。ガイドラインは次から構成されます。

- 38 ページの『ビジネス・オブジェクトの構造』
- 39 ページの『ビジネス・オブジェクトの属性プロパティー』
- 42 ページの『アプリケーション固有情報』
- 43 ページの『ビジネス・オブジェクト動詞』

ビジネス・オブジェクト定義を正しく作成することにより、データ・ハンドラーがビジネス・オブジェクトと XML 文書との相互の変換を正しく行うことができます。XML データ・ハンドラーのビジネス・オブジェクトの作成方法の詳細については、60 ページの『DTD からのビジネス・オブジェクト定義の作成』を参照してください。

ビジネス・オブジェクトの構造

DTD またはスキーマ文書を表すには、少なくとも以下の 2 つのビジネス・オブジェクト定義が必要です。

- *トップレベル・ビジネス・オブジェクト定義*は、DTD またはスキーマ文書を定義する情報を表し、以下の属性が含まれている必要があります。
  - XML バージョンを表す属性 `XMLDeclaration`

    この属性のアプリケーション固有情報には、`type=pi` タグの設定が必要です。
  - DTD またはスキーマ文書内の root エレメントを表す属性

    この属性には、タイプとして単一カーディナリティーのビジネス・オブジェクトを指定し、そのタイプが DTD またはスキーマ文書の root エレメントのビジネス・オブジェクト定義である必要があります。 XML ODA は、この root エレメントの名前を `Root` ODA 構成プロパティーから取得します。アプリケーション固有情報に、`elem_name` タグを使用してこのエレメントの名前がリストされている必要があります。

    注: `elem_name` タグは、ビジネス・オブジェクト属性のアプリケーション固有情報内の XML エレメントの名前のみを必要とした以前の構文に代わるものです。XML データ・ハンドラーは、既存のビジネス・オブジェクト定義との後方互換性のために以前の構文もサポートします。ただし、XML ODA でビジネス・オブジェクト定義を生成するときには新規構文を使用します。
- XML 定義文書の root エレメントを表す *root エレメント・ビジネス・オブジェクト定義*。 XML ODA の `Root` 構成プロパティーを使用して、root エレメントとするエレメントを指定できます。構成プロパティーには、root エレメント内の各 XML コンポーネントの属性が含まれます。

DTD またはスキーマ文書のビジネス・オブジェクト定義を使用して XML データ・ハンドラーにより処理されるビジネス・オブジェクトは、次の追加規則に準拠していることが必要です。

- XML 文書内のすべてのタグは、ビジネス・オブジェクト内に関連する属性が存在する必要があります。ビジネス・オブジェクト定義によりビジネス・オブジェクト属性のタイプが規定され、その属性に対応するアプリケーション固有情報が格納されています。この情報は、XML エレメントの構造および内容により決定されます。
- XML エレメントのビジネス・オブジェクト定義では、XML 属性を表すすべての属性は、ほかの属性より*前* にあることが必要です。 XML データ・ハンドラーは、いずれかの XML エレメントがあった場合、その属性がビジネス・オブジェクト定義内の*最初* の属性であることを前提としています。

注: ビジネス・オブジェクトには必要なデータが設定されて、XML データ・ハンドラーが有効な XML 文書を作成できるようにする必要があります。データなしでデータ・ハンドラー・ビジネス・オブジェクトを送信しないようにします。

本書では、DTD およびスキーマ文書のビジネス・オブジェクト定義の構造について次の情報を提供します。

データ・モデル	詳細
文書タイプ定義 (DTD)	48 ページの『DTD のビジネス・オブジェクト構造』
スキーマ文書	61 ページの『スキーマ文書のビジネス・オブジェクト構造』

ビジネス・オブジェクトの属性プロパティー

ビジネス・オブジェクト定義により属性が定義されます。各属性には、その属性に関する情報を提供する各種のプロパティーがあります。このセクションでは、XML データ・ハンドラーがこれらのプロパティーの一部をどのように解釈するかについて説明するとともに、ビジネス・オブジェクト定義を変更するためにプロパティーを設定する方法について説明します。

Name 属性プロパティー

各ビジネス・オブジェクト属性には、固有の名前を付ける必要があります。XML エレメントまたは XML 属性の名前は、常に elem_name または attr_name タグ内で指定されます。この場合、属性のアプリケーション固有情報の elem_name (または attr_name) タグに指定された XML エレメント (または属性) の名前に特殊文字が含まれます。ただし、(これらの特殊文字を使用できない) ビジネス・オブジェクト属性の名前では、特殊文字は省略されます。

Type 属性プロパティー

各ビジネス・オブジェクト属性は、次のように、整数、ストリング、このオブジェクトに含まれる子ビジネス・オブジェクトのタイプなど、1 つのタイプを持っていることが必要です。

- DTD の場合、子エレメントまたは 1 つ以上の FIXED 以外の属性を持つ XML エレメントは、ビジネス・オブジェクトとして取り扱われます。PCDATA 値のみを持つ XML エレメントは、XML エレメントが単一カーディナリティーによりその親に含まれる場合は、属性として取り扱われます。複数カーディナリティーにより含まれる場合は、ビジネス・オブジェクトと見なされます。それはビジネス・オブジェクト定義は複数カーディナリティーのスカラー値 (例えば、String 値の配列) をサポートしないからです。
- スキーマ文書の場合、各ビジネス・オブジェクト属性には、ストリングまたはこのオブジェクトに含まれる子ビジネス・オブジェクトのタイプを設定する必要があります。複数の子エレメントまたは複合タイプを含む XML エレメントは、ビジネス・オブジェクトとして取り扱われます。単純タイプの値のみを持つ XML エレメントは、単一カーディナリティーによりその親に含まれている場合には、ビジネス・オブジェクト属性として取り扱われます。複数カーディナリティーに

より含まれる場合は、ビジネス・オブジェクトと見なされます。それはビジネス・オブジェクト定義は複数カーディナリティーのスカラー値 (例えば、String 値の配列) をサポートしないからです。

注: 単純属性はすべて、`String` タイプです。

Key 属性および Foreign Key 属性のプロパティー

各ビジネス・オブジェクトには、1 つ以上の基本キー属性が必要です。基本キー属性は、属性に対する Key プロパティーを true に設定することにより指定されます。Foreign Key プロパティーの設定はオプショナルで、XML 文書の構造に依存します。このセクションでは、次に示す Key 属性および Foreign Key 属性のプロパティーに関連する情報について説明します。

- 『ビジネス・オブジェクト定義のキーの指定』
- 『キーおよび「必須性」の処理』

ビジネス・オブジェクト定義のキーの指定: 以前のバージョンの XML ビジネス・オブジェクト定義生成ツール (XMLBorgen、Edifecs SpecBuilder、および XML ODA など) では、生成ツールが、ObjectEventId 属性を親 XML ビジネス・オブジェクトのキーとして指定していました。しかし、今回のリリースより、Business Object Designer Express では ObjectEventId 属性がキーとして指定されているビジネス・オブジェクト定義を保管できなくなりました。

この制限により、現行バージョンの XML ODA は次に示すアクションを行ないます。

- それぞれの子ビジネス・オブジェクトで、最初の属性をキーとして設定します。
- 親ビジネス・オブジェクトでは、キー属性を設定しません。

XML ODA が生成する親ビジネス・オブジェクト定義にキーを指定するためには、ビジネス・オブジェクト定義を Business Object Designer Express に送り、そのビジネス・オブジェクト定義を分析して、キーとして指定する適切な属性を判別する必要があります。ビジネス・オブジェクト定義のキー属性の変更は、Business Object Designer Express でそのビジネス・オブジェクト定義を保管する*前* に行なう必要があります。

注: XML ODA は、以前の XML ビジネス・オブジェクト定義生成ツール (XMLBorgen や Edifecs SpecBuilder など) に代わるものです。そのため、ObjectEventId が親ビジネス・オブジェクトのキー属性として指定されることを回避するためのこれらの特別なステップは、XML ODA でのみ行なうことができます。以前の XML ビジネス・オブジェクト定義生成ツール (以前のバージョンの XML ODA も含む) で生成した、既存の XML ビジネス・オブジェクト定義がある場合、それらのビジネス・オブジェクト定義では引き続き、ObjectEventId をキーとして使用する可能性があります。これらのビジネス・オブジェクトを現行リリースにマイグレーションする場合は、そのビジネス・オブジェクト定義を調べてください。ビジネス・オブジェクト定義で適切なキー属性の設定をしないと、イベント・シーケンス機能のパフォーマンスに対して好ましくない影響を及ぼす可能性があります。

キーおよび「必須性」の処理: 本書では次の情報を提供すると共に、キーと「必須性」の関係についても説明します。

データ・モデル	詳細
文書タイプ定義 (DTD) スキーマ文書	49 ページの『DTD のビジネス・オブジェクト属性プロパティー』 70 ページの『スキーマ文書のビジネス・オブジェクト属性プロパティー』

Required 属性プロパティー

このプロパティーが単一カーディナリティーの子ビジネス・オブジェクトを含む属性に対して指定されると、XML データ・ハンドラーは親ビジネス・オブジェクトにこの属性の子ビジネス・オブジェクトが含まれていることを要求します。Cardinality、Key、および Foreign Key 属性プロパティーは、属性の Required プロパティーの設定に影響を与えることがあります。

本書では、必須かどうかについて次の情報を提供します。

データ・モデル	詳細
文書タイプ定義 (DTD) スキーマ文書	49 ページの『DTD のビジネス・オブジェクト属性プロパティー』 70 ページの『スキーマ文書のビジネス・オブジェクト属性プロパティー』

Cardinality 属性プロパティー

Cardinality プロパティーは、ビジネス・オブジェクト定義をタイプとしている属性の場合に、許容される子ビジネス・オブジェクトの数を示します。このプロパティーの設定は、XML 文書およびそのエレメントの構造に依存します。また、この設定は、属性が必須であることを必要とするかどうか (Required プロパティーが true に設定されているかどうか) にも影響します。

本書では、カーディナリティーと「必須性」の関係について次の情報を提供します。

データ・モデル	詳細
文書タイプ定義 (DTD) スキーマ文書	49 ページの『DTD のビジネス・オブジェクト属性プロパティー』

特殊属性値

ビジネス・オブジェクト属性には 1 つの値があり、この値のタイプは、その属性の Type プロパティーと一致しています。さらに、属性は、次の 2 つの特別な値のいずれか 1 つを持つことができます。

- CxIgnore

  XML データ・ハンドラーは、統合ブローカーからビジネス・オブジェクトを受け取ると、CxIgnore の値を持つ属性をすべて 無視します。これらの属性はデータ・ハンドラーに不可視の状態になります。したがって、データ・ハンドラーは、対応する XML エレメントを生成しません。つまり、データ・ハンドラーは

この属性に対して XML *タグを作成しません* (空のタグも作成しません)。XML データ・ハンドラーは、ビジネス・オブジェクト属性に対応する XML タグを持たない XML 入力を受け取ると、その属性に CxIgnore の値を割り当てます。

- CxBlank

  XML データ・ハンドラーは、統合ブローカーからビジネス・オブジェクトを受け取ると、CxBlank 属性値を、その属性の Type プロパティーに基づいて処理します。

  – データ・ハンドラーは、複合属性 (つまり、Type プロパティーに別のビジネス・オブジェクト定義の名前が設定されている属性) の場合、複合属性に CxBlank 値がないことを前提としています。
  – 単純属性 (つまり、Type プロパティーに `String` データ・タイプが設定されている属性) の場合、データ・ハンドラーは、XML 文書に空のタグを作成します。DTD を基にした XML 文書の場合、空の二重引用符 (″ ″) が、CxBlank と同等の PCDATA として使用されます。

XML データ・ハンドラーは、空のタグを持つ XML 入力を受け取ると、対応するビジネス・オブジェクト属性に CxBlank の値を割り当てます。

アプリケーション固有情報

ビジネス・オブジェクト定義内のアプリケーション固有情報により、データ・ハンドラーにビジネス・オブジェクト定義を XML 文書に変換する指示が与えられます。アプリケーション固有情報により、データ・ハンドラーがビジネス・オブジェクトを正しく処理できるようになります。したがって、XML データ・ハンドラーに関して新しいビジネス・オブジェクトを作成するかまたは既存のビジネス・オブジェクトを変更する場合は、ビジネス・オブジェクト定義内のアプリケーション固有情報がデータ・ハンドラーが予期する構文規則に一致することを確認します。XML データ・ハンドラーは次の種類のアプリケーション固有情報を使用することができます。

- ビジネス・オブジェクト・レベルのアプリケーション固有情報は、ビジネス・オブジェクト定義全体に関する情報を提供します。
- 属性レベルのアプリケーション固有情報は、特定の属性に関する情報を提供します。

注: XML データ・ハンドラーはアプリケーション固有情報を使用して、XML 文書のコンポーネントとビジネス・オブジェクト内の属性を突き合わせます。アプリケーション固有情報の最大長は 255 文字です。アプリケーション固有情報の値が 255 文字を超える場合には、DTD またはスキーマ文書を再構築し、ビジネス・オブジェクトを再生成することが必要です。

本書では、アプリケーション固有情報について次の情報を提供します。

データ・モデル	詳細
文書タイプ定義 (DTD)	50 ページの『DTD の XML コンポーネントのアプリケーション固有情報』
スキーマ文書	71 ページの『スキーマ文書内の XML コンポーネントのアプリケーション固有情報』

ビジネス・オブジェクト動詞

ビジネス・オブジェクトを XML 文書に変換する場合、XML データ・ハンドラーは動詞に関する XML を生成せず、また XML 文書をビジネス・オブジェクトに変換する際に動詞の設定も行いません。しかし、動詞の情報は次のいずれかの方法で保持されます。

- DTD またはスキーマ文書の中で、動詞に対応するエレメントを作成し、動詞に対応するビジネス・オブジェクト属性を作成することができます。次にビジネス・インテグレーション・システムの内容を作成して、動詞をビジネス・オブジェクト属性にコピーできます。その後データ・ハンドラーが動詞を XML エレメントに変換し、これにより動詞が XML 文書に保持されます。XML 文書が戻されると、ビジネス・インテグレーション・システムがビジネス・オブジェクト属性の値に従って動詞を設定できます。
- ビジネス・オブジェクトと動詞の特定の組み合わせに対応して DTD またはスキーマ文書を作成し、各ビジネス・オブジェクト要求を、そのビジネス・オブジェクトと動詞の組み合わせに対応する DTD またはスキーマ文書に関連付けることができます。XML 文書がビジネス・オブジェクトに変換され、呼び出し元に戻ったとき、コネクターは、DTD またはスキーマ文書に対応する動詞を設定することができます。
- 呼び出し元コネクターで動詞についての情報を提供できる場合は、ビジネス・オブジェクトに動詞を設定してから、統合ブローカーに送ることができます。

XML データ・ハンドラーの構成

XML データ・ハンドラーを構成するには、その構成情報が XML 子メタオブジェクトに設定されている必要があります。

注: XML データ・ハンドラーを構成するには、ビジネス・オブジェクト定義を作成または変更してデータ・ハンドラーをサポートするようにすることも必要です。詳細については、37 ページの『ビジネス・オブジェクト定義の要件』を参照してください。

XML データ・ハンドラー向けに、IBM からデフォルト子メタオブジェクト MO_DataHandler_DefaultXMLConfig が提供されています。このメタオブジェクト内の各属性により、XML データ・ハンドラーのために構成プロパティーが定義されます。表 12 で、この子メタオブジェクト内の属性について説明します。

表 12. XML データ・ハンドラーの子メタオブジェクト属性

属性名	説明	納入時のデフォルト値
BOPrefix	ビジネス・オブジェクト名を作成するために、デフォルトの NameHandler クラスで使用されるプレフィックス。デフォルト値は、関連するビジネス・オブジェクト定義の名前に一致するよう変更する必要があります。属性値は大文字小文字を区別します。	XMLTEST

表 12. XML データ・ハンドラーの子メタオブジェクト属性 (続き)

属性名	説明	納入時のデフォルト値
ClassName	指定された MIME タイプで使用するためにロードするデータ・ハンドラー・クラスの名前。トップレベルのデータ・ハンドラー・メタオブジェクトには、指定された MIME タイプと名前が一致し、タイプが XML 子メタオブジェクトである属性があります (表 12 で説明)。	com.crossworlds. DataHandlers. text.xml
DefaultEscapeBehavior	属性値に特殊文字が含まれている場合、XML データ・ハンドラーはエスケープ処理の実行が必要となります。属性のアプリケーション固有情報に escape タグが設定されていない 場合、XML データ・ハンドラーは、DefaultEscapeBehavior プロパティーを調べることにより、エスケープ処理を実行するかどうか決定します。詳細については、 58 ページの『特殊文字が含まれる XML エレメントまたは属性の場合』を参照してください。	true
DTDPath	データ・ハンドラーが文書タイプ定義 (DTD) またはスキーマ (XSD) へのパスを構成するのに使用します。	なし
DummyKey	キー属性。データ・ハンドラーは使用しませんが、ビジネス・インテグレーション・システムには必要です。	1
EntityResolver	DTD やスキーマなどの外部エンティティーの参照を処理するのに使用するクラス名。この属性に関する値の詳細については、 36 ページの『エンティティー・リゾルバー』を参照してください。	なし
IgnoreUndefinedAttributes	この属性が false に設定されている場合は、XML データ・ハンドラーがすべての XML 属性をビジネス・オブジェクト定義と突き合わせて検証するときに、未定義の属性があると、例外がスローされます。この属性が true に設定さている場合、XML データ・ハンドラーは未定義の XML 属性をすべて無視し、警告を生成します。	true
IgnoreUndefinedElements	この属性が false に設定されていると、XML データ・ハンドラーはすべての XML エレメントをビジネス・オブジェクト定義と照合し、アプリケーション固有情報に定義されていないエレメントを検出すると、例外をスローします。この属性が true に設定されていると、XML データ・ハンドラーはすべての未定義 XML エレメント (および未定義エレメント内のすべての属性) を無視し、警告を発行します。	false
InitialBufferSize	ビジネス・オブジェクトを XML に変換する際に使用するバッファーの初期サイズを定義します。この値を、XML ビジネス・オブジェクトのサイズ (バイト単位) に設定してください。この値に大きな数値を設定すると、ビジネス・オブジェクトから直列化 XML への変換が速くなります。	2 MB (2,097,152 KB)

表 12. XML データ・ハンドラーの子メタオブジェクト属性 (続き)

属性名	説明	納入時のデフォルト値
NameHandlerClass	XML 文書の内容からビジネス・オブジェクトの名前を判別するのに使用するクラス名。独自のカスタム・ネーム・ハンドラーを作成する場合は、この属性のデフォルト値を変更します。詳細については、 96 ページの『カスタム XML ネーム・ハンドラーの作成』を参照してください。	com. crossworlds. DataHandlers.xml. TopElementNameHandler
Parser	XML 文書の SAX 準拠パーサーのパッケージ名。	なし
UseNewLine	新しい行に出力 XML 内の各タグを付ける場合は true に設定します。(XML データ・ハンドラーは、改行 (LF) および復帰 (CR) の形式で追加の内容を XML 文書に付加します。) XML 出力を変更しない場合は、false に設定します。	false
Validation	この値はデータ・ハンドラーに使用されて、検証パーサーを使用することを指定します。データ・ハンドラーはこのために、xerces パーサーの http://xml.org/sax/features/validation フィーチャーを true に設定します。検証パーサーを使用するには、デフォルト値を true に変更する必要があります。 Validation が true に設定されていると、XML パーサーは XML 文書を下記と照合します。 • DTD が存在する場合は DTD • スキーマ文書が指定されている場合は、スキーマ文書。この場合、XML パーサーはスキーマの完全チェックは実行しません。 • DTD とスキーマ文書の両方が指定されている場合は、両方	false
ObjectEventId	プレースホルダー。データ・ハンドラーは使用しませんが、ビジネス・インテグレーション・システムで必要です。	なし

表 12 の「納入時のデフォルト値」列には、納入時のビジネス・オブジェクトの対応する属性の Default Value プロパティーの値がリストされています。使用する環境を調べてこれらの属性の Default Value プロパティーに適切な値を設定する必要があります。少なくとも ClassName と BOPrefix 属性にはデフォルト値が設定されている必要があります。

注: ビジネス・オブジェクト定義を変更するには、Business Object Designer Express を使用します。

さまざまな MIME タイプ/サブタイプの組み合わせでそれぞれ同じ XML データ・ハンドラー構成を使用する場合、これらの組み合わせで使用できる子メタオブジェクトは 1 つです。コネクターでさまざまな MIME タイプに関して異なる XML データ・ハンドラー構成が必要の場合、各データ・ハンドラー・インスタンスに対して別々の子メタオブジェクトを作成する必要があります。XML データ・ハンドラーの複数の構成を作成するには、次の手順で行います。

- デフォルトの XML 子メタオブジェクトをコピーして名前を変更する。新しい子メタオブジェクトの名前には、MIME タイプにサブタイプを設定することをお勧めします。例えば、デフォルトの XML データ・ハンドラーと SGML バージョンの両方をサポートするには、デフォルトの XML 子メタオブジェクトをコピーして、MO_DataHandler_DefaultSGMLConfig という名前を付けます。
- 各 XML 子メタオブジェクト内の属性のデフォルト値を設定して、データ・ハンドラー・インスタンスを構成する。

  各 MIME タイプ/サブタイプの組み合わせに関するトップレベルのデータ・ハンドラー・メタオブジェクト内の属性を作成します。例えば、XML と SGML をサポートする場合、text_xml と text_xml_sgml の MIME タイプを作成できます。これらの属性は、それぞれ関連する子メタオブジェクトを表すことになります。

  また、XML データ・ハンドラーを構成して同じデータ・ハンドラーの複数のインスタンスをサポートすることもできます。この場合は、text_xml_*subtype* という名前の別のトップレベルの属性名を作成することができます。ここで、*subtype* は、text_xml_AppA の場合のようなアプリケーション・エンティティー名、または別の適切な名前とすることができます。

データ・ハンドラーの構成方法の詳細については、 25 ページの『データ・ハンドラーの構成』を参照してください。

図 12 に、トップレベルのデータ・ハンドラー・メタオブジェクトとそれに対応する子メタオブジェクトの例を示します。トップレベルのメタオブジェクト、MO_DataHandler_XMLSample には 4 つの属性がありますが、子メタオブジェクトは 3 つしかないことに注意してください。これは、属性 Application_xml_AppC が、属性 text_xml_AppB と同じ子メタオブジェクトを使用して、該当するデータ・ハンドラーを起動するからです。

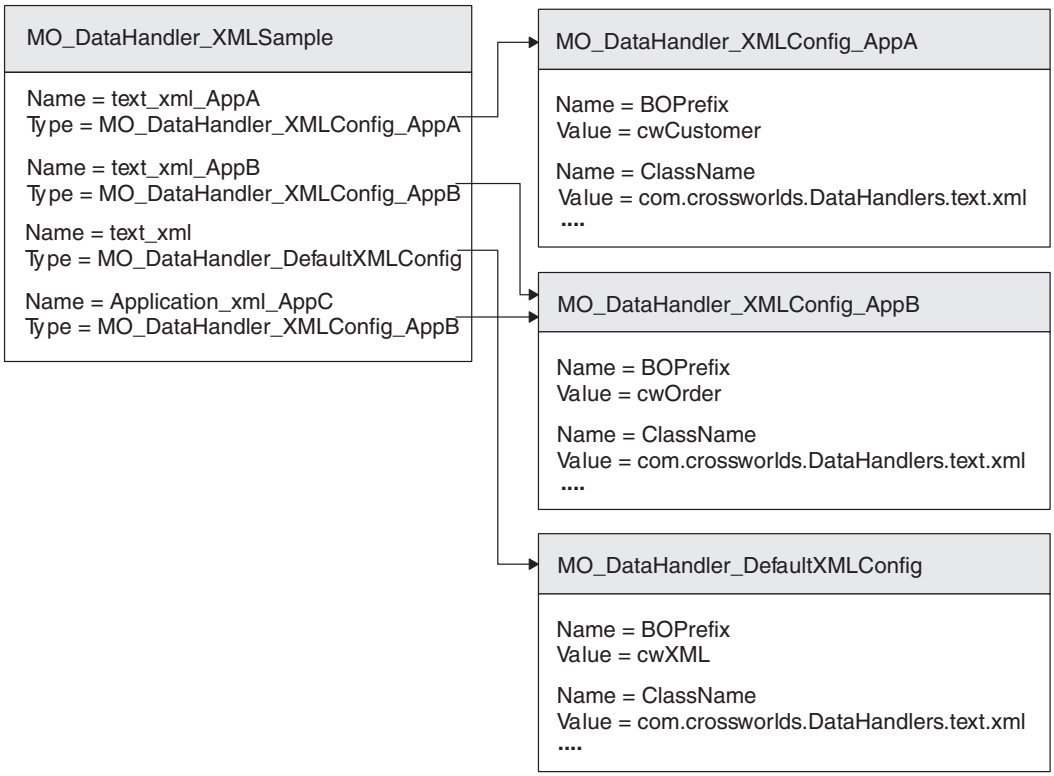

図 12. 複数の XML データ・ハンドラーのメタオブジェクトの例

DTD を使用する XML 文書

文書タイプ定義 (DTD) は、スキーマと呼ばれる XML 文書のテンプレートを規定するための特別な構文を提供する XML 文書用データ・モデルです。この DTD は .dtd 拡張子を持つファイルです。XML 文書のスキーマを表すビジネス・オブジェクト定義は DTD 内の情報を使用することにより、文書の構造を保存し、記録します。このセクションでは、ビジネス・オブジェクト定義に対応する構造情報の DTD からの引き出しに関する次の内容について説明します。

- 『DTD に基づくビジネス・オブジェクト定義の要件』
- 60 ページの『DTD からのビジネス・オブジェクト定義の作成』

DTD に基づくビジネス・オブジェクト定義の要件

DTD に対応するビジネス・オブジェクト定義が XML データ・ハンドラーの要件に適合していることを確認するためには、このセクションで示すガイドラインを使用します。ガイドラインは次のように構成されます。

- 48 ページの『DTD のビジネス・オブジェクト構造』
- 49 ページの『DTD のビジネス・オブジェクト属性プロパティー』
- 50 ページの『DTD の XML コンポーネントのアプリケーション固有情報』

ビジネス・オブジェクト定義を正しく作成することにより、データ・ハンドラーがビジネス・オブジェクトと XML 文書との相互の変換を正しく行うことができま

す。XML データ・ハンドラーのビジネス・オブジェクトの作成方法の詳細については、 61 ページの『スキーマ文書を使用する XML 文書』を参照してください。

DTD のビジネス・オブジェクト構造

DTD を表すには、少なくとも 38 ページの『ビジネス・オブジェクトの構造』で説明した 2 つのビジネス・オブジェクト定義が必要です。 DTD では、これらのビジネス・オブジェクト定義に以下の追加要件があります。

- *トップレベル・ビジネス・オブジェクト*は XML DTD を表し、以下の属性を含むことができます。
  - DOCTYPE 宣言を表す属性 DocType

    XML ODA がトップレベル・ビジネス・オブジェクト定義に DocType 属性を生成するかどうかは、DocTypeOrSchemaLocation 構成プロパティーの設定によって異なります。詳細については、 58 ページの『XML DOCTYPE 宣言の場合』および 60 ページの『サポートされる DTD 構造』を参照してください。
  - XML バージョンを表す属性 XMLDeclaration

    この属性のアプリケーション固有情報には、type=pi タグの設定が必要です。詳細については、 85 ページの『XML 処理命令の場合』を参照してください。
  - DTD 内の root エレメントを表す属性

    38 ページの『ビジネス・オブジェクトの構造』で説明したように、この属性には、タイプとして単一カーディナリティーのビジネス・オブジェクトを指定し、そのタイプが root エレメントのビジネス・オブジェクト定義である必要があります。アプリケーション固有情報に、elem_name タグを使用してこのエレメントの名前がリストされている必要があります。詳細については、 79 ページの『XML エレメントの場合』を参照してください。
- DTD の root エレメントを表す *root エレメント・ビジネス・オブジェクト定義*。

DTD に基づき、ビジネス・オブジェクト定義を使用して XML データ・ハンドラーにより処理されるビジネス・オブジェクトは、次の規則にも準拠していることが必要です。

- XML 文書内のすべてのタグは、ビジネス・オブジェクト定義内の属性と関係付けられます。この規則の例外は、FIXED 属性です。デフォルトでは、FIXED 属性には静的データが設定されているため、この属性は、ビジネス・オブジェクト定義には*含まれません*。ただし、FIXED 属性をビジネス・オブジェクト定義に含めたい場合には、そのための属性を手動でビジネス・オブジェクト定義に追加することができます。

注: ビジネス・オブジェクトの一般要件のリストについては、 37 ページの『ビジネス・オブジェクト定義の要件』を参照してください。

XML 文書の DTD の例を以下に示します。 DTD は Order という名前で、アプリケーション Order エンティティーに対応するエレメントを含みます。

```
<!--Order -->
<!-- Element Declarations -->
<!ELEMENT Order (Unit+)>
<!ELEMENT Unit (PartNumber?, Quantity, Price, Accessory*)>
```

```
<!ELEMENT PartNumber (#PCDATA)>
<!ELEMENT Quantity (#PCDATA)>
<!ELEMENT Price (#PCDATA)>
<!ELEMENT Accessory (Quantity, Type)>
<!ATTLIST Accessory
     Name CDATA >
<!ELEMENT Type (#PCDATA)>
```

図 13 に、Order DTD に関連する XML 文書に対応して作成されることがあるビジネス・オブジェクトの構造を示します。 Order DTD 内の各 XML エレメントとエレメント属性には、対応するビジネス・オブジェクト属性があります。トップレベル・ビジネス・オブジェクトには、XML 宣言、DOCTYPE、およびトップレベルの Order エレメントの各属性が格納されます。また、Name エレメント属性は、Accessory ビジネス・オブジェクト内の最初の属性であることにも注意してください。

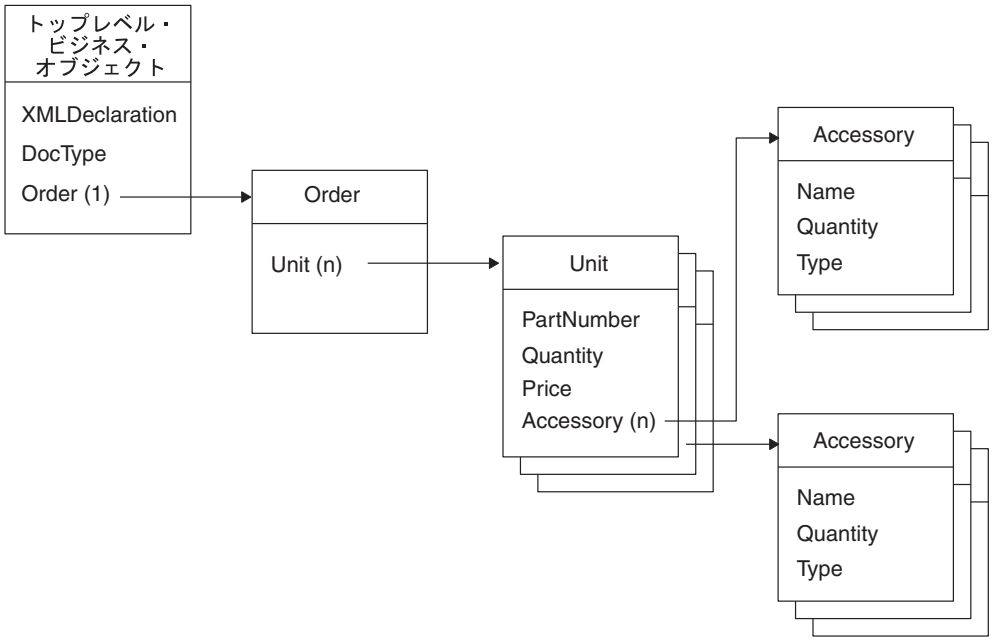

図 13. Order DTD を使用する XML 文書のビジネス・オブジェクトの例

DTD のビジネス・オブジェクト属性プロパティー

XML 文書用のビジネス・オブジェクト定義が DTD に基づいている場合、ビジネス・オブジェクト属性プロパティーには、39 ページの『ビジネス・オブジェクトの属性プロパティー』で説明した制約があります。さらに、DTD 構文により、ビジネス・オブジェクト属性の「必須性」が決まる場合があります。「必須性」とは、カーディナリティーや属性がキーであるかどうかなどの要因の組み合わせであり、XMLデータ・ハンドラーがその属性を必要とするかどうかを指定します。属性が必須な場合には、Required 属性プロパティーを true に設定することが必要です。

Required 属性プロパティーの設定は、以下に示すように、XML エレメントおよび属性の指定のほか、Cardinality、Key、および Foreign Key 属性プロパティーの設定に依存します。

- ビジネス・オブジェクト属性のカーディナリティーは、DTD 内の ELEMENT フラグメントにより決定されます。このカーディナリティーは、属性が必須であるかどうかに影響します。表 13 に、DTD で宣言されたエレメントの各組み合わせに対応するカーディナリティーと「必須性」の概要を示します。

表 13. DTD に対応するカーディナリティーと「必須性」

DTD ELEMENT フラグメント	カーディナリティー	必須
指定なし	1	はい
?	1	いいえ
+	N	はい
*	N	いいえ

- ビジネス・オブジェクト属性が基本キーか外部キーかを決定するのは、DTD 内の ATTLIST フラグメントです。キーの存在は、属性が必須であるかどうかに影響します。表 14 に、ATTLIST フラグメント内の構文が、DTD で宣言された属性の各組み合わせに対して、ビジネス・オブジェクト属性の「必須性」にどのように影響するかを示します。

表 14. DTD に対応するキーと「必須性」

DTD ATTLIST フラグメント	キー	必須	コメント
#IMPLIED	なし	いいえ	
#REQUIRED	なし	はい	
ID {#IMPLIED | #REQUIRED}	あり	いいえ	#IMPLIED、#REQUIRED は無視
IDREF {#IMPLIED, #REQUIRED}	Foreign Key	#IMPLIED か #REQUIRED が指定されているかどうかで異なる	
NMTOKEN {#IMPLIED | #REQUIRED}	あり	いいえ	#IMPLIED、#REQUIRED は無視

DTD の XML コンポーネントのアプリケーション固有情報

このセクションでは、DTD に基づくビジネス・オブジェクト定義のためのアプリケーション固有情報フォーマットの次の内容について説明します。

- 『ビジネス・オブジェクト・レベルのアプリケーション固有情報』
- 54 ページの『配列属性のアプリケーション固有情報』
- 54 ページの『属性のアプリケーション固有情報』

ビジネス・オブジェクト・レベルのアプリケーション固有情報: XML データ・ハンドラーは次のタイプのビジネス・オブジェクトを使用して、DTD から生成された異なる種類の XML エレメントを表します。

- 51 ページの『DTD に基づく通常のビジネス・オブジェクト定義』
- 51 ページの『DTD に基づく混合ビジネス・オブジェクト定義』
- 51 ページの『DTD に基づくラッパー・ビジネス・オブジェクト定義』

これらのタイプのビジネス・オブジェクトは、ビジネス・オブジェクト・レベルでアプリケーション固有情報により識別されます。

DTD に基づく通常のビジネス・オブジェクト定義: *通常*のビジネス・オブジェクトは XML エレメントを表します。このタイプのビジネス・オブジェクトでは、ビジネス・オブジェクト・レベルでのアプリケーション固有情報により、ビジネス・オブジェクトが表す XML エレメントの名前を識別します。例えば、XML エレメントが次のように定義されていると仮定します。

```
<!ELEMENT Unit(...)>
```

関連するビジネス・オブジェクト定義に対応するビジネス・オブジェクト・レベルでのアプリケーション固有情報は次のとおりです。

```
[BusinessObjectDefinition]
Name = MyApp_Unit
AppSpecificInfo = elem_name=Unit
[Attribute]
...
```

DTD に基づく混合ビジネス・オブジェクト定義: *混合* ビジネス・オブジェクトは、文字データ (#PCDATA) およびその他のサブエレメントが混合した内容を格納している混合 XML エレメントを表します。混合タイプの XML エレメントの DTD での表現は、次のようになります。

```
<!ELEMENT (#PCDATA | CONTAINED_ELEMENT1 | CONTAINED_ELEMENTN)*>
```

例えば、Cust XML エレメントが DTD で次のように定義されていると仮定します。

```
<!ELEMENT Cust(#PCDATA | Address | Phone)*>
```

混合タイプの XML エレメントを表すため、混合タイプのビジネス・オブジェクト定義を使用します。混合ビジネス・オブジェクト定義の場合、そのビジネス・オブジェクト・レベルのアプリケーション固有情報は、次のコンポーネントから構成されます。

- 混合 XML エレメントの名前
- タグ type=MIXED

Cust エレメントを表すビジネス・オブジェクト定義 MyApp_Cust の場合、ビジネス・オブジェクト・レベルでのアプリケーション固有情報は次のようになります。

```
[BusinessObjectDefinition]
Name = MyApp_Cust
AppSpecificInfo = Cust;type=MIXED;
```

DTD に基づくラッパー・ビジネス・オブジェクト定義: ラッパー・ビジネス・オブジェクトは、反復選択項目リストを表します。このタイプのビジネス・オブジェクト定義は、XML エレメントに任意の順序と任意のカーディナリティーになり得る子がある場合に必要です。ラッパー・ビジネス・オブジェクトは、子エレメントの順序とカーディナリティーを XML 文書に保存します。

選択リストの XML エレメントの場合、DTD 定義は次のようになります。

```
(CONTAINEDELEMENT1 | ... | CONTAINEDELEMENTN)*
```

DTD 内の選択リスト XML エレメント定義の例を次に示します。

```
<!ELEMENT CUST( U | I | B )* )>
```

このエレメントは、任意の順序となりえるオプションの 3 つのサブエレメントを含んでいます。各サブエレメントは単純エレメントです。図 14 に、このタイプの XML 文書を示します。

```
<CUST>
   <U>.....
   </U>
   <B>.....
   </B>
   <I>.....
   </I>
   <B>.....
   </B>
   <U>.....
   </U>
...
```

図 14. 反復選択リストの XML 文書内容

DTD 内に定義された選択リスト XML エレメントを表すため、ビジネス・オブジェクト定義は階層型です。次のビジネス・オブジェクト定義から構成されています。

- 親ビジネス・オブジェクト定義

  この親ビジネス・オブジェクト定義には、複数カーディナリティーのビジネス・オブジェクト配列を表す 1 つの属性があります。この属性は、そのタイプとして、関連付けられたラッパー・ビジネス・オブジェクトのビジネス・オブジェクト定義を保持しています。親ビジネス・オブジェクト定義には、次のアプリケーション固有情報が設定されています。

  – ビジネス・オブジェクト・レベルでは、親ビジネス・オブジェクト定義は、アプリケーション固有情報の中に、選択リストの名前を保持しています。
  – 属性レベルでは、複数カーディナリティーの属性 (親ビジネス・オブジェクト定義内) により、オプショナル選択エレメントが次のフォーマットで指定されます。

    (*choiceElement1*|...|*choiceElementN*)

    ここで、*choiceElement1*...*choiceElementN* は、定義された選択エレメントそれぞれに対応します。各選択エレメントはパイプ文字 (|) で区切り、タグ全体は括弧で囲むことが必要です。

- ラッパー・ビジネス・オブジェクト定義

  ラッパー・ビジネス・オブジェクト定義には、選択リスト・エレメントに定義された各選択エレメントに対応した属性がそれぞれ設定されています。ビジネス・オブジェクト・レベルでのアプリケーション固有情報は必要ありません。

53 ページの図 15 に、選択リスト XML エレメントに対応するビジネス・オブジェクト定義の階層構造を示します。実行時には、各子ビジネス・オブジェクトがラッパー・ビジネス・オブジェクトのインスタンスとなり、1 つの属性にのみデータが設定されます。例として、図 14 の XML 文書のビジネス・オブジェクトには 5 つの子があり、それぞれに該当する属性が設定されています。

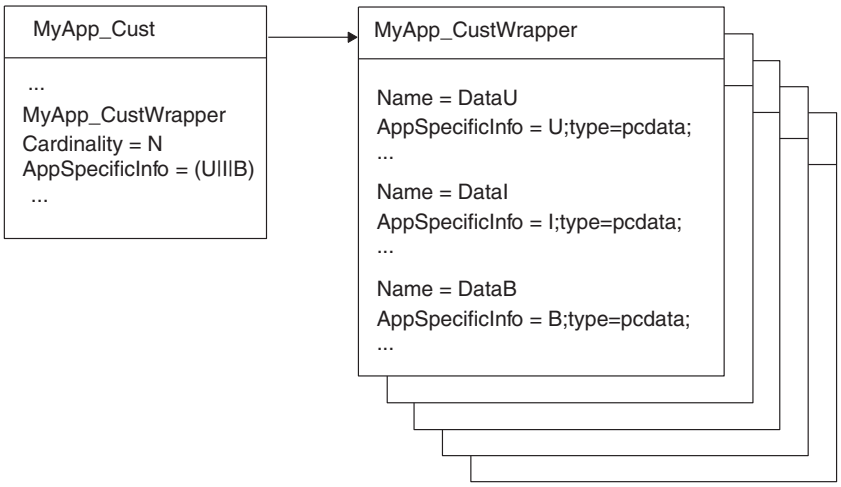

図 15. 選択リスト XML エレメントに対応するビジネス・オブジェクト定義の階層構造

図 16 に、親ビジネス・オブジェクト定義である MyApp_Cust とそのアプリケーション固有情報を示します。

```
[BusinessObjectDefinition]
Name = MyApp_Cust
AppSpecificInfo =

   [Attribute]
   Name = CustWrapper
   Type = MyApp_CustWrapper
   Cardinality = N
   AppSpecificInfo = attr_name=CustWrapper;(U|I|B)
[End]
```

図 16. 選択リスト・エレメントの親ビジネス・オブジェクト定義

ラッパー・ビジネス・オブジェクト定義 MyApp_CustWrapper には 3 つの属性があり、それぞれ 1 つの選択エレメントに対応しています。各選択エレメントには文字データが格納されているため、各属性に対応するアプリケーション固有情報により、以下の内容が指定されます。

- エレメントの名前
- タグ type=pcdata

注: 文字データに対応する属性について詳しくは、 56 ページの『PCDATA のみを持つ XML エレメントの場合』を参照してください。

図 17 に、この XML 文書に対応するラッパー・ビジネス・オブジェクト定義を示します。

```
[BusinessObjectDefinition]
Name = MyApp_CustWrapper
AppSpecificInfo =

   [Attribute]
   Name = DataU
   Type = String
   AppSpecificInfo = attr_name=U;type=pcdata;
   [End]

   [Attribute]
   Name = DataI
   Type = String
   AppSpecificInfo = attr_name=I;type=pcdata;
   [End]

   [Attribute]
   Name = DataB
   Type = String
   AppSpecificInfo = attr_name=B;type=pcdata;
   [End]
```

図 17. 選択リスト・エレメントのラッパー・ビジネス・オブジェクト定義

配列属性のアプリケーション固有情報: ビジネス・オブジェクト属性が他のエレメントを含む XML エレメントを表す場合、アプリケーション固有情報にはエレメントの名前が設定されている必要があります。例えば、DeliveryDate という名前の属性にビジネス・オブジェクト・タイプがあり、DATETIME という名前のエレメントを表す場合、アプリケーション固有情報には、次のようにエレメントの名前が設定されています。

```
Name = DeliveryDate
Relationship = Containment
Cardinality = n
AppSpecificInfo = DATETIME
```

属性のアプリケーション固有情報: ビジネス・オブジェクト定義の属性は、以下の XML コンポーネントを表すことができます。

- 55 ページの『XML エレメントの場合』
- 56 ページの『PCDATA のみを持つ XML エレメントの場合』
- 56 ページの『XML 属性の場合』
- 57 ページの『文字データおよび属性を持つ XML エレメントの場合』
- 58 ページの『特殊文字が含まれる XML エレメントまたは属性の場合』
- 58 ページの『XML DOCTYPE 宣言の場合』
- 59 ページの『CDATA セクションの場合』
- 59 ページの『XML コメントの場合』
- 59 ページの『XML 処理命令の場合』

表 25 に、これらのさまざまな XML コンポーネントの属性レベルのアプリケーション固有情報のタグと、タグの詳細が記載されている本書のセクションを示します。

表 15. 属性のアプリケーション固有情報のタグ

ビジネス・オブジェクト属性の表現	アプリケーション固有情報	詳細
XML エレメント	elem_name=*name of XML element*	『XML エレメントの場合』
PCDATA のみを持つ XML エレメント	elem_name=*name of XML element*;type=pcdata	56 ページの『PCDATA のみを持つ XML エレメントの場合』
XML エレメントの属性	attr_name=*name of XML attribute* type=attribute	56 ページの『XML 属性の場合』
文字データと属性を持つ XML エレメント	type=pcdata;notag	57 ページの『文字データおよび属性を持つ XML エレメントの場合』
特殊文字を持つ XML エレメントまたは属性	escape=true	84 ページの『特殊文字が含まれる XML エレメントまたは属性の場合』
DOCTYPE 宣言	type=doctype	58 ページの『XML DOCTYPE 宣言の場合』
CDATA セクション	type=cdata	59 ページの『CDATA セクションの場合』
XML 文書に追加されるコメント	type=comment	84 ページの『XML コメントの場合』
処理命令	type=pi	85 ページの『XML 処理命令の場合』

注: 属性のアプリケーション固有情報には、(a | b | c) という形式のタグを使用して、反復選択を表す複数カーディナリティーの属性を指定することもできます。詳細については、 51 ページの『DTD に基づくラッパー・ビジネス・オブジェクト定義』を参照してください。

XML エレメントの場合: XML エレメントを表すすべての単純 (String) ビジネス・オブジェクト属性には、アプリケーション固有情報に elem_name タグを使用して、関連するエレメントを識別する必要があります。

```
elem_name=name of XML element
```

例えば、ビジネス・オブジェクト属性 CustLName が単純 XML 属性を表す場合には、そのアプリケーション固有情報は次のようになります。

```
Name = CustLName
AppSpecificInfo = elem_name=CustLName;
```

XML エレメント名には特殊文字 (ピリオドやハイフンなど) を使用できます。ただし、ビジネス・オブジェクト属性名にこれらの特殊文字を使用することはできません。したがって、XML エレメント名を elem_name タグに指定する必要があります。ビジネス・オブジェクト属性に名前を付けるとき、XML ODA は XML エレメントの名前から特殊文字を除去し、それらを下線 (_) 文字に置換します。

以下の例では、XML エレメントのアプリケーション固有情報で指定された名前は実際の XML エレメントの名前とは異なっています。これは、属性に特殊文字が含まれているためです。

```
Name = Phone_Tag
AppSpecificInfo = elem_name=Phone#Tag;
```

XML エレメントの実際の名前にはポンド記号 (#) が含まれています。ポンド記号をビジネス・オブジェクト属性の名前に使用することはできません。このため、アプリケーション固有情報の elem_name タグで実際の XML エレメント名を指定します。関連したビジネス・オブジェクト属性の名前では、ポンド記号は下線に置換されます。

PCDATA のみを持つ XML エレメントの場合: 文字データのみを持つ XML エレメントが、PCDATA エレメント内容指定子のみを持つエレメントと混合されています。PCDATA のみが格納されている XML エレメントを表すビジネス・オブジェクト属性は、アプリケーション固有情報に次の type タグを使用する必要があります。

```
type=pcdata
```

この場合、エレメント名はアプリケーション固有情報の最初のフィールドで、type パラメーターが 2 番目のフィールドです。

例えば、PCDATA のみを格納している PartNumber という名前のエレメントは、DTD で次のように定義されています。

```
<!ELEMENT PartNumber (#PCDATA)>
```

ビジネス・オブジェクト定義内の対応する属性には、次のアプリケーション固有情報が設定されています。

```
Name = PartNumber
AppSpecificInfo = elem_name=PartNumber;type=pcdata;
```

アプリケーション固有情報にテキスト notag も設定されている場合、XML データ・ハンドラーは XML マークアップを生成しません。データ・ハンドラー自体の属性値のみを XML 文書に追加します。詳細については、57 ページの『文字データおよび属性を持つ XML エレメントの場合』を参照してください。

XML 属性の場合: ビジネス・オブジェクト属性が XML エレメントの属性を表す場合、アプリケーション固有情報に次のタグを設定する必要があります。

- attr_name タグ

  attr_name=*attrName*

  XML 属性名には特殊文字 (ピリオドやハイフンなど) を使用できます。ただし、ビジネス・オブジェクト属性名にこれらの特殊文字を使用することはできません。したがって、XML 属性名を attr_name タグに指定する必要があります。ビジネス・オブジェクト属性に名前を付けるとき、XML ODA は XML 属性の名前から特殊文字を除去します。

- type タグ

  type=attribute

  この type タグは、関連するビジネス・オブジェクト属性の目的を XML 属性として識別します。

注: 38 ページの『ビジネス・オブジェクトの構造』で説明したように、XML 属性を表すすべてのビジネス・オブジェクト属性は、ビジネス・オブジェクト定義内で XML エレメントを表すビジネス・オブジェクト属性の*前に* 置かれる必要があります。

例えば、ID という名前のビジネス・オブジェクト属性が ID という名前の XML 属性を表す場合、アプリケーション固有情報は次のとおりです。

```
Name = ID
AppSpecificInfo = attr_name=ID;type=attribute;
```

type=attribute タグを使用している別の例については、『文字データおよび属性を持つ XML エレメントの場合』を参照してください。

文字データおよび属性を持つ XML エレメントの場合: XML エレメントが PCDATA または CDATA のみ を含み、1 つ以上の XML 属性を持っている場合、そのビジネス・オブジェクト定義には次の属性が設定されていることが必要です。

- 各 XML 属性に対応するビジネス・オブジェクト属性

  属性名は、XML 属性の名前と一致していることが必要です。属性レベルのアプリケーション固有情報には、attr_name タグと type=attribute タグが設定されていることが必要です。詳細については、56 ページの『XML 属性の場合』を参照してください。

- PCDATA または CDATA エレメント内容指定子に関連付けられた文字データ用のビジネス・オブジェクト属性

  この属性には、親 XML エレメントに関係付けられたデータが設定されています。アプリケーション固有情報は以下を含んでいることが必要です。

  – 適切な type=*typename* タグ (ただし *typename* は pcdata または cdata です) とそれに続くセミコロン (;)
  – notag キーワード。これにより、XML データ・ハンドラーによる開始タグの重複生成 (ビジネス・オブジェクト用と属性用) が防止できます。 XML データ・ハンドラーは、ビジネス・オブジェクト属性ごとに XML 開始タグと終了タグを作成します。ただし、この属性のアプリケーション固有情報に notag が指定されている場合にはタグの作成はありません。

例えば、Price という名前の XML エレメントに、Currency という名前の属性があり、Price 用のデータを必要としていると仮定します。

```
<!ELEMENT Price (#PCDATA)>
<!ATTLIST Price Currency NMTOKEN #IMPLIED>
```

Price エレメントには XML 属性があるため、そのビジネス・オブジェクト定義に、Currency 用のビジネス・オブジェクト属性を作成することが必要です。さらに、Price データを保持するために別の属性も必要です。Price データの属性には、そのアプリケーション固有情報に notag を指定して、データ・ハンドラーがこの属性の開始および終了タグを作成しないようにする必要があります。

Price 子ビジネス・オブジェクトは、次のようになります。

```
[BusinessObjectDefinition]
Name = Price
AppSpecificInfo = Price

   [Attribute]
   Name = Currency
   Type = String
   AppSpecificInfo = attr_name=Currency;type=attribute;
   ...
   [End]
```

```
[Attribute]
Name = Price
Type = String
AppSpecificInfo = Price;type=pcdata;notag
...
[End]
```

この場合、データ・ハンドラーは、Price データの新しい XML エレメントは生成せず、データを親エレメントに追加するだけです。

特殊文字が含まれる XML エレメントまたは属性の場合: エスケープ処理を必要とする内容の XML エレメントまたは XML 属性を表すビジネス・オブジェクト属性の場合、そのアプリケーション固有情報には次のタグが設定されていることがあります。

```
escape=true
```

属性が表す XML エレメントが、次の特殊文字のいずれかを含む値を持つ場合、その属性にはエスケープ処理が必要です。

- 単一引用符 (')
- 二重引用符 (")
- アンパーサンド (&)
- より小符号 (<)
- より大符号 (>)

アプリケーション固有情報に escape=true タグが設定されている場合を除き、属性はエスケープ処理の*対象外* です。このタグは、既存のアプリケーション固有情報の最後に入れることが必要です。例えば次のようになります。

```
[Attribute]
Name=Data
Type=String
AppSpecificInfo=Price;type=pcdata;escape=true
[End]
```

属性のアプリケーション固有情報に escape タグが*設定されていない* 場合、XML データ・ハンドラーは、DefaultEscapeBehavior プロパティーを調べることにより、エスケープ処理を実行するかどうか決定します。

- DefaultEscapeBehavior が true であれば、XML データ・ハンドラーはすべての 属性値に対してエスケープ処理を実行します。
- DefaultEscapeBehavior が false であれば、XML データ・ハンドラーは、アプリケーション固有情報に escape タグが設定されている属性に対してのみ エスケープ処理を実行します。

XML DOCTYPE 宣言の場合: ビジネス・オブジェクト属性が prolog 内の文書タイプ宣言を表す場合、アプリケーション固有情報には次の type タグが設定されていることが必要です。

```
type=doctype
```

例えば、ビジネス・オブジェクト属性 DocType が DOCTYPE エレメントを表す場合には、そのアプリケーション固有情報は次のようになります。

```
Name = DocType
AppSpecificInfo = type=doctype;
```

DocType 属性が、次の値の場合、

```
DOCTYPE CUSTOMER "customer.dtd"
```

データ・ハンドラーは次の XML を生成します。

```
<!DOCTYPE CUSTOMER "customer.dtd">
```

このアプリケーション固有情報は、XML 文書の汎用エンティティー宣言を組み込むためにも使用されます。しかし、内部 DTD またはパラメーター・エンティティー宣言の組み込みに関する明示的なサポートはありません。これらを文書に組み込むには、アプリケーション固有情報に type=doctype が指定されている属性の値にテキスト全体を設定します。

CDATA セクションの場合: ビジネス・オブジェクト属性が XML 文書内の CDATA セクションを表す場合、アプリケーション固有情報には次の type タグが設定されていることが必要です。

```
type=cdata
```

例えば、ビジネス・オブジェクト属性 UserArea が CDATA セクションを表す場合、そのアプリケーション固有情報は次のようになります。

```
Name = UserArea
AppSpecificInfo = type=cdata;
```

XML コメントの場合: XML データ・ハンドラーがビジネス・オブジェクトを XML 文書に変換するとき、XML 文書へのコメントの追加を XML データ・ハンドラーに指示することができます。データ・ハンドラーによるコメントの追加を可能にするには、次のステップを実行します。

- ビジネス・オブジェクト定義内に、コメントを表す単一または複数のビジネス・オブジェクト属性を作成します。

  注: XML ODA は、XML コメント用ビジネス・オブジェクト属性を自動的には生成しません。このような属性は、90 ページの『ビジネス・オブジェクト定義の手動での作成』で説明されている手順により、手動で追加する必要があります。

- 属性レベルのアプリケーション固有情報に次の type タグを設定します。

  ```
  type=comment
  ```

- 実際のビジネス・オブジェクトに、この属性の値として、コメント・テキストを指定します。

例えば、Comment という名前のビジネス・オブジェクト属性が、データ・ハンドラーが XML 文書に追加する XML コメントを表す場合、Comment 属性は次のようになります。

```
Name = Comment
AppSpecificInfo = type=comment;
```

属性の値が「Customer information update from application A」であれば、次の XML が生成されます。

```
<!--Customer information update from application A-->
```

XML 処理命令の場合: ビジネス・オブジェクト属性が処理命令を表す場合、アプリケーション固有情報には次の type タグが設定されていることが必要です。

```
type=pi
```

例えば、XMLDeclaration という名前のビジネス・オブジェクト属性が prolog 内の XML 宣言を表す場合、アプリケーション固有情報は次のとおりです。

```
Name = XMLDeclaration
AppSpecificInfo = type=pi;
```

属性が、次の値の場合、

```
xml version = "1.0"
```

XML データ・ハンドラーは次の XML を生成します。

```
<?xml version="1.0"?>
```

DTD からのビジネス・オブジェクト定義の作成

DTD は XML 文書のフォーマットを記述します。したがって、ビジネス・オブジェクト定義のために必要な情報を取得する上で、DTD は非常に有用です。DTD 内の構造情報をビジネス・オブジェクト定義に変換するため、XML Object Discovery Agent (ODA) を使用することができます。XML ODA については、89 ページの『ビジネス・オブジェクト定義を作成するための XML ODA の使用』を参照してください。

注: XML データ・ハンドラーの旧バージョンには、DTD からビジネス・オブジェクト定義を作成するための 2 つのツール、Edifecs SpecBuilder および XMLBorgen ユーティリティー (使用しないことが推奨されています) が含まれていました。XML ODA によりこれら 2 つのツールの機能が実現できます。

サポートされる DTD 構造

XML ODA は次の DTD 構造をサポートします。

- エンティティー: 参照されると常に、次の形式のユーザー定義エンティティーが認識されて置換されます。

  ```
  <!ENTITY % name value>
  ```

  注: XML ODA は、改行文字 (¥n)、復帰文字 (¥r) またはタブ文字 (¥t) が、ENTITY タグとその内容の間またはエレメント・タグもしくは属性タグとその内容の間に存在する場合にのみ、その文字を削除します。

- 外部 DTD: XML ODA は外部 DTD への参照 (1 つの DTD が 1 つ以上の別の DTD を参照) をサポートしています。外部 DTD はローカル・ファイル・システム上にある場合にのみ 解決できます。検索のため、URL にアクセスすることはできません。どちらのツールも必ずローカル・ファイル・システム上で外部 DTD への参照を見つけることを試みます。これらの参照を無視することはありません。
- ANY ディレクティブ: XML ODA は、ANY を内容として持つエレメントに対して、String 属性を作成します。例えば、DTD が次の構成である場合、

  ```
  <!ELEMENT SCENE (ANY) >
  ```

  対応するビジネス・オブジェクト定義は次のとおりです。

  ```
  [Attribute]
  Name = SCENE
  Type = String
  ```

```
Cardinality = 1
MaxLength = 255
IsKey = false
IsForeignKey = false
IsRequired = true
AppSpecificInfo = SCENE;type=pcdata;
[End]
```

- Prolog: XML データ・ハンドラーは、DOCTYPE および XML 宣言などの prolog 情報の挿入を、これらのエレメントに対応する属性がビジネス・オブジェクト定義に含まれている場合に限って実行します。ただし、データ・ハンドラーは、DOCTYPE の名前のみ 挿入し、他のメタ情報があっても挿入はしません。

サポートされない DTD 構造

XML ODA はほとんどの DTD を処理できます。ただし、次の DTD 構造はサポートされていません。

- 条件セクション: 次のような構造のセクションです。

```
<![INCLUDE[ information to be included ]]>
<![IGNORE[ information to be ignored ]]>
```

- ネーム・スペース: xxx:yyy という形式のタグは、単にタグ xxx:yyy として扱われ、ネーム・スペース xxx に属するタグ yyy としては扱われません。

スキーマ文書を使用する XML 文書

XML スキーマ は、XML 文書のテンプレート (スキーマ) を定義するためにスキーマ文書 (.xsd 拡張子) を使用する XML 文書用データ・モデルです。DTD とは異なり、スキーマ文書は、スキーマを記述するために、XML 文書と同じ構文を使用します。XML 文書のスキーマを表すビジネス・オブジェクト定義は、文書の構造を保存および記録するために、スキーマ文書内の情報を使用します。このセクションでは、ビジネス・オブジェクト定義に対応する構造情報のスキーマ文書からの引き出しに関する次の内容について説明します。

- 『スキーマ文書に基づくビジネス・オブジェクト定義の要件』
- 86 ページの『スキーマ文書からのビジネス・オブジェクト定義の作成』

スキーマ文書に基づくビジネス・オブジェクト定義の要件

スキーマ文書を表すビジネス・オブジェクト定義が XML データ・ハンドラーの要件に適合していることを確認するには、このセクションで示すガイドラインを使用します。ガイドラインは次のように構成されています。

- 『スキーマ文書のビジネス・オブジェクト構造』
- 70 ページの『スキーマ文書のビジネス・オブジェクト属性プロパティー』
- 71 ページの『スキーマ文書内の XML コンポーネントのアプリケーション固有情報』

スキーマ文書のビジネス・オブジェクト構造

スキーマ文書に基づき、ビジネス・オブジェクト定義を使用して XML データ・ハンドラーにより処理されるビジネス・オブジェクトは、次の規則に準拠していることが必要です。

- スキーマ文書には次のビジネス・オブジェクト定義が含まれる必要があります。
  - schema エレメントを表すトップレベル・ビジネス・オブジェクト定義。

– スキーマ文書の root エレメントを表す root エレメント・ビジネス・オブジェクト定義。スキーマ文書内の root エレメントを表す、通常、混合、またはラッパー・ビジネス・オブジェクト定義。

詳細については、『スキーマ文書に必要なビジネス・オブジェクト定義』を参照してください。

- 通常、混合、またはラッパー・ビジネス・オブジェクト定義は、含まれている XML コンポーネントを表すこともできます。

XML 文書用のスキーマ文書の例を図 18 に示します。このスキーマ・文書の名前は Order で、アプリケーション Order エンティティーに対応するエレメントを格納しています。このスキーマ文書のサンプルは、前述の DTD 文書のサンプルと同じビジネス・オブジェクト構造を記述しています。 49 ページの図 13 に、Order スキーマ文書に関連する XML 文書に対応して作成されることがあるビジネス・オブジェクト定義の構造を示します。

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
        elementFormDefault="qualified">

        <xs:complexType name="AccessoryType">
                <xs:sequence>
                        <xs:element ref="Quantity"/>
                        <xs:element ref="Type"/>
                </xs:sequence>
                <xs:attribute name="Name" type="xs:string" use="required"/>
        </xs:complexType>
        <xs:element name="Order">
                <xs:complexType>
                        <xs:sequence>
                                <xs:element name="Unit" type="UnitType"
                    maxOccurs="unbounded"/>
                        </xs:sequence>
                </xs:complexType>
        </xs:element>
        <xs:element name="PartNumber" type="xs:string"/>
        <xs:element name="Price" type="xs:string"/>
        <xs:element name="Quantity" type="xs:string"/>
        <xs:element name="Type" type="xs:string"/>
        <xs:complexType name="UnitType">
                <xs:sequence>
                        <xs:element ref="PartNumber" minOccurs="0"/>
            <xs:element ref="Quantity"/>
            <xs:element ref="Price"/>
            <xs:element name="Accessory" type="AccessoryType" minOccurs="0"
                   maxOccurs="unbounded"/>
                </xs:sequence>
    </xs:complexType>
</xs:schema>
```

図 18. XML スキーマ文書のサンプル

スキーマ文書に必要なビジネス・オブジェクト定義: スキーマ文書を表すには、少なくとも 38 ページの『ビジネス・オブジェクトの構造』で説明した 2 つのビジネス・オブジェクト定義が必要です。スキーマ文書では、これらのビジネス・オブジェクト定義に以下の追加要件があります。

- トップレベル・ビジネス・オブジェクトは schema エレメントを表し、以下の属性が含まれている必要があります。

– XML バージョンを表す属性 XMLDeclaration

この属性のアプリケーション固有情報には、type=pi タグの設定が必要です。詳細については、 85 ページの『XML 処理命令の場合』を参照してください。

– スキーマ文書内の root エレメントを表す属性

38 ページの『ビジネス・オブジェクトの構造』で説明したように、この属性には、タイプとして単一カーディナリティーのビジネス・オブジェクトを指定し、そのタイプが root エレメントのビジネス・オブジェクト定義である必要があります。アプリケーション固有情報に、elem_name タグを使用してこのエレメントの名前がリストされている必要があります。詳細については、 79 ページの『XML エレメントの場合』を参照してください。

- スキーマ文書の root エレメントを表す *root* エレメント・ビジネス・オブジェクト定義。スキーマ文書にはグローバル・レベルで定義されたいくつかの XML コンポーネントを組み込めるので、XML ODA の Root 構成プロパティーを使用して、root エレメントとするエレメントを指定できます。root エレメント・ビジネス・オブジェクト定義では、次の属性を設定することができます。
  – 単一または複数のスキーマ文書の場所を表す schemaLocation 属性 (オプション)

    XML ODA がトップレベル・ビジネス・オブジェクト定義に schemaLocation 属性を生成するかどうかは、DocTypeOrSchemaLocation 構成プロパティーの設定によって異なります。詳細については、 85 ページの『XML スキーマ・ロケーションの場合』を参照してください。

  – root エレメント内の各 XML コンポーネントの属性

注: ビジネス・オブジェクトの一般要件のリストについては、 37 ページの『ビジネス・オブジェクト定義の要件』を参照してください。

これらの必須のビジネス・オブジェクト定義には、ターゲット・ネーム・スペースおよびコンポーネント名の制限を定義する、ビジネス・オブジェクト・レベルのアプリケーション固有情報が必要です。詳細については、 71 ページの『ビジネス・オブジェクト・レベルのアプリケーション固有情報』を参照してください。

62 ページの図 18 の XML schema エレメントは次のとおりです。

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
        elementFormDefault="qualified">
```

図 19 に、このスキーマ・エレメントを表すトップレベル・ビジネス・オブジェクト定義 TopLevel を示します。

```
[BusinessObjectDefinition]
Name=TopLevel
AppSpecificInfo=elem_fd=qualified;attr_fd=unqualified
...
   [Attribute]
   Name=XMLDeclaration
   Type=String
   AppSpecificInfo=type=pi;
   ...
   [End]
   [Attribute]
   Name=Order
   Type=TopLevel_Order
   AppSpecificInfo=elem_name=Order;
   ...
   [End]
...
[End]
```

図 19. トップレベル・ビジネス・オブジェクト定義のサンプル

XML ODA が図 19 のトップレベル・ビジネス・オブジェクト定義を生成した場合、次の ODA 構成プロパティーが設定されています。

ODA 構成プロパティー	プロパティーの値
BOPrefix	設定なし
TopLevel	"TopLevel"
Root	"Order"
DoctypeorSchemaLocation	true

root エレメント・ビジネス・オブジェクト定義の例は、76 ページの図 26 を参照してください。

スキーマ文書に基づく通常のビジネス・オブジェクト定義: *通常の* ビジネス・オブジェクト定義は、次の XML 構造のいずれかを表します。

- 子エレメントの sequence グループを持つ複合タイプ (名前付きまたは無名) が設定された XML エレメント。

  sequence グループの各子エレメントは、ビジネス・オブジェクト定義の中では属性として表されます。詳細については、81 ページの『複合タイプの XML エレメントの場合』を参照してください。

  注: 複合タイプに choice または all グループが含まれる場合は、ラッパー・ビジネス・オブジェクト定義で表す必要があります。詳細については、67 ページの『スキーマ文書に基づくラッパー・ビジネス・オブジェクト定義』を参照してください。
- 属性を持つ XML エレメント。

このタイプのビジネス・オブジェクト定義では、ビジネス・オブジェクト・レベルのアプリケーション固有情報に特別な情報は必要ありません。通常のビジネス・オブジェクト定義の属性は、XML 複合タイプ内に定義されているエレメントを表します。

注: 子エレメントの sequence グループを持つ複合タイプの場合、sequence グループ内の各子エレメントは、ビジネス・オブジェクト定義の中では属性として表現されます。

例えば、Unit という XML エレメントが次のように定義されていると仮定します。

```
<xsd:element name="Unit">
    <xsd:complexType>
    ...
    </xsd:complexType>
</element>
```

次のビジネス・オブジェクト定義は、Unit エレメントを表します。

```
[BusinessObjectDefinition]
Name = MyApp_Unit
AppSpecificInfo =
[Attribute]
   ...
```

スキーマ文書に基づく混合ビジネス・オブジェクト定義: *混合* ビジネス・オブジェクト定義は、文字データおよびその他のサブエレメントが混合した内容を格納している混合 XML エレメントを表します。スキーマ文書は、混合タイプの XML エレメントを、次のように、mixed 属性が true に設定された複合タイプとして記述します。

```
<xsd:complexType mixed="true">
    <xsd:sequence>
        <xsd:element name="subElement1" type="subElementType"/>
        ...
    </xsd:sequence>
</xsd:complexType>
```

注: このセクションで説明する混合ビジネス・オブジェクトは、エレメントと文字データの繰り返しリスト用に使用されます。選択リストに文字データが含まれない場合には、 76 ページの『修飾されたコンポーネント名』 で説明するラッパー・ビジネス・オブジェクトを使用してください。

この複合タイプでは mixed 属性が true に設定されるため、このタイプにより定義される 1 つ以上のサブエレメントの他に文字データを格納することができます。mixed 属性が false に設定されている複合タイプの場合、文字データを入れることはできません。

図 20 に、スキーマ文書で定義される混合タイプの XML エレメントの例を示します。

```
<xsd:complexType name="Cust" mixed="true">
    <xsd:sequence>
        <xsd:element name="Name"/>
        <xsd:element name="Address"/>
        <xsd:element name="Phone"/>
    </xsd:sequence>
</xsd:complexType>
```

図 20. 混合タイプの XML エレメント用のスキーマ文書サンプル

混合タイプの XML エレメントを表すには、次の 2 つのビジネス・オブジェクト定義が必要です。

- 親ビジネス・オブジェクト定義

  この親ビジネス・オブジェクト定義には、複数カーディナリティーのビジネス・オブジェクト配列を表す 1 つの属性があります。この属性は、そのタイプとして、関連付けられたラッパー・ビジネス・オブジェクトのビジネス・オブジェクト定義を保持しています。親ビジネス・オブジェクト定義には、次のアプリケーション固有情報が設定されています。

  – ビジネス・オブジェクト・レベルのアプリケーション固有情報は、次のコンポーネントから構成されます。
    - 関連する混合タイプの XML エレメントの名前と、それに続くセミコロン (;)
    - タグ type=MIXED
  – 複数カーディナリティーの属性のアプリケーション固有情報には、次のタグの設定が必要です。

    (*mixedTypeElement*|*subElement1*|...|*subElementN*)

    ここで、
    - *mixedTypeElement* は、関連する混合タイプの XML エレメントの名前です。
    - *subelement1...subElementN* は、複合タイプに定義された各サブエレメントに対応します。

    各サブエレメントはパイプ文字 (|) で区切り、タグ全体は括弧で囲むことが必要です。

- ラッパー・ビジネス・オブジェクト定義

  ラッパー・ビジネス・オブジェクト定義には、混合データ用の属性が設定されています。

  – 文字データ用の 1 つの属性。この属性には、type=pcdata タグ (文字データを指すため) および notag タグ (データが別のエレメントではなく、現在のエレメントに関係することを示すため) が必要です。
  – 混合タイプのエレメントに定義された各サブエレメントに対応する属性。各属性に、type=pcdata タグ (単純タイプを表すことを示すため) が必要です。

  type=pcdata タグについて詳しくは、81 ページの『複合タイプの XML エレメントの場合』を参照してください。

  注: このラッパー・ビジネス・オブジェクト定義には、ビジネス・オブジェクト・レベルでのアプリケーション固有情報は必要ありません。

ビジネス・オブジェクト定義 MyApp_Cust が、65 ページの図 20 に示した Cust 混合タイプ・エレメントを表す場合、そのアプリケーション固有情報は次のようになります。

```
[BusinessObjectDefinition]
Name = MyApp_Cust
AppSpecificInfo = type=MIXED;

   [Attribute]
   Name=Cust_wrapper1
   Type=MyApp_CustWrapper1
```

```
Cardinality=n
AppSpecificInfo=(Cust|Address|Phone)
...
[End]
```

スキーマ文書に基づくラッパー・ビジネス・オブジェクト定義: ラッパー・ビジネス・オブジェクト定義は、繰り返し選択リストを表します。このタイプのビジネス・オブジェクト定義は、XML エレメントに任意の順序と任意のカーディナリティーになり得る子がある場合に必要です。ラッパー・ビジネス・オブジェクトは、子エレメントの順序とカーディナリティーを特定の XML 文書に保存します。

注: このセクションで説明するラッパー・ビジネス・オブジェクトは、エレメントを含み、文字データは含まない 繰り返し選択リスト用に使用されます。選択リストの文字データが含まれる場合には、混合ビジネス・オブジェクトを使用してください。詳細については、65 ページの『スキーマ文書に基づく混合ビジネス・オブジェクト定義』を参照してください。

スキーマ文書は、選択リストの XML エレメントを、次のモデル・グループのいずれかを含む複合タイプとして記述することができます。

- エレメントの順序なしリストを定義する choice グループ。リスト内のエレメントの数は 1 つのみであることが必要です。

  ```
  <xsd:complexType>
      <xsd:choice>
      ...
      </choice>
  </complexType>
  ```

  choice グループを含む複合タイプの XML エレメントを表すため、XML データ・ハンドラーは、DTD で定義された選択リスト XML エレメント用の階層型ビジネス・オブジェクト定義と同じ階層型ビジネス・オブジェクト定義を想定します。

  – ラッパー・ビジネス・オブジェクトをタイプとする複数カーディナリティーの単一の属性を格納している親ビジネス・オブジェクト定義。
  – choice グループ内の各サブエレメントに対応する属性を格納しているラッパー・ビジネス・オブジェクト定義。

  これらのビジネス・オブジェクト定義には、DTD 用の選択リスト XML エレメントの場合と同じフォーマットで、ビジネス・オブジェクト・レベルのアプリケーション固有情報が格納されています。詳細については、51 ページの『DTD に基づくラッパー・ビジネス・オブジェクト定義』を参照してください。

- 各エレメントがそれぞれ 1 回以上出現するエレメントの順序なしリストを定義する all グループ。

  ```
  <xsd:complexType>
      <xsd:all>
      ...
      </xsd:all>
  </xsd:complexType>
  ```

  all グループを含む複合タイプの XML エレメントを表すため、XML データ・ハンドラーは、次のビジネス・オブジェクト定義を持つ階層型ビジネス・オブジェクト定義を想定します。

– ラッパー・ビジネス・オブジェクトをタイプとする*複数カーディナリティー*の単一の属性を格納している親ビジネス・オブジェクト定義。
– all グループ内の各サブエレメントに対応する属性を格納しているラッパー・ビジネス・オブジェクト定義

これらのビジネス・オブジェクト定義には、DTD 用の選択リスト XML エレメントの場合と同じフォーマットで、ビジネス・オブジェクト・レベルのアプリケーション固有情報が格納されています。詳細については、51 ページの『DTD に基づくラッパー・ビジネス・オブジェクト定義』を参照してください。

カーディナリティーを指示するため、これらのモデル・グループでは、minOccurs および maxOccurs 属性により、出現の制限をサポートしています。詳細については、70 ページの表 18を参照してください。

スキーマ文書内の XML エレメント定義の例を次に示します。

```
<xsd:element name="CUST">
    <xsd:complexType>
        <xsd:choice minOccurs="0" maxOccurs="unbounded">
            <xsd:element ref="U"/>
            <xsd:element ref="I"/>
            <xsd:element ref="B"/>
        </xsd:choice>
    </xsd:complexType>
</xsd:element>
```

このエレメントには 3 つのオプションのサブエレメントが含まれ (ただしリストのサブエレメントが必ず 1 つは存在しなければなりません)、その出現順序は任意です。52 ページの図 14 に、このタイプの XML 文書を示します。53 ページの図 16 に、この XML 文書に対応する親ビジネス・オブジェクト定義を示します。54 ページの図 17 に、ラッパー・ビジネス・オブジェクト定義を示します。

スキーマ文書に基づく混合ビジネス・オブジェクト定義における型置換: *型置換*によって、個々の XML 文書インスタンス内で基本タイプの代わりに派生型を使用できます。型置換が行われると、1 つのデータ型を用いた宣言に適合するエレメントは、それを拡張または制限する任意のデータ型を持つことができます。以下のスキーマ定義では、ShirtType および HatType は基本的な ProductType の派生型です。

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
<xsd:element name="items" type="ItemsType"/>
<xsd:complexType name="ItemsType">
  <xsd:sequence>
            <xsd:element ref="product" maxOccurs="unbounded"/>
          </xsd:sequence>
</xsd:complexType>
<xsd:element name="product" type="ProductType"/>
<xsd:complexType name="ProductType">
    <xsd:sequence>
          <xsd:element name="number" type="xsd:string"/>
            <xsd:element name="name" type="xsd:string"/>
      </xsd:sequence>
</xsd:complexType>
<xsd:complexType name="ShirtType">
    <xsd:complexContent>
    <xsd:extension base="ProductType">
          <xsd:sequence>
                <xsd:element name="size" type="xsd:string"/>
                <xsd:element name="color" type="xsd:string"/>
          </xsd:sequence>
    </xsd:extension>
    </xsd:complexContent>
</xsd:complexType>
<xsd:complexType name="HatType">
    <xsd:complexContent>
    <xsd:extension base="ProductType">
      <xsd:sequence>
                <xsd:element name="size" type="xsd:string"/>
              </xsd:sequence>
            </xsd:extension>
        </xsd:complexContent>
    </xsd:complexType>
</xsd:schema>
```

図 21. 型置換を用いたスキーマのサンプル

上記のスキーマを基にした場合、以下の XML 文書が有効です。

```
<items xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
<product>
      <number>999</number>
      <name>Special Seasonal</name>
</product>
<product xsi:type="ShirtType">
      <number>557</number>
      <name>Short-Sleeved Linen Blouse</name>
      <size>M</size>
      <color>blue</color>
</product>
<product xsi:type="HatType">
      <number>443</number>
      <name>Four-Gallon Hat</name>
      <size>L</size>
</product>
</items>
```

図 22. XML 文書内の派生型

ProductType が出現する箇所では、その派生型の ShirtType および HatType (xsi:type 属性によって表される) を代わりに使用することができます。型置換が行われる XML 文書を表すため、XML データ・ハンドラーは、ラッパー・ビジネス・オブジェクトを XML 文書の子の属性として作成します。このラッパー・ビジネス・オブ

ジェクトは、複合型 (69 ページの図 21 内の ProductType) およびその派生型 (ShirtType および HatType) に対応する子の属性を持ちます。表 16 および表 17 に、前述の XML スキーマから生成されるビジネス・オブジェクト定義を示します。

表 16. ItemsType のビジネス・オブジェクト定義

属性名	型	カーディナリティー	ASI
Product	ProductTypeWrapper	N	(product);typeSub=true

表 17. ProductTypeWrapper のビジネス・オブジェクト定義

属性	型	カーディナリティー	ASI
ProductType	ProductType	1	Elem_name=product; xsiType=ProductType
ShirtType	ShirtType	1	Elem_name=product; xsiType=ShirtType;
HatType	HatType	1	Elem_name=product; xsiType=HatType;

スキーマ文書のビジネス・オブジェクト属性プロパティー

XML 文書用のビジネス・オブジェクト定義がスキーマ文書に基づいている場合、ビジネス・オブジェクト属性プロパティーには、 39 ページの『ビジネス・オブジェクトの属性プロパティー』で説明した制約があります。さらに、スキーマ文書の構文により、ビジネス・オブジェクト属性の「必須性」が決まる場合があります。「必須性」とは、カーディナリティーや属性がキーであるかどうかなどの要因の組み合わせであり、XML データ・ハンドラーがその属性を必要とするかどうかを指定します。属性が必須な場合には、Required 属性プロパティーを true に設定することが必要です。

Required 属性プロパティーの設定は、以下に示すように、XML エレメントおよび属性の指定のほか、Cardinality、Key、および Foreign Key 属性プロパティーの設定に依存します。

- ビジネス・オブジェクト属性のカーディナリティーは、スキーマ文書内の出現インディケーターにより決定されます。このカーディナリティーは、属性が必須であるかどうかに影響します。表 18 に、スキーマ文書で宣言されたエレメントの各組み合わせに対応するカーディナリティーと「必須性」の概要を示します。

表 18. スキーマ文書に対応するカーディナリティーと「必須性」

スキーマ・フラグメントの出現インディケーター	カーディナリティー	必須
指定なし	1	はい
maxOccurs > 1	N	はい
maxOccurs = ″unbounded″	N	はい
minOccurs=0	影響なし	いいえ
minOccurs>1	N	はい

- ビジネス・オブジェクト属性が必須であるかどうかは、スキーマ文書内の use 属性によっても決まります。表 19 に、use 属性の各値に対応する「必須性」の概要を示します。

表 19. スキーマ文書に対応する「必須性」

スキーマ・フラグメントの出現属性: use	カーディナリティー	必須
指定なし	影響なし	いいえ
use=required	影響なし	はい

- ビジネス・オブジェクト属性が基本キーか外部キーかは、スキーマ文書内の id 属性により決まります。キーの存在は、属性が必須であるかどうかに影響します。表 20 に、id 属性の値が、どのようにビジネス・オブジェクト属性の「必須性」に影響するかを示します。

表 20. スキーマ文書に対応するキーと「必須性」

スキーマ・フラグメントの属性: id	キー	必須	コメント
id=ID	あり	いいえ	

スキーマ文書内の XML コンポーネントのアプリケーション固有情報

このセクションでは、スキーマ文書に基づくビジネス・オブジェクト定義のためのアプリケーション固有情報フォーマットの次の内容について説明します。

- 『ビジネス・オブジェクト・レベルのアプリケーション固有情報』
- 78 ページの『属性アプリケーション固有情報』

ビジネス・オブジェクト・レベルのアプリケーション固有情報: XML データ・ハンドラーは次のタイプのビジネス・オブジェクト定義を使用して、スキーマ文書に定義されている異なる種類の root XML エレメントを表します。これらのビジネス・オブジェクト定義のタイプは、ビジネス・オブジェクト・レベルでアプリケーション固有情報により識別されます。

表 21. ビジネス・オブジェクト・レベルのアプリケーション固有情報のタグ

アプリケーション固有情報内のタグ	説明	詳細
target_ns	スキーマ文書のターゲット・ネーム・スペースを指定します。	72 ページの『スキーマ・ネーム・スペース』
attr_fd	属性名を修飾するかどうかを指定します。	76 ページの『修飾されたコンポーネント名』
elem_fd	ローカルに宣言されたエレメント名を修飾するかどうかを指定します。	76 ページの『修飾されたコンポーネント名』

注: ビジネス・レベルのアプリケーション固有情報には、type=MIXED タグを組み込むこともできます。詳細については、65 ページの『スキーマ文書に基づく混合ビジネス・オブジェクト定義』を参照してください。

スキーマ・ネーム・スペース: DTD とは異なり、スキーマ文書は、1 つ以上のネーム・スペースの定義を必要とします。ネーム・スペース は、XML 文書内のエレメント名、エレメント・タイプ、および属性のコンテキストを規定します。ネーム・スペースは Uniform Resource Identifier (URI) であり、HTTP、FTP、およびその他のパスが含まれます。表 22 に、スキーマ・コンポーネント間で参照を解決するためにスキーマ文書で宣言できるネーム・スペースを示します。

表 22. XML ネーム・スペース

ネーム・スペース	説明	名前	共通プレフィックス
XML スキーマ・ネーム・スペース	XML スキーマ定義言語 (XSDL) で使用される element、schema、simpleType などすべてのコンポーネントを定義します。	http://www.w3.org/2001/XMLSchema	xsd、xs
XML スキーマ・インスタンス・ネーム・スペース	スキーマ・インスタンスに関連付けられた 4 つの属性 type、nil、schemaLocation、noNamespaceSchemaLocation を定義します。	http://www.w3.org/2001/XMLSchema-instance	xsi
ユーザー定義のネーム・スペース	グローバル宣言により宣言または定義されたすべてのコンポーネント (element、attribute、type、group など) を定義します。 **注:** ローカルに宣言されたエレメントでは、ターゲット・ネーム・スペースを使用する場合と、しない場合があります。詳細については、 76 ページの『修飾されたコンポーネント名』を参照してください。	ユーザー定義	ユーザー定義

注: XML ODA は、複数のターゲット・ネーム・スペースを持つスキーマ文書をサポートしています。

各スキーマ文書では 1 つのターゲット・ネーム・スペース を宣言でき、targetNamespace タグを使用して、ネーム・スペースがどのグローバル・コンポーネント (エレメント、属性、タイプ、またはグループ) に属するかを指定します。スキーマ・エレメントに targetNamespace タグを設定する場合、そのスキーマ文書用に生成された各ビジネス・オブジェクト定義のアプリケーション固有情報に target_ns タグを設定して、XML 文書で宣言されたターゲット・ネーム・スペースを指定する必要があります。

target_ns=*URI address for target namespace*

target_ns タグは、次のようにしてすべての ビジネス・オブジェクト定義のビジネス・オブジェクト・レベルのアプリケーション固有情報に設定する必要があります。

- トップレベル・ビジネス・オブジェクト定義の場合、target_ns タグの値は、XML schema エレメントの targetNamespace 属性が指定する値を指定します。
- XML エレメントを表す各ビジネス・オブジェクト定義にも、アプリケーション固有情報内に target_ns タグを組み込む必要があります。すべてのグローバル・コンポーネント (エレメント、属性、タイプなど) はスキーマ文書のターゲット・

ネーム・スペースに属するので、これらのコンポーネントを表すビジネス・オブジェクト定義も、スキーマ文書のターゲット・ネーム・スペースを指定します。

注: 以前のバージョンの XML データ・ハンドラーでは、トップレベル・ビジネス・オブジェクト定義にネーム・スペース・プレフィックスの属性を設定し、属性レベルのアプリケーション固有情報に適切な type=defaultNS タグと type=xmlns タグを使用する必要がありました。このようにネーム・スペース・プレフィックスを定義する仕組みは置き換えられましたが、XML データ・ハンドラーは、既存のビジネス・オブジェクト定義との後方互換性のためにこの仕組みのサポートを継続しています。新規のビジネス・オブジェクト定義では、このセクションで説明したように target_ns タグを使用します。XML ODA は、target_ns タグを使用するよう変更されました。

例えば、図 23 のスキーマ文書は、(xsd プレフィックスを持つ) XML スキーマ・ネーム・スペースとターゲット・ネーム・スペース (デフォルトのネーム・スペース) を記述しています。

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
   targetNamespace="http://www.ibm.com/ns1" xmlns="http://www.ibm.com/ns1">

   <xsd:complexType name="TaxInfoType">
      <xsd:sequence>
         <xsd:element name="SSN" type="string">
         </xsd:element>

         <xsd:element name="State" type="string">
         </xsd:element>
      </xsd:sequence>
   </xsd:complexType>

   <xsd:element name="Customer">
      <xsd:complexType>
         <xsd:sequence>
            <xsd:element name="TaxInfo" type="TaxInfoType">
            </xsd:element>

            <xsd:element name="BillTo" type="xsd:string">
            </xsd:element>
         </xsd:sequence>

         <xsd:attribute name="Name" type="xsd:string">
         </xsd:attribute>

         <xsd:attribute name="ID" type="xsd:string">
         </xsd:attribute>
      </xsd:complexType>
   </xsd:element>
</xsd:schema>
```

図 23. スキーマ文書のサンプル Schema1.xsd

XML ODA は、このスキーマ文書に対して BOPrefix_TopLevel、BOPrefix_TopLevel_Customer、および BOPrefix_TopLevel_TaxInfoType という 3 つのビジネス・オブジェクト定義を生成します (ここで、BOPrefix および TopLevel はこれらの ODA 構成プロパティーの値です)。これらの 3 つのビジネス・オブジェクト定義では、ビジネス・オブジェクト・レベルのアプリケーション固有情報に次の行が設定されます。

```
target_ns=http://www.ibm.com/ns1;elem_fd=unqualified;attr_fd=unqualified
```

注: 図 23 の schema エレメントには elementFormDefault 属性も attributeFormDefault 属性も含まれていないので、このアプリケーション固有情報には unqualified に設定した elem_fd タグと attr_fd タグが組み込まれます。詳細については、 76 ページの『修飾されたコンポーネント名』を参照してください。

1 つのスキーマ文書では 1 つのターゲット・ネーム・スペースのみを定義できます。ただし、import エレメントを使用すると、別のスキーマ文書のターゲット・ネーム・スペースで定義されたエレメントや属性を組み込むことができます。

73 ページの図 23 で定義した Schema1.xsd 文書に基づくスキーマ文書を図 24 に示します。このスキーマ文書は、TaxInfoType 複合タイプ、BillTo エレメント、および Name 属性を宣言する ns2 ネーム・スペースをインポートします。

```
<?xml version "1.0" encoding="UTF-8"?>
<xsd:schema smlns:xsd="http://www.w3.org/2001/XMLSchema"
   targetNamespace="http://www.example.com/ns1" xmlns="http://www.example.com/ns1"
   attributeFormDefault="qualified" elementFormDefault="qualified"
   xmlns:ns2="http://www.example.com/ns2">

   <xsd:import schemaLocation="Schema2.xsd"
      namespace="http://www.example.com/ns2"/>

   <xsd:element name="Customer2">
      <xsd:complexType>
         <xsd:sequence>
            <xsd:element name="TaxInfo" type="ns2:TaxInfoType">
            </xsd:element>

            <xsd:element ref="ns2:BillTo">
            </xsd:element>
         </xsd:sequence>

         <xsd:attribute ref="ns2:Name">
         </xsd:attribute>

         <xsd:attribute name="ID" type="xsd:string">
         </xsd:attribute>
      </xsd:complexType>
   </xsd:element>
</xsd:schema>
```

図 24. ターゲット・ネーム・スペースのインポート

root エレメントのビジネス・オブジェクト定義 Customer2 は、TaxInfo、BillTo、および Name XML コンポーネントを表す属性に、この代替ネーム・スペースを次のように指定する必要があります。

- TaxInfo 属性は、タイプとして TaxInfoType を表すビジネス・オブジェクト定義を持ちます。タイプは ns2 (http://www.example.com/ns2) ネーム・スペースで定義されます (75 ページの図 25 を参照)。
- BillTo 属性は、アプリケーション固有情報に elem_ns タグを設定して、関連する BillTo エレメントのソースとして ns2 ネーム・スペースを指定します。
- Name 属性は、アプリケーション固有情報に attr_ns タグを設定して、関連する XML 属性 Name のソースとして ns2 ネーム・スペースを指定します。

図 25 に、ns2 ネーム・スペース内の TaxInfoType、BillTo、および Name XML コンポーネントを定義するスキーマ文書を示します。

```
<?xml version "1.0" encoding="UTF-8"?>
<schema smlns:xsd="http://www.w3.org/2001/XMLSchema"
   targetNamespace="http://www.example.com/ns2"
   attributeFormDefault="qualified" elementFormDefault="qualified"
   xmlns:ns2="http://www.example.com/ns2">

   <complexType name="TaxInfoType">
      <sequence>
         <element name="SSN" type="string">
         </element>
         <element name="State" type="string">
         </element>
      </sequence>
   </complexType>

   <attribute name="Name" type="string"
   </attribute>

   <complexType name="AddressType">
      <sequence>
         <element name="Zip" type="string">
         </element>
         <element name="Street" type="string">
         </element>
      </sequence>
   </complexType>

   <element name="BillTo" type="ns2:AddressType">
   </element>
</schema>
```

図 25. 2 番目のネーム・スペースの定義

図 25 に、Customer2 root エレメントのビジネス・オブジェクト定義を示します。

```
[BusinessObjectDefinition]
Name=TopLevel_Customer2
AppSpecificInfo=target_ns=http://www.example.com/ns1;elem_fd=qualified;
   attr_fd=qualified
...
   [Attribute]
   Name=Name
   Type=String
   AppSpecificInfo=attr_name=Name;type=attribute;attr_ns=http://www.example.com/ns2
   ...
   [End]

   [Attribute]
   Name=ID
   Type=String
   AppSpecificInfo=attr_name=ID;type=attribute
   ...
   [End]

   [Attribute]
   Name=schemaLocation
   Type=String
   AppSpecificInfo=attr_name=schemaLocation;type=xsischemalocation
   ...
   [End]

   [Attribute]
   Name=TaxInfo
   Type=TopLevel_TaxInfoType
   AppSpecificInfo=elem_name=TaxInfo
   ...
   [End]

   [Attribute]
   Name=BillTo
   Type=TopLevel_AddressType
   AppSpecificInfo=elem_name=BillTo;elem_ns=http://www.example.com/ns2
   ...
   [End]
...
[End]
```

図 26. root エレメント・ビジネス・オブジェクト定義のサンプル

修飾されたコンポーネント名: XML 文書の中で、ネーム・スペースに関連付けられたコンポーネント名は、次のように修飾されるか、または修飾されません。

- 非修飾名にはプレフィックスがなく、いずれのネーム・スペースの一部でもありません。
- 修飾名は、次のいずれかです。
  - コンポーネント名には、ネーム・スペースに関連するプレフィックスが含まれます。

    1 つ以上のネーム・スペースにプレフィックスを割り当てることができます。ネーム・スペースのプレフィックスは xmlns:*prefix* タグを使用して宣言します。ここで、*prefix* は宣言されたプレフィックスです。スキーマ文書内のコンポーネント定義では、これらのコンポーネント名は*修飾*されます。これは、コンポーネント名の先頭にプレフィックスが付加されるためです (*prefix*:*componentName*)。
  - 名前はプレフィックスを含みませんが、デフォルトのネーム・スペースの一部となります (エレメントのみ)。

デフォルトのネーム・スペース は、コンポーネント名にプレフィックスを含まないコンポーネントに関連付けられるネーム・スペースを指定します。 デフォルトのネーム・スペースは、xmlns タグを使用して宣言します。

XML データ・ハンドラーが XML からビジネス・オブジェクトへの変換を正しく処理するためには、XML 文書用のネーム・スペースとスキーマ文書用のネーム・スペースが次のように一致していることが必要です。

- スキーマ文書がデフォルトのネーム・スペースを指定している場合は、XML 文書もデフォルトのネーム・スペースを指定することが必要です。
- スキーマ文書がデフォルトのネーム・スペースを持たない 場合には、XML 文書もデフォルトのネーム・スペースを持つことができません。

schema エレメントの elementFormDefault 属性は、ローカルに宣言されたエレメント名を修飾するかどうかを指定します。デフォルトでは、ローカルに宣言されたエレメントは修飾されず、デフォルトのネーム・スペースに属します。 表 23 に示すように、elementFormDefault 属性の値は、ビジネス・オブジェクト・レベルのアプリケーション固有情報内の elem_fd タグの値を決定します。

表 23. elem_fd タグの設定

elementFormDefault の値	**elem_fd タグの値**
"unqualified" (または属性がまったく指定されない)	elem_fd=unqualified
"qualified"	elem_fd=qualified

例えば、 73 ページの図 23 のスキーマ文書の schema エレメントには、elementFormDefault 属性は含まれません。したがって、このスキーマ文書のすべてのビジネス・オブジェクト定義のビジネス・オブジェクト・レベルのアプリケーション固有情報 (BOPrefix_TopLevel、BOPrefix_TopLevel_Customer、および BOPrefix_TopLevel_TaxInfoType。ここで、BOPrefix と TopLevel はこれらの ODA 構成プロパティーの値) には、次のタグが含まれます。

```
elem_fd=unqualified
```

注: スキーマ文書の schema エレメントに次の属性が含まれる場合、これらの 3 つのビジネス・オブジェクト定義のビジネス・オブジェクト・レベルのアプリケーション固有情報には、これと同じタグが含まれます。

```
elementFormDefault="unqualified"
```

個々の XML エレメントに form 属性が含まれる場合、この form 属性の値は elementFormDefault 属性のすべての設定をオーバーライドします。

schema エレメントの attributeFormDefault 属性は、エレメント名を修飾するかどうかを指定します。デフォルトでは、属性名は修飾されず、いずれのネーム・スペースにも属しません。表 24 に示すように、attributeFormDefault 属性の値は、ビジネス・オブジェクト・レベルのアプリケーション固有情報の attr_fd タグの値を決定します。これは、elementFormDefault 属性の値が elem_fd タグの値を決定するのと同様です。

表 24. attr_fd タグの設定

attributeFormDefault の値	elem_fd タグの値
"unqualified" (または属性がまったく指定されない)	attr_fd=unqualified
"qualified"	attr_fd=qualified

例えば、 73 ページの図 23 のスキーマ文書の schema エレメントには、attributeFormDefault 属性は*含まれません*。したがって、このスキーマ文書のすべて のビジネス・オブジェクト定義のビジネス・オブジェクト・レベルのアプリケーション固有情報 (BOPrefix_TopLevel、BOPrefix_TopLevel_Customer、および BOPrefix_TopLevel_TaxInfoType。ここで、BOPrefix と TopLevel はこれらの ODA 構成プロパティーの値) には、次のタグが含まれます。

```
attr_fd=unqualified
```

注: スキーマ文書の schema エレメントに次の属性が含まれる場合、これらの 3 つのビジネス・オブジェクト定義のビジネス・オブジェクト・レベルのアプリケーション固有情報には、これと同じタグが含まれます。

```
attributeFormDefault="unqualified"
```

属性アプリケーション固有情報: ビジネス・オブジェクト定義の属性は、以下の XML コンポーネントを表すことができます。

- 79 ページの『XML エレメントの場合』
- 81 ページの『複合タイプの XML エレメントの場合』
- 83 ページの『XML 属性の場合』
- 85 ページの『XML 処理命令の場合』
- 84 ページの『XML コメントの場合』
- 84 ページの『特殊文字が含まれる XML エレメントまたは属性の場合』
- 85 ページの『XML スキーマ・ロケーションの場合』

表 25 に、これらのさまざまな XML コンポーネントの属性レベルのアプリケーション固有情報のタグと、タグの詳細が記載されている本書のセクションを示します。

表 25. 属性のアプリケーション固有情報のタグ

ビジネス・オブジェクト属性の表現	アプリケーション固有情報	詳細
XML エレメント	elem_name=*name of XML element* elem_ns=*namespace for element's definition* elem_fd=*value of* form *attribute*	『XML エレメントの場合』
複合タイプの XML エレメント	type=pcdata	81 ページの『複合タイプの XML エレメントの場合』
XML エレメントの属性	attr_name=*name of XML attribute* type=attribute attr_ns=*namespace for attribute's definition* attr_fd=*value of* form *attribute*	83 ページの『XML 属性の場合』
特殊文字を持つ XML エレメントまたは属性	escape=true	84 ページの『特殊文字が含まれる XML エレメントまたは属性の場合』
XML 文書に追加されるコメント	type=comment	84 ページの『XML コメントの場合』
処理命令	type=pi	85 ページの『XML 処理命令の場合』
XML インスタンスの schemaLocation または noNamespaceSchemaLocation 属性	type=xsischemalocation type=xsinoNSlocation	85 ページの『XML スキーマ・ロケーションの場合』

注: 属性のアプリケーション固有情報には、(a | b | c) という形式のタグを使用して、反復選択を表す複数カーディナリティーの属性を指定することもできます。詳細については、 67 ページの『スキーマ文書に基づくラッパー・ビジネス・オブジェクト定義』を参照してください。

XML エレメントの場合: ビジネス・オブジェクト属性が XML エレメントを表す場合、アプリケーション固有情報に次の elem_name タグを設定して、関連するエレメントを識別する必要があります。

elem_name=*name of XML element*

XML エレメント名には特殊文字 (ピリオドやハイフンなど) を使用できます。ただし、ビジネス・オブジェクト属性名にこれらの特殊文字を使用することはできません。したがって、XML エレメント名を elem_name タグに指定する必要があります。ビジネス・オブジェクト属性に名前を付けるとき、XML ODA は XML エレメントの名前から特殊文字を除去します。

ビジネス・オブジェクト属性は、次の場合に XML エレメントを表すことができます。

- XML エレメントが XML 複合タイプである (または XML エレメントを含む) 場合

この場合、ビジネス・オブジェクト属性は*複合* 属性であり、そのデータ型は XML 複合タイプを表すビジネス・オブジェクト定義です。 (属性のアプリケーション固有情報内の) elem_name タグには、XML エレメントの名前 (または複合タイプ) が含まれます。

- XML エレメントが XML 複合タイプの一部である場合

  この場合、ビジネス・オブジェクト属性は、タイプ String の*単純* 属性です。アプリケーション固有情報には elem_tag (複合タイプの XML エレメント名を含む) および type=pcdata タグが設定されています。詳細については、81 ページの『複合タイプの XML エレメントの場合』を参照してください。

XML エレメントを表すビジネス・オブジェクト属性は、そのアプリケーション固有情報に、次のタグも組み込むことができます。

- elem_fd タグは、XML エレメントの form 属性の設定を指定します。この属性は、ローカルに宣言されたエレメント名が修飾されるかどうかを示します。XML エレメントが属性 form="qualified" を指定した場合、elem_fd の値は form 属性の値に設定されます。

  例えば、ローカルに宣言された XML エレメントが次のように定義されていると仮定します。

  ```
  <xsd:element ref="Name" form="qualified"></xsd:element>
  ```

  関連するビジネス・オブジェクト属性は、次のフォーマットを使用します。

  ```
  [Attribute]
  Name=ns2:Name
  Type=String
  AppSpecificInfo=elem_name=Name;elem_fd=qualified;
  ...
  ```

  XML エレメントに form 属性が指定*されていない* 場合、(schema エレメントの) elementFormDefault 属性の値によって、エレメント名が修飾されるかどうかが決定します。詳細については、76 ページの『修飾されたコンポーネント名』を参照してください。

- XML エレメントのターゲット・ネーム・スペースがスキーマ文書のターゲット・ネーム・スペースとは異なる 場合、elem_ns タグが XML エレメントのターゲット・ネーム・スペースを指定します。スキーマ文書に複数のネーム・スペースが使用される場合、このタグが必要です。あるスキーマ文書で参照される XML エレメントが*別のスキーマ文書* のターゲット・ネーム・スペースで定義されている場合、このタグはそのネーム・スペース名をリストします。

  例えば、複合タイプの XML エレメントが次のように定義されていると仮定します。

  ```
  <xsd:element ref="ns2:BillTo"></xsd:element>
  ```

  関連するビジネス・オブジェクト属性は、次のフォーマットを使用します。

  ```
  [Attribute]
  Name=BillTo
  Type=String
  AppSpecificInfo=elem_name=BillTo;elem_ns=http:/www.imb.com/ns2;
  ...
  ```

2 番目のネーム・スペースに定義された XML エレメントを含むスキーマ文書の詳細な例については、72 ページの『スキーマ・ネーム・スペース』を参照してください。

複合タイプの XML エレメントの場合: ビジネス・オブジェクト定義が XML 複合タイプ (complexType) を表す場合、この複合タイプの内容は、このビジネス・オブジェクト定義内の単純 (String) 属性によって表されます。これらのビジネス・オブジェクト属性のアプリケーション固有情報には、elem_name タグを組み込んで、エレメント名を指定する必要があります。

注: このタグの詳細については、79 ページの『XML エレメントの場合』を参照してください。

XML 複合タイプでは、ビジネス・オブジェクト属性は表 26 に示す複合タイプの内容を表すことができます。

表 26. XML 複合タイプの内容

複合タイプの エレメントのタイプ	説明	属性のアプリケーション 固有情報
単純な XML エレメント	文字のみ を含むエレメント。他のエレメントや XML 属性は格納できません。これが可能なのは複合タイプに限られます。	type=pcdata
単純な内容	文字データのみ (エレメントなし)。	type=pcdata;notag
属性を持つ XML エレメント	文字データおよび サブエレメントと属性の組み合わせの*両方* を含むエレメント。	なし。 このタイプの XML エレメントは、単一のビジネス・オブジェクト属性としてではなく、ビジネス・オブジェクト定義を使用して表す必要があります。詳細については、64 ページの『スキーマ文書に基づく通常のビジネス・オブジェクト定義』を参照してください。

type=pcdata タグを使用する例として、73 ページの図 23 のスキーマ文書には、単純な XML エレメントのみを持つ TaxInfoType 複合タイプの定義が含まれます。TaxInfoType XML 複合タイプのビジネス・オブジェクト定義 (BOPrefix_TopLevel_TaxInfoType。ここで BOPrefix と TopLevel は、それぞれの名前を持つ ODA 構成プロパティーの値) には、2 つの属性が含まれます。各属性のアプリケーション固有情報には、図 27 に示すように、関係する XML エレメントの名前が含まれます。この複合タイプには単純な XML エレメントのみが含まれる (サブエレメントや属性は含まれない) ので、対応するビジネス・オブジェクト属性のアプリケーション固有情報に type=pcdata タグも設定する必要があります。

```
[BusinessObjectDefinition]
Name=BOPrefix_TopLevel_TaxInfoType
AppSpecificInfo=target_ns=;elem_fd=unqualified;attr_fd=unqualified
...
[End]

[Attribute]
Name=SSN
Type=String
AppSpecificInfo=elem_name=SSN;type=pcdata
...
[End]

[Attribute]
Name=State
Type=String
AppSpecificInfo=elem_name=SSN;type=pcdata
...
[End]
```

図 27. 単純なエレメントを持つ XML 複合タイプのビジネス・オブジェクト定義のサンプル

notag キーワードを type=pcdata タグとともに使用する例として、Price という XML エレメントが PriceType 複合タイプであり、単純な内容のみ、つまり文字データのみが含まれると仮定します。この例で、simpleContent エレメントには Currency という名前の属性が定義され、Price 用のデータが必要です。

```
<xsd:element name="Price" type="PriceType">
    <xsd:complexType name="PriceType">
        <xsd:simpleContent>
            <xsd:extension base="xsd:decimal">
                <xsd:attribute name="Currency" type="xsd:NMTOKEN"/>
            </xsd:extension>
        </xsd:simpleContent>
    </xsd:complexType>
</element>
```

PriceType 複合タイプのビジネス・オブジェクト定義には、単純な内容に関連付けられた文字データを保持するための属性が設定されています。この属性のアプリケーション固有情報には、次のタグを設定する必要があります。

```
type=pcdata;notag
```

notag キーワードにより、XML データ・ハンドラーによる開始タグの重複生成 (ビジネス・オブジェクト定義用と属性用) を防止できます。 XML データ・ハンドラーは、ビジネス・オブジェクト属性ごとに XML 開始タグと終了タグを作成します。ただし、この属性のアプリケーション固有情報に notag が指定されている場合にはタグの作成はありません。

Price 子ビジネス・オブジェクト定義は、次のようになります。

```
[BusinessObjectDefinition]
Name = Price
AppSpecificInfo =

[Attribute]
Name = Currency
Type = String
AppSpecificInfo = attr_name=Currency;type=attribute;
...
[End]

[Attribute]
```

```
Name = Price
Type = String
AppSpecificInfo = elem_name=Price;type=pcdata;notag
...
[End]
```

Price データを保持するにはビジネス・オブジェクト属性が必要です。Price データの属性には、そのアプリケーション固有情報に notag を指定して、データ・ハンドラーがこの属性の開始および終了タグを作成しないようにする必要があります。この場合、データ・ハンドラーは、Price データの新しい XML エレメントは生成せず、データを親エレメントに追加するだけです。

さらに、ビジネス・オブジェクト定義には、Currency 属性値を保持するための別の属性が必要です。単純な内容に属性が格納されている場合には、ビジネス・オブジェクト定義に、各 XML 属性に対応する属性も設定されていることが必要です。属性のアプリケーション固有情報には、type=attribute タグが設定されていることが必要です。詳細については、『XML 属性の場合』を参照してください。

XML 属性の場合: ビジネス・オブジェクト定義が XML エレメントまたは複合タイプを表す場合、スキーマ文書がこのエレメント用に宣言するすべての属性は、ビジネス・オブジェクト定義内の属性として表す必要があります。ビジネス・オブジェクト属性が XML エレメントの属性を表す場合、アプリケーション固有情報に次のタグを設定する必要があります。

- attr_name タグ

  attr_name=*name of XML attribute*

- type タグ

  type=attribute

注: これらのタグの詳細については、56 ページの『XML 属性の場合』を参照してください。type=attribute タグの使用例については、81 ページの『複合タイプの XML エレメントの場合』を参照してください。

XML 属性を表すビジネス・オブジェクト属性は、そのアプリケーション固有情報に、次のタグも組み込むことができます。

- attr_fd タグは、XML 属性の form 属性の設定を指定します。この属性は、属性名が修飾されるかどうかを示します。 XML 属性で form="unqualified" の代わりに属性 form="qualified" が指定されている場合、attr_fd は ″form″ 属性に指定された値を持ちます。

  例えば、XML 属性が次のように定義されていると仮定します。

  ```
  <xsd:attribute ref="Name" form="qualified"></xsd:attribute>
  ```

  関連するビジネス・オブジェクト属性は、次のフォーマットを使用します。

  ```
  [Attribute]
  Name=ns2:Name
  Type=String
  AppSpecificInfo=attr_name=Name;type=attribute;attr_fd=qualified
  ...
  ```

XML 属性に form 属性が指定されていない場合、(schema エレメントの) attributeFormDefault 属性の値によって、属性名が修飾されるかどうかが決定します。詳細については、 76 ページの『修飾されたコンポーネント名』を参照してください。

- XML 属性のターゲット・ネーム・スペースがスキーマ文書のターゲット・ネーム・スペースとは異なる 場合、attr_ns タグが XML 属性のターゲット・ネーム・スペースを指定します。スキーマ文書に複数のネーム・スペースが使用される場合、このタグが必要です。あるスキーマ文書で参照される XML 属性が*別のスキーマ文書* のターゲット・ネーム・スペースで定義されている場合、このタグはそのネーム・スペース名をリストします。

  例えば、XML 属性が次のように定義されていると仮定します。

  ```
  <xsd:attribute ref="ns2:Name"></xsd:attribute>
  ```

  関連するビジネス・オブジェクト属性は、次のフォーマットを使用します。

  ```
  [Attribute]
  Name=Name
  Type=String
  AppSpecificInfo=attr_name=Name;attr_ns=http://www.example.com/ns2;
     type=attribute
  ...
  ```

  2 番目のネーム・スペースに定義された XML 属性を含むスキーマ文書の詳細な例については、 72 ページの『スキーマ・ネーム・スペース』を参照してください。

特殊文字が含まれる XML エレメントまたは属性の場合: 特殊文字を含む XML エレメントまたは XML 属性を表すビジネス・オブジェクト属性は、XML データ・ハンドラーによるエスケープ処理を必要とします。データ・ハンドラーにエスケープ処理の必要性を通知するには、ビジネス・オブジェクト属性のアプリケーション固有情報に次のタグを設定する必要があります。

```
escape=true
```

エスケープ処理の対象である XML 文書が、そのスキーマの記述にスキーマ文書を使用している場合、エスケープ処理を指定するステップは、スキーマの記述に DTD を使用している XML 文書に対してエスケープ処理を指定するステップと同じです。詳細については、 58 ページの『特殊文字が含まれる XML エレメントまたは属性の場合』を参照してください。

XML コメントの場合: XML データ・ハンドラーがビジネス・オブジェクトを XML 文書に変換するとき、XML 文書への XML コメントの追加を XML データ・ハンドラーに指示することができます。これを実行するには、属性のアプリケーション固有情報に次のタグを設定します。

```
type=comment
```

XML ODA は、XML コメント用ビジネス・オブジェクト属性を自動的には生成しません。このような属性は、手動で追加する必要があります。スキーマ文書によって記述されている XML 文書にコメントを追加するステップは、DTD によって記述されている XML 文書にコメントを定義するステップと同じです。詳細については、 59 ページの『XML コメントの場合』を参照してください。

XML 処理命令の場合: XML 文書に XML 処理命令が含まれる場合、スキーマ文書に関連するビジネス・オブジェクト定義に、処理値を保持するための属性を設定する必要があります。この属性のアプリケーション固有情報には、次の type タグを設定する必要があります。

```
type=pi
```

例えば、XML データ・ハンドラーが XML 文書をビジネス・オブジェクトに変換するときには、トップレベル・ビジネス・オブジェクト定義の特別な属性 XMLDeclaration に XML バージョンを置きます。 64 ページの図 19 のトップレベル・ビジネス・オブジェクトは、TopLevel_Customer ビジネス・オブジェクト定義内の XMLDeclaration 属性を示しています。type=pi タグの詳細については、 59 ページの『XML 処理命令の場合』を参照してください。

XML スキーマ・ロケーションの場合: XML 文書が、そのスキーマ文書のスキーマ・ロケーションを指す場合、その XML 文書は次の情報を格納していることが必要です。

- XML スキーマ・インスタンス・ネーム・スペースの宣言と、xsi プレフィックスのこのネーム・スペースへのマッピング
- 次の XML スキーマ・インスタンス属性の 1 つの組み込み
  - xsi:schemaLocation

    この属性は、ネーム・スペースの名前をスキーマ・ロケーションに関連付けます。スキーマ文書がターゲット・ネーム・スペースを使用する場合には、このターゲット・ネーム・スペースの名前は、xsi:schemaLocation 属性 (XML 文書内) により指定されるネーム・スペースの名前と一致していることが必要です。
  - xsi:noNamespaceSchemaLocation

    この属性により、スキーマ・ロケーションが識別されます。スキーマ文書がターゲット・ネーム・スペースを使用していない場合には、1 つのスキーマ・ロケーションを特定するため、XML 文書には noNamespaceSchemaLocation 属性が設定されています。

例えば、XML 文書に、図 28 に示したネーム・スペース宣言があると仮定します。

```
<order xmlns="http://sampleDoc.org.ord"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.example.com/order order.xsd">
   ...
/<order>
```

図 28. XML 文書内のスキーマ・ロケーション定義サンプル

スキーマ文書には、ターゲット・ネーム・スペースを定義している次のネーム・スペース宣言が含まれる場合があります。

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
     xmlns="http://www.example.com/order"
     targetNamespace="http://www.example.com/order">
```

この XML 文書を表すビジネス・オブジェクトが schemaLocation 属性の情報を保持するためには、ビジネス・オブジェクト定義にこのスキーマ・ロケーション情報

のための属性を設定する必要があります。root エレメント・ビジネス・オブジェクト定義では、このスキーマ・ロケーション情報は次のように表現されます。

- schemaLocation 属性を使用している XML 文書の場合には、トップレベル・ビジネス・オブジェクト定義には、次のプロパティーを持つ属性が設定されている必要があります。
  - 属性名は schemaLocation で、この属性のタイプは String です。
  - この属性は、そのアプリケーション固有情報の中に次の type タグを持っていることが必要です。

    ```
    type=xsischemalocation
    ```
  - このビジネス・オブジェクト属性の値は、schemaLocation 属性の値です。
- noNamespaceSchemaLocation 属性を使用している XML 文書の場合には、root エレメント・ビジネス・オブジェクト定義には、次のプロパティーを持つ属性が設定されている必要があります。
  - 属性名は noNamespaceSchemaLocation で、この属性のタイプは String です。
  - この属性は、そのアプリケーション固有情報の中に次の type タグを持っていることが必要です。

    ```
    type=xsinoNSlocation
    ```

この XML 文書を表すスキーマ文書の root エレメント・ビジネス・オブジェクト定義には、スキーマ・ロケーションに対応する次の属性が含まれます。

```
[Attribute]
Name=schemaLocation
Type=String
AppSpecificInfo=type=xsischemalocation;
```

注: 属性が schemaLocation または noNamespaceSchemaLocation XML 属性を表す場合、アプリケーション固有情報に type=attribute タグを設定する必要はありません。

XML ODA は、DoctypeorSchemaLocation ODA 構成プロパティーの値に基づいて、トップレベル・ビジネス・オブジェクトに schemaLocation 属性を作成するか、noNamespaceSchemaLocation 属性を作成するかを決定します。

スキーマ文書からのビジネス・オブジェクト定義の作成

スキーマ文書は XML 文書の内容を規定し、制約します。したがって、ビジネス・オブジェクト定義のために必要な情報を取得する上で、スキーマ文書は非常に有用です。スキーマ文書の構造情報をビジネス・オブジェクト定義に変換するため、XML Object Discovery Agent (ODA) を使用することができます。XML ODA については、89 ページの『ビジネス・オブジェクト定義を作成するための XML ODA の使用』を参照してください。

サポートされるスキーマ文書構造

XML ODA はスキーマのエレメント宣言をオブジェクト定義の属性にマップします。ビジネス・オブジェクト属性のタイプは、XML エレメント宣言で指定したタイプによって異なります。単純タイプは String Java タイプにマップされます。複合タイプはビジネス・オブジェクト定義にマップされます。

XML ODA は、次のスキーマ文書構造をサポートしています。

- 単純タイプ: 単純タイプは、アトミック (組み込みまたは制限を使用して派生される)、リスト、共用体のいずれかのタイプです。XML ODA は上記のすべてのタイプをビジネス・オブジェクト属性の String タイプにマップします。詳細については、81 ページの『複合タイプの XML エレメントの場合』を参照してください。
- エレメント・ワイルドカード any: XML ODA は、any ワイルドカードをビジネス・オブジェクト定義の単純属性にマップします。実行時に、ユーザーはデータに関する知識に基づいてこの属性を構成する必要があります (属性を構成するには、該当するアプリケーション固有情報を追加します)。
- anyAttribute エレメント: このエレメントにより、ユーザーは XML 文書を拡張し、スキーマに指定されていないエレメントも扱えるようになります。 XML ODA は、ユーザーにこのような属性の名前を指定するように求め、次にその属性を、ビジネス・オブジェクト定義の中の単純属性にマップします (209 ページの図 44を参照)。ODA は、anyAttribute ワイルドカードに対応するどの namespace 属性も processContents 属性も無視します。
- sequence グループ: このモデル・グループでは、子エレメントが特定の順序で並ぶことが必要です。 XML ODA は、sequence グループ内の子エレメントを、ビジネス・オブジェクト定義内の属性にマップします。これら属性のタイプは、XML の type 属性から決定されます。
- choice グループ: このモデル・グループは、1 つのインスタンスに出現する対応準拠エレメント宣言は 1 つのみでなければならないことを示します。 XML ODA は、choice グループ内の子エレメントを、ラッパー・ビジネス・オブジェクト定義内の属性にマップします。親ビジネス・オブジェクト定義は、ラッパー・ビジネス・オブジェクトをタイプとする複数カーディナリティーの単一の属性を格納しています。詳細については、67 ページの『スキーマ文書に基づくラッパー・ビジネス・オブジェクト定義』を参照してください。
- all グループ: XML ODA は、all グループ内の子エレメントを、ラッパー・ビジネス・オブジェクト定義内の属性にマップします。親ビジネス・オブジェクト定義は、ラッパー・ビジネス・オブジェクトをタイプとする複数カーディナリティーの単一の属性を格納しています。詳細については、67 ページの『スキーマ文書に基づくラッパー・ビジネス・オブジェクト定義』を参照してください。
- include エレメント: include の機能は、組み込まれる側のスキーマ文書に格納されているすべての定義および宣言を、組み込む側のスキーマ文書に投入することです。 XML ODA では、組み込まれるスキーマ文書の完全パスを指定することが必要です。この完全パスは、組み込まれるスキーマ文書がファイル・システムに実際に入るロケーションであり、その URI ではありません。スキーマ文書のロケーションを XML ODA プロパティー FileName により指定する場合には、組み込まれるスキーマ文書すべてがこのロケーションに存在していることが必要です。

  注: スキーマ文書を組み込む場合は、組み込まれる側のスキーマ文書では、組み込む (include する) 側のスキーマ文書と同じ名前のグローバル・エレメントを宣言しない でください。
- restriction エレメント: このエレメントにより、ユーザーは制限によって複合タイプを派生させることができます。つまり、複合タイプの属性や内容を除去ま

たは制限することによって複合タイプを制限できます。 XML ODA は、以下のコンテキストにおいて restriction エレメントをサポートします。

– 単純な内容の制限の場合、XML ODA は、単純な内容に対して定義されている制限、および派生型のファセットを含む制限のマッピングを実行しません。派生型に定義されている属性に対しては制限を実行します。

– 複合的な内容の制限の場合、XML ODA は次のように、基本複合タイプの属性およびコンテンツ・モデルの除去または制限をサポートします。

  - コンテンツ・モデルの制限の場合、ODA は、基本複合タイプおよび派生型の*両方* に対してビジネス・オブジェクト定義を作成します。ただし、XML ODA は、スキーマ構文が有効であり、制限について W3C Schema 規格ガイドラインに従っていることを前提とします。そのため XML ODA は、派生型が基本タイプの有効な制限であるかどうかを*検証しません*。

  - 属性の制限の場合、XML ODA は基本タイプのすべての属性が派生型に伝えられることを前提とします。そのためビジネス・オブジェクト定義には、(制限に含まれない) 基本タイプに定義されている属性に加え、制限に定義されているすべての XML 属性に対応する属性があります。

- `import` エレメント: このエレメントにより、XML 文書は他のネーム・スペースに属するコンポーネントを参照することができます。属性のアプリケーション固有情報に `elem_ns` タグまたは `attr_ns` タグを指定することによって、ビジネス・オブジェクト定義はインポートされたネーム・スペースを識別できます。詳細については、72 ページの『スキーマ・ネーム・スペース』を参照してください。

- 複合タイプの導出: XML ODA は、制限および拡張を使用して派生された複合タイプをサポートします。この構造の場合、XML ODA は派生コンテンツ・モデルに基づいて、派生複合タイプのビジネス・オブジェクト定義を作成します。

- 型置換: XML ODA プロパティー `TypeSubstitution` は、スキーマ内の xsi:type 属性に基づく型置換をサポートします。ユーザーが ODA 内のドロップダウンで `TypeSubstitution` を True に設定すると、XML ODA は、ASI ″typeSub=true″ を持つ追加ラッパー・オブジェクトと、基本型およびその派生型をこのラッパー・オブジェクトの子として表現するビジネス・オブジェクトを生成します。詳細については、68 ページの『スキーマ文書に基づく混合ビジネス・オブジェクト定義における型置換』を参照してください。

サポートされないスキーマ文書構造

XML ODA は、ほとんどのスキーマ文書を処理できます。ただし、次のスキーマ文書構造はサポートされていません。

- エレメント置換: XML データ・ハンドラーは実行時のエレメント置換をサポート*しません*。スキーマでのエレメント置換を指定するには、グローバル・エレメント宣言で `substitutionGroup` 属性を使用する必要があります。

- タイプおよびグループの再定義: `redefine` エレメントにより、タイプ (単純または複合) またはグループ (エレメントまたは属性) が、特定の動作のみサポートされるように再定義することができます。

- デフォルト属性と固定属性: XML ODA はエレメント宣言および属性宣言でデフォルト属性と固定属性を無視します。エレメント宣言または属性宣言でこのいず

れかの属性が指定されており、インスタンスにエレメントまたは属性が含まれていない 場合、XML ODA は対応するビジネス・オブジェクト属性にデータを取り込みません。

ビジネス・オブジェクト定義の作成

XML 文書は、その構造を定義するため、DTD またはスキーマ文書を持つことができます。XML 文書内のエレメントを表すビジネス・オブジェクト定義には、文書の構造に関する情報が設定されていることが必要です。XML データ・ハンドラーにより処理されるビジネス・オブジェクトを作成するため、XML データ・ハンドラーは、XML 文書を処理するための構造情報が設定されたビジネス・オブジェクト定義を見つける能力を備えていることが必要です。XML 文書用のビジネス・オブジェクト定義は、次のいずれかの方法で作成できます。

- 『ビジネス・オブジェクト定義を作成するための XML ODA の使用』
- 90 ページの『ビジネス・オブジェクト定義の手動での作成』

どちらの方法でも、Business Object Designer Express ツールを使用することが必要です。このセクションでは、XML 文書用ビジネス・オブジェクト定義を生成するために Business Object Designer Express を使用する方法の概要について説明します。Business Object Designer Express の詳細な説明については、「*ビジネス・オブジェクト開発ガイド*」を参照してください。

注: XML データ・ハンドラーを使用する一部のコネクターには、コンテンツ・ビジネス・オブジェクトを子として組み込むトップレベルのラッパー・ビジネス・オブジェクトが必要です。したがって、使用するコネクターが必要とするビジネス・オブジェクトの構造によっては、子として生成されたビジネス・オブジェクトを別のビジネス・オブジェクトに追加する必要がある場合があります。コネクターのビジネス・オブジェクトの構造についての詳細は、各コネクターの資料を参照してください。

ビジネス・オブジェクト定義を作成するための XML ODA の使用

XML Object Discovery Agent (ODA) は、XML 文書のためのビジネス・オブジェクト定義を、その DTD またはスキーマ文書に基づいて作成します。ODA は DTD またはスキーマ文書を検査して、XML 文書構造についての情報を取得します。次に、ODA は、ビジネス・インテグレーション・システムにロードできるファイルに、ビジネス・オブジェクト定義を書き込みます。

注: DTD もスキーマ文書も*持たない* XML 文書の場合には、この文書に対応するビジネス・オブジェクト定義を手動で作成することができます。詳細については、90 ページの『ビジネス・オブジェクト定義の手動での作成』を参照してください。

XML ODA ユーティリティーは XML データ・ハンドラーの要件に適合するビジネス・オブジェクト定義を作成します。ODA は必要な `ObjectEventId` 属性をすべてのビジネス・オブジェクト定義に追加します。また、リポジトリーのバージョン番号を指定すると、これをビジネス・オブジェクト定義の先頭に追加します。これはビジネス・オブジェクト定義を WebSphere Business Integration Express にインポートするのに必要です。これらのビジネス・オブジェクト定義を、さらに編集する必

要はありません。何らかの理由で編集が必要な場合には、211 ページの『ビジネス・オブジェクト定義の情報の変更』を参照してください。

XML ODA の使用方法については、197 ページの『XML ODA の使用』を参照してください。この付録では、XML ODA のインストールと構成方法について説明します。また、ビジネス・オブジェクト定義を作成するために、Business Object Designer Express 上で XML ODA を使用する方法についても説明します。 Business Object Designer Express の起動については、「*ビジネス・オブジェクト開発ガイド*」を参照してください。

ビジネス・オブジェクト定義の手動での作成

このセクションでは、手動でビジネス・オブジェクト定義を作成して XML 文書を表す方法を説明します。ビジネス・オブジェクト定義は、その属性およびアプリケーション固有情報を含めて、正しく定義する必要があります。

注: DTD もスキーマ文書も持たない XML 文書の場合には、この文書に対応するビジネス・オブジェクト定義を手動で作成することが必要です。DTD またはスキーマ文書が存在する場合には、ビジネス・オブジェクト定義の作成に XML ODA を使用することを IBM はお勧めします。

DTD またはスキーマ文書用の XML 文書フォーマットの記述により、XML ODA が作成する ビジネス・オブジェクト定義が規定されます。 34 ページの表 9 に、本書の中で、XML 文書のフォーマットを説明しているセクションを示します。各セクションには、該当するスキーマを記述するための対応するデータ・モデルが記載されています。これらのセクションで説明されているように、ビジネス・オブジェクト定義は、XML データ・ハンドラーの要件に適合しています。したがって、ビジネス・オブジェクト定義を手動で作成する必要がある場合には、該当するセクションの記載に従ってください。

次の手順では、*ElementTypeName* はビジネス・オブジェクト構造 (属性またはビジネス・オブジェクト) で表される XML エレメントのタイプです。XML 文書に基づいて、ビジネス・オブジェクトを定義するには、次の手順で行います。

1. トップレベル・ビジネス・オブジェクト定義を作成します。このビジネス・オブジェクト定義の名前は、XML 文書の最高位のエレメント (例えば、DTD またはスキーマ文書の名前) であり、*BOPrefix_TopLevelName* のフォーマットでなければなりません。

   注: 一部のコネクターでは、コンテンツ・ビジネス・オブジェクトを子として組み込むラッパー・ビジネス・オブジェクトが必要です。詳しくは、51 ページの『DTD に基づくラッパー・ビジネス・オブジェクト定義』を参照してください。
2. トップレベル・ビジネス・オブジェクト定義に、XML エレメントに対応する属性を作成します。

   DTD またはスキーマ文書に基づくトップレベル・ビジネス・オブジェクト定義には、次の属性が必要です。

   - `XMLDeclaration`: 属性のアプリケーション固有情報は `type=pi` です。

- DTD またはスキーマ文書の root エレメントを表す属性: この属性には単一カーディナリティーの子ビジネス・オブジェクトが含まれ、そのアプリケーション固有情報は elem_name タグを使用してエレメント名を指定します。

これらの必須属性の一般情報は、 38 ページの『ビジネス・オブジェクトの構造』を参照してください。さらに、本書では、DTD およびスキーマ文書に基づくトップレベル・ビジネス・オブジェクト定義の構造について、次の情報を提供します。

データ・モデル	詳細
文書タイプ定義 (DTD)	48 ページの『DTD のビジネス・オブジェクト構造』
スキーマ文書	62 ページの『スキーマ文書に必要なビジネス・オブジェクト定義』

3. トップレベル・ビジネス・オブジェクト定義の子オブジェクトである、root エレメント・ビジネス・オブジェクト定義を作成します。これには root XML エレメントの属性が含まれます。このビジネス・オブジェクト定義の名前は、XML 文書の root エレメントであり、*BOPrefix_TopLevelName_RootElementName* のフォーマットであることが必要です。
4. root エレメント・ビジネス・オブジェクト定義の中に、各下位エレメントに対応するビジネス・オブジェクト属性を作成します。次のことに留意してください。
   - ビジネス・オブジェクトの属性名は、XML エレメント (または属性) 名と同じである必要はありません。アプリケーション固有情報を使用してエレメント (または属性) 名を指定します。
   - XML 属性は、ビジネス・オブジェクト定義内の最初の属性でなければなりません。
   - タイプの判別
     - String タイプは、エレメント内容や関連する属性リスト宣言のない、カーディナリティー 1 のエレメントです。
     - ビジネス・オブジェクト・タイプは、カーディナリティー n の下位エレメント、あるいはエレメント内容または関連する属性仕様を含む下位エレメントです。
   - String タイプ属性、混合タイプ・エレメント、またはカーディナリティー n の選択リスト・エレメントに関しては、アプリケーション固有情報が必要です。

     このアプリケーション固有情報についての一般情報は、 42 ページの『アプリケーション固有情報』を参照してください。さらに、本書では、DTD およびスキーマ文書に基づくビジネス・オブジェクト属性のアプリケーション固有情報について、次の情報を提供します。

データ・モデル	詳細
文書タイプ定義 (DTD)	50 ページの『DTD の XML コンポーネントのアプリケーション固有情報』

データ・モデル	詳細
スキーマ文書	71 ページの『スキーマ文書内の XML コンポーネントのアプリケーション固有情報』

注: スキーマ定義を持つ XML 文書の場合、root エレメント・ビジネス・オブジェクト定義にスキーマ・ロケーションのための属性も必要である場合があります。詳細については、 62 ページの『スキーマ文書に必要なビジネス・オブジェクト定義』を参照してください。

5. すべての下位エレメントに関する子ビジネス・オブジェクト定義を作成します。上記の規則に従います。

ビジネス・オブジェクトの XML 文書への変換

ビジネス・オブジェクトを XML 文書に変換するために、XML データ・ハンドラーはビジネス・オブジェクト定義内の属性を順にループします。ビジネス・オブジェクトとその子に現れる属性の順序に基づいて XML を再帰的に生成します。

XML データ・ハンドラーは、ビジネス・オブジェクトを次の手順で処理して XML 文書にします。

1. データ・ハンドラーは XML データを格納する文書を作成します。
2. データ・ハンドラーは、トップレベル・ビジネス・オブジェクト定義内のアプリケーション固有情報を調べて、子メタオブジェクト (ビジネス・オブジェクト・レベルのアプリケーション固有情報の `cw_mo_label` タグ内に名前がリストされている子メタオブジェクト) の有無を判断します。データ・ハンドラーは、これらの属性を XML 文書に組み込みません。
3. データ・ハンドラーはビジネス・オブジェクト定義の残りの属性をループします。

   データ・ハンドラーは次の規則で、各属性の XML を生成します。

   - XML データ・ハンドラーは、単純な `String` タイプ属性ごとに XML 開始タグと終了タグを作成します。ただし、この属性のアプリケーション固有情報に `notag` が指定されている場合にはタグの作成はありません。開始タグを開いて、処理されるビジネス・オブジェクト属性が XML 属性を表さない場合、XML データ・ハンドラーは開始タグを閉じます (つまり、XML 文書に文字「>」を追加します。
   - 属性がビジネス・オブジェクトを表す場合、XML データ・ハンドラーは開始タグを開き、再帰呼び出しを行って子ビジネス・オブジェクト内の属性を検索し、子ビジネス・オブジェクトの属性に関する XML を生成して、エレメントの終了タグを生成します。XML データ・ハンドラーは、属性レベルの アプリケーション固有情報 `elem_name` をエレメント名として使用します。複数カーディナリティーの属性の場合、配列内の各ビジネス・オブジェクト・インスタンスについて、この処理を繰り返します。
   - 属性にアプリケーション固有情報 `xsiType` が含まれる場合、XML データ・ハンドラーは、`xsi:type=ValueofXsiType` という形式で現在のエレメント用の属性を書き出します。例えば、ビジネス・オブジェクト属性が

xsiType=ShirtType を指定する場合、対応する XML 属性は xsi:type="ShirtType" となります。型置換の詳細については、 68 ページの『スキーマ文書に基づく混合ビジネス・オブジェクト定義における型置換』を参照してください。

- XML マークアップを表す属性について、データ・ハンドラーは属性のアプリケーション固有情報を使用して、表 27 に示すように、XML を生成します。

表 27. XML マークアップを表す属性に関する XML 出力

XML エンティティー・ビジネス・オブジェクト属性が表す項目	XML 出力	例	アプリケーション固有情報
処理命令	<?*AttrValue*?>	<?xml version="1.0"?>	type=pi
DTD	<!*AttrValue*>	<!DOCTYPE CUSTOMER "customer.dtd">	type=doctype
エレメント	<*ElementName*>... </*ElementName*>	DTD に基づく XML 文書の場合: <CUSTOMER>... </CUSTOMER> スキーマ文書に基づくXML 文書の場合: <element name=CUSTOMER...> </element>	type=pcdata
XML 属性	AttrName= "*AttrValue*"	DTD に基づく XML 文書の場合: Seqno="1" スキーマ文書に基づくXML 文書の場合: <element name=CUSTOMER... Seqno="1"...> </element>	type=attribute
CDATA セクション	<![CDATA[*AttrValue*]]>	<![CDATA [<HTML>Text</HTML>]]>	type=cdata
コメント	<!--*CommentText* -->	<!--Customer information from source application A-->	type=comment
スキーマ・ロケーション (ターゲット・ネーム・スペース付き)	<*elementName* xmlns="*URI_path*" xmlns:xsi= "http://www.w3.org/ 2001/XMLSchema- instance" xsi:schemaLocation= "*URI_for_schema schema_location*" ...	85 ページの図 28 を参照	type= xsischemalocation
スキーマ・ロケーション (ターゲット・ネーム・スペースなし)	<*elementName* xmlns="*URI_path*" xmlns:xsi= "http://www.w3.org/ 2001/XMLSchema- instance" xsi:noNamespace SchemaLocation= "*schema_location*" ...	<order xmlns=″http://sampleDoc.org.ord″ xmlns:xsi=″http://www.w3.org/2001/ XMLSchema-instance″ xsi:noNamespaceSchemaLocation= ″order.xsd″> ... /<order>	type=xsinoNSlocation

- あるエレメントと関連する文字データ (DTD 内の PCDATA) を含む属性の場合、マークアップは生成されず、属性自体の値のみが文書に追加されます (アプリケーション固有情報に notag タグが存在する場合)。

- エレメントまたは属性の文字データを含む属性の場合、データ・ハンドラーはすべての特殊文字を、表 28 に示すように、適切なエスケープ・シーケンスで置換します (アプリケーション固有情報に `escape=true` タグが存在する場合)。

表 28. 特殊文字と XML 表現

特殊文字	XML エスケープ・シーケンス
アンパーサンド (&)	`&amp`
より小 (<)	`&lt`
より大 (>)	`&gt`
単一引用符 (‘)	`&apos`
二重引用符 (“)	`&quot`

注: 属性が、単一引用符、二重引用符、あるいは「&」、「<」、または「>」文字を含む XML エレメントを表す場合、属性にはエスケープ処理が必要です。属性は、そのアプリケーション固有情報内に値 `escape=true` が設定されていないと、エスケープ処理されません。このアプリケーション固有情報は、他のすべてのテキストの末尾に配置する必要があります。

- 以下のいずれかの条件に当てはまる場合、属性はスキップされます。
  - 属性に値 `CxIgnore` がある。
  - 属性名が、ビジネス・オブジェクト定義のアプリケーション固有情報の `cw_mo_` タグにリストされている。

  これらの属性の XML は生成されません。
- 値が `CxBlank` の単純 `String` 属性は、空のタグとして XML 文書に組み込まれます。 DTD を基にした XML 文書の場合、空の二重引用符 (″ ″) が `CxBlank` と同等の `PCDATA` として使用されます。ただし、データ・ハンドラーは、複合属性 (タイプがビジネス・オブジェクトである属性) には、値 `CxBlank` が使用されないことを前提としています。

4. データ・ハンドラーが変換を完了すると、XML 文書を呼び出し元に戻します。

注: ビジネス・オブジェクト内の動詞情報は、XML 文書への変換で失われます。動詞の保持の詳細については、43 ページの『ビジネス・オブジェクト動詞』を参照してください。

XML 文書のビジネス・オブジェクトへの変換

このセクションでは、XML データ・ハンドラーが XML 文書をビジネス・オブジェクトに変換する方法に関して次のことを説明します。

- 『XML 文書の要件』
- 95 ページの『直列化データの処理』

XML 文書の要件

XML データ・ハンドラーは、XML 文書について、次のことを前提としています。

- XML 文書は適切な形式になっている。

- XML 文書は SAX パーサー要件に対応している。XML データ・ハンドラーで使用されるデフォルトのパーサーでは、XML 文書に XML 宣言が組み込まれていることが必要です。
- XML 文書の構造は、その対応するビジネス・オブジェクトの構造と一致しなければならない。さらに、文書内のエレメントの順序は、ビジネス・オブジェクト内の属性の順序と一致している必要があります。

直列化データの処理

XML 文書をビジネス・オブジェクトに変換する場合、XML データ・ハンドラーは、ビジネス・オブジェクトが XML 文書の構造に従い、37 ページの『ビジネス・オブジェクト定義の要件』で説明されているビジネス・オブジェクト定義の要件に適合することを前提としています。ビジネス・オブジェクトに所定のエレメント名の属性が存在しない場合、XML データ・ハンドラーはエラーを返します。

XML 文書をビジネス・オブジェクトに変換するために、XML データ・ハンドラーは次の手順を実行します。

1. 呼び出し元コネクターが変換メソッドにビジネス・オブジェクトを引き渡すと、データ・ハンドラーはこのビジネス・オブジェクトを使用して処理を続行します。呼び出し元がビジネス・オブジェクトを引き渡さない 場合、データ・ハンドラーはビジネス・オブジェクト名を判別して、XML 文書内のデータを格納するビジネス・オブジェクトを作成します。

   ビジネス・オブジェクト名を判別するために、データ・ハンドラーはネーム・ハンドラーを起動します。デフォルトのネーム・ハンドラーは、`BOPrefix` メタオブジェクト属性、アンダースコアー、および root エレメントの値を組み合わせて、トップレベル・ビジネス・オブジェクト名を作成します。例えば、XML 文書に `<!DOCTYPE Customer>` が含まれ、`BOPrefix` 属性が `MyApp` であれば、名前は `MyApp_Customer` となります。別の動作を構成するカスタム・ネーム・ハンドラーを設定することができます。
2. データ・ハンドラーは `Parser` メタオブジェクト属性の値を検索して、XML 文書の構文解析に使用する SAX パーサーを判別します。 XML データ・ハンドラーがどの SAX パーサーを使用するかについては、35 ページの『SAX パーサー』を参照してください。データ・ハンドラーがパーサーの名前を判別する場合、パーサーをインスタンス化します。
3. データ・ハンドラーは、イベント・ハンドラー (DTD に基づく XML 文書の場合は、エンティティー・リゾルバー) をパーサーに登録します。イベント・ハンドラーは、各 XML エレメントと属性を処理するコールバック・メソッドです。

   注: エンティティー・リゾルバーは、DTD 文書内の外部エンティティー参照を処理します。データ・ハンドラーが有効なクラス名を持つ `EntityResolver` オプションを見つけられないない場合は、`com.crossworlds.DataHandlers.xml.DefaultEntityResolver` を使用します。このエンティティー・リゾルバーは、外部参照をすべて無視します。
4. データ・ハンドラーは、パーサーを起動して XML 文書を構文解析します。
   - データ・ハンドラーは、子メタオブジェクト (ビジネス・オブジェクトのアプリケーション固有情報の `cw_mo_`*`label`* タグに名前がリストされている子メタ

オブジェクト) の有無を判別します。データ・ハンドラーは、ビジネス・オブジェクトのこれらの属性を設定する処理を実行しません。

- エレメントのタイプに応じて、パーサーのイベント・ハンドラーは属性プロパティーのビジネス・オブジェクト定義を照会し、それに従ってエレメント・データを処理します。実行が停止するのは、パーサーから致命的エラーが返されるか、またはビジネス・オブジェクト定義内で XML データ内のエレメントを検索できない場合のみです。

5. ビジネス・オブジェクトが完了すると、データ・ハンドラーはビジネス・オブジェクトを呼び出し元に返します。

注: XML 文書内で検索されたすべてのエレメントと属性について、データ・ハンドラーはビジネス・オブジェクト内で属性が検出されることを予期します。ビジネス・オブジェクト定義に所定のエレメントまたは属性名に関する属性がない場合、データ・ハンドラーはエラーを返します。この規則の例外は FIXED タイプの属性です。この属性は、ビジネス・オブジェクト定義では必要ありません。FIXED 属性がビジネス・オブジェクト定義に存在しない場合、FIXED 属性が XML 文書内で検出されても実行は停止しません。

XML データ・ハンドラーのカスタマイズ

XML データ・ハンドラーのカスタマイズは以下により実行できます。

- 『カスタム XML ネーム・ハンドラーの作成』
- 98 ページの『カスタム・エンティティー・リゾルバーの作成』

カスタム XML ネーム・ハンドラーの作成

XML データ・ハンドラーはネーム・ハンドラーを呼び出して、XML メッセージからビジネス・オブジェクトの名前を抽出します。 XML データ・ハンドラーに組み込まれているデフォルトのネーム・ハンドラーは、次のタグを探します。

```
<!DOCTYPE Name>
```

このタグと BOPrefix メタオブジェクト属性から、データ・ハンドラーはビジネス・オブジェクトの名前を生成します。 XML データ・ハンドラーは、データ・ハンドラー・メタオブジェクトに格納されている NameHandlerClass 属性の値を使用して、起動するネーム・ハンドラーを判別します。ネーム・ハンドラーを別の方法で機能させる必要がある場合には、次を実行する必要があります。

1. NameHandler クラスを拡張して、カスタム・ネーム・ハンドラーを作成する。
2. XML データ・ハンドラーのメタオブジェクト内の NameHandlerClass 属性のデフォルト値を更新して、カスタム・クラスを使用するように XML データ・ハンドラーを構成する。

次のサンプル・コードは、DataHandler クラスを拡張して、XML データ・ハンドラーのカスタム・データ・ハンドラー CustomDataHandler を作成します。

```
package com.crossworlds.DataHandlers.xml;

// DataHandler Dependencies
import com.crossworlds.DataHandlers.
     Exceptions.MalformedDataException;
import com.crossworlds.DataHandlers.NameHandler;
import com.crossworlds.DataHandlers.DataHandler;
```

```
// Java classes
import java.io.*;
import java.lang.Exception;

/************************************************************
 * CustomNameHandler class. This class extends the Namehandler
 * class and implements method:
 *      getBOName( Reader serializedData, String subType )
 * The method getBOName contains the logic to extract the BOName
************************************************************/
public class CustomNameHandler extends NameHandler
{

         /**
          * This method generates the business object name from
          * the data extracted from the 'serializedData' arg.
          * In this case, it is up to the caller to create
          * the BOName.
          */

         public String  getBOName( Reader serializedData,
                                   String subType )
           throws MalformedDataException
         {
           // The NameHandler uses DataHandler tracing. If the
           // DataHandler is not set, the NameHandler won't run.
           if (dh == null)
            return null;
           // Log a message
           dh.traceWrite(
           "Entering CustomNameHandler.getBOName for subtype '"
             + subType + "'.", 4);

           // This method parses the XML document and extracts the
           // business object name from the following tag in
           // the XML doc:
           //        <cml title=
           // For example, in:
           //        <cml title="cholestrol" id="cml_cholesterol">
           // the business object name is 'cholestrol'.

           // Log a message
           dh.traceWrite(
             "Name resolution will be done using <cml title= ",4);
           String name = null;

           try
           {
             // Read line of data from the Reader object
             LineNumberReader lineReader =
               new LineNumberReader( serializedData );
             serializedData.mark( 1000 );
             String line = lineReader.readLine();
             while ( line != null )
             {
               // search for <cml title= in the line
               int start = line.indexOf("<cml title=");
               if ( start != -1 )
               {
                 start += 12;
                 // search for the ending quotes for the tile tag
                 int end = line.indexOf('\"', start);

                 // extract name from line
                 name = line.substring(start, end);
                 break;
```

```
            }
            line = lineReader.readLine();
          }

          if ( name == null || name.length() == 0 )
            throw new MalformedDataException(
             "Error: can't determine the BusinessObject Name.");

        }

        catch(Exception e)
        {
          throw new MalformedDataException( e.getMessage() );
        }
        serializedData.reset();
        return name;
      }
}
```

カスタム・エンティティー・リゾルバーの作成

XML データ・ハンドラーで使用される SAX パーサーはエンティティー・リゾルバーを呼び出して、XML 文書内の外部エンティティー (参照される DTD) を検索します。XML データ・ハンドラーに組み込まれるエンティティー・リゾルバーは、外部参照を無視するか、またはローカル・ファイル・システム上で検索できます。別の方法を指定して外部エンティティーを検出する必要がある場合は、カスタム・エンティティー・リゾルバー・クラスを作成する必要があります。

XML データ・ハンドラーは、XML データ・ハンドラー・メタオブジェクトに格納されている `EntityResolver` 属性の値を使用して、起動するエンティティー・リゾルバーを判別します。

第 4 章 Request-Response データ・ハンドラー

Request-Response データ・ハンドラーは、要求ビジネス・オブジェクトと応答ビジネス・オブジェクトに異なるデータ・フォーマットが必要なシナリオを処理します。Request-Response データ・ハンドラーにより、これらの 2 つのフォーマットを異なるものにすることができます。この章では、Request-Response データ・ハンドラーが情報を処理する方法と、ビジネス・オブジェクト定義が Request-Response データ・ハンドラーによって処理されるように定義する方法を説明します。また、Request-Response データ・ハンドラーの構成方法についても説明します。

この章を構成するセクションは次のとおりです。

- 『概要』
- 107 ページの『ビジネス・オブジェクト定義の要件』
- 111 ページの『Request-Response データ・ハンドラーの構成』
- 115 ページの『要求データ・ハンドラーによるビジネス・オブジェクトの変換』
- 116 ページの『応答データ・ハンドラーによるビジネス・オブジェクトの変換』
- 117 ページの『エラー処理』
- 117 ページの『Request-Response データ・ハンドラーのカスタマイズ』

注: Request-Response データ・ハンドラーは、`CwDataHandler.jar` ファイルに含まれる基本データ・ハンドラーの 1 つです。このデータ・ハンドラーのインストール方法およびそのソース・コードの保管先の詳細については、25 ページの『第 2 章 データ・ハンドラーのインストールと構成』を参照してください。

概要

Request-Response データ・ハンドラーは、異なるフォーマットの要求データおよび応答データのサポートを提供することが主な役割であるデータ変換モジュールです。つまり、呼び出し側コンテキスト (アダプターや Server Access Interface など) が次の 2 つの異なるデータ・ハンドラーを呼び出せるようにします。

- *要求データ・ハンドラー* は、要求を*開始* する WebSphere Business Integration システム・コンポーネントのデータ変換を行います。
- *応答データ・ハンドラー* は、要求に*応答* する WebSphere Business Integration システム・コンポーネントのデータ変換を行います。

他のデータ・ハンドラーを使用すると、呼び出し側コンテキストは、アプリケーションまたはアクセス・クライアントとデータを送受信するときにデータが同じフォーマットであると想定します。したがって、呼び出し側コンテキストは、単一のデータ・ハンドラーを呼び出して要求および応答ビジネス・オブジェクトの変換を行うよう構成されます。

ただし、XML を入力として受け入れ、カスタム・フォーマット化された文書を出力として戻すレガシー・アプリケーションを使用する場合があります。既存のデータ・ハンドラーでは、この状態に容易に対処できません。しかし、このレガシー・

アプリケーションと通信するアダプターを構成して、Request-Response データ・ハンドラーを呼び出すことができます。このデータ・ハンドラーは、入力と出力に以下の異なるデータ・ハンドラーを呼び出すよう構成できます。

- 要求ビジネス・オブジェクトを処理する IBM WebSphere Business Integration Data Handler for XML (XML データ・ハンドラー)
- 応答ビジネス・オブジェクトを処理するカスタム・データ・ハンドラー

要求処理では、統合ブローカーがこのアダプターに要求を送信するとき、アダプターは Request-Response データ・ハンドラーを呼び出して、要求ビジネス・オブジェクトを送信します。Request-Response データ・ハンドラーはその構成を検査して、このビジネス・オブジェクトを XML 文書に変換するために XML データ・ハンドラーを呼び出す必要があるかどうかを判断します。このビジネス・オブジェクトが変換されると、Request-Response データ・ハンドラーは XML 文書をアダプターに戻し、アダプターはそれをレガシー・アプリケーションに発送します。

図 29 に、Request-Response データ・ハンドラーによって実行されるビジネス・オブジェクトからストリングへの変換の例を示します。

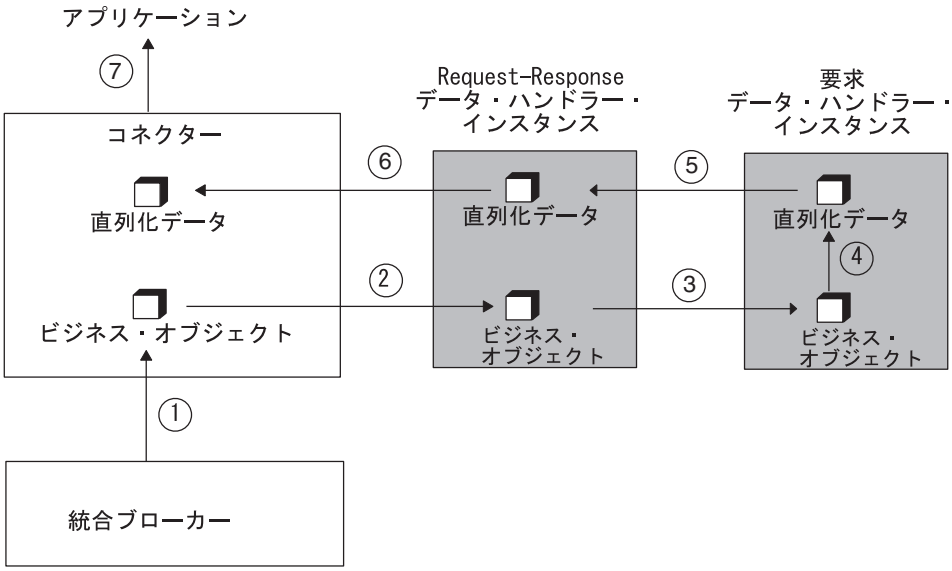

図 29. Request-Response データ・ハンドラーによるビジネス・オブジェクトからストリングへの変換

その後、アダプターがレガシー・アプリケーションから応答を受け取る場合があります。この応答は、レガシー・アプリケーションのカスタム・フォーマットになります。アダプターは再び Request-Response データ・ハンドラーを呼び出し、応答データを送信します。Request-Response データ・ハンドラーはその構成を検査して、この応答データをビジネス・オブジェクトに変換するためにカスタム・データ・ハンドラーを呼び出す必要があるかどうかを判断します。このデータが変換されると、Request-Response データ・ハンドラーはビジネス・オブジェクトをアダプターに戻し、アダプターはそれを統合ブローカーに発送します。

また、Request-Response データ・ハンドラーは、ICS 統合ブローカーが 1 つのビジネス・オブジェクト・タイプをコラボレーション・ポートに通知し、1 つ以上の異なるビジネス・オブジェクトを受け取ることも可能にします。例えば、アクセス・

クライアントがカスタマー・オブジェクトをコラボレーションに送信し、そのカスタマーの保留注文オブジェクトの配列を受け取ることができます。

Request-Response データ・ハンドラーは、text/requestresponse MIME タイプの直列化データをサポートします。その直列化データは、テキスト・データであっても、バイナリー・データであってもかまいません。ただし、デフォルトのトップレベルのメタオブジェクト (MO_DataHandler_Default または MO_Server_DataHandler) は、text/requestresponse MIME タイプをサポートしません。したがって、アクセス・クライアントまたはコネクターが Request-Response データ・ハンドラーを呼び出すことができるようにするには、text/requestresponse MIME タイプをサポートするように適切なトップレベルのメタオブジェクトを変更する必要があります。詳細については、 112 ページの『トップレベルのメタオブジェクトの構成』を参照してください。

この概要のセクションでは、Request-Response データ・ハンドラーの次の内容について説明します。

- 『Request-Response データ・ハンドラー・コンポーネント』
- 102 ページの『Request-Response データ・ハンドラーの機能』
- 106 ページの『要求および応答ビジネス・オブジェクトの処理』

Request-Response データ・ハンドラー・コンポーネント

データ・ハンドラーは、以下のどちらかの方法で直列化データを受け取ることができます。

- 直列化データおよび 空のビジネス・オブジェクトを受け取る。データ・ハンドラーは、このビジネス・オブジェクトに直列化データを取り込みます。
- 直列化データのみ を受け取る。データ・ハンドラーは、まずビジネス・オブジェクトを作成しなければ、直列化データを取り込むことができません。

Request-Response データ・ハンドラーは、ネーム・ハンドラーを使用して、作成するトップレベル・ビジネス・オブジェクトの名前を作成します。Request-Response データ・ハンドラーのコンポーネントとその関係を図 30 に示します。

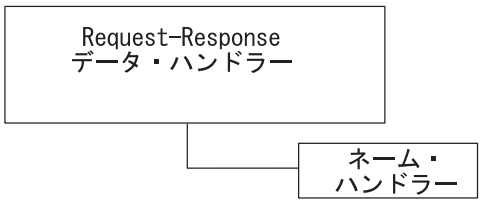

図 30. Request-Response データ・ハンドラー・コンポーネント

データ・ハンドラーは、Request-Response データ・ハンドラーの子メタオブジェクト内の NameHandlerClass 属性の値に基づいてネーム・ハンドラーのインスタンスを呼び出します。

- クラス名を指定しない場合 (NameHandlerClass 属性が空である場合)、データ・ハンドラーはデフォルトのネーム・ハンドラーを使用します。デフォルトのネーム・ハンドラーは、要求用に構成されたデータ・ハンドラーによって生成されたトップレベル・ビジネス・オブジェクト名の先頭に BOPrefix 属性値を付加します。データ・ハンドラーはエラーをログに記録し、例外を生成します。

例えば、ユーザーが REQUESTTEST のデフォルトの BOPrefix を指定し、要求データ・ハンドラーがビジネス・オブジェクト Customer を生成する場合、Request-Response データ・ハンドラーは REQUESTTEST_Customer というトップレベルのオブジェクトを作成し、その子オブジェクトのいずれかを Customer オブジェクトに取り込みます。

- NameHandlerClass 属性にクラス名が用意されている場合、Request-Response データ・ハンドラーはこのネーム・ハンドラーを使用してビジネス・オブジェクト名を決定します。

  カスタム・ネーム・ハンドラーを指定するには、NameHandlerClass をカスタム・ネーム・ハンドラー・クラスの名前に設定してください。カスタム・ネーム・ハンドラーの作成方法については、117 ページの『Request-Response データ・ハンドラーのカスタマイズ』を参照してください。

この製品で提供されるバージョンのメタオブジェクトでは、NameHandlerClass 属性は空です。このため、Request-Response ネーム・ハンドラーはデフォルトのネーム・ハンドラーを使用します。

Request-Response データ・ハンドラーの機能

Request-Response データ・ハンドラーは、次のどちらの場合にも役立ちます。

- 『イベント処理のサポート』
- 105 ページの『要求処理のサポート』

イベント処理のサポート

イベント処理には、アプリケーション・ビジネス・エンティティーへの変更を示すイベントの発生を統合ブローカーに通知することが含まれます。イベント通知では、データ・ハンドラーの呼び出し側コンテキストがデータ・ハンドラーを呼び出します。このデータ・ハンドラーは、直列化データをビジネス・オブジェクトに変換します (ビジネス・オブジェクトは統合ブローカーに送信されます)。このストリングからビジネス・オブジェクトへの変換は、要求データ・ハンドラーが実行します。これは、このデータ・ハンドラーが要求 (入力) フォーマットからビジネス・オブジェクトへの変換を処理するためです。

イベント処理は、同期と非同期のいずれかです。ただし、非同期イベント処理ではアダプター (特にアダプターのコネクター・コンポーネント) は統合ブローカーからの応答を待機しないため、以下のようになります。

- 同期イベント処理が発生するのは、アクセス・クライアントが IBM WebSphere InterChange Server (特にこのサーバー内のコラボレーション) にイベントが発生したことを通知する場合です。アクセス・クライアントは、ICS からの応答を待機せず、ICS も応答を待機しません。この場合、(ICS 内の) Server Access Interface は必要なデータ・ハンドラーを呼び出します。
- 非同期イベント処理が発生するのは、アダプター (特にアダプターのコネクター・コンポーネント) がアプリケーションまたはテクノロジーからイベントを受信し、そのイベントを (ビジネス・オブジェクトの形式で) 統合ブローカーに送信して、何かが発生したことを通知する場合です。ただし、コネクターは統合ブローカーからの応答を待機しません。このため、Request-Response データ・ハンドラーは非同期イベント処理の場合には有用ではありません。

注: データ・ハンドラーの呼び出し側コンテキストの詳細については、 7 ページの『データ・ハンドラーの呼び出し用のコンテキスト』を参照してください。

例えば、次のステップは同期イベント処理を示しています。ここでは、Request-Response データ・ハンドラーによって (ICS 内の) Server Access Interface が、あるフォーマット (要求フォーマット) でデータを ICS に送信し、異なる形式のデータ (応答フォーマット) を受信する方法について説明します。これによってアクセス・クライアントは、カスタマー XML 文書を送信し、そのカスタマーの保留注文が含まれる XML 文書を受信するというシナリオを実行できます。

1. アクセス・クライアントが、ICS 内のコラボレーションによって実行されるイベントを (要求フォーマットのデータとして) 送信します。
2. ICS が Request-Response データ・ハンドラーの新規インスタンスを作成し、それを要求フォーマットのデータに渡します。
3. Request-Response データ・ハンドラーが構成済みの要求データ・ハンドラーを呼び出し、その要求データ・ハンドラーが要求フォーマットのデータをビジネス・オブジェクトに変換します。

   Request-Response データ・ハンドラーは `getBO()` メソッドを使用して、受信した直列化データを処理します。次のいずれかの条件が真の場合、Request-Response データ・ハンドラーは直列化データ上で要求データ・ハンドラーを呼び出します。

   - `getBO()` メソッドが引き数として直列化データのみ を受信する場合、メソッドはこのデータを新規要求と見なします。
   - `getBO()` メソッドが引き数としてトップレベル・ビジネス・オブジェクトおよび 直列化データを受信する場合、メソッドはトップレベル・ビジネス・オブジェクトの子ビジネス・オブジェクトを検査して、次のアクションを決定します。

     すべての子ビジネス・オブジェクトに CxIgnore 値 (子ビジネス・オブジェクトが空 であることを示します) が含まれる場合、データ・ハンドラーは直列化データが新規要求を表すと見なします。

     注: トップレベル・ビジネス・オブジェクト内の子ビジネス・オブジェクトを取り込む場合、データ・ハンドラーは、この子オブジェクトが元の要求を表し、その結果として直列化データが応答を表すと見なします。したがって、データ・ハンドラーは応答データ・ハンドラーのインスタンスを作成して、データを処理します。詳細については、 105 ページの『要求処理のサポート』を参照してください。

   どちらの場合も、Request-Response データ・ハンドラーは要求データ・ハンドラーのインスタンスを作成し、要求データ・ハンドラーを使用して要求フォーマットのデータを要求ビジネス・オブジェクトに変換します。Request-Response データ・ハンドラーは、要求データ・ハンドラーに要求フォーマットのデータを渡します。要求データ・ハンドラーは、対応する要求ビジネス・オブジェクトを戻します。
4. Request-Response データ・ハンドラーは、次のようにして要求ビジネス・オブジェクトをトップレベル・ビジネス・オブジェクトの子として追加します。

- データ・ハンドラーの getBO() メソッドがトップレベル・ビジネス・オブジェクトを受信していない 場合、新規のトップレベル・ビジネス・オブジェクトを作成し、それに要求ビジネス・オブジェクトを追加する必要があります。この新規ビジネス・オブジェクトに名前を付けるために、データ・ハンドラーはネーム・ハンドラーを呼び出します。詳細については、 101 ページの『Request-Response データ・ハンドラー・コンポーネント』を参照してください。
- データ・ハンドラーの getBO() メソッドがトップレベル・ビジネス・オブジェクトを受信した 場合、それに要求ビジネス・オブジェクトを追加します。

5. Request-Response データ・ハンドラーは、トップレベル・ビジネス・オブジェクトを (アクセス・クライアントが指定した) コラボレーションに同期的に戻します。
6. コラボレーションはトップレベル・ビジネス・オブジェクトを受け取り、(要求ビジネス・オブジェクトが含まれる) 子オブジェクトを取り出し、なんらかのビジネス・プロセスを実行します。
7. コラボレーションは新規の応答ビジネス・オブジェクトを作成し、それをトップレベル・ビジネス・オブジェクトに追加してから、正常に戻ります。

   このトップレベル・ビジネス・オブジェクトは、コラボレーションが Request-Response データ・ハンドラーから受け取ったビジネス・オブジェクトです。コラボレーションがこのビジネス・オブジェクトを更新すると、ビジネス・オブジェクトには元の要求ビジネス・オブジェクトと新たに作成した応答ビジネス・オブジェクトの両方が含まれます。
8. ICS が変更されたトップレベル・ビジネス・オブジェクトを Request-Response データ・ハンドラーに渡します。
9. Request-Response データ・ハンドラーは構成済みの応答データ・ハンドラーを呼び出し、応答ビジネス・オブジェクトを応答フォーマットのデータに変換します。

   Request-Response データ・ハンドラーは getStringFromBO() メソッドを使用して、受信したトップレベル・ビジネス・オブジェクトを処理します。トップレベル・ビジネス・オブジェクト内の複数の子ビジネス・オブジェクトを取り込む場合、データ・ハンドラーは、トップレベル・ビジネス・オブジェクトに元の要求ビジネス・オブジェクトとその応答ビジネス・オブジェクトの*両方* が含まれ、その結果として応答ビジネス・オブジェクトを変換する必要があると見なします。したがって、データ・ハンドラーは応答データ・ハンドラーのインスタンスを作成して、トップレベル・ビジネス・オブジェクト内で最後に定義された子ビジネス・オブジェクトを応答ビジネス・オブジェクトとして処理します。

   注: getStringFromBO() が受信したトップレベル・ビジネス・オブジェクト内の子ビジネス・オブジェクトを 1 つだけ取り込む場合、データ・ハンドラーは、この子オブジェクトが新規の要求を表すと見なします。したがって、データ・ハンドラーは要求データ・ハンドラーのインスタンスを作成して、要求ビジネス・オブジェクトを (要求フォーマットの) 直列化データに変換します。詳細については、 105 ページの『要求処理のサポート』を参照してください。

10. Request-Response データ・ハンドラーが応答フォーマットのデータを呼び出し元 (ICS 内の Server Access Framework) に戻します。
11. ICS は応答フォーマットのデータ (応答ビジネス・オブジェクトに基づきます) をアクセス・クライアントに戻します。

コネクターも、executeCollaboration() メソッドを使用して同期イベント処理を実行できます。ただし、一般的に、コネクターはポーリングなどのイベント検出機構を使用して非同期イベント処理を実行します。

要求処理のサポート

要求処理には、統合ブローカーから要求を受け取り、アプリケーション・ビジネス・エンティティー内で適切な変更を開始することが含まれます。アクセス・クライアント (同期) またはコネクター (非同期) によって開始されるイベント処理とは異なり、要求処理は統合ブローカーによって開始され、コネクターのみと通信します (アクセス・クライアントとは通信しません)。

要求処理では、データ・ハンドラーの呼び出し側コンテキストがデータ・ハンドラーを呼び出して、直列化データをビジネス・オブジェクトに変換します (ビジネス・オブジェクトは統合ブローカーに送信されます)。これはストリングからビジネス・オブジェクトへの変換です。

例えば、次のステップは IBM WebSphere InterChange Server 統合ブローカーおよびテクノロジー・アダプターにおける要求処理を示しています。このテクノロジー・アダプターは、Request-Response データ・ハンドラーを使用して要求データおよび応答データを処理するよう構成されています。この例では、要求データのフォーマットと応答データのフォーマットは異なります。

1. コラボレーションがトップレベル・ビジネス・オブジェクトの新規インスタンスを作成し、要求ビジネス・オブジェクトを子として追加します。
2. コラボレーションは要求を (トップレベル・ビジネス・オブジェクトの形式で) テクノロジー・コネクターに送信し、コネクターはそのトップレベル・ビジネス・オブジェクトを Request-Response データ・ハンドラーに渡します。
3. Request-Response データ・ハンドラーは構成済みの要求データ・ハンドラーを呼び出し、要求ビジネス・オブジェクトを要求フォーマットのデータに変換します。

   Request-Response データ・ハンドラーは getStringFromBO() メソッドを使用して、受信したトップレベル・ビジネス・オブジェクトを処理します。トップレベル・ビジネス・オブジェクト内の子ビジネス・オブジェクトを 1 つだけ取り込む場合、Request-Response データ・ハンドラーは、この子オブジェクトが新規の要求を表すと見なします。したがって、データ・ハンドラーは要求データ・ハンドラーのインスタンスを作成して、要求ビジネス・オブジェクトを (要求フォーマットの) 直列化データに変換します。

   注: getStringFromBO() が受け取ったトップレベル・ビジネス・オブジェクト内の複数の子ビジネス・オブジェクトを取り込む場合、データ・ハンドラーは、トップレベル・ビジネス・オブジェクトに元の要求ビジネス・オブジェクトとその応答ビジネス・オブジェクトの*両方* が含まれ、その結果として応答ビジネス・オブジェクトを変換する必要があると見なします。したがって、データ・ハンドラーは応答データ・ハンドラーのインスタンス

を作成して、トップレベル・ビジネス・オブジェクト内で最後に定義された子ビジネス・オブジェクトを応答ビジネス・オブジェクトとして処理します。詳細については、102 ページの『イベント処理のサポート』を参照してください。

4. テクノロジー・コネクターが要求フォーマットのデータをアプリケーションに送信します。
5. アプリケーションはいくつかのタスクを実行し、応答フォーマットのデータをテクノロジー・コネクターに戻します。
6. テクノロジー・コネクターは、元のトップレベル・ビジネス・オブジェクトと応答フォーマットのデータの両方を Request-Response データ・ハンドラーに渡します。
7. Request-Response データ・ハンドラーは構成済みの応答データ・ハンドラーを呼び出し、その応答データ・ハンドラーが応答フォーマットのデータをビジネス・オブジェクトに変換します。

   Request-Response データ・ハンドラーは `getBO()` メソッドを使用して、受信したトップレベル・ビジネス・オブジェクトと直列化データを処理します。トップレベル・ビジネス・オブジェクト内の子ビジネス・オブジェクトを取り込む場合、Request-Response データ・ハンドラーは、この子オブジェクトが元の要求を表し、その結果として、受信した直列化データが応答フォーマットであると見なします。したがって、データ・ハンドラーは応答データ・ハンドラーのインスタンスを作成して、直列化データを応答ビジネス・オブジェクトに変換します。

   注: `getBO()` が受信したトップレベル・ビジネス・オブジェクト内のすべての子ビジネス・オブジェクトに CxIgnore 値 (子ビジネス・オブジェクトが空であることを示します) が含まれる場合、データ・ハンドラーは直列化データが新規要求を表すと見なし、要求データ・ハンドラーのインスタンスを作成してデータを処理します。詳細については、102 ページの『イベント処理のサポート』を参照してください。
8. Request-Response データ・ハンドラーは、応答ビジネス・オブジェクトをトップレベル・ビジネス・オブジェクトの子として追加してから、このトップレベル・ビジネス・オブジェクトを呼び出し元 (テクノロジー・コネクター) に戻します。
9. テクノロジー・コネクターは更新されたトップレベル・ビジネス・オブジェクトをコラボレーションに戻します。
10. コラボレーションはトップレベル・ビジネス・オブジェクトを受け取り、その応答ビジネス・オブジェクトの内容をビジネス・プロセスに取り込みます。

要求および応答ビジネス・オブジェクトの処理

Request-Response データ・ハンドラーを使用して、要求ビジネス・オブジェクトを適切な要求フォーマットに変換したり、応答フォーマットのデータを応答ビジネス・オブジェクトに変換するには、表 29 に示すステップを実行する必要があります。

表 29. Request-Response データ・ハンドラーの使用

ステップ	詳細
1. ビジネス・オブジェクトの構造を記述しているビジネス・オブジェクト定義が存在し、実行中の Request-Response データ・ハンドラー (およびそのコンポーネントのデータ・ハンドラー) から使用できること。	『ビジネス・オブジェクト定義の要件』
2. Request-Response データ・ハンドラーが運用環境向けに構成されていること。	111 ページの『Request-Response データ・ハンドラーの構成』
3. 適切なデータ操作を実行するために、Request-Response データ・ハンドラーが呼び出し側コンテキスト (コネクターまたはアクセス・クライアント) から呼び出されること。	
a) データ操作: この要求を開始するコンポーネントから要求を受け取り、それを適切なフォーマットに変換する。	115 ページの『要求データ・ハンドラーによるビジネス・オブジェクトの変換』
b) データ操作: 要求に応答するコンポーネントから応答を受け取り、それを適切なフォーマットに変換する。	116 ページの『応答データ・ハンドラーによるビジネス・オブジェクトの変換』

ビジネス・オブジェクト定義の要件

Request-Response データ・ハンドラーを使用するには、データ・ハンドラーに必要な構造を提供するようにビジネス・オブジェクト定義を作成または修正する必要があります。ただし、他のデータ・ハンドラーとは異なり、メタデータを持たせるようにビジネス・オブジェクト定義を変更する必要はありません。このセクションでは、Request-Response データ・ハンドラーと連動するビジネス・オブジェクト定義の作成に必要な情報を示します。特に、以下の情報を取り上げます。

- 『要求/応答ビジネス・オブジェクト構造の理解』
- 110 ページの『Request-Response データ・ハンドラーのビジネス・オブジェクト定義の作成』

要求/応答ビジネス・オブジェクト構造の理解

Request-Response データ・ハンドラーは、要求または応答データ・ハンドラーとビジネス・オブジェクトを送受信するときにビジネス・オブジェクト定義を使用します。Request-Response データ・ハンドラーは、処理可能なビジネス・オブジェクトの構造に対して特定の要件を設定します。データ・ハンドラーに渡されるビジネス・オブジェクトは、1 つの要求子オブジェクトと 1 つ以上の応答子オブジェクトを持つ必要があります。これらの子オブジェクトは、自身を処理するデータ・ハンドラーの要件に適合する必要があります。

注: Request-Response データ・ハンドラーでは、ビジネス・オブジェクト定義またはその属性にアプリケーション固有情報を設定する必要はありません。

要求/応答ビジネス・オブジェクトを表すビジネス・オブジェクトの構造を図 31 に示します。

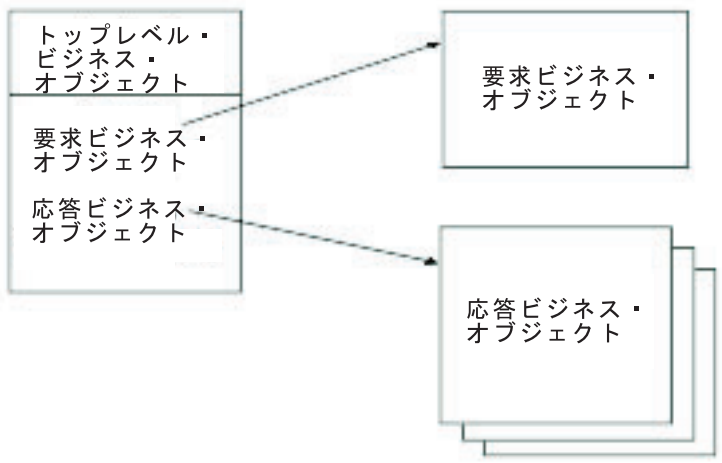

図 31. 要求/応答ビジネス・オブジェクトのビジネス・オブジェクト構造

ビジネス・オブジェクト定義を Request-Response データ・ハンドラーの要件に適合したものにするためには、以下の各ビジネス・オブジェクトごとに提供されるガイドラインを使用してください。

- 『トップレベル・ビジネス・オブジェクト』
- 109 ページの『要求ビジネス・オブジェクト』
- 110 ページの『応答ビジネス・オブジェクト』

トップレベル・ビジネス・オブジェクト

Request-Response データ・ハンドラーは、トップレベル・ビジネス・オブジェクトが呼び出し側コンテキストとの間で送受信する情報を保持しているものとして動作します。表 30 では、Request-Response データ・ハンドラーがビジネス・オブジェクトのプロパティーをどのように解釈するかを説明し、また、Request-Response データ・ハンドラーで使用できるようにビジネス・オブジェクトを修正する場合のプロパティーの設定方法を説明しています。

表 30. トップレベル・ビジネス・オブジェクト定義のプロパティー

プロパティー名	説明
Name	各ビジネス・オブジェクト定義には、固有の名前を付ける必要があります。これらのビジネス・オブジェクト定義の先頭に標準プレフィックスを付加することをお勧めします。トップレベル・ビジネス・オブジェクトの名前は、次のようにメッセージ標準によって異なります。 *BusObj プレフィックス + 応答ビジネス・オブジェクト*
Version	ビジネス・オブジェクト定義の現行バージョンを表す定数。現在の値は 1.0.0 です。
アプリケーション固有情報	アプリケーション固有情報は不要です。

このトップレベル・ビジネス・オブジェクトには次の属性が必要です。

- 単一の要求ビジネス・オブジェクトを保持するための単一カーディナリティーの属性

この要求ビジネス・オブジェクトは、要求データ・ハンドラーに必要なすべてのビジネス・オブジェクト要件に適合する必要があります。

- 応答ビジネス・オブジェクトを保持するための単一カーディナリティーの属性

  この応答ビジネス・オブジェクトは、応答データ・ハンドラーに必要なすべてのビジネス・オブジェクト要件に適合する必要があります。

注:

1. ターゲット・アプリケーションが複数の タイプの応答ビジネス・オブジェクトを戻す可能性がある場合は、トップレベル・ビジネス・オブジェクトには応答ビジネス・オブジェクトのそれぞれのタイプごとに 1 つの子ビジネス・オブジェクトが含まれている必要があります。

   例えば、ターゲット・アプリケーションが Customer XML 文書または OrderUpdate XML 文書のいずれかを戻す可能性がある場合は、トップレベル・ビジネス・オブジェクト定義には 2 つの属性が含まれている必要があります。1 つは Customer XML 文書を表すビジネス・オブジェクト定義を保持する属性、もう 1 つは OrderUpdate XML 文書を表すビジネス・オブジェクト定義を保持する属性です。データ・ハンドラーは応答ビジネス・オブジェクトを受け取ると、適切な属性にデータを取り込みます。
2. データ・ハンドラーは、要件に適合しない トップレベル・ビジネス・オブジェクトを受け取った場合や、関与している他のデータ・ハンドラーに渡されたビジネス・オブジェクトまたは文書をそのデータ・ハンドラーが変換できなかった場合に正常に終了しません。

要求ビジネス・オブジェクト

Request-Response データ・ハンドラーは、要求データ・ハンドラーの要求情報を保持するため、トップレベル・ビジネス・オブジェクトの最初の属性が要求ビジネス・オブジェクトであるものとして動作します。この属性は単一カーディナリティーである必要があります。表 31 では、Request-Response データ・ハンドラーがこのビジネス・オブジェクト定義のプロパティーをどのように解釈するかを説明し、また、Request-Response データ・ハンドラーで使用できるようにビジネス・オブジェクトを修正する場合のこれらのプロパティーの設定方法を説明しています。

表 31. 要求ビジネス・オブジェクト定義のプロパティー

プロパティー名	説明
Name	各ビジネス・オブジェクト定義には、固有の名前を付ける必要があります。この名前は、要求データ・ハンドラーが処理するビジネス・オブジェクト定義名と一致しなければなりません。
Version	ビジネス・オブジェクト定義の現行バージョンを表す定数。現在の値は 1.0.0 です。
アプリケーション固有情報	使用される要求データ・ハンドラーによって異なります。

注: 要求ビジネス・オブジェクトのフォーマットの詳細については、要求データ・ハンドラーとして動作するデータ・ハンドラーの資料を参照してください。

例えば、RequestDataHandlerMimeType を text/xml として指定した場合、要求ビジネス・オブジェクトとして定義する子オブジェクトは、XML データ・ハンドラーとの互換性を持つ必要があります。

応答ビジネス・オブジェクト

Request-Response データ・ハンドラーは、応答データ・ハンドラーの応答情報を保持するため、トップレベル・ビジネス・オブジェクトの 2 番目以降の属性に応答ビジネス・オブジェクトがあるものとして動作します。この属性は単一カーディナリティーである必要があります。応答データ・ハンドラーが複数のタイプのビジネス・オブジェクトを戻す可能性がある場合、トップレベル・ビジネス・オブジェクトはそれぞれのタイプごとに 1 つの属性を持ちます。表 32 では、Request-Response データ・ハンドラーがこのビジネス・オブジェクト定義のプロパティーをどのように解釈するかを説明し、また、Request-Response データ・ハンドラーで使用できるようにビジネス・オブジェクトを修正する場合のこれらのプロパティーの設定方法を説明しています。

表 32. 応答ビジネス・オブジェクト定義のプロパティー

プロパティー名	説明
Name	各ビジネス・オブジェクト定義には、固有の名前を付ける必要があります。この名前は、応答データ・ハンドラーが処理するビジネス・オブジェクト定義名と一致しなければなりません。
Version	ビジネス・オブジェクト定義の現行バージョンを表す定数。現在の値は 1.0.0 です。
アプリケーション固有情報	使用される要求データ・ハンドラーによって異なります。

注: 要求ビジネス・オブジェクトのフォーマットの詳細については、要求データ・ハンドラーとして動作するデータ・ハンドラーの資料を参照してください。

例えば、ResponseDataHandlerMimeType を text/abc として指定した場合、要求ビジネス・オブジェクトとして定義する子オブジェクトは、abc MIME タイプを処理できるカスタム・データ・ハンドラーとの互換性を持つ必要があります。

Request-Response データ・ハンドラーのビジネス・オブジェクト定義の作成

このセクションでは、Request-Response データ・ハンドラーの要件に適合する構造を表すビジネス・オブジェクト定義の作成方法について説明します。Business Object Designer Express を使用して、必要に応じてビジネス・オブジェクト定義から属性を追加または削除し、属性プロパティーを編集します。

107 ページの『要求/応答ビジネス・オブジェクト構造の理解』 で説明したように、Request-Response データ・ハンドラーでは次のビジネス・オブジェクト定義を作成する必要があります。

- 111 ページの『トップレベル・ビジネス・オブジェクト定義の作成』
- 111 ページの『その他のビジネス・オブジェクト定義の作成』

トップレベル・ビジネス・オブジェクト定義の作成

Request-Response データ・ハンドラー用のトップレベル・ビジネス・オブジェクト定義を作成するには、Business Object Designer Express を使用して手動でビジネス・オブジェクト定義を作成する必要があります。

1. トップレベル・ビジネス・オブジェクト定義を作成します。

   このトップレベル・ビジネス・オブジェクト定義の構造については、108 ページの『トップレベル・ビジネス・オブジェクト』を参照してください。

2. トップレベル・ビジネス・オブジェクトに、子ビジネス・オブジェクトを作成します。トップレベル・ビジネス・オブジェクト内で、表 33 に示すビジネス・オブジェクトの子オブジェクト属性を作成します。

表 33. Request-Response データ・ハンドラー用のビジネス・オブジェクト

属性	注	ビジネス・オブジェクト
要求ビジネス・オブジェクト	要求に関する情報が含まれます。	109 ページの『要求ビジネス・オブジェクト』
応答ビジネス・オブジェクト	要求の応答に関する情報が含まれます。	110 ページの『応答ビジネス・オブジェクト』

その他のビジネス・オブジェクト定義の作成

要求および応答ビジネス・オブジェクト定義を作成するには、次のいずれかの方法を使用できます。

- Object Discovery Agent (ODA) を使用して、直列化データをビジネス・オブジェクト定義としてエクスポートできます。XML 文書などのデータ・フォーマット用にいくつかの ODA があります。

  XML ODA の詳細については、197 ページの『XML ODA の使用』を参照してください。その他の ODA の詳細については、対応するアダプター・ガイドを参照してください。

- Business Object Designer Express を使用して、データ用のビジネス・オブジェクト定義を手動で作成できます。

  詳細については、『トップレベル・ビジネス・オブジェクト定義の作成』を参照してください。

Request-Response データ・ハンドラーの構成

Request-Response データ・ハンドラーは、次のようにしてメタオブジェクトの階層から構成プロパティーを受け取ります。

- 親メタオブジェクトを使用することにより、コネクターまたは Server Access Interface は文書の MIME タイプに基づいてデータ・ハンドラーのインスタンスを生成できます。
- 子メタオブジェクトには、データ・ハンドラーのクラス名、作成するビジネス・オブジェクトのプレフィックスなど、文書内のデータの処理に必要なすべての情報が含まれています。

トップレベルのメタオブジェクトの構成

親メタオブジェクトに含まれる MIME タイプは、サポートされる MIME タイプと、それをサポートするデータ・ハンドラーを示します。提供されるトップレベルのメタオブジェクトには、Request-Response データ・ハンドラーの入力は組み込まれていません。コネクターまたはアクセス・クライアントで Request-Response データ・ハンドラーを使用するには、text/requestresponse MIME タイプの属性をトップレベルのメタオブジェクト MO_Server_DataHandler または MO_DataHandler_Default に追加する必要があります。この属性は、タイプ MO_DataHandler_DefaultRequestResponseConfig である必要があります。

次のビジネス・オブジェクト定義のフラグメントは、text/requestresponse 属性の定義を示しています。

```
[Attribute]
Name = text.requestresponse
Type = MO_DataHandler_DefaultRequestResponseConfig
ContainedObjectVersion = 1.0.0
Relationship = Containment
Cardinality = 1
MaxLength = 1
IsKey = false
IsForeignKey = false
IsRequired = false
IsRequiredServerBound = false
[End]
```

注: トップレベルのメタオブジェクトおよびその変更方法の詳細については、26 ページの『トップレベルのメタオブジェクト』を参照してください。

子メタオブジェクトの構成

Request-Response データ・ハンドラーを構成するには、その構成情報が Request-Response 子メタオブジェクトとして提供されることを確認する必要があります。

注: Request-Response データ・ハンドラーを構成するには、ビジネス・オブジェクト定義を作成または変更してデータ・ハンドラーをサポートするようにすることも必要です。詳細については、107 ページの『ビジネス・オブジェクト定義の要件』を参照してください。

Request-Response データ・ハンドラー向けに、IBM からデフォルト子メタオブジェクト MO_DataHandler_DefaultRequestResponseConfig が提供されています。このメタオブジェクトの各属性は、Request-Response データ・ハンドラーの構成プロパティーを定義しています。表 34 で、この子メタオブジェクト内の属性について説明します。

表 34. Request-Response データ・ハンドラーの子メタオブジェクト属性

属性名	説明	納入時のデフォルト値
BOPrefix	トップレベル・ビジネス・オブジェクト名を作成するために、デフォルトの NameHandler クラスで使用されるプレフィックス。デフォルト値は、関連するビジネス・オブジェクト定義の名前に一致するよう変更する必要があります。属性値は大文字小文字を区別します。	REQUESTTEST

表 34. *Request-Response* データ・ハンドラーの子メタオブジェクト属性 (続き)

属性名	説明	納入時のデフォルト値
ClassName	指定された MIME タイプで使用するためにロードするデータ・ハンドラー・クラスの名前。トップレベルのデータ・ハンドラー・メタオブジェクトは、属性の名前が指定された MIME タイプと一致し、そのタイプが Request-Response 子メタオブジェクトである必要があります (この表で説明しています)。	com.crossworlds.DataHandlers.text.requestresponse
NameHandlerClass	要求文書の内容からトップレベル・ビジネス・オブジェクトの名前を決定するために用いるネーム・ハンドラー・クラスの名前。独自のカスタム・ネーム・ハンドラーを作成する場合は、この属性のデフォルト値を変更します。詳細については、 96 ページの『カスタム XML ネーム・ハンドラーの作成』を参照してください。	com. crossworlds.DataHandlers.xml.TopElementNameHandler
RequestDataHandlerMimeType	このデータ・ハンドラーによって処理される要求の MIME タイプ。Request-Response データ・ハンドラーはこの MIME タイプを使用して、要求ビジネス・オブジェクトや要求文書の処理用にインスタンスを生成するデータ・ハンドラーを決定します。	text/xml
ResponseDataHandlerMimeType	このデータ・ハンドラーによって処理される応答の MIME タイプ。Request-Response データ・ハンドラーはこの MIME タイプを使用して、応答ビジネス・オブジェクトや応答文書の処理用にインスタンスを生成するデータ・ハンドラーを決定します。	text/xml
ObjectEventId	プレースホルダー。データ・ハンドラーは使用しませんが、ビジネス・インテグレーション・システムで必要です。	なし

表 34 の「納入時のデフォルト値」列には、納入時のビジネス・オブジェクトの対応する属性の Default Value プロパティーの値がリストされています。使用する環境を調べてこれらの属性の Default Value プロパティーに適切な値を設定する必要があります。少なくとも ClassName と BOPrefix 属性にはデフォルト値が設定されている必要があります。

MO_DataHandler_DefaultRequestResponseConfig 子メタオブジェクトを作成するには、Business Object Designer Express を使用して次のフォーマットのビジネス・オブジェクト定義を作成します。

```
[BusinessObjectDefinition]
Name = MO_DataHandler_DefaultRequestResponseConfig
Version = 1.0.0

   [Attribute]
   Name = ClassName
   Type = String
   Cardinality = 1
   MaxLength = 255
   IsKey = false
   IsForeignKey = false
   IsRequired = false
   DefaultValue = com.crossworlds.DataHandlers.text.requestresponse
   IsRequiredServerBound = false
   [End]
```

```
[Attribute]
Name = NameHandlerClass
Type = String
MaxLength = 255
IsKey = false
IsForeignKey = false
IsRequired = false
IsRequiredServerBound = false
[End]

[Attribute]
Name = RequestDataHandlerMimeType
Type = String
MaxLength = 255
IsKey = false
IsForeignKey = false
IsRequired = false
DefaultValue = text/xml
IsRequiredServerBound = false
[End]

[Attribute]
Name = ResponseDataHandlerMimeType
Type = String
MaxLength = 255
IsKey = false
IsForeignKey = false
IsRequired = false
DefaultValue = text/xml
IsRequiredServerBound = false
[End]

[Attribute]
Name = BOPrefix
Type = String
MaxLength = 255
IsKey = false
IsForeignKey = false
IsRequired = false
DefaultValue = Wrapper
IsRequiredServerBound = false
[End]

[Attribute]
Name = DummyKey
Type = String
MaxLength = 1
IsKey = true
IsForeignKey = false
IsRequired = false
DefaultValue = 1
IsRequiredServerBound = false
[End]

[Attribute]
Name = ObjectEventId
Type = String
Cardinality = 1
MaxLength = 255
IsKey = false
IsForeignKey = false
IsRequired = false
IsRequiredServerBound = false
[End]

[Verb]
Name = Create
[End]
```

```
   [Verb]
   Name = Delete
   [End]
   [Verb]
   Name = Retrieve
   [End]
   [Verb]
   Name = Update
   [End]
[End]
```

この子メタオブジェクト・ファイルの格納場所については、 25 ページの『データ・ハンドラーの構成』を参照してください。

要求データ・ハンドラーによるビジネス・オブジェクトの変換

要求データ・ハンドラー は、要求を開始する WebSphere Business Integration システム・コンポーネントのデータ変換を行います。

- 要求処理において、統合ブローカーはコネクターに送信された要求ビジネス・オブジェクトの形式で要求を開始します。

WebSphere InterChange Server

統合ブローカーが InterChange Server (ICS) である場合、コラボレーションが要求ビジネス・オブジェクトを作成し、それを要求処理のために適切なコネクターに送信します。

したがって、要求データ・ハンドラーは要求に対してビジネス・オブジェクトからストリングへの変換を行います。つまり、要求ビジネス・オブジェクトを直列化データに変換します。この直列化データは、コネクターのアプリケーション (またはアクセス・クライアント) が入力として受け入れる *要求フォーマット* のデータです。

- イベント処理において、呼び出し側コンテキスト (アクセス・クライアントまたはコネクター) は統合ブローカーに送信された直列化データの形式で要求を開始します。

WebSphere InterChange Server

統合ブローカーが ICS である場合、アクセス・クライアントを使用して同期イベント処理を開始できます。詳細については、 102 ページの『イベント処理のサポート』を参照してください。

したがって、要求データ・ハンドラーは要求に対してストリングからビジネス・オブジェクトへの変換を行います。つまり、直列化データを要求ビジネス・オブジェクトに変換します。この直列化データは、アクセス・クライアントまたはコネクターのアプリケーションが出力として生成する要求フォーマットのデータです。

Request-Response データ・ハンドラーは、子メタオブジェクト内の RequestDataHandlerMimeType プロパティーに基づいて、要求データ・ハンドラーとして呼び出すデータ・ハンドラーを決定します。 RequestDataHandlerMimeType プロパティーには、要求データ・ハンドラーがサポートする MIME タイプが含まれます。RequestDataHandlerMimeType が初期化されていない 場合、Request-Response データ・ハンドラーはエラーをログに記録し、例外を生成します。このため、RequestDataHandlerMimeType プロパティーを初期化する必要があります。

応答データ・ハンドラーによるビジネス・オブジェクトの変換

応答データ・ハンドラー は、要求に応答する WebSphere Business Integration システム・コンポーネントのデータ変換を行います。

- 要求処理において、コネクターのアプリケーションはコネクターに送信された直列化データの形式で要求に応答し、それを統合ブローカーに送信します。

  WebSphere InterChange Server

  統合ブローカーが InterChange Server (ICS) である場合、コラボレーションがコネクターから応答ビジネス・オブジェクトを受信します。

  したがって、応答データ・ハンドラーは応答に対してストリングからビジネス・オブジェクトへの変換を行います。つまり、直列化データを応答ビジネス・オブジェクトに変換します。この直列化データは、コネクターのアプリケーション (またはアクセス・クライアント) が出力として生成する応答フォーマット のデータです。
- イベント処理において、統合ブローカーは呼び出し側コンテキスト (アクセス・クライアントまたはコネクター) に送信された応答ビジネス・オブジェクトの形式で要求に応答します。

  WebSphere InterChange Server

  統合ブローカーが ICS である場合、ICS 内のコラボレーションが応答ビジネス・オブジェクトを生成します。次に、ICS はこの応答ビジネス・オブジェクトを呼び出し側コンテキストに戻します。詳細については、 102 ページの『イベント処理のサポート』を参照してください。

  したがって、応答データ・ハンドラーは応答に対してビジネス・オブジェクトからストリングへの変換を行います。つまり、応答ビジネス・オブジェクトを直列化データに変換します。この直列化データは、アクセス・クライアントまたはコネクターのアプリケーションが入力として受け入れる応答フォーマットのデータです。

Request-Response データ・ハンドラーは、子メタオブジェクト内の ResponseDataHandlerMimeType プロパティーに基づいて、応答データ・ハンドラーとして呼び出すデータ・ハンドラーを決定します。 ResponseDataHandlerMimeType プロパティーには、応答データ・ハンドラーがサポートする MIME タイプが含まれ

ます。ResponseDataHandlerMimeType が初期化されていない 場合、Request-Response データ・ハンドラーはエラーをログに記録し、例外を生成します。このため、ResponseDataHandlerMimeType プロパティーを初期化する必要があります。

エラー処理

Request-Response データ・ハンドラーは、要求を処理できない場合に例外をスローします。Request-Response データ・ハンドラーから出される例外メッセージには、エラー発生時のデータ・ハンドラーの動作内容と、トランザクションに含まれるコンポーネント (別のデータ・ハンドラーや JCDK) から戻されたすべてのメッセージが示されます。これらの例外は、最初にデータ・ハンドラーを呼び出したコンポーネントに (おそらく例外としてではなく別のデータ構造として) 伝搬されます。

Request-Response データ・ハンドラーは、データ・ハンドラー・フレームワークが提供するサービスを使用して、エラーの記録およびメッセージのトレースを行います。エラー・メッセージと警告は ICS ログに記録されます。データ・ハンドラーは、メッセージを Java コンソールには記録しません。

Request-Response データ・ハンドラーのカスタマイズ

特殊ネーム・ハンドラーを作成することによって、Request-Response データ・ハンドラーをカスタマイズすることができます。 Request-Response データ・ハンドラーは、ネーム・ハンドラーを呼び出して、作成するビジネス・オブジェクトの名前を取得します。データ・ハンドラーは、データ・ハンドラー・メタオブジェクトに格納された NameHandlerClass 属性の値を用いて起動するネーム・ハンドラーを決定します。 Request-Response データ・ハンドラーに組み込まれているデフォルトのネーム・ハンドラーは、応答データ・ハンドラーから戻されるビジネス・オブジェクトの名前の先頭に BOPrefix 属性の値を付加します。異なる方法で機能するネーム・ハンドラーが必要な場合は、次のようにします。

1. NameHandler クラスを拡張して、カスタム・ネーム・ハンドラーを作成します。
2. Request-Response データ・ハンドラーのメタオブジェクト内の NameHandlerClass 属性のデフォルト値を更新することにより、カスタム・ネーム・ハンドラー・クラスを使用するように Request-Response データ・ハンドラーを構成します。

カスタム・データ・ハンドラーの作成方法については、 171 ページの『カスタム・ネーム・ハンドラーの作成』を参照してください。

第 5 章 FixedWidth データ・ハンドラー

FixedWidth データ・ハンドラーは、ビジネス・オブジェクトを固定幅ストリングおよびストリームに変換し、固定幅ストリングおよびストリームをビジネス・オブジェクトに変換します。この章では、FixedWidth データ・ハンドラーが固定幅文書を処理する方法と、ビジネス・オブジェクトがデータ・ハンドラーによって処理されるように定義する方法を説明します。 FixedWidth データ・ハンドラーの要件を満たすビジネス・オブジェクトをインプリメントする際の指針として、この情報を利用することができます。また、この章では、FixedWidth データ・ハンドラーの構成方法も説明します。この章を構成するセクションは次のとおりです。

- 『概要』
- 120 ページの『FixedWidth データ・ハンドラーの構成』
- 124 ページの『FixedWidth 文書へのビジネス・オブジェクトの変換』
- 125 ページの『ビジネス・オブジェクトへの FixedWidth 文書の変換』

注: FixedWidth データ・ハンドラーは、CwDataHandler.jar ファイルに含まれる基本データ・ハンドラーの 1 つです。このデータ・ハンドラーのインストール方法およびそのソース・コードの保管先の詳細については、25 ページの『第 2 章 データ・ハンドラーのインストールと構成』を参照してください。

概要

FixedWidth データ・ハンドラーは、データ変換モジュールです。主に、ビジネス・オブジェクトを固定幅フィールド形式のストリングまたはストリームに変換し、また逆の変換を行います。固定幅ストリングまたはストリームは、text/fixedwidth MIME タイプの直列化データです。データ・ハンドラーは、固定幅フィールドを用いたテキスト・データを構文解析します。フィールド長は、各ビジネス・オブジェクト属性の MaxLength プロパティーで指定されています。このプロパティーの値は、ビジネス・オブジェクト定義に格納されています。

デフォルトのトップレベル・コネクターのメタオブジェクト (MO_DataHandler_Default) は、text/fixedwidth MIME タイプをサポートします。そのため、MO_DataHandler_Default データ・ハンドラー・メタオブジェクトをサポートするように構成されたコネクターは、FixedWidth データ・ハンドラーを呼び出すことができます。アクセス・クライアントがこのデータ・ハンドラーを呼び出すことができるようにするためには、text/fixedwidth MIME タイプをサポートするように MO_Server_DataHandler メタオブジェクトを変更する必要があります。詳細については、175 ページの『トップレベルのメタオブジェクトの変更』を参照してください。

FixedWidth データ・ハンドラーの機能

FixedWidth データ・ハンドラーは、ビジネス・オブジェクト定義内の属性の MaxLength プロパティーの値を使用して、データを読み書きする方法を決定します。MaxLength は、右そろえまたは左そろえのための埋め込みも含めた属性値の最大文字数を定義します。

桁そろえのためにデータに追加またはデータから除去するスペースに使用する埋め込み文字を構成することができます。埋め込み文字は、ビジネス・オブジェクトをストリングに変換するときに追加され、ストリングをビジネス・オブジェクトに変換するときに除去されます。また、左そろえ、または右そろえされるように、ビジネス・オブジェクト属性値の桁そろえを構成することもできます。これにより、属性値データが意味のある文字を保存できるようになります。Alignment メタオブジェクト属性の値を表 35 で説明します。

表 35. 桁そろえメタオブジェクト属性の値

値	説明
LEFT	左側をトリムし、右側をパッドします。
RIGHT	右側をトリムし、左側をパッドします。
BOTH	ストリングをビジネス・オブジェクトに変換するときに、左右両側をトリムします。ビジネス・オブジェクトをストリングに変換するときは、埋め込み文字で右側をパッドします。

さらに、Truncation メタオブジェクト属性を使用すれば、ビジネス・オブジェクト値を MaxLength に切り捨てるか、または属性値が MaxLength よりも長いストリングであればエラーを生成するように、FixedWidth データ・ハンドラーを構成することができます。ビジネス・オブジェクト名、動詞、およびカーディナリティー 1 または n の子オブジェクトのビジネス・オブジェクト・カウントに用いられる長さ (サイズ) を設定することもできます。

ストリング属性の MaxLength プロパティーの値を設定するには、Business Object Designer Express を使用します。他のタイプ (整数、倍精度浮動小数点数など) の MaxLength プロパティーの値を変更するには、ビジネス・オブジェクト定義をファイルに保存し、そのファイルを手動で編集した後、変更した定義をビジネス・インテグレーション・システムにインポートする必要があります。

ビジネス・オブジェクトと FixedWidth ストリング処理

FixedWidth データ・ハンドラーは、表 36 に示す操作を実行します。

表 36. FixedWidth データ・ハンドラーのデータ操作

データ・ハンドラー操作	詳細
呼び出し元からビジネス・オブジェクトを受け取り、ビジネス・オブジェクトを FixedWidth ストリングまたはストリームに変換し、その FixedWidth データを呼び出し元に渡します。	124 ページの『FixedWidth 文書へのビジネス・オブジェクトの変換』
呼び出し元からストリングまたはストリームを受け取り、ビジネス・オブジェクトを作成し、そのビジネス・オブジェクトを呼び出し元に返します。	125 ページの『ビジネス・オブジェクトへの FixedWidth 文書の変換』

FixedWidth データ・ハンドラーの構成

FixedWidth データ・ハンドラーを構成するには、以下のステップを実行します。

- FixedWidth 子メタオブジェクトの属性に適切な値を入力します。
- データ・ハンドラーをサポートするように、ビジネス・オブジェクト定義を作成または変更します。

上記の各ステップについて、以下のセクションで詳しく説明します。

FixedWidth 子メタオブジェクトの構成

FixedWidth データ・ハンドラーを構成するには、構成情報を FixedWidth 子メタオブジェクトに供給する必要があります。FixedWidth データ・ハンドラーについて、IBM では MO_DataHandler_DefaultFixedWidthConfig 子メタオブジェクトを提供しています。このメタオブジェクトの各属性は、FixedWidth データ・ハンドラーの構成プロパティーを定義しています。表 37 で、この子メタオブジェクトの属性について説明します。

表 37. FixedWidth データ・ハンドラーの子メタオブジェクト属性

メタオブジェクト属性名	意味	納入時のデフォルト値
ClassName	ロードするデータ・ハンドラー・クラスの名前。トップレベル・データ・ハンドラー・メタオブジェクトの属性名と一致する MIME タイプで使用します。この属性のタイプは、FixedWidth 子メタオブジェクトです。	com.crossworlds.DataHandlers.text.fixedwidth
Alignment	PadCharacter 属性を追加または除去します。イベント処理では、埋め込み文字がトリムされます。要求処理では、埋め込み文字が追加されます。設定可能な値は、BOTH、LEFT、および RIGHT です。例えば「LEFT」桁そろえを指定すると、ビジネス・オブジェクト属性の値が属性値のスペースの一番左に移動します。イベント通知の 「BOTH」桁そろえを指定すると、埋め込み文字が左右両側でトリムされます。要求処理の「RIGHT」桁そろえを指定すると、右側に埋め込み文字がパッドされます。	BOTH
BOCountSize	処理するビジネス・オブジェクトの総数に割り当てられるスペースを指定します。	3
BONameSize	ビジネス・オブジェクトの名前に割り当てられるスペースを指定します。	50
BOVerbSize	動詞に割り当てられるスペースを指定します。	20
CxBlank	ビジネス・オブジェクトからの変換を行う場合、FixedWidth データ・ハンドラーでは、属性の値が CxBlank であるビジネス・オブジェクト属性を検出すると、CxBlank メタオブジェクト属性の Default Value プロパティーのために構成された値を固定幅文書に書き込みます。また、ビジネス・オブジェクトへの変換を行う場合は、FixedWidth データ・ハンドラーでは、固定幅文書内でこの CxBlank メタオブジェクト属性の値を検出すると、CxBlank メタオブジェクト属性の Default Value プロパティーのために構成された値をビジネス・オブジェクト属性の値に割り当てます。ビジネス・オブジェクトは、値 CxBlank を含まない 基本キーを、実行時に少なくとも 1 つは持っていなければなりません。	CxBlank 値

表 37. FixedWidth データ・ハンドラーの子メタオブジェクト属性 (続き)

メタオブジェクト属性名	意味	納入時のデフォルト値
CxIgnore	ビジネス・オブジェクトからの変換を行う場合、FixedWidth データ・ハンドラーでは、値が CxIgnore であるビジネス・オブジェクト属性を検出すると、CxIgnore メタオブジェクト属性の Default Value プロパティーのために構成された値を固定幅文書に書き込みます。また、ビジネス・オブジェクトへの変換を行う場合は、FixedWidth データ・ハンドラーでは、固定幅文書内でこの CxIgnore メタオブジェクト属性の値を検出すると、CxIgnore メタオブジェクト属性の Default Value プロパティーのために構成された値をビジネス・オブジェクト属性の値に割り当てます。ビジネス・オブジェクトは、値 CxIgnore を含まない 基本キーを、実行時に少なくとも 1 つは持っていなければなりません。	CxIgnore 値
DummyKey	ビジネス・インテグレーション・システムで必要とされるキー属性。	1
OmitObjectEventId	ビジネス・オブジェクトからストリングへの変換時、およびストリングからビジネス・オブジェクトへの変換時に ObjectEventId データが含まれているかどうかを判別するためのブール値。	false
PadCharacter	桁そろえのために追加または除去するスペースを指示します。どのような文字でも埋め込み文字として指定することができます。	#
Truncation	文字の除去の有無を設定します。true の場合、MaxLength よりも大きいビジネス・オブジェクトの属性値は、要求処理時に MaxLength に合わせて切り捨てられます。false の場合は、エラーがログに記録され、フォーマットは停止します。	false
ObjectEventId	プレースホルダー。データ・ハンドラーは使用しませんが、ビジネス・インテグレーション・システムで必要です。	なし

表 37 の「納入時のデフォルト値」列には、納入時のビジネス・オブジェクトの対応する属性の Default Value プロパティーの値がリストされています。ご使用の環境を調べて、システムおよび FixedWidth 文書に適した値をそれらの属性の Default Value プロパティーに設定してください。少なくとも ClassName 属性には、デフォルト値を設定しておく必要があります。

注: ビジネス・オブジェクト定義を変更するには、Business Object Designer Express を使用します。

ビジネス・オブジェクトの要件

FixedWidth データ・ハンドラーは、いくつかの前提条件の下にビジネス・オブジェクト構造体を取り扱います。そのため、FixedWidth データ・ハンドラーを用いて変換を行う場合は、ビジネス・オブジェクトを作成するときに以下の規則に従ってください。

- ビジネス・オブジェクト定義のすべての属性に、適切な MaxLength プロパティー値を設定する。これにより FixedWidth データ・ハンドラーは、ビジネス・オブジェクトから FixedWidth フォーマットへ、または FixedWidth フォーマットからビジネス・オブジェクトへのデータ変換を正しく処理することができます。

- ビジネス・オブジェクト階層のすべてのレベルのすべてのビジネス・オブジェクトに、ObjectEventId 属性が含まれていることを確認します。 Business Object Designer Express は、ビジネス・オブジェクト定義を保管するときに自動的にこの作業を行いますが、要件が満たされていることを確認する必要があります。

ビジネス・オブジェクトの構造

ビジネス・オブジェクト構造体については、FixedWidth データ・ハンドラー向けの要件はありません。データ・ハンドラーは、MaxLength 属性プロパティーに値が設定されていればどのようなビジネス・オブジェクトも処理できます。

データ・ハンドラーが処理するビジネス・オブジェクトの名前は、ビジネス・インテグレーション・システムで認められていればどのような名前でもかまいません。

ビジネス・オブジェクトの属性プロパティー

ビジネス・オブジェクトのアーキテクチャーには、属性に適用されるさまざまなプロパティーが含まれています。表 38 では、FixedWidth データ・ハンドラーがこれらのプロパティーをどのように解釈するかを説明し、また、ビジネス・オブジェクトを修正する場合のプロパティーの設定方法を説明しています。

表 38. FixedWidth データ・ハンドラーを使用して変換されるビジネス・オブジェクトの属性プロパティー

プロパティー名	説明
Name	各ビジネス・オブジェクト属性には、固有の名前を付ける必要があります。
Type	各ビジネス・オブジェクト属性には、整数、ストリング、または下位の子ビジネス・オブジェクトのタイプなどのタイプを設定する必要があります。
Key	FixedWidth データ・ハンドラーは使用しません。
MaxLength	属性値が含まれるフィールドの幅を決定します。
Foreign Key	FixedWidth データ・ハンドラーは使用しません。
Required	FixedWidth データ・ハンドラーは使用しません。
Default Value	FixedWidth データ・ハンドラーは使用しません。
Cardinality	カーディナリティー 1 およびカーディナリティー n のオブジェクトをサポートします。

ビジネス・オブジェクト・アプリケーション固有情報

FixedWidth データ・ハンドラーでは、ビジネス・オブジェクトまたはその属性にアプリケーション固有情報を設定する必要はありません。ただし、データ・ハンドラーは cw_mo_ タグがあるかどうかについては検査します。このタグは、コネクターが用いる子メタオブジェクトを指示するためにビジネス・オブジェクトが使用する場合があります。データ・ハンドラーは、ビジネス・オブジェクトのアプリケーション固有情報に含まれている cw_mo_ タグで識別された属性をすべて無視します。cw_mo_ タグの詳細については、163 ページの『ビジネス・オブジェクトからの変換のインプリメント』を参照してください。

既存のビジネス・オブジェクト定義の使用

FixedWidth データ・ハンドラーは、ビジネス・オブジェクトがデータ・ハンドラーの要件を満たす形式でデータを提供していれば、どのようなビジネス・オブジェクトでも FixedWidth ストリングに変換できます。FixedWidth データ・ハンドラーの

唯一の要件は、各ビジネス・オブジェクト属性に MaxLength 値が指定されていることです。既存のビジネス・オブジェクトを使用するのであれば、MaxLength に適切な値を指定する変更が必要となる場合があります。

この要件を満たしていれば、既存のビジネス・オブジェクトも FixedWidth データ・ハンドラーによって変換できますが、処理するデータのタイプごとに独自のビジネス・オブジェクトを作成することをお勧めします。サンプルのビジネス・オブジェクト、または別のインプリメンテーションで同じアプリケーションをサポートするために開発されたビジネス・オブジェクトを使用する場合は、自身が開発を行うインプリメンテーションに必要な属性のみが含まれるように、必要に応じて定義を修正してください。

したがって、既存のビジネス・オブジェクトを、ご使用のデータと密接に対応した形式に変換するには、アプリケーションが必要とするデータおよびデータ・ハンドラーが必要とする情報のみを提供するように、ビジネス・オブジェクトを変更してください。既存のビジネス・オブジェクトを FixedWidth データ・ハンドラーで利用できるようにするためには、以下の操作を実行します。

1. ターゲット・アプリケーションの機能分析を実行し、結果を既存のビジネス・オブジェクトと比較して、必要となるビジネス・オブジェクト定義のフィールドを決定します。
2. Business Object Designer Express を使用して、必要に応じてビジネス・オブジェクト定義へ属性を追加、またはビジネス・オブジェクト定義から属性を削除します。

FixedWidth 文書へのビジネス・オブジェクトの変換

ビジネス・オブジェクトを FixedWidth 文書に変換するため、FixedWidth データ・ハンドラーはビジネス・オブジェクトの属性を順次ループ処理します。ビジネス・オブジェクトとその子において属性が現れる順番に従って、固定幅ストリング内で再帰的にフィールドを生成します。

FixedWidth データ・ハンドラーは、以下のようにしてビジネス・オブジェクトを FixedWidth 文書に加工します。

1. データ・ハンドラーは、ビジネス・オブジェクトのデータを格納するための固定幅ストリングを作成する。
2. データ・ハンドラーは、ビジネス・オブジェクト名および動詞を固定幅ストリングに追加する。ビジネス・オブジェクトの名前は、変換メソッドへの引き数として指定することができます。
3. データ・ハンドラーは、ビジネス・オブジェクト定義のアプリケーション固有情報を調べて、子メタオブジェクト (ビジネス・オブジェクトのアプリケーション固有情報の cw_mo_ タグ内に名前がリストされているもの) があるかどうかを判断する。データ・ハンドラーは、FixedWidth 文書にこれらの属性を組み込みません。cw_mo_ タグの詳細については、163 ページの『ビジネス・オブジェクトからの変換のインプリメント』を参照してください。
4. データ・ハンドラーは、OmitObjectEventId という名前のメタオブジェクト属性を調べる。この属性が true に設定されている場合、データ・ハンドラーはビジネス・オブジェクトの ObjectEventId データを FixedWidth 文書に組み込みません。

5. データ・ハンドラーは、順番に残りのビジネス・オブジェクト属性をループ処理し、それぞれの単純属性のストリングに正しく埋め込み文字を追加する。配列属性については、データ・ハンドラーは以下の操作を実行します。
   - 属性が単一カーディナリティー属性を表す場合、データ・ハンドラーは属性名および 1 のカウントをストリングに追加した後、子ビジネス・オブジェクトを再帰的に処理して、ストリングに各属性の値を追加する。
   - 属性が複数カーディナリティー配列を表す場合、データ・ハンドラーは属性名および子オブジェクトのカウントをストリングに追加した後、それぞれの子ビジネス・オブジェクトを再帰的に処理して、ストリングに各属性の値を追加する。
6. データ・ハンドラーが変換を完了したら、直列化データが呼び出し元へ戻されます。データ・ハンドラーは、呼び出し元から要求された形式 (`String` または `InputStream`) でデータを返します。

注: `Truncation` メタオブジェクト属性の「デフォルト値」プロパティーの値が `true` に設定されている場合、`MaxLength` よりも長いビジネス・オブジェクトの属性値は `MaxLength` に合わせて切り捨てられます。 `Truncation` が `false` に設定されているときに、属性値が `MaxLength` よりも長い場合、フォーマットは終了し、エラーがログに記録されます。

ビジネス・オブジェクトへの FixedWidth 文書の変換

このセクションでは、FixedWidth データ・ハンドラーが FixedWidth 文書をビジネス・オブジェクトに変換する方法について以下の項目を説明します。

- 『FixedWidth ストリングの要件』
- 126 ページの『直列化データの処理』

FixedWidth ストリングの要件

FixedWidth データ・ハンドラーは、ストリングをビジネス・オブジェクトに変換するときに、以下の前提条件の下に処理を行います。

- ビジネス・オブジェクト名は、データの最初のフィールドとして現れる。
- 動詞は、データの 2 番目のフィールドとして現れる。
- すべての属性は、ビジネス・オブジェクト定義で現れる順番に従って提供される。
- 各ビジネス・オブジェクトには、`ObjectEventId` 属性が存在する。`CxIgnore` の値が設定されている場合にも、ビジネス・オブジェクトには `ObjectEventId` のエントリーが必要です。実行時にビジネス・オブジェクトのインスタンスを区別するため、データ・ハンドラーが使用するからです。

固定幅ストリングのフォーマットは、次のようになっています。

```
[Bus_Obj_Name<blank_space_padding_for_size >]
[Verb<blank_space_padding_for_size >]
[Attr1<blank_space_padding_for_size >]
[Attr2...<blank_space_padding_for_size >]
[Number-of-child-object_instances<blank_space_padding_for_size >]
[Child_Object_Name<blank_space_padding_for_size >]
```

```
[Child_Object_Verb<blank_space_padding_for_size >]
[Child_Object_Attr1<blank_space_padding_for_size >]
[Child_Object_Attr2...<blank_space_padding_for_size >]
<EndBO:Bus_Obj_Name>
```

このフォーマットでは、最初の 2 つのフィールド (Bus_Obj_Name および Verb) は、FixedWidth 子メタオブジェクト内の BONameSize および BOVerbSize 属性で指定された長さのフィールドを作成するようにパッドされます。以降の属性は、各ビジネス・オブジェクト属性の MaxLength プロパティーで指定された長さのフィールドを作成するようにパッドされます。

その属性で CxIgnore が使用可能であれば、FixedWidth データ・ハンドラーで用いられるフィールドの長さは、少なくとも 8 文字以上でなければなりません。属性が CxIgnore の値を含んでいないことが保証されていれば、MaxLength は 8 文字より少なくてもかまいません。

コネクターが固定幅フォーマットのファイルを読み取る場合、望ましい値を生成するように CxIgnore メタオブジェクト属性、および CxBlank メタオブジェクト属性を構成する必要があります。これらのストリングは、最小の MaxLength 属性に影響します。MaxLength の最小値は、両方に対応する必要があります。

直列化データの処理

FixedWidth データ・ハンドラーは、以下のようにして FixedWidth 文書をビジネス・オブジェクトに加工します。

1. データ・ハンドラーは、データを格納するためのビジネス・オブジェクトを作成する。ビジネス・オブジェクトのタイプがコネクターによって変換メソッドに渡されるか、またはデータ・ハンドラーがストリングの最初のフィールドからビジネス・オブジェクトの名前を取り出します。タイプ・パラメーターとデータの内容が一致しない場合、データ・ハンドラーはエラーを生成します。
2. データ・ハンドラーは、ビジネス・オブジェクトに動詞を設定します。データ・ハンドラーは、トップレベル・ビジネス・オブジェクト用の動詞が、データの 2 番目のフィールドにあることを想定しています。
3. データ・ハンドラーは、子メタオブジェクト (ビジネス・オブジェクトのアプリケーション固有情報の cw_mo_ タグに名前がリストされているもの) があるかどうかを判別します。データ・ハンドラーは、ビジネス・オブジェクトのこれらの属性を設定する処理を実行しません。cw_mo_ タグの詳細については、 163 ページの『ビジネス・オブジェクトからの変換のインプリメント』を参照してください。
4. データ・ハンドラーは、OmitObjectEventId という名前のメタオブジェクト属性を調べる。この属性が true に設定されている場合、データ・ハンドラーは ObjectEventId 属性にデータを取り込む処理を実行しません。
5. データ・ハンドラーは、残りのビジネス・オブジェクト属性を設定するため、ビジネス・オブジェクト定義で指定された各属性の MaxLength に基づいてデータを構文解析する。データから属性値を取り出し、ビジネス・オブジェクト内の単純属性の値として取り込みます。

   データ・ハンドラーは、以下のようにして配列属性を処理します。

- 属性が単一カーディナリティー属性であれば、データ・ハンドラーは属性リストを再帰的に構文解析し、属性値を設定し、子ビジネス・オブジェクトを親ビジネス・オブジェクトに追加する。
- 属性が複数カーディナリティー配列であれば、データ・ハンドラーはそれぞれの子属性リストの属性を再帰的に構文解析し、子ビジネス・オブジェクトを親ビジネス・オブジェクトに追加する。

第 6 章 Delimited データ・ハンドラー

Delimited データ・ハンドラーは、ビジネス・オブジェクトを区切り形式のストリングおよびストリームへ変換し、区切り形式のストリングおよびストリームをビジネス・オブジェクトへ変換します。この章では、Delimited データ・ハンドラーが区切りデータを処理する方法、およびデータ・ハンドラーが処理するビジネス・オブジェクトの定義方法について説明します。この情報は、既存ビジネス・オブジェクトを変更するとき、またはデータ・ハンドラーの要件に準拠する新規ビジネス・オブジェクトをインプリメントするときの手引きとして使用できます。また、この章では、Delimited データ・ハンドラーの構成方法についても説明します。この章を構成するセクションは次のとおりです。

- 『概要』
- 130 ページの『Delimited データ・ハンドラーの構成』
- 134 ページの『ビジネス・オブジェクトの区切りデータへの変換』
- 135 ページの『区切りデータのビジネス・オブジェクトへの変換』

注: Delimited データ・ハンドラーは、CwDataHandler.jar ファイルに含まれる基本データ・ハンドラーの 1 つです。このデータ・ハンドラーのインストール方法およびそのソース・コードの保管先の詳細については、25 ページの『第 2 章 データ・ハンドラーのインストールと構成』を参照してください。

概要

Delimited データ・ハンドラーは、主にビジネス・オブジェクトと区切り形式ストリングおよびストリーム間の変換を行うデータ変換モジュールです。区切り形式のストリングおよびストリームは、text/delimited MIME タイプの直列化データです。データ・ハンドラーは、ビジネス・オブジェクト・データの個々のフィールドを区切るために指定された区切り文字を基にしてテキスト・データを解析します。このタイプのデータ変換は主に、マシン読み取りの効率が最も重要な要素である場合に使用されます。

デフォルトのトップレベル・コネクター・メタオブジェクト (MO_DataHandler_Default) は、text/delimited MIME タイプをサポートします。したがって、MO_DataHandler_Default メタオブジェクトを使用するように構成されたコネクターは、Delimited データ・ハンドラーを呼び出すことができます。アクセス・クライアントがこのデータ・ハンドラーを呼び出せる必要がある場合は、text/delimited MIME タイプをサポートするように、トップレベル・サーバー・メタオブジェクト (MO_Server_DataHandler) を変更する必要があります。詳細については、175 ページの『トップレベルのメタオブジェクトの変更』を参照してください。

Delimited データ・ハンドラーの機能

Delimited データ・ハンドラーを使用すると、次のストリングを設定できます。

- 区切り文字: データ・ハンドラーは区切り文字を使用して、区切られたデータ内のそれぞれのフィールドを区切ります。Delimiter メタオブジェクト属性を、データ内の該当の区切り文字に設定できます。デフォルトではデータ・ハンドラーは、データの読み書き方法を決定するために、属性プロパティー MaxLength の代りに波形記号 (~) を区切り文字として使用します。MaxLength は、属性値の文字の最大数 (テキストの右寄せまたは左寄せが可能な場合の埋め込みを含む) を定義するビジネス・オブジェクト属性プロパティーです。MaxLength は、リポジトリー内のビジネス・オブジェクトの定義から読み取られます。したがって、ビジネス・オブジェクトの主な要件は、各ストリング属性の MaxLength を適切な値に設定することです。
- エスケープ・ストリング: データ・ハンドラーは、エスケープ・ストリングを使用して、区切り文字とエスケープ・ストリングをエスケープするためのストリングを構成します。エスケープ・ストリングにより、属性値データに、区切り文字およびエスケープに類似のストリングを含めることができます。エスケープ・ストリングを構成するように、Escape メタオブジェクト属性を設定できます。デフォルトでは、データ・ハンドラーはエスケープ・ストリングとして円記号 (¥) を使用します。

ビジネス・オブジェクトとストリング処理

Delimited データ・ハンドラーは、表 39 にリストされている操作を実行します。

表 39. Delimited データ・ハンドラーのデータ操作

データ・ハンドラー操作	詳細
呼び出し元からビジネス・オブジェクトを受け取り、ビジネス・オブジェクトを区切りストリングまたはストリームに変換し、直列化データを呼び出し元へ渡します。	134 ページの『ビジネス・オブジェクトの区切りデータへの変換』
呼び出し元からストリングまたはストリームを受け取り、ビジネス・オブジェクトを作成し、そのビジネス・オブジェクトを呼び出し元に返します。呼び出し元から区切りストリングまたはストリームを受け取り、ビジネス・オブジェクトを構築し、ビジネス・オブジェクトを呼び出し元へ渡します。	135 ページの『区切りデータのビジネス・オブジェクトへの変換』

Delimited データ・ハンドラーの構成

Delimited データ・ハンドラーを構成するには、次の手順に従ってください。

- Delimited 子メタオブジェクトの適切な属性値を入力します。
- データ・ハンドラーをサポートするように、ビジネス・オブジェクト定義を作成または変更します。

上記の各ステップについて、以下のセクションで詳しく説明します。

Delimited 子メタオブジェクトの構成

Delimited データ・ハンドラーを構成するには、その構成情報が Delimited 子メタオブジェクトに提供されることを確認する必要があります。Delimited データ・ハンドラーについて、IBM では MO_DataHandler_DefaultDelimitedConfig 子メタオブジ

ェクトを提供しています。このメタオブジェクトの各属性は、Delimited データ・ハンドラーの構成プロパティーを定義しています。表 40 で、この子メタオブジェクト内の属性について説明します。

表 40. Delimited データ・ハンドラーの子メタオブジェクト属性

メタオブジェクト属性名	意味	納入時のデフォルト値
ClassName	指定された MIME タイプで使用するためにロードするデータ・ハンドラー・クラスの名前。トップレベルのデータ・ハンドラー・メタオブジェクトには、指定された MIME タイプと名前が一致し、タイプが Delimited 子メタオブジェクトである属性があります (表 40 で説明)。	com.crossworlds.DataHandlers.text.delimited
CxBlank	特殊なビジネス・オブジェクト属性値 Blank (CxBlank 定数) の Delimited データに同等値を設定します。詳細については、 133 ページの『CxBlank』を参照してください。	CxBlank 定数 (ブランク・スペース)
CxIgnore	特殊なビジネス・オブジェクト属性値 Ignore (CxIgnore 定数) の Delimited データ に同等値を設定します。詳細については、 132 ページの『CxIgnore』を参照してください。	CxIgnore 定数 (空ストリング)
Delimiter	ビジネス・オブジェクト・データをファイルに書き込むときにビジネス・オブジェクト属性内の値を区切るために使用されるストリング、またはファイルをビジネス・オブジェクトに変換するときに属性に対応するデータのフィールドを区切ることが想定されるストリング。この値には複数の文字を含めることができます。	~ (波形記号)
DummyKey	ビジネス・インテグレーション・システムで必要とされるキー属性。	1
Escape	区切り文字およびエスケープ文字がビジネス・オブジェクト属性値に含まれている場合に、それらをエスケープするために使用されるストリング。この値は 1 文字だけです。	¥ (円記号)
OmitObjectEventId	ビジネス・オブジェクトからストリングへの変換時、およびストリングからビジネス・オブジェクトへの変換時に ObjectEventId データが含まれているかどうかを判別するためのブール値。	false
ObjectEventId	ビジネス・インテグレーション・システムに必要なプレースホルダー属性。	なし

表 40 の「納入時のデフォルト値」列には、納入時のビジネス・オブジェクトの対応する属性の Default Value プロパティーの値がリストされています。環境を調べ、それらの属性の Default Value プロパティーをシステムおよび区切り文書に適した値に設定する必要があります。少なくとも ClassName 属性には、デフォルト値を設定しておく必要があります。

注: ビジネス・オブジェクト定義を変更するには、Business Object Designer Express を使用します。

ビジネス・オブジェクトの要件

Delimited データ・ハンドラーは、処理するビジネス・オブジェクトについて想定します。したがって、Delimited データ・ハンドラーを使用して変換のためのビジネス・オブジェクトを作成するときには、次の規則に従ってください。

- Delimited データ・ハンドラーがデータを正しく変換できるように、必ずすべてのビジネス・オブジェクト属性に名前を付けます。
- ビジネス・オブジェクト階層のすべてのレベルのすべてのビジネス・オブジェクトに、ObjectEventId 属性が含まれていることを確認します。 Business Object Designer Express は、ビジネス・オブジェクト定義を保管するときに自動的にこの作業を行いますが、要件が満たされていることを確認する必要があります。

子メタオブジェクトの Delimiter 属性は、属性フィールドを区切るために使用される区切り文字を構成します。デフォルト値は波形記号 (~) です。

区切り文字とエスケープ・ストリングをエスケープするためのストリングを構成するために、子メタオブジェクト属性 Escape を設定できます。エスケープ・ストリングにより、属性値データに、区切り文字およびエスケープに類似のストリングを含めることができます。

ビジネス・オブジェクトの構造

Delimited データ・ハンドラーのビジネス・オブジェクトの構造に関する要件はありません。データ・ハンドラーは、いずれのビジネス・オブジェクトも処理できます。

データ・ハンドラーが処理するビジネス・オブジェクトには、統合ブローカーで許可される任意の名前を付けることができます。

ビジネス・オブジェクトの属性プロパティー

ビジネス・オブジェクトのアーキテクチャーには、属性に適用されるさまざまなプロパティーが含まれています。表 41 では、Delimited データ・ハンドラーがこれらのいくつかのプロパティーを解釈する方法、およびビジネス・オブジェクトの変更時にプロパティーを設定する方法について説明します。

表 41. Delimited データ・ハンドラーを使用して変換されるビジネス・オブジェクトの属性プロパティー

プロパティー名	説明
Name	すべてのビジネス・オブジェクト属性に固有の名前が必要です。
Type	各ビジネス・オブジェクト属性には、整数、ストリング、または下位の子ビジネス・オブジェクトのタイプなどのタイプを設定する必要があります。
Key	Delimited データ・ハンドラーでは使用されません。
MaxLength	Delimited データ・ハンドラーでは使用されません。
Foreign Key	Delimited データ・ハンドラーでは使用されません。
Required	Delimited データ・ハンドラーでは使用されません。
Default Value	Delimited データ・ハンドラーでは使用されません。
Cardinality	カーディナリティー 1 およびカーディナリティー n のオブジェクトをサポートします。

ビジネス・オブジェクトの属性には、CxIgnore または CxBlank の特殊値を指定できます。Delimited データ・ハンドラーは、以下のセクションで説明するとおり、属性がこれらの値を持つときは特別な処理ステップを実行します。

CxIgnore: CxIgnore メタオブジェクト属性は、Ignore 属性値 (CxIgnore 定数) の Delimited データに同等値を設定します。デフォルトでは、CxIgnore メタオブジ

ェクト属性が、CxIgnore 定数の値に設定されます。データ・ハンドラーは、CxIgnore メタオブジェクト属性を次のように使用します。

- ビジネス・オブジェクトから 変換する場合、Delimited データ・ハンドラーは、属性の値として CxIgnore 定数が含まれているビジネス・オブジェクト属性を検出すると必ず、この CxIgnore メタオブジェクト属性の値 (Default Value プロパティーに構成されている値) を Delimited データに書き込みます。
- ビジネス・オブジェクトへ 変換する場合、Delimited データ・ハンドラーは、Delimited データでこの CxIgnore メタオブジェクト属性の値 (その Default Value プロパティーに構成されている値) を検出すると必ず、CxIgnore 定数をビジネス・オブジェクト属性の値に割り当てます。

注: ビジネス・オブジェクトは、値 CxIgnore を含まない 基本キーを、実行時に少なくとも 1 つは持っていなければなりません。

CxBlank: CxBlank メタオブジェクト属性は、Blank 属性値 (CxBlank 定数) の Delimited データに同等値を設定します。デフォルトでは、CxBlank メタオブジェクト属性が、CxBlank 定数の値に設定されます。データ・ハンドラーは、CxBlank メタオブジェクト属性を次のように使用します。

- ビジネス・オブジェクトから 変換する場合、Delimited データ・ハンドラーは、属性値として CxBlank 定数が含まれているビジネス・オブジェクト属性を検出すると必ず、この CxBlank メタオブジェクト属性の値 (Default Value プロパティーに構成されている値) を Delimited データに書き込みます。
- ビジネス・オブジェクトへ 変換する場合、Delimited データ・ハンドラーは、Delimited データでこの CxBlank メタオブジェクト属性の値 (その Default Value プロパティーに構成されている値) を検出すると必ず、CxBlank 定数をビジネス・オブジェクト属性の値に割り当てます。

注: ビジネス・オブジェクトは、値 CxBlank を含まない 基本キーを、実行時に少なくとも 1 つ は持っていなければなりません。

ビジネス・オブジェクト・アプリケーション固有情報

Delimited データ・ハンドラーには、ビジネス・オブジェクトまたはそれらの属性に関するアプリケーション固有情報は必要ありません。ただし、データ・ハンドラーは cw_mo_ タグがあるかどうかについては検査します。このタグは、コネクターが用いる子メタオブジェクトを指示するためにビジネス・オブジェクトが使用する場合があります。データ・ハンドラーは、ビジネス・オブジェクトのアプリケーション固有情報に含まれている cw_mo_ タグで識別された属性をすべて無視します。cw_mo_ タグの詳細については、163 ページの『ビジネス・オブジェクトからの変換のインプリメント』を参照してください。

既存のビジネス・オブジェクト定義の使用

Delimited データ・ハンドラーは、ビジネス・オブジェクトがデータ・ハンドラーの要件に適合する形式でデータを引き渡す限り、いずれのビジネス・オブジェクトも区切りストリングに変換できます。Delimited データ・ハンドラーの唯一の要件は、データ・ハンドラーが区切りファイルを読み取る必要がある場合に、その個々のフィールドが構成済みの区切り文字で区切られていることです。

この要件を満たしていれば、既存のビジネス・オブジェクトも Delimited データ・ハンドラーによって変換できますが、処理するデータのタイプごとに独自のビジネス・オブジェクトを作成することをお勧めします。サンプルのビジネス・オブジェクト、または別のインプリメンテーションで同じアプリケーションをサポートするために開発されたビジネス・オブジェクトを使用する場合は、自身が開発を行うインプリメンテーションに必要な属性のみが含まれるように、必要に応じて定義を修正してください。

したがって、既存のビジネス・オブジェクトを、ご使用のデータと密接に対応した形式に変換するには、アプリケーションが必要とするデータおよびデータ・ハンドラーが必要とする情報のみを提供するように、ビジネス・オブジェクトを変更してください。既存のビジネス・オブジェクトを Delimited データ・ハンドラーで利用できるようにするためには、以下の操作を実行します。

1. ターゲット・アプリケーションの機能分析を実行し、結果を既存のビジネス・オブジェクトと比較して、必要となるビジネス・オブジェクト定義のフィールドを決定します。
2. Business Object Designer Express を使用して、必要に応じてビジネス・オブジェクト定義へ属性を追加、またはビジネス・オブジェクト定義から属性を削除します。

ビジネス・オブジェクトの区切りデータへの変換

ビジネス・オブジェクトをストリングへ変換するために、Delimited データ・ハンドラーは、ビジネス・オブジェクトの属性を順番にループします。また、ビジネス・オブジェクト内での属性とその子の出現順に、区切りフォーマットを再帰的に生成します。ビジネス・オブジェクトの名前は、変換メソッドに引き数として渡されます。

Delimited データ・ハンドラーは、ビジネス・オブジェクトを次のように処理して、区切りデータへ変換します。

1. データ・ハンドラーは、ビジネス・オブジェクト内のデータを含むストリングを作成します。
2. データ・ハンドラーは、ストリングの最初のトークンとしてビジネス・オブジェクト名を追加し、ストリングの 2 番目のトークンとして動詞を追加します。
3. データ・ハンドラーは、ビジネス・オブジェクト定義のアプリケーション固有情報を調べて、子メタオブジェクト (ビジネス・オブジェクトのアプリケーション固有情報の `cw_mo_` タグ内に名前がリストされているもの) があるかどうかを判断します。データ・ハンドラーは、区切りデータにこれらの属性を含めません。`cw_mo_` タグの詳細については、163 ページの『ビジネス・オブジェクトからの変換のインプリメント』を参照してください。
4. データ・ハンドラーは、`OmitObjectEventId` という名前のメタオブジェクト属性を調べる。これが `true` に設定されている場合は、区切りデータにビジネス・オブジェクトの `ObjectEventId` データを含めません。
5. データ・ハンドラーは、残りのビジネス・オブジェクト属性を順にループし、それぞれの単純属性ごとに値をストリングへ追加し、各属性の後に構成済みの区切り文字を追加する。コンテナー属性の場合、データ・ハンドラーは次のことを行います。

- 属性がカーディナリティー 1 のコンテナーである場合、データ・ハンドラーでは、属性カウントをストリングへ追加し、続いて子ビジネス・オブジェクトを再帰的に処理して、各属性ごとに値を追加する。
- 属性がカーディナリティー n のコンテナーである場合、データ・ハンドラーは、コンテナー内の子オブジェクトのカウントをストリングへ追加し、続いて各子ビジネス・オブジェクトを再帰的に処理して、各属性の値をストリングへ追加する。

6. データ・ハンドラーが変換を完了したら、直列化データが呼び出し元へ戻されます。データ・ハンドラーは、呼び出し元が要求した形式 (String または InputStream) でデータを戻します。

データ・ハンドラーが生成する形式は、次のパターンに準拠します。

```
Bus_Obj_Name<delimiter>Verb<delimiter>Attr1<delimiter>Attr2<delimiter>
Number_of_child_object_instances<delimiter>Child_Object_Name<delimiter>Verb
<delimiter>Attr1<delimiter>Attr2<EndBO:Bus_Obj_Name>
```

属性値内の区切り文字に類似のストリングの前には、エスケープ・ストリングが付加されます。エスケープ・ストリングは、子メタオブジェクトの `Escape` 属性を使用して構成されます。

区切りデータのビジネス・オブジェクトへの変換

このセクションでは、Delimited データ・ハンドラーが区切りデータをビジネス・オブジェクトへ変換する方法に関する以下の情報を提供します。

- 『区切りストリングの要件』
- 136 ページの『直列化データの処理』

区切りストリングの要件

ストリングまたはストリームを変換するときには、Delimited データ・ハンドラーは以下のことを前提とします。

- データには、`Delimiter` メタオブジェクト属性に指定された区切り文字が含まれている。
- ビジネス・オブジェクト名は、データの最初のフィールドに含まれている。
- 動詞は、データの 2 番目のフィールドとして現れる。
- 属性は、ビジネス・オブジェクト定義内での出現順になっている。
- 子コンテナー内のすべてのオブジェクトが同じタイプである。
- データには、カーディナリティー `n` の各コンテナー内の子オブジェクトの数を表すトークンが含まれている。
- 各ビジネス・オブジェクトには、`ObjectEventId` 属性が存在する。`CxIgnore` の値が設定されている場合にも、ビジネス・オブジェクトには `ObjectEventId` のエントリーが必要です。実行時にビジネス・オブジェクトのインスタンスを区別するため、データ・ハンドラーが使用するからです。

データ内に複数のオブジェクトがある場合は、オブジェクト間に新しい文字 (例えば、スペース、タブ、改行、または復帰など) を挿入しないでください。

Delimited データ・ハンドラーは、Delimited 形式のファイルを読み取る際、以下のような特別な処理ステップを使用して、ビジネス・オブジェクト属性に CxIgnore または CxBlank 属性値を割り当てます。

- データ・ハンドラーは、Delimited データで以下のいずれかの条件が検出されると必ず、それに対応する属性値としてこの CxIgnore 定数 (null) を割り当てます。
  - CxIgnore メタオブジェクト属性の値 (その Default Value プロパティーに構成されている値)
  - 空ストリングの値 (″ ″)
- データ・ハンドラーは、CxBlank メタオブジェクト属性が構成されていて、しかもこの構成済みの値を対応する Delimited データで検出した場合にのみ、CxBlank 定数を属性値として割り当てます。

注: エスケープ・ストリングと区切り文字は、必ず異なる値にしてください。これらは、Delimiter データ・ハンドラーの子メタオブジェクト内の Escape および Delimiter メタオブジェクト属性で構成されます。

次の行には、区切り形式のストリングの例を示してあります。構文は次のとおりです。

```
Bus_Obj_Name<delimiter >Verb<delimiter >Attr1<delimiter >Attr2<delimiter >
Number_of_child_object_instances<delimiter >Child_Object_Name<delimiter >
Verb<delimiter >Attr1<delimiter >Attr2<EndBO:Bus_Obj_Name>
```

次のサンプルでは、波形記号 (~) の区切り文字を使用しています。

```
Customer~Create~p1~p2~p3~1~CustomerAddress~Create~q1~q2~q3~q4~q5~q6~q7~q8~q9~q10~3~
PhoneInfo~Create~r1~r2~r3~r4~r5~r6~r7~PhoneInfo~Create~r1~r2~r3~r4~r5~r6~r7~
PhoneInfo~Create~r1~r2~r3~r4~r5~r6~r7
```

直列化データの処理

Delimited データ・ハンドラーは、次のように区切りデータを処理して、ビジネス・オブジェクトに変換します。

1. データ・ハンドラーは、データの最初のトークンからビジネス・オブジェクト名を取得し、データを格納するビジネス・オブジェクトを作成する。
2. データ・ハンドラーは、ビジネス・オブジェクトに動詞を設定します。データ・ハンドラーは、トップレベル・ビジネス・オブジェクトの動詞が、区切りデータの 2 番目のトークンに含まれていると想定します。子ビジネス・オブジェクトには動詞が設定されていない可能性があることに注意してください。
3. データ・ハンドラーは、子メタオブジェクト (ビジネス・オブジェクトのアプリケーション固有情報の cw_mo_ タグに名前がリストされているもの) があるかどうかを判別します。データ・ハンドラーは、ビジネス・オブジェクトのこれらの属性を設定する処理を実行しません。cw_mo_ タグの詳細については、163 ページの『ビジネス・オブジェクトからの変換のインプリメント』を参照してください。
4. データ・ハンドラーは、OmitObjectEventId という名前のメタオブジェクト属性を調べる。この属性が true に設定されている場合、データ・ハンドラーは ObjectEventId 属性にデータを取り込む処理を実行しません。

5. データ・ハンドラーは、データを解析し、データからのトークン値を使用して、ビジネス・オブジェクトに残りの単純属性の値を取り込む。データ・ハンドラーは、次のようにコンテナー属性を処理します。
   - 属性が単一カーディナリティーである場合は、データ・ハンドラーがストリング内の属性トークンを再帰的に解析し、ビジネス・オブジェクトに属性値を設定し、子ビジネス・オブジェクト・コンテナーを親ビジネス・オブジェクトに追加する。
   - 属性が複数カーディナリティーである場合は、データ・ハンドラーが各子オブジェクトの属性トークンを再帰的に解析し、子ビジネス・オブジェクトに属性値を設定し、子ビジネス・オブジェクト・コンテナーを親ビジネス・オブジェクトに追加する。

第 7 章 NameValue データ・ハンドラー

NameValue データ・ハンドラーは、ビジネス・オブジェクトを名前と値のペア形式のストリングまたはストリームに変換し、名前と値のペア形式のストリングまたはストリームをビジネス・オブジェクトに変換します。この章では、NameValue データ・ハンドラーが NameValue データを処理する方法、および NameValue データ・ハンドラーが処理するビジネス・オブジェクトの定義方法について説明します。この情報は、NameValue データ・ハンドラーの要件に準拠するビジネス・オブジェクトをインプリメントする際の手引きとして使用できます。また、この章では、NameValue データ・ハンドラーの構成方法についても説明します。この章を構成するセクションは次のとおりです。

- 『概要』
- 140 ページの『NameValue データ・ハンドラーの構成』
- 144 ページの『ビジネス・オブジェクトの NameValue データへの変換』
- 145 ページの『NameValue データのビジネス・オブジェクトへの変換』

注: NameValue データ・ハンドラーは、`CwDataHandler.jar` ファイルに含まれる基本データ・ハンドラーの 1 つです。このデータ・ハンドラーのインストール方法およびそのソース・コードの保管先の詳細については、25 ページの『第 2 章 データ・ハンドラーのインストールと構成』を参照してください。

概要

NameValue データ・ハンドラーは、データ変換モジュールで、主な役割はビジネス・オブジェクトと名前と値のペア形式のストリングまたはストリーム間の変換を行います。NameValue 形式のストリングおよびストリームは、`text/namevalue` MIME タイプの直列化データです。 NameValue データ・ハンドラーは、テスト目的でビジネス・オブジェクト・ファイルを生成するために使用できます。

データ・ハンドラーは、名前が付いたフィールドを基にして、直列化データを解析します。例えば、このデータ・ハンドラーのテキスト・ファイルに、ビジネス・オブジェクト・タイプ (BusinessObject=BOname)、動詞 (Verb=verbName)、属性の数 (AttributeCount=numericValue)、および属性値 (`AttributeName=`*`Value`*) を識別するフィールドが含まれています。データ・ハンドラーは、データを解析するために名前と値の情報を使用します。

デフォルトのトップレベル・コネクター・メタオブジェクト (`MO_DataHandler_Default`) は、`text/namevalue` MIME タイプをサポートします。したがって、`MO_DataHandler_Default` メタオブジェクトを使用するように構成されたコネクターは、NameValue データ・ハンドラーを呼び出すことができます。アクセス・クライアントがこのデータ・ハンドラーを呼び出すことができるようにするためには、`text/namevalue` MIME タイプをサポートするように `MO_Server_DataHandler` メタオブジェクトを変更する必要があります。詳細については、175 ページの『トップレベルのメタオブジェクトの変更』を参照してください。

NameValue データ・ハンドラーは、表 42 にリストされている操作を実行します。

表 42. NameValue データ・ハンドラーのデータ操作

データ・ハンドラー操作	詳細
呼び出し元からビジネス・オブジェクトを受け取り、ビジネス・オブジェクトを NameValue ストリングまたはストリームに変換し、直列化データを呼び出し元へ渡します。	144 ページの『ビジネス・オブジェクトの NameValue データへの変換』
呼び出し元からストリングまたはストリームを受け取り、ビジネス・オブジェクトを作成し、そのビジネス・オブジェクトを呼び出し元に返します。	145 ページの『NameValue データのビジネス・オブジェクトへの変換』

NameValue データ・ハンドラーの構成

NameValue データ・ハンドラーを構成するには、次の手順に従ってください。

- NameValue 子メタオブジェクトの適切な属性値を入力します。
- データ・ハンドラーをサポートするように、ビジネス・オブジェクト定義を作成または変更します。

上記の各ステップについて、以下のセクションで詳しく説明します。

NameValue 子メタオブジェクトの構成

NameValue データ・ハンドラーを構成するには、その構成情報が NameValue 子メタオブジェクトに提供されることを確認する必要があります。NameValue データ・ハンドラーについて、IBM では MO_DataHandler_DefaultNameValueConfig 子メタオブジェクトを提供しています。このメタオブジェクトの各属性は、NameValue データ・ハンドラーの構成プロパティーを定義しています。表 43 では、この子メタオブジェクトの属性について説明しています。

表 43. NameValue データ・ハンドラーの子メタオブジェクト属性

メタオブジェクト属性名	説明	納入時のデフォルト値
ClassName	指定された MIME タイプで使用するためにロードするデータ・ハンドラー・クラスの名前。トップレベルのデータ・ハンドラー・メタオブジェクトの属性は、その名前が、指定された MIME タイプと一致し、そのタイプは、NameValue 子メタオブジェクトです (表 43 で説明しています)。	com.crossworlds.DataHandlers.text.namevalue
CxBlank	特殊なビジネス・オブジェクト属性値 Blank (CxBlank 定数) の NameValue データに、同等値を設定します。詳細については、 143 ページの『CxBlank』を参照してください。	CxBlank 定数
CxBlankValue	*この属性は使用しないでください。*NameValue データが CxBlank 属性値を表す方法をデータ・ハンドラーに通知するために、CxBlank メタオブジェクト属性 (上記) を使用してください。	ブランク・スペース
CxIgnore	特殊なビジネス・オブジェクト属性値 Ignore (CxIgnore 定数) の NameValue データに同等値を設定します。詳細については、 142 ページの『CxIgnore』を参照してください。	CxIgnore 定数
DefaultVerb	ビジネス・オブジェクトの動詞。	Create

表 43. NameValue データ・ハンドラーの子メタオブジェクト属性 (続き)

メタオブジェクト属性名	説明	納入時のデフォルト値
DummyKey	ビジネス・インテグレーション・システムで必要とされるキー属性。	1
SkipCxIgnore	要求処理時に、特殊属性値 CxIgnore は、メタオブジェクト属性 SkipCxIgnore に基づいて次のように処理されます。詳細については、142 ページの『CxIgnore』を参照してください。	false
ValidateAttrCount	データ・ハンドラーが、ビジネス・オブジェクト・データ内の属性のカウントが含まれているトークンを検索 (または出力ストリングへ追加) します。	true
ObjectEventId	プレースホルダー。データ・ハンドラーは使用しませんが、ビジネス・インテグレーション・システムで必要です。	なし

表 43 の「納入時のデフォルト値」列には、納入時のビジネス・オブジェクトの対応する属性の Default Value プロパティーの値がリストされています。環境を調べ、それらの属性の Default Value プロパティーを、システムと名前と値のペア形式の文書に適した値に設定する必要があります。

注: ビジネス・オブジェクト定義を変更するには、Business Object Designer Express を使用します。

ビジネス・オブジェクトの要件

NameValue データ・ハンドラーは、処理するビジネス・オブジェクトを想定します。したがって、NameValue データ・ハンドラーを使用して変換のためのビジネス・オブジェクトを渡す時には、次の規則に従ってください。

- 必ずすべてのビジネス・オブジェクト属性に Name プロパティーを設定します。これにより、NameValue データ・ハンドラーは、ビジネス・オブジェクトから NameValue 形式へのデータ変換、および NameValue 形式からビジネス・オブジェクトへのデータ変換を正しく処理できます。
- ビジネス・オブジェクト階層のすべてのレベルのすべてのビジネス・オブジェクトに、ObjectEventId 属性が含まれていることを確認します。 Business Object Designer Express は、ビジネス・オブジェクト定義を保管するときに自動的にこの作業を行いますが、要件が満たされていることを確認する必要があります。

ビジネス・オブジェクトの構造

NameValue データ・ハンドラーのビジネス・オブジェクトの構造に関する要件はありません。データ・ハンドラーは、いずれのビジネス・オブジェクトも処理できます。

データ・ハンドラーが処理するビジネス・オブジェクトの名前は、ビジネス・インテグレーション・システムで認められていればどのような名前でもかまいません。

ビジネス・オブジェクトの属性プロパティー

ビジネス・オブジェクトのアーキテクチャーには、属性に適用されるさまざまなプロパティーが含まれています。表 44 では、NameValue データ・ハンドラーがこれらのいくつかのプロパティーを解釈する方法、およびビジネス・オブジェクト定義

の変更時にプロパティーを設定する方法について説明します。

表 44. NameValue データ・ハンドラーを使用して変換されるビジネス・オブジェクトの属性プロパティー

プロパティー名	説明
Name	各ビジネス・オブジェクト属性には、固有の名前を付ける必要があります。
Type	各ビジネス・オブジェクト属性には、整数、ストリング、または下位の子ビジネス・オブジェクトのタイプなどのタイプを設定する必要があります。単純属性はすべて、タイプ・ストリングである必要があります。
Key	NameValue データ・ハンドラーでは使用されません。
MaxLength	NameValue データ・ハンドラーでは使用されません。
Foreign Key	NameValue データ・ハンドラーでは使用されません。
Required	NameValue データ・ハンドラーでは使用されません。
Default Value	NameValue データ・ハンドラーでは使用されません。
Cardinality	カーディナリティー 1 およびカーディナリティー n のオブジェクトをサポートします。

ビジネス・オブジェクトの属性には、CxIgnore または CxBlank の特殊値を指定できます。NameValue データ・ハンドラーは、以下のセクションで説明するとおり、属性がこれらの値を持つときは特別な処理ステップを実行します。

CxIgnore: CxIgnore メタオブジェクト属性は、Ignore 属性値 (CxIgnore 定数) の NameValue データに同等値を設定します。デフォルトでは、CxIgnore メタオブジェクト属性が、CxIgnore 定数の値に設定されます。データ・ハンドラーは、CxIgnore メタオブジェクト属性を次のように使用します。

- ビジネス・オブジェクトから 変換する場合、NameValue データ・ハンドラーは、属性の値として CxIgnore 定数が含まれているビジネス・オブジェクト属性を検出すると必ず、この CxIgnore メタオブジェクト属性の値 (Default Value プロパティーに構成されている値) を NameValue データに書き込みます。
- ビジネス・オブジェクトへ 変換する場合、NameValue データ・ハンドラーは、NameValue データで以下のいずれかの条件が検出されると必ず、この CxIgnore 定数をビジネス・オブジェクト属性の値に割り当てます。
  - CxIgnore メタオブジェクト属性の値 (その Default Value プロパティーに構成されている値)
  - 空ストリングの値
  - 対応していない値

注: ビジネス・オブジェクトは、値 CxIgnore を含まない 基本キーを、実行時に少なくとも 1 つは持っていなければなりません。

NameValue データ・ハンドラーが CxIgnore 値に設定された属性を処理する方法を指定して構成することができます。例えば次のようになります。

- 要求処理時に CxIgnore 値が含まれている属性を処理するか、またはそれらを無視するように、データ・ハンドラーを構成できます。

- CxIgnore 値が含まれている属性の項目を作成しないように決定できます。これにより、コネクターはイベント通知時にビジネス・オブジェクト・データを正しく処理できます。これは、オブジェクト・タイプ用のダミー・ファイルを作成するときに役立ちます。

要求処理時、データ・ハンドラーは、ビジネス・オブジェクトの直列化バージョンを作成します。この時点では、特殊属性値 CxIgnore の処理は、次のように子メタオブジェクト属性 SkipCxIgnore に基づきます。

- SkipCxIgnore が false に設定されている場合、データ・ハンドラーは、属性の値として CxIgnore 定数が含まれているビジネス・オブジェクト属性を検出すると必ず、この CxIgnore メタオブジェクト属性の値を NameValue データに書き込みます。
- SkipCxIgnore が true に設定されている場合、データ・ハンドラーは、CxIgnore の値を持つすべての属性を無視し、それらの NameValue データを生成しません。

注: SkipCxIgnore が true に設定されると、NameValue データ・ハンドラーは双方向ではなくなります。つまり、ビジネス・オブジェクトからストリングへの変換中に生成されたストリングについて、ストリングからビジネス・オブジェクトへの変換を実行することはできません。

CxBlank: CxBlank メタオブジェクト属性は、Blank 属性値 (CxBlank 定数) の NameValue データに同等値を設定します。デフォルトでは、CxBlank メタオブジェクト属性が、CxBlank 定数の値に設定されます。データ・ハンドラーは、CxBlank メタオブジェクト属性を次のように使用します。

- ビジネス・オブジェクトから 変換する場合、NameValue データ・ハンドラーは、属性の値として CxBlank 定数が含まれているビジネス・オブジェクト属性を検出すると必ず、この CxBlank メタオブジェクト属性の値 (Default Value プロパティーに構成されている値) を NameValue データに書き込みます。
- ビジネス・オブジェクトへ 変換する場合、NameValue データ・ハンドラーは、NameValue データでこの CxBlank メタオブジェクト属性の値 (その Default Value プロパティーに構成されている値) を検出すると必ず、CxBlank 定数をビジネス・オブジェクト属性の値に割り当てます。

注: ビジネス・オブジェクトは、値 CxBlank を含まない 基本キーを、実行時に少なくとも 1 つは持っていなければなりません。

ビジネス・オブジェクト・アプリケーション固有情報

NameValue データ・ハンドラーには、ビジネス・オブジェクトまたはそれらの属性に関するアプリケーション固有情報は必要ありません。ただし、データ・ハンドラーは cw_mo_ タグがあるかどうかについては検査します。このタグは、コネクターが用いる子メタオブジェクトを指示するためにビジネス・オブジェクトが使用する場合があります。データ・ハンドラーは、ビジネス・オブジェクトのアプリケーション固有情報に含まれている cw_mo_ タグで識別された属性をすべて無視します。cw_mo_ タグの詳細については、163 ページの『ビジネス・オブジェクトからの変換のインプリメント』を参照してください。

既存のビジネス・オブジェクト定義の使用

NameValue データ・ハンドラーは、ビジネス・オブジェクトがデータ・ハンドラーの要件に適合する形式でデータを引き渡す場合に限り、ビジネス・オブジェクトを NameValue 直列化データに変換できます。NameValue データ・ハンドラーは、データの各部分にそれ自体を識別する名前 (`BusinessObject=Customer`、`Verb=Create`、および `CustomerName=JDoe` など) を付けることを必要とします。属性にはそのような名前を付ける必要があるので、NameValue データ・ハンドラーで使用できます。

NameValue データ・ハンドラーでは、この要件を満たす既存ビジネス・オブジェクトを変換できますが、処理するデータのタイプごとに独自のビジネス・オブジェクトを作成してみることをお勧めします。サンプルのビジネス・オブジェクト、または別のインプリメンテーションで同じアプリケーションをサポートするために開発されたビジネス・オブジェクトを使用する場合は、自身が開発を行うインプリメンテーションに必要な属性のみが含まれるように、必要に応じて定義を修正してください。

したがって、既存のビジネス・オブジェクトを、ご使用のデータと密接に対応した形式に変換するには、アプリケーションが必要とするデータおよびデータ・ハンドラーが必要とする情報のみを提供するように、ビジネス・オブジェクトを変更してください。既存のビジネス・オブジェクトを NameValue データ・ハンドラーで利用できるようにするためには、以下の操作を実行します。

1. ターゲット・アプリケーションの機能分析を実行し、結果を既存のビジネス・オブジェクトと比較して、必要となるビジネス・オブジェクト定義のフィールドを決定します。
2. Business Object Designer Express を使用して、必要に応じてビジネス・オブジェクト定義へ属性を追加、またはビジネス・オブジェクト定義から属性を削除します。

ビジネス・オブジェクトの NameValue データへの変換

ビジネス・オブジェクトをストリングまたはストリームへ変換するために、NameValue データ・ハンドラーは、ビジネス・オブジェクトの属性を順番にループします。また、ビジネス・オブジェクト内での属性とその子の出現順に、名前と値のペアを再帰的に生成します。ビジネス・オブジェクトの名前は、変換メソッドに引き数として渡されます。

NameValue データ・ハンドラーは、ビジネス・オブジェクトを次のように処理して、NameValue データへ変換します。

1. データ・ハンドラーは、ビジネス・オブジェクト内のデータを含むストリングを作成します。
2. ビジネス・オブジェクト名を指定するために、データ・ハンドラーは `BusinessObject=`*`Name`* をストリングへ追加します。
3. 動詞を指定するために、データ・ハンドラーは `Verb=`*`Verb`* をストリングへ追加します。

4. メタオブジェクト属性 ValidateAttrCount が true に設定されている場合、データ・ハンドラーは、AttributeCount=*Count* をストリングへ追加します。この名前と値のペアは、ビジネス・オブジェクト・データ内の属性の数を指定します。
5. データ・ハンドラーは、ビジネス・オブジェクト定義のアプリケーション固有情報を調べて、子メタオブジェクト (ビジネス・オブジェクトのアプリケーション固有情報の cw_mo_ タグ内に名前がリストされているもの) があるかどうかを判断します。データ・ハンドラーは、NameValue データにこれらの属性を含めません。cw_mo_ タグの詳細については、 163 ページの『ビジネス・オブジェクトからの変換のインプリメント』を参照してください。
6. データ・ハンドラーは、残りのビジネス・オブジェクト属性を順にループし、単純な各属性ごとに名前と値のペアをストリングへ追加します。コンテナー属性の場合、データ・ハンドラーは次のことを行います。
   - 属性がカーディナリティー 1 のコンテナーである場合、データ・ハンドラーでは、属性名とカウント 1 をストリングへ追加し、続いて子ビジネス・オブジェクトを再帰的に処理して、各属性ごとに名前と値のペアをストリングへ追加します。
   - 属性がカーディナリティー n のコンテナーである場合、データ・ハンドラーは、属性名とコンテナー内の子オブジェクトの数をストリングへ追加し、続いて各子ビジネス・オブジェクトを再帰的に処理して、各属性の名前と値のペアをストリングへ追加します。
7. データ・ハンドラーが変換を完了したら、直列化データが呼び出し元へ戻されます。データ・ハンドラーは、呼び出し元が要求した形式 (String または InputStream) でデータを戻します。

子メタオブジェクト属性 ValidateAttrCount が true に設定されている場合、データ・ハンドラーは、ビジネス・オブジェクト内の属性のカウントが含まれているトークンを出力データへ追加します。データ・ハンドラーは、復帰を出力データへ追加します。終了結果は 図 32 (146 ページ) のようになります。

NameValue データのビジネス・オブジェクトへの変換

このセクションでは、NameValue データ・ハンドラーが名前と値のペア形式のストリングまたはストリームをビジネス・オブジェクトへ変換する方法についての以下の情報を提供します。

- 『NameValue ストリングの要件』
- 147 ページの『直列化データの処理』

NameValue ストリングの要件

NameValue データ・ハンドラーは、直列化データについて次のように想定します。

- ビジネス・オブジェクト名は、最初の名前と値のペアに含まれている。
- 動詞は、2 番目の名前と値のペアに含まれている。
- データには、ビジネス・オブジェクトに含まれている各子ごとに、子オブジェクトのインスタンスの数を表すトークンが含まれている。
- 各ビジネス・オブジェクトには、ObjectEventId 属性が存在する。

属性カウントを表すトークンは、オプションです。子メタオブジェクト属性 ValidateAttrCount が true に設定されている場合、データ・ハンドラーは、ビジネス・オブジェクト内の属性のカウントが含まれているトークンを検索します。属性カウントが指定されている場合は、ビジネス・オブジェクト定義内の属性の数を正確に反映する必要があります。

NameValue データ・ハンドラーは、名前と値の形式のファイルを読み取る際、以下のような特別な処理ステップを使用して、 ビジネス・オブジェクト属性に CxIgnore または CxBlank 属性値を割り当てます。

- データ・ハンドラーは、NameValue データで以下のいずれかの条件が検出されると必ず、それに対応する属性値としてこの CxIgnore 定数 (null) を割り当てます。
  - CxIgnore メタオブジェクト属性の値 (その Default Value プロパティーに構成されている値)
  - 空ストリングの値 (″ ″)
  - ビジネス・オブジェクト属性の NameValue データの対応しない値
- データ・ハンドラーは、CxBlank メタオブジェクト属性が構成されていて、しかもこの構成済みの値を、対応する NameValue データで検出したときにのみ、CxBlank 定数を属性値として割り当てます。

図 32 に、NameValue 形式の直列化データの例を示します。

```
BusinessObject=Customer
  Verb=Update
          AttributeCount=7
          CustomerID=103
          CustomerName=Thai Inc.
          Cust_Phone_Number=CxIgnore
          ProductName=GoodProduct
          Address=2
            BusinessObject=Address
              Verb=Update
              AttributeCount=3
              AddressID=105
              AddressLine=CxIgnore
              ObjectEventID=12345
            BusinessObject=Address
              Verb=Delete
              AttributeCount=3
              AddressID=106
              AddressLine=2758 Forest Avenue
              ObjectEventID=CxIgnore
          Item=1
            BusinessObject=Item
              Verb=Update
              ItemID=107
              ItemName=CxIgnore
              ObjectEventID=Obj_201
          ObjectEventID=SampleConnector_894927711_2
```

図 32. NameValue データの例

この例では、項目が次のものを表しています。

- BusinessObject は、処理する親または子ビジネス・オブジェクトの名前です。

- Verb は、親または子ビジネス・オブジェクトを送るために使用される要求のタイプです (例えば、Create、Update)。
- AttributeCount は、そのレベルにある親または子ビジネス・オブジェクトの属性の合計数です。
- CustomerID、CustomerName、Cust_Phone_Number、および ProductName は、親ビジネス・オブジェクトの属性の名前です。各親ビジネス・オブジェクト属性の値は、属性名に従います。
- Address = 2 は、Address 子ビジネス・オブジェクトの 2 つのインスタンスが存在することを示します。Address は、親オブジェクト内の Address 子ビジネス・オブジェクトを参照する属性名です。
- Item = 1 は、Item 属性に Item ビジネス・オブジェクトの単一のインスタンスが含まれていることを示します。
- AddressID および AddressLine は、Address 子ビジネス・オブジェクトの属性の名前です。各子ビジネス・オブジェクト属性の値は、属性名に従います。
- ObjectEventID=Obj_201 は、子ビジネス・オブジェクト Item のシステム生成 ID です。
- ObjectEventID=SampleConnector_894927711_2 は、親ビジネス・オブジェクト Customer のシステム生成 ID です。

直列化データの処理

NameValue データ・ハンドラーは、次のように、名前と値のペア形式のストリングまたはストリームをビジネス・オブジェクトへ変換します。

1. データ・ハンドラーが、ストリングまたはストリーム内のデータを含むビジネス・オブジェクトを作成します。
2. データ・ハンドラーは、ビジネス・オブジェクトに動詞を設定します。データ・ハンドラーは、トップレベル・ビジネス・オブジェクトの動詞が、データの 2 番目の名前と値のペアに含まれていると想定します。子ビジネス・オブジェクトには動詞が設定されていない可能性があることに注意してください。
3. ValidateAttrCount 子メタオブジェクト属性が true に設定されている場合、データ・ハンドラーは、ファイル内の属性の数がビジネス・オブジェクト定義内の属性の数に一致していることを確認します。
4. データ・ハンドラーが、直列化データを解析します。
   - 最初に、子メタオブジェクトが存在するかどうかを判別します (その名前は、ビジネス・オブジェクトのアプリケーション固有情報の cw_mo_ タグにリストされています)。データ・ハンドラーは、ビジネス・オブジェクトのこれらの属性を設定する処理を実行しません。cw_mo_ タグの詳細については、163 ページの『ビジネス・オブジェクトからの変換のインプリメント』を参照してください。
   - ビジネス・オブジェクトに残りの単純な属性の値を取り込みます。データ・ハンドラーは、次のようにコンテナー属性を処理します。

     属性が単一カーディナリティーである場合は、データ・ハンドラーが属性リスト内の属性を再帰的に解析し、子ビジネス・オブジェクト・コンテナーを親ビジネス・オブジェクトに追加します。

属性が複数カーディナリティーである場合は、データ・ハンドラーが各子属性リスト内の属性を再帰的に解析し、子ビジネス・オブジェクト・コンテナーを親ビジネス・オブジェクトに追加します。

データ・ハンドラーは名前と値の関連付けを行うので、ストリングからビジネス・オブジェクトへの変換の順序どおりに、直列化データ内の属性を指定できます。

第 2 部 カスタム・データ・ハンドラー

第 8 章 カスタム・データ・ハンドラーの作成

この章では、カスタム・データ・ハンドラーをインプリメントして WebSphere Business Integration Adapter と共に使用する方法、またはアクセス・クライアントと共に使用する方法についての情報を提供します。IBM からの納入時のカスタム・データ・ハンドラーは、ビジネス・オブジェクトを特定の直列化データ形式に変換し、特定形式の直列化データをビジネス・オブジェクトに変換します。この章を構成するセクションは次のとおりです。

- 『データ・ハンドラー開発過程の概要』
- 153 ページの『データ・ハンドラー開発用ツール』
- 155 ページの『データ・ハンドラーの設計』
- 156 ページの『データ・ハンドラー基本クラスの拡張』
- 156 ページの『メソッドのインプリメント』
- 171 ページの『カスタム・ネーム・ハンドラーの作成』
- 173 ページの『JAR ファイルへのデータ・ハンドラーの追加』
- 174 ページの『データ・ハンドラー・メタオブジェクトの作成』
- 176 ページの『その他のビジネス・オブジェクトのセットアップ』
- 177 ページの『コネクターの構成』
- 177 ページの『国際化データ・ハンドラー』

データ・ハンドラー開発過程の概要

カスタム・データ・ハンドラーを開発するには、データ・ハンドラーのソース・ファイルをコード化し、その他のタスク (例えば、データ・ハンドラーのメタオブジェクトの開発) を完了します。カスタム・データ・ハンドラーを作成するタスクには、次の一般的なステップが含まれます。

1. 直列化データの形式と、変換するビジネス・オブジェクトの構造に基づいて、データ・ハンドラーを設計します。
2. クラスを拡張するクラスを作成します。

   `com.crossworlds.DataHandlers.DataHandler`
3. 特定のデータ形式とビジネス・オブジェクト間のデータ変換を行う抽象メソッドをインプリメントします。
4. クラスをコンパイルし、それを `CustDataHandler.jar` ファイルへ追加します。
5. データ・ハンドラー・メタオブジェクトを作成します。
6. データ・ハンドラーの要件および呼び出し元の要件を満たすビジネス・オブジェクト定義を開発します。

図 33 に、データ・ハンドラーの開発過程の概要を図示します。この図は、特定のトピックに関する情報を見つけることができる各章のクイック・リファレンスです。開発チームがデータ・ハンドラーの開発に携わる場合は、チームのメンバーが分担して、データ・ハンドラー開発のいくつかの主要タスクを並行して行うことができます。

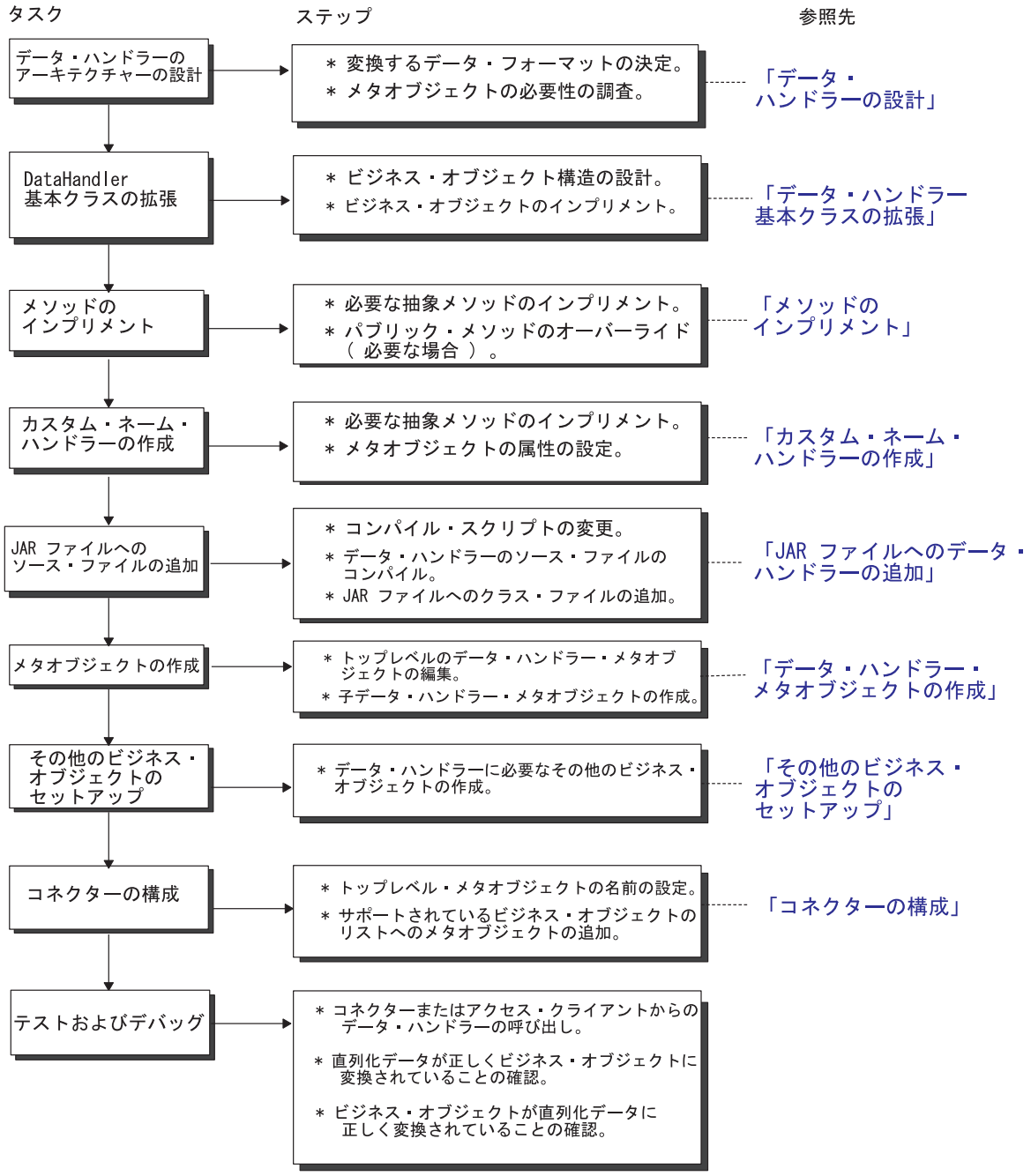

図 33. データ・ハンドラー開発過程の概要

データ・ハンドラー開発用ツール

データ・ハンドラーは Java で作成するので、Windows または UNIX システムで開発できます。表 45 に、IBM がデータ・ハンドラー開発用に提供しているツールをリストします。

表 45. データ・ハンドラー開発用ツール

開発ツール	説明
Adapter 開発キット (ADK)	次のものが含まれています。 • サンプル・データ・ハンドラー • `DataHandler` クラスを拡張するためのスタブ・ファイル
Data Handler API	カスタム・データ・ハンドラーを作成するために拡張する単一のクラス `DataHandler`
Java Connector Development Kit (JCDK)	ビジネス・オブジェクトを処理する Java クラスが含まれています。

Adapter 開発キット

Adapter 開発キット (ADK) は、アダプター開発を支援するファイルおよびサンプルを提供します。Adapter 開発キットは、多くの Object Discovery Agent (ODA)、コネクター、およびデータ・ハンドラーを含むアダプター・コンポーネントにサンプルを提供します。ADK が提供するサンプルは、製品ディレクトリーの DevelopmentKits サブディレクトリーにあります。

注: ADK は WebSphere Business Integration Adapters 製品の一部であり、専用の別個のインストーラーが必要です。そのため、ADK 内の開発サンプルを利用するためには、WebSphere Business Integration Adapters 製品をインストールした上で、ADK をインストールする必要があります。ADK が Windows システムのみで利用可能であることに注意してください。表 46 に、ADK がデータ・ハンドラー開発用に提供するサンプルと、それらが存在する DevelopmentKits ディレクトリーのサブディレクトリーをリストします。

表 46. データ・ハンドラー開発用 ADK サンプル

Adapter 開発キットのコンポーネント	DevelopmentKits サブディレクトリー
データ・ハンドラーのサンプル	`edk¥DataHandler`

サンプル・データ・ハンドラー

データ・ハンドラーの開発を支援するため、ADK では、次の製品ディレクトリーにいくつかのサンプル・データ・ハンドラー用のコードが格納されています。

```
DevelopmentKits¥edk¥DataHandler¥Samples
```

表 47 に、ADK から提供されるサンプル・データ・ハンドラーをリストします。

表 47. EDK に含まれているサンプル・データ・ハンドラー

名前	説明
delimited.java	ビジネス・オブジェクトを Delimited ストリングに変換し、Delimited ストリングをビジネス・オブジェクトに変換します。
fixedwidth.java	ビジネス・オブジェクトを FixedWidth ストリングに変換し、FixedWidth ストリングからビジネス・オブジェクトに変換します。
namevalue.java	ビジネス・オブジェクトを NameValue ストリングに変換し、NameValue ストリングをビジネス・オブジェクトに変換します。

注: これらのサンプルは検査目的には役立ちますが、DataHandler クラスでサポートされているすべての機能性の例を提供するわけではありません。

開発ファイル

DevelopmentKits¥edk¥DataHandler ディレクトリーには、カスタム・データ・ハンドラーの開発を支援するいくつかのファイルが入っています。これらのファイルを表 48 にリストします。

表 48. データ・ハンドラー開発ファイル

データ・ハンドラー開発ファイル	詳細
StubDataHandler.java	156 ページの『データ・ハンドラー基本クラスの拡張』
makeDataHandler.bat (Windows システム)	173 ページの『JAR ファイルへのデータ・ハンドラーの追加』

Data Handler API

Data Handler API は、DataHandler という単一の Java クラスを提供します。抽象 DataHandler 基本クラスにより、カスタム・データ・ハンドラーの開発が容易になります。このクラスには、入力データから抽出した値をビジネス・オブジェクトに取り込むメソッドや、ビジネス・オブジェクトをストリングまたはストリームへ直列化するメソッドが含まれています。また、クラスには、カスタム・データ・ハンドラーが使用できるユーティリティー・メソッドが含まれています。カスタム・データ・ハンドラーは、DataHandler クラスから派生させます。

DataHandler クラスに用意されているメソッドの詳細については、181 ページの『第 9 章 データ・ハンドラーの基本クラス・メソッド』を参照してください。

Java Connector Development Kit

ビジネス・オブジェクトを扱うために、データ・ハンドラーは Java Connector Development Kit (JCDK) のクラスからのメソッドを使用する必要があります。データ・ハンドラーを開発するときには、追加の JCDK クラス (CxCommon.CxObjectContainerInterface や CxCommon.CxObjectAttr など) をインポ

ートする必要があります。 JCDK メソッドの参照情報については、WebSphere Business Integration Adapters 資料セットの「*コネクター開発ガイド (Java 用)*」を参照してください。

注: JCDK は、下位の Java コネクター・ライブラリーです。そのメソッドの資料は、「*コネクター開発ガイド (Java 用)*」の個別のパーツに含まれています。

データ・ハンドラーの設計

カスタム・データ・ハンドラーの記述を始める前に、次のことを明確に理解しておくことをお勧めします。

- データ・ハンドラーが変換するファイルのデータ形式
- ビジネス・オブジェクト・モデル

  特に、次の方法を確認しておく必要があります。
  - ビジネス・オブジェクト・インスタンスからの値の抽出
  - ビジネス・オブジェクト・インスタンスの作成と、ファイルからの値の取り込み

メタデータ主導型データ・ハンドラーの作成

カスタム・データ・ハンドラーをメタデータ主導型にするには、使用するデータ・ハンドラーを識別する情報を動的に指定する必要があります。メタデータ主導型データ・ハンドラーの詳細については、22 ページの『メタデータ主導型ハンドラーの設計』を参照してください。

データ・ハンドラー・メタオブジェクトの使用

設計上必要な決定事項の 1 つに、データ・ハンドラーがその構成を初期化するためにメタオブジェクトを使用するかどうかの決定があります。

注: メタオブジェクトの詳細については、25 ページの『データ・ハンドラーの構成』を参照してください。

メタオブジェクトを使用するかどうかを決定するときには、次のことを考慮してください。

- メタオブジェクトを使用すると、データ・ハンドラーを動的に構成できます。この設計戦略により、呼び出されるコンテキストに基づいて構成できる柔軟なデータ・ハンドラーを作成できます。

  メタオブジェクトを使用するデータ・ハンドラーを呼び出すには、呼び出し元がデータ・ハンドラーの関連付けられた MIME タイプを `createHandler()` メソッドに渡します。 MIME タイプで呼び出されると、`createHandler()` は、新たにインスタンス化されたデータ・ハンドラーを、子メタオブジェクトの構成情報を使用して初期化します。
- メタオブジェクトのアクセスと検索に関連するオーバーヘッドがあります。データ・ハンドラーでは、その構成情報が変更されない場合 (ハードコーディングが可能)、または関連付けられた MIME タイプを持たないデータを変換する場合に、メタオブジェクトを回避できます。

メタオブジェクトを*使用しない* データ・ハンドラーを呼び出すには、呼び出し元がデータ・ハンドラーのクラス名のみ を createHandler() メソッドに渡します。クラス名で呼び出されると、createHandler() は、そのクラスのデータ・ハンドラーをインスタンス化し、関連付けられたメタオブジェクトを検索しません。

メタオブジェクトを使用するようにカスタム・データ・ハンドラーを設計する場合は、これらのメタオブジェクトを、データ・ハンドラー・インプリメンテーションの一部として作成する必要があります。詳細については、174 ページの『データ・ハンドラー・メタオブジェクトの作成』を参照してください。

データ・ハンドラー基本クラスの拡張

カスタム・データ・ハンドラーを作成するには、データ・ハンドラー基本クラス (DataHandler) を拡張して、独自のデータ・ハンドラー・クラス を作成します。DataHandler クラスには、開発支援のためのユーティリティー・メソッドのほか、変換 (ストリングからビジネス・オブジェクトへ、およびビジネス・オブジェクトからストリングへ) を実行するメソッドが含まれます。EDK には、カスタム・データ・ハンドラー用のスタブ・コードと makefile が含まれています。スタブ・ファイルには、空のクラスを定義する Java コードが含まれています。この空のクラスには、インプリメントする必要がある全メソッドがリストされています。このスタブ・ファイルをテンプレートとして使用して、カスタム・データ・ハンドラーを作成することができます。

スタブ・ファイルを使用してデータ・ハンドラーのソース・ファイルを作成するには、次のようにします。

1. StubDataHandler.java ファイルをコピーし、その名前が定義するデータ・ハンドラー・クラスの名前と一致するように、名前を変更します。

   スタブ・ファイルは、製品ディレクトリー内の DevelopmentKits¥edk¥DataHandler サブディレクトリー内にあります。これには、データ・ハンドラー・パッケージ com.crossworlds.DataHandlers をインポートする import ステートメントが含まれています。また、Java Connector Development Kit からもいくつかのクラスをインポートします。
2. StubDataHandler キーワードを、カスタム・データ・ハンドラーをインプリメントするクラスの名前に変更します。

   例えば、次の行では、DataHandler クラスを拡張して、HtmlDataHandler という名前のカスタム・データ・ハンドラー・クラスを作成します。

   ```
   public class HtmlDataHandler extends DataHandler
   ```

メソッドのインプリメント

データ・ハンドラーを開発するには、DataHandler クラスの次のメソッドをインプリメントします。

- 抽象 DataHandler メソッド (必須)
- パブリック DataHandler メソッド (オプション)

カスタム・データ・ハンドラーのメソッドは、 156 ページの『データ・ハンドラー基本クラスの拡張』で作成した DataHandler クラスの Java ソース・ファイルに入ります。

注: 呼び出し元が複数のスレッドに対して DataHandler クラスのキャッシュ・インスタンスを再利用する場合は、クラス・スレッド・セーフを作成する必要があります。これが必要かどうかを判断するためのスレッド化モデルの詳細については、「*コネクター開発ガイド (Java 用)*」を参照してください。

抽象メソッドのインプリメント

データ・ハンドラー基本クラス DataHandler には、表 49 にある抽象メソッドがあります。これらは、カスタム・データ・ハンドラーの DataHandler クラスにインプリメントする必要があります。

表 49. DataHandler クラスの抽象メソッド

データ変換	直列化データの形式	DataHandler メソッド
ストリングからビジネス・オブジェクトへの変換	Reader オブジェクトを使用して、直列化データをビジネス・オブジェクトに変換します。	getBO() - 抽象
ビジネス・オブジェクトからストリングへの変換	ビジネス・オブジェクトを InputStream オブジェクトに変換します。	getStreamFromBO()
	ビジネス・オブジェクトを String オブジェクトに変換します。	getStringFromBO()
	ビジネス・オブジェクトをバイト配列に変換します。	getByteArrayFromBO()

注: カスタム・データ・ハンドラーを使用して DataHandler クラスを拡張する StubDataHandler.java ファイルのコピーには、インプリメントする必要がある抽象メソッドの宣言が含まれています。

次のセクションでは、各抽象 DataHandler メソッドのインプリメンテーション情報を提供します。

ビジネス・オブジェクトへの変換のインプリメント

抽象 getBO() メソッドでは、ストリングからビジネス・オブジェクトへの変換を実行します。つまり、Java Reader オブジェクトから抽出したデータをビジネス・オブジェクトに取り込みます。 getBO() メソッドには、次のように 2 つのバージョンがあります。

- getBO(Reader*serializedData*, BusinessObjectInterface*theObj*, Object*config*)

  入力引き数には、直列化データとビジネス・オブジェクトへの参照を含む Reader オブジェクトが組み込まれます。メソッドは、*theBusObj* ビジネス・オブジェクトに、*serializedData* データを取り込みます。

- getBO(Reader*serializedData*, Object*config*)

入力引き数には、直列化データを含む Reader オブジェクトが組み込まれます。このメソッドは、データからビジネス・オブジェクト・タイプ (ビジネス・オブジェクト定義) の名前を判別し、そのタイプのビジネス・オブジェクト・インスタンスを作成および充てんします。

注: コネクターのコンテキストで呼び出されるデータ・ハンドラーをサポートするには、(DataHandler を拡張する) データ・ハンドラー・クラスに getBO() メソッドの*両方* のバージョンをインプリメントする必要があります。アクセス・クライアントからのみ 呼び出されるデータ・ハンドラーをサポートするには (IBM WebSphere InterChange Server 統合ブローカーのみ)、getBO() メソッドの 2 番目の形式のみをインプリメントする必要があります。Server Access Interface では、getBO() のこの 2 番目の形式のみを使用します。

getBO() メソッドを使用すると、呼び出し元は、構成情報 (*config* パラメーター) が含まれたオプションのオブジェクトを渡すことができます。この情報は、データ・ハンドラーのメタオブジェクトで指定された構成情報に加えられるものです。例として、構成オブジェクトは、テンプレート・ファイルや、データ・ハンドラーが使用する URL を指すことができます。

注: ビジネス・オブジェクトに変換する場合、getBO() メソッドは、親ビジネス・オブジェクトのアプリケーション固有情報に記述されている cw_mo_*label* タグで識別できるすべての属性に値が*取り込まれていない* ことを確認する必要があります。 cw_mo_*label* タグの詳細については、163 ページの『ビジネス・オブジェクトからの変換のインプリメント』を参照してください。

抽象 getBO() メソッドの目的は、Reader オブジェクトに含まれる直列化データをビジネス・オブジェクトに取り込むことです。これにより、getBO() のパブリック・バージョンはサポートされるいずれかの形式で直列化データを受け取り、そのデータを Reader オブジェクトに変換し、抽象 getBO() メソッドを呼び出して、ストリングからビジネス・オブジェクトへの変換を実際に実行することができます。パブリック getBO() メソッドについての詳細は、185 ページの『getBO() - パブリック』 を参照してください。

図 34 に、getBO() メソッドの 2 番目の形式の基本的なインプリメンテーションを示します。この例には、固定幅データを含む Reader オブジェクトからのデータをビジネス・オブジェクトに変換するときのステップを示しています。

1. getBO() メソッドは、Reader オブジェクトのデータを String オブジェクトに変換します。次に、ユーザー定義の getBOFromString() メソッドを呼び出して、ビジネス・オブジェクトのインスタンスを作成します。
2. getBOFromString() メソッドは、String 内の最初の固定幅トークンに基づいて作成するビジネス・オブジェクトのタイプを判別し、そのタイプのビジネス・オブジェクト・インスタンスを作成します。String 内の 2 番目の固定幅トークンから動詞を取得し、その動詞をビジネス・オブジェクトに設定します。このメソッドは次に、ユーザー定義の parseAttributeList() メソッドを呼び出して、String の残りの部分を解析し、ビジネス・オブジェクトに値を取り込みます。
3. parseAttributeList() メソッドは、String を解析し、ビジネス・オブジェクトに再帰的に取り込みます。メソッドは、オブジェクト・タイプの属性を検出したら、オブジェクトが単一カーディナリティーか複数カーディナリティーかを判別します。さらに、getMultipleCard() を呼び出して、配列内のビジネス・オブジ

エクトを再帰的に処理し、getSingleCard() を呼び出して、単一カーディナリティーの子ビジネス・オブジェクトを処理します。

ヒント: データ・ハンドラーは、ビジネス・オブジェクトからデータを抽出し、コネクターが行うのと同じ方法でデータをビジネス・オブジェクトに取り込みます。例えば、次のサンプル・コードでは、getBOFromString() メソッドが JavaConnectorUtil.createBusinessObject() を呼び出してビジネス・オブジェクト・インスタンスを作成し、BusinessObjectInterface.setVerb() を呼び出して動詞を設定しています。ビジネス・オブジェクトの処理方法については、「*コネクター開発ガイド (Java 用)*」を参照してください。

```
public BusinessObjectInterface getBO(Reader serializedData,
    Object config)
    throws Exception
{
    clear(config);
    BusinessObjectInterface resultObj = null;

    // Create a String object from the Reader, then use the string
    // method
    int conversionCheck;
    char[] temp = new char[2000];
    StringBuffer tempStringBuffer = new StringBuffer(1000);

    while ( (conversionCheck = serializedData.read(temp)) != -1 )
        tempStringBuffer.append(new String (temp, 0, conversionCheck));

    mBOString = new String(tempStringBuffer);
    mBOStringLength = mBOString.length();

    resultObj = getBOFromString(null);
    return resultObj;
}


// Gets business object name and verb and creates a bus obj instance
private BusinessObjectInterface getBOFromString(String pvBOType)
    throws Exception
{
    BusinessObjectInterface returnObj = null;
    String lvBOName = null;
    String lvVerb = null;

    lvBOName = this.getNextToken(mBONameSize, true);
    lvVerb = this.getNextToken(mBOVerbSize, true);

    if( (pvBOType != null) && (lvBOName.compareTo(pvBOType) != 0) ) {
        throw new Exception(...);
    }
    else
    {
        returnObj = JavaConnectorUtil.createBusinessObject(lvBOName);
    }

    returnObj.setVerb(lvVerb);

    parseAttributeList(returnObj);

    return returnObj;
}
```

図 34. getBO() メソッドの例 (1/4)

```
// Parse String to populate the attributes in the business object
protected void parseAttributeList(BusinessObjectInterface pvBO)
    throws Exception
{
    if ( mEndOfBOString )
        throw new Exception(...);
    else if( pvBO == null )
        throw new Exception(...);

    int lvAttrNum = pvBO.getAttrCount();

    String lvAttrName = null;
    String lvAttrValue = null;
    int lvAttrMaxLength = 0;

    try {
        for (int lvAttrIndex = 0; lvAttrIndex < lvAttrNum;
                lvAttrIndex++)
        {
            CxObjectAttr lvAttrSpec = pvBO.getAttrDesc(lvAttrIndex);
            lvAttrName = lvAttrSpec.getName();

            // Determine if the attribute is a simple attribute or a
            // business object container.
            if (lvAttrSpec.isObjectType())
            {
                // Get the next token based on the BOCountSize
                lvAttrMaxLength = mBOCountSize;
                lvAttrValue =  this.getNextToken(mBOCountSize, true);
                String lvBOType = lvAttrSpec.getTypeName();
                Object lvAttrBOValue = null;

                if (lvAttrSpec.isMultipleCard())
                {
                    this.getMultipleCard(pvBO,lvAttrIndex,lvBOType,
                        lvAttrValue);
                }
                else {
                    this.getSingleCard(pvBO,lvAttrIndex,lvBOType,
                        lvAttrValue);
                }
            }
            else
            {
                // Get the next token based on the MaxLength of the attribute
                lvAttrMaxLength = lvAttrSpec.getMaxLength();
                if (lvAttrMaxLength > 0)
                    lvAttrValue =  this.getNextToken(lvAttrMaxLength, false);
                else
                    lvAttrValue = null;
```

図 34. getBO() メソッドの例 (2/4)

```
                // For simple String attribute, set to null, set to
                // configured CxIgnore or CxBlank values, or set to the
                // attribute value
                if (lvAttrValue == null )
                    pvBO.setAttrValue(lvAttrIndex, null);
                else if (lvAttrValue.equals(mCxIgnore)||
                      lvAttrValue.equals(""))
                    pvBO.setAttrValue(lvAttrIndex, null);
                else if (lvAttrValue.equals(mCxBlank)||
                      lvAttrValue.equals(" "))
                    pvBO.setAttrValue(lvAttrIndex, "");
                else
                    pvBO.setAttrValue(lvAttrIndex, lvAttrValue);
            }
        }
    }
}


// Populates a child container with values in the String
protected void getMultipleCard(BusinessObjectInterface pvParentBO,
    int pvParentAttrIndex, String pvBOType, String pvObjectCountString)
    throws CW_BOFormatException, Exception
{
    try {
        if ( pvObjectCountString.equals(mCxIgnore) )
        {
            // trace message
        }
        else {
            int lvObjectCount = Integer.parseInt(pvObjectCountString);
            if ( lvObjectCount == 0)
            {
                // trace message with the number of objects in container
            }
            else if (lvObjectCount > 0)
            {
                // There is at least one instance of the object in the string
                BusinessObjectInterface lvChildBO = null;

                // For each instance of the child object, parse the attribute
                // list, and then add the object container to the parent.
                for (int lvObjectIndex = 0; lvObjectIndex < lvObjectCount;
                      lvObjectIndex++)
                {
                    lvChildBO = getBOFromString(pvBOType);
                    pvParentBO.setAttrValue(pvParentAttrIndex,lvChildBO);
                }
            }
        }
    }
}
```

図 34. getBO() メソッドの例 (3/4)

```
// Populates a single cardinality child with values in the String
protected void  getSingleCard(BusinessObjectInterface pvParentBO,
    int pvParentAttrIndex, String pvBOType, String pvObjectCountString)
    throws CW_BOFormatException, Exception
{
    try {
        BusinessObjectInterface lvChildBO = null;

        // Check the object count token
        // If it does not match "1", assume that the child object should
        //  be null
        if (pvObjectCountString.equals("1"))
        {
            // The string contains a single instance of the child
            lvChildBO = getBOFromString(pvBOType);
            pvParentBO.setAttrValue(pvParentAttrIndex, lvChildBO);
        }
        else if ( pvObjectCountString.equals(mCxIgnore) ||
              pvObjectCountString.equals("0"))
        {
            // Validate that the object count token is valid
        }
        else
            throw new CW_BOFormatException(...);
    }
}
```

図 34. getBO() メソッドの例 (4/4)

ビジネス・オブジェクトからの変換のインプリメント

表 50 の抽象メソッドは、ビジネス・オブジェクトからストリングへの変換を実行します。つまり、ビジネス・オブジェクトから特定の形式の直列化データを作成します。

表 50. ビジネス・オブジェクトからストリングへの変換をインプリメントするための抽象メソッド

抽象メソッド	説明
getStringFromBO()	ビジネス・オブジェクト内のデータを String オブジェクトに変換します。
getStreamFromBO()	ビジネス・オブジェクト内のデータを InputStream オブジェクトに変換します。
getByteArrayFromBO()	ビジネス・オブジェクト内のデータをバイト配列に変換します。

ビジネス・オブジェクトから変換する目的は、ビジネス・オブジェクト内のデータをすべて直列化形式にすることです。ただし、場合によっては、一部のビジネス・オブジェクト・データを直列化データに含めてはならない こともあります。例えば、ビジネス・オブジェクトは、子メタオブジェクトを使用して、コネクターの動的構成情報を保持することがあります。

IBM では、構成メタデータまたは動的メタデータ (あるいはその両方) 用に、プレフィックス cw_mo_*label* で始まるアプリケーション固有情報をすべて予約しています。データ・ハンドラーがビジネス・オブジェクトからの変換時に無視する 必要が

ある属性を示すために、親ビジネス・オブジェクトのビジネス・オブジェクト定義では、そのアプリケーション固有情報内に次のタグを指定します。

cw_mo_*label* =*child_meta-object_attribute_name*

label は、cw_mo_ タグの目的を詳細に指定するために定義するストリングです。また、*child_meta-object_attribute_name* は、無視する属性の名前を表しています。この属性には通常、子メタオブジェクトが含まれています。cw_mo_*label* タグを複数指定する場合は、セミコロン (;) で区切ります。

getStringFromBO()、getStreamFromBO()、および getByteArrayFromBO() メソッドをカスタム・データ・ハンドラー用にインプリメントするときには、これらのメソッドが次のようにして、データ・ハンドラーにコネクター固有の属性をスキップオーバーさせる必要があります。

1. 変換の対象となるビジネス・オブジェクトのビジネス・オブジェクト定義におけるアプリケーション固有情報の中に、cw_mo_*label* タグが存在するかどうかを確認します (*label* は、無視する属性を指定するためにユーザーが入力するストリングです)。
2. cw_mo_*label* タグが存在する場合は、このタグが示すストリング (*child_meta-object_attribute_name*) を検索します。等号 (=) を囲む空白文字は無視してください。
3. ビジネス・オブジェクトの属性をループするときには、各属性名を *child_meta-object_attribute_name* ストリングと比較してください。この名前を持つ属性をスキップオーバーします。

次のコードは、属性をスキップオーバーする方法を示しています。

```
List configObjList =
     com.crossworlds.DataHandlers.text.namevalue.listMOAttrNames(BusObj);

//this list contains attribute names, which are configuration objects
for ( attributes in BusObj )
{
          String attrName = BusObj.getAttrDisc(i).getName();
          if ( configObjList.contains(attrName) )
          {
            //skip
            continue;
          }
}
```

例えば、MyCustomer という名前のビジネス・オブジェクトが、子メタオブジェクトを使用して、コネクター固有のルーティング情報を保持しているとします。このメタオブジェクトが CustConfig という名前の属性で表されている場合、MyCustomer には、そのアプリケーション固有情報に次のタグが含まれている可能性があります。

cw_mo_cfg=CustConfig

カスタム・データ・ハンドラーは、ビジネス・オブジェクトからの変換時にアプリケーション固有情報を調べて、MyCustomer に関連付けられているビジネス・オブジェクト定義の有無を検査します。次に、cw_mo_cfg タグを検索して、CustConfig 属性のスキップオーバーが必要かどうかを判別します。データ・ハンドラーから結果として生じた直列化データには、CustConfig 子メタオブジェクトは含まれていません。

注: IBM から提供されたデータ・ハンドラーは、ビジネス・オブジェクトからの変換時に cw_mo_*label* タグで指定されたすべての 属性をスキップオーバーします。

カスタムのデータ・ハンドラーを開発して cw_mo_*label* タグを処理することが必要なのは、このデータ・ハンドラーが子メタオブジェクトやその他の動的オブジェクトを使用する場合のみ です。

getStringFromBO() メソッドのインプリメンテーション: getStringFromBO() メソッドは、ビジネス・オブジェクトからストリングへの変換を実行します。つまり、ビジネス・オブジェクトのデータを String オブジェクトに変換します。このメソッドの場合、呼び出し元は、変換するビジネス・オブジェクトを受け渡します。図 35 に、FixedWidth データ・ハンドラーによってインプリメントされた getStringFromBO() メソッドを示します。このメソッドは、固定幅フィールドの String を作成します。

この例は、ビジネス・オブジェクトから Reader オブジェクトへのデータ変換のステップを示しています。

1. getStringFromBO() メソッドは、setAttrList() を呼び出して、ビジネス・オブジェクト内の属性を再帰的にループします。setAttrList() メソッドが単純な属性を検出すると、setSimpleToken() メソッドを呼び出して、値を設定します。
2. setSimpleToken() メソッドは、属性値を StringBuffer オブジェクトに追加し、StringBuffer を String オブジェクトに変換して、ビジネス・オブジェクト全体を表す StringBuffer へストリングを追加します。setAttrList() がビジネス・オブジェクトを処理した時、getStringFromBO() メソッドは、StringBuffer を String オブジェクトへ変換し、String を呼び出し元へ戻します。

```
public String getStringFromBO(BusinessObjectInterface theObj,
    Object config)
    throws Exception
{
    traceWrite(
      "Entering getStringFromBO(BusinessObjectInterface, Object) "
      + " for object type " + theObj.getName(),
      JavaConnectorUtil.LEVEL4);

    clear(config);
    String lvBOString = null;
    setAttrList(theObj);
    lvBOString = mBOStringBuffer.toString();

    traceWrite(
      "Exiting getStringFromBO(BusinessObjectInterface, Object) "
       + " for object type " + theObj.getName(),
       JavaConnectorUtil.LEVEL4);

    return lvBOString;
}


protected void setAttrList(BusinessObjectInterface pvBO) throws Exception
{
    traceWrite(
      "Entering setAttrList(BusinessObjectInterface) for object "
      + pvBO.getName(), JavaConnectorUtil.LEVEL4);

    int lvAttrNum = pvBO.getAttrCount();

    String lvAttrName = null;
    String lvAttrValue = null;
    int lvAttrMaxLength = 0;

    // Add the business object name and verb to the fixed width format
    //  String
    this.setSimpleToken( mBONameSize, pvBO.getName());
    this.setSimpleToken( mBOVerbSize, pvBO.getVerb());

    try {
        List moAttrNames = listMOAttrNames( pvBO );
        int lvAttrCount = pvBO.getAttrCount();

        ATTRIBUTE_WALK: for (int lvAttrIndex = 0;
              lvAttrIndex < lvAttrCount; ++lvAttrIndex)
        {
            CxObjectAttr lvAttrSpec = pvBO.getAttrDesc(lvAttrIndex);
            lvAttrName = lvAttrSpec.getName();

            // Check if the current attribute is a simple (String) type
            //  or a contained object.
            if (lvAttrSpec.isObjectType())
            {
                //skip child objects designated as meta objects
                if( moAttrNames.contains( lvAttrName ) )
                {
                    continue ATTRIBUTE_WALK;
                }
```

図 35. getStringFromBO() メソッドの例 (1/5)

```
if (lvAttrSpec.isMultipleCard())
{
    CxObjectContainerInterface lvAttrMultiCardBOValue =
  (CxObjectContainerInterface) pvBO.getAttrValue(lvAttrIndex);

    if (lvAttrMultiCardBOValue == null)
    {
        traceWrite(
    "setAttrList found empty multiple cardinality container "
    + lvAttrSpec.getTypeName(), JavaConnectorUtil.LEVEL5);

        // Add the count to the fixed width String
        this.setSimpleToken( mBOCountSize, "0");
    }
    else
    {
        int lvObjectCount =
          lvAttrMultiCardBOValue.getObjectCount();
        traceWrite(
      "setAttrList found multiple cardinality container "
      + lvAttrSpec.getTypeName() + " with "
      + lvObjectCount + " instances",
      JavaConnectorUtil.LEVEL5);

        // Add the count to the fixed width String
        this.setSimpleToken( mBOCountSize,
          Integer.toString(lvObjectCount));

        // Add each object in the container to the fixed
        //  width String.
        for (int lvContObjectIndex = 0;
              lvContObjectIndex < lvObjectCount;
              ++lvContObjectIndex)
            setAttrList(
        lvAttrMultiCardBOValue.getBusinessObject(
            lvContObjectIndex));
    }
}
else
{
    BusinessObjectInterface lvAttrSingleCardBOValue =
  (BusinessObjectInterface) pvBO.getAttrValue(lvAttrIndex);

    if (lvAttrSingleCardBOValue == null)
    {
         traceWrite(
    "setAttrList found empty single cardinality container "
    + lvAttrSpec.getTypeName(), JavaConnectorUtil.LEVEL5);

         // Add the count to the fixed width String
         this.setSimpleToken( mBOCountSize, "0");
    }
```

図 35. getStringFromBO() メソッドの例 (2/5)

```
                else
                {
                    traceWrite(
                  "setAttrList found single cardinality container "
                  + lvAttrSpec.getTypeName(),
                  JavaConnectorUtil.LEVEL5);

                    // Add the count to the fixed width String
                    this.setSimpleToken( mBOCountSize, "1");
                    setAttrList(lvAttrSingleCardBOValue);
                }
            }
        }
        else
        {
            lvAttrValue = (String) pvBO.getAttrValue(lvAttrIndex);
            lvAttrMaxLength = lvAttrSpec.getMaxLength();

            if (lvAttrMaxLength > 0)
                this.setSimpleToken(lvAttrMaxLength, lvAttrValue);
           }
        }
    }
    catch (CxObjectNoSuchAttributeException e)
    {
        throw new Exception(e.getMessage());
    }

    traceWrite(
        "Exiting setAttrList(BusinessObjectInterface) for object "
        + pvBO.getName(), JavaConnectorUtil.LEVEL4);
}


protected void setSimpleToken( int pvCellSize, String pvTokenValue)
    throws Exception
{
    traceWrite( "Entering setSimpleToken(int, String)",
      JavaConnectorUtil.LEVEL4);

    StringBuffer lvNewBuffer = new StringBuffer(pvCellSize);
    int lvTokenLength = 0;
    int lvCxIgnoreLength = mCxIgnore.length();
    int lvCxBlankLength = mCxBlank.length();

    int lvPadNumber = 0;


    // Check the token value to see if it is null
    if (pvTokenValue == null)
    {
        // For this case, we add the configured CxIgnore value to the
        //  fixed width String if it fits in the cell.
        if (!mTruncation && lvCxIgnoreLength > pvCellSize)
            throw new Exception(
                " Null attribute value encountered where cell size is "
                + pvCellSize + ", size of CxIgnore value is "
                + lvCxIgnoreLength + "and trucation is not allowed. "
                + "Please check your MO format configuration.");
```

図 35. getStringFromBO() メソッドの例 (3/5)

```
        else
        {
            lvPadNumber = pvCellSize - lvCxIgnoreLength;
            lvNewBuffer.append(mCxIgnore);
        }

    }
    else if (pvTokenValue.equals(""))
    {
        // For this case, the configured CxBlank value is added to the
        //  fixed width String if it fits in the cell.
        if (! mTruncation && lvCxBlankLength > pvCellSize)
            throw new Exception(
              " Blank attribute value encountered where cell size is "
              + pvCellSize + ", size of CxBlank value is "
              + lvCxBlankLength + "and trucation is not allowed. "
              + "Please check your MO format configuration.");
        else
        {
            lvPadNumber = pvCellSize - lvCxBlankLength;
            lvNewBuffer.append(mCxBlank);
        }
    }
    else
    {
        // For this case, actually add the token value to the fixed
        // width String, unless the data is too long for the cell.
        lvTokenLength = pvTokenValue.length();
        if (!mTruncation && lvTokenLength > pvCellSize )
            throw new Exception(
              " Attribute value encountered where cell size is "
              + pvCellSize + ", size of token value is "
              + lvTokenLength + "and trucation is not allowed. "
              + "Please check your MO format configuration.");
        else
        {
            lvNewBuffer.append(pvTokenValue);
            lvPadNumber = pvCellSize - lvTokenLength;
        }
    }

    if (lvPadNumber <= 0 && mTruncation)
        // Token is longer than the cell and truncation is allowed,
        //  so the characters up to pvCellSize are added
        lvNewBuffer.setLength(pvCellSize);
    else if (lvPadNumber > 0)
    {
        // Pad the cell based on the configuration option chosen
        if ( mAlignment.equals(fixedwidth.AlignmentLeft) ||
              mAlignment.equals(fixedwidth.AlignmentBoth))
            this.padRight(lvNewBuffer, lvPadNumber);
        else if (mAlignment.equals(fixedwidth.AlignmentRight))
            this.padLeft(lvNewBuffer, lvPadNumber);
    }
```

図 35. getStringFromBO() メソッドの例 (4/5)

```
    String lvNewBuffString =  lvNewBuffer.toString();

    // Note that this may cause a performance issue when the tracing
    //  level is low, but in most cases it should not as any one token
    //  is *usually* not very long.
    traceWrite(
      "Adding the following token to the fixed width String: "
      + lvNewBuffString, JavaConnectorUtil.LEVEL5);

    // After the cell has been fully formatted, append to fixed width
    //  String being built
    mBOStringBuffer.append(lvNewBuffString);

    traceWrite( "Exiting setSimpleToken(int, String)",
      JavaConnectorUtil.LEVEL4);
}
```

図 35. getStringFromBO() メソッドの例 (5/5)

getStreamFromBO() メソッドのインプリメンテーション: getStreamFromBO() メソッドは、ビジネス・オブジェクトのデータを InputStream オブジェクトに変換します。図 36 に、getStreamFromBO() メソッドのインプリメンテーションを示します。このインプリメンテーションでは、getStreamFromBO() が getStringFromBO() を呼び出して、ビジネス・オブジェクト・データを含む String オブジェクトを作成し、次に String を InputStream へ変換します。メソッドは、ビジネス・オブジェクト内のデータを表す InputStream オブジェクトを戻します。

```
public InputStream getStreamFromBO(BusinessObjectInterface theObj,
    Object config)
    throws Exception
{
    clear(config);

    String BOstring;
    BOstring = getStringFromBO(theObj, config);

    return new ByteArrayInputStream( BOstring.getBytes() );
}
```

図 36. getStreamFromBO() メソッドの例

パブリック・メソッドのオーバーライド

抽象 DataHandler メソッド (インプリメントする必要がある) に加えて、DataHandler クラスの一部のパブリック・メソッド (表 51 を参照) についても、カスタム・データ・ハンドラーで最適に動作するようにオーバーライドする必要があります。

表 51. DataHandler クラスのパブリック・メソッド

パブリック DataHandler メソッド	説明
getBO() - パブリック	(いずれかの形式の) 直列化データをビジネス・オブジェクトに変換します。
getBOName()	直列化データからビジネス・オブジェクトの名前を取得します。

表 51. *DataHandler* クラスのパブリック・メソッド *(続き)*

パブリック DataHandler メソッド	説明
getBooleanOption()	データ・ハンドラーから Boolean 構成オプションの値を取得します。
getOption()	データ・ハンドラーから構成オプションの値を取得します。
setOption()	データ・ハンドラーに構成オプションを設定します。
traceWrite()	データ・ハンドラーの適切なコンテキスト用にトレース書き込みメソッドを呼び出します。このとき、コンテキストはコネクターまたは Server Access Interface です。

カスタム・ネーム・ハンドラーの作成

データ・ハンドラーは、ネーム・ハンドラー を呼び出して、直列化データからビジネス・オブジェクト定義の名前を抽出できます。このタスクは、データ・ハンドラーの呼び出し元が、直列化データを取り込むビジネス・オブジェクトを受け渡さない ときに、ストリングからビジネス・オブジェクトへの変換時に必要となります。この場合は、データ・ハンドラーがビジネス・オブジェクトを作成してから、取り込みが可能になります。ビジネス・オブジェクトを作成するには、データ・ハンドラーが関連したビジネス・オブジェクト定義の名前を認識しておく必要があります。このビジネス・オブジェクト名を取得するのはネーム・ハンドラーです。

注: 現在、XML データ・ハンドラーは、ネーム・ハンドラーを使用して、作成するビジネス・オブジェクトの名前を取得しています。

カスタム・ネーム・ハンドラーを作成およびインプリメントするタスクには、次の一般的なステップが含まれます。

1. `NameHandler` クラスを拡張するクラスを作成します。
2. 直列化データを読み取り、ビジネス・オブジェクト名を戻す抽象 `getBOName()` メソッドをインプリメントします。
3. クラスをコンパイルし、それを `DataHandlers¥CustDataHandler.jar` ファイルへ追加します。詳細については、 173 ページの『JAR ファイルへのデータ・ハンドラーの追加』を参照してください。
4. `NameHandlerClass` 属性のデフォルト値をデータ・ハンドラー用のメタオブジェクトに設定します。

NameHandler クラスの拡張

カスタム・ネーム・ハンドラーを作成するには、ネーム・ハンドラー基本クラス (`NameHandler`) を拡張して、独自のネーム・ハンドラー・クラス を作成します。`NameHandler` クラスには、直列化データからビジネス・オブジェクトの名前を抽出するメソッドが含まれます。このネーム・ハンドラー基本クラスのパッケージは、`com.crossworlds.DataHandlers.NameHandler` です。

ネーム・ハンドラー・クラスを派生させるには、以下のステップを実行します。

1. NameHandler クラスを拡張するネーム・ハンドラー・クラスを作成します。
2. 次のコマンドを入力して、ネーム・ハンドラー・クラス・ファイルに NameHandler パッケージのクラスをインポートします。

import

3. NameHandler クラス内の抽象メソッドである getBOName() をインプリメントします。詳細については、『getBOName() メソッドのインプリメント』を参照してください。

NameHandler クラスの定義は、次のとおりです。

```
// Imports
import java.lang.String;
import java.io.Reader;
import com.crossworlds.DataHandlers.Exceptions.MalformedDataException;

public abstract class NameHandler {

// Constructors
public NameHandler() { }

// Methods
public abstract String getBOName(Reader serializedData,
String boPrefix)
throws MalformedDataException;
}

/* This method was introduced so that the NameHandler would have
 * a reference to the DataHandler. The DataHandler base calls this
 * method after it instantiated the NameHandler:
 * eg. nh = (NameHandler)Class.forName(className).newInstance();
 *     nh.setDataHandler(this);
*/
public final void setDataHandler( DataHandler dataHandler )
{
dh = dataHandler;
}
```

独自のネーム・ハンドラーを作成するには、NameHandler 抽象基本クラスを拡張します。

getBOName() メソッドのインプリメント

NameHandler クラスを拡張するには、getBOName() メソッドをインプリメントする必要があります。このメソッドは、直列化データを読み取り、そのデータに関連するビジネス・オブジェクトの名前を戻します。このメソッドの構文は次のとおりです。

```
public abstract String getBOName(Reader serializedData, String BOPrefix)
throws MalformedDataException
```

ここで、

serializedData
: メッセージを含むオブジェクトへの参照。

BOPrefix
: ビジネス・オブジェクト定義の名前のビジネス・オブジェクト・プレフィックスが含まれている String 値。この引き数は、メタオブジェクト (存在する場合) 内の BOPrefix 属性に設定できます。

メタオブジェクト属性の設定

カスタム・ネーム・ハンドラーを使用するようにデータ・ハンドラーに通知するには、メタオブジェクト属性の Default Value プロパティーを完全クラス名に設定す

る必要があります。データ・ハンドラーはこの後、構成オプションの 1 つから実行時にクラス名を取得できます。デフォルトでは、このメタオブジェクト属性は NameHandlerClass という名前です。 XML のデータ・ハンドラーに関連付けられた子メタオブジェクトには、この属性が含まれています。IBM からの納入時、この属性のデフォルト値として、デフォルトのネーム・ハンドラー・クラスの名前が指定されています。データ・ハンドラーにカスタム・ネーム・ハンドラーを使用させるには、拡張するデータ・ハンドラーに関連付けられた子メタオブジェクトで、NameHandlerClass 属性の Default Value プロパティーを更新する必要があります。

JAR ファイルへのデータ・ハンドラーの追加

新規データ・ハンドラーのコードを完成させたら、クラスをコンパイルし、それを Java アーカイブ・ファイル (jar) に追加する必要があります。カスタム・データ・ハンドラーを格納するために、ファイル CustDataHandler.jar が提供されます。この JAR ファイルは、製品ディレクトリーの DataHandlers サブディレクトリー内にあります。データ・ハンドラー・クラスを突き止めるため、createHandler() メソッドは、納入時のデータ・ハンドラーが含まれている CwDataHandler.jar ファイルを検索した後、この JAR ファイルを検索します。

注: Java コードをコンパイルするには、マシンに Java Development Kit (JDK) がインストールされている必要があります。JDK の必要なバージョンおよびそのインストール方法については、ご使用の製品のインストール情報を参照してください。

カスタム・データ・ハンドラーを CustDataHandler.jar へ追加するには、次のようにしてください。

1. データ・ハンドラー・コンパイル・スクリプトを編集して、Java ソース・ファイルの名前を追加します。

   このデータ・ハンドラー・コンパイル・スクリプトは、次の製品ディレクトリーのサブディレクトリーにあります。

   ```
   DevelopmentKits¥edk¥DataHandler
   ```

   Windows

   Windows システムでは、データ・ハンドラー・コンパイル・スクリプトは make_datahandler.bat という名前です。 Java ソース・ファイルの名前を、次のように行に追加します。

   ```
   set SOURCE_FILES_DH=
   ```

   UNIX

   UNIX システムでは、データ・ハンドラー・コンパイル・スクリプトは make_datahandler という名前です。 Java ソース・ファイルの名前を、次のように行に追加します。

   ```
   SOURCE_FILES_DH=
   ```

2. データ・ハンドラーのコンパイル・スクリプトを実行して、Java ファイルを .class ファイルにコンパイルします。
3. 次のコマンドを使用して、CustDataHandler.jar ファイルへ新しいクラスを追加します。

   ```
   jar -vf CustDataHandler.jar input_files
   ```

   ここで *input_files* は、追加するクラス・ファイルのリストです。

データ・ハンドラー・メタオブジェクトの作成

データ・ハンドラー・メタオブジェクトを使用するカスタム・データ・ハンドラーを作成する場合は、次のことを行う必要があります。

- カスタム・データ・ハンドラーの構成情報の属性を含む子データ・ハンドラー・メタオブジェクトを作成します。
- サポートされている MIME タイプを含むようにトップレベルのデータ・ハンドラー・メタオブジェクトを変更して、データ・ハンドラーの構成がデータ・ハンドラーのインスタンス化時に実行できるようにします。
- ビジネス・インテグレーション・システムでメタオブジェクトをセットアップします。

注: データ・ハンドラー設計にメタオブジェクトを使用するかどうかを判別する方法については、155 ページの『データ・ハンドラー・メタオブジェクトの使用』を参照してください。

子メタオブジェクトの作成

子メタオブジェクトには、データ・ハンドラーの構成情報が含まれています。createHandler() メソッドは、この情報を使用して、新たにインスタンス化されたデータ・ハンドラーを初期化します。このプロセスの詳細情報については、15 ページの『MIME タイプの使用』を参照してください。

カスタム・データ・ハンドラー用の子メタオブジェクトを作成するには、次のようにします。

1. 子メタオブジェクトを作成して、データ・ハンドラーのインスタンスを表します。

   この子メタオブジェクトを作成するときは、Business Object Designer Express ツールを使用できます。この子メタオブジェクトには、データ・ハンドラーに必要な構成情報を定義するための属性が含まれている必要があります。最低でも、子メタオブジェクトには ClassName 属性が必要 です。
2. ClassName 属性にデータ・ハンドラー・クラスの名前を指定する必要があるかどうかを判別します。
   - デフォルト形式であるデータ・ハンドラーの場合は、ClassName の値を指定する必要はありません。デフォルト形式は次のとおりです。

     ```
     com.crossworlds.DataHandlers.MimeTypeString
     ```

     ただし、ClassName 属性のデフォルト値に、データ・ハンドラーのクラス名を指定できます。

- デフォルト形式でないデータ・ハンドラー・クラスの場合は、データ・ハンドラー・インスタンス用に完全クラス名を指定する必要があります。それ以外の場合は、createHandler() メソッドが、データ・ハンドラーをインスタンス化しようとしても、データ・ハンドラー・クラスを突き止めることができません。

3. 子メタオブジェクト内部にある適切な属性のデフォルト値を設定し、データ・ハンドラー・インスタンスがデータを処理する方法を構成します。

トップレベルのメタオブジェクトの変更

呼び出し元が MIME タイプを createHandler() メソッドに提供すると、createHandler() は、以下のステップでインスタンス化するデータ・ハンドラーを判別します。

1. データ・ハンドラーに関連付けられているトップレベルのメタオブジェクトの名前を突き止めます。
2. 変換するデータに一致する MIME タイプについて、このトップレベル・メタオブジェクトを調べます。
3. この MIME タイプが存在する場合は、関連付けられた子メタオブジェクトの名前を検出します。これには、構成情報が含まれています。

このプロセスの詳細説明については、15 ページの『MIME タイプの使用』を参照してください。この処理を正常に実行するため、createHandler() は、データに関連付けられている MIME タイプを探し出すことができる必要があります。したがって、トップレベルのデータ・ハンドラー・メタオブジェクトを編集して、データ・ハンドラーが変換するデータの MIME タイプに応じた属性を設定することが必要です。この属性には、次のものを含める必要があります。

- データ・ハンドラーの関連付けられた MIME タイプの MIME タイプ・ストリングに一致する名前

  MIME タイプの名前に使用できるのは、英数字と下線 (_) のみ です。そのほかの特殊文字は無効です。MIME タイプにピリオド (.) がある場合は、下線に置き換えてください。
- 単一カーディナリティーのビジネス・オブジェクト
- データ・ハンドラーを表す子メタオブジェクトの名前であるタイプ

例として、図 37 に、カスタム HTML データ・ハンドラー用に構成されているトップレベルのコネクター・メタオブジェクトを示します。

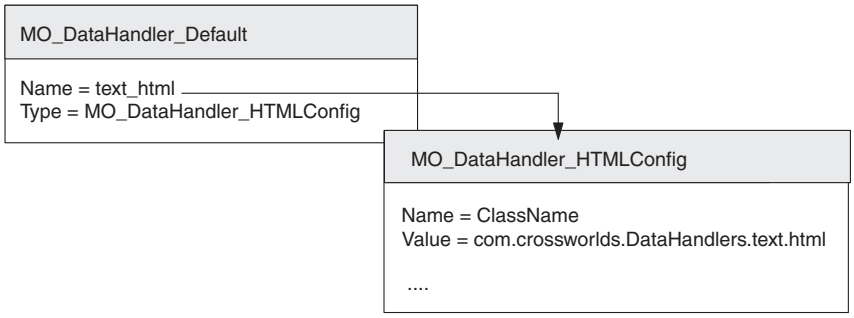

図 37. カスタム・データ・ハンドラーのトップレベル・コネクター・メタオブジェクトの例

図 37 では、コネクター用のデフォルト・トップレベル・メタオブジェクト (MO_DataHandler_Default) を、新しい MIME タイプである HTML をサポートするように変更してあります。この MIME タイプのサポートでは、MO_DataHandler_Default メタオブジェクトに、次の属性プロパティーが含まれています。

```
Attribute Name = text_html
Attribute Type = MO_DataHandler_HTMLConfig
```

要確認: MIME タイプの名前は、英数字 および下線 (_) のみに制限されています。それ以外の特殊文字は、MIME タイプには使用できません。

ビジネス・インテグレーション・システムでのメタオブジェクトのセットアップ

データ・ハンドラー・メタオブジェクトを作成した場合、以下のように、WebSphere Business Integration システムにこれらのメタオブジェクトをセットアップする必要があります。

1. 新規メタオブジェクトをリポジトリーにロードします。
2. データ・ハンドラーが呼び出されるコンテキストに応じて、適切なメタオブジェクトを変更します。
   - データ・ハンドラーがコネクターのコンテキストで実行される場合は、データ・ハンドラーのメタオブジェクトを子としてトップレベルのデータ・ハンドラー・メタオブジェクトへ追加します。次に、トップレベルのデータ・ハンドラー・メタオブジェクトのサポートを、コネクター定義へ追加します。
   - データ・ハンドラーが Server Access Interface のコンテキストで実行される場合は、データ・ハンドラーのメタオブジェクトを子としてトップレベルのサーバー・メタオブジェクト MO_Server_Datahandler へ追加します。

その他のビジネス・オブジェクトのセットアップ

データ・ハンドラーのコーディングに加えて、データ・ハンドラーのビジネス・オブジェクトをセットアップする必要があります。データ・ハンドラーの要件および呼び出し元の要件を満たすビジネス・オブジェクトを作成します。

ヒント: データ・ハンドラーに必要なビジネス・オブジェクトは、データ・ハンドラーを使用するコネクターのサポート・オブジェクト・リストに含まれていることを確認してください。

コネクターの構成

カスタム・データ・ハンドラーがコネクターのコンテキストで使用される場合は、トップレベルのコネクター・メタオブジェクトの名前を取得するように各コネクターを構成する必要があります。コネクターは、データ・ハンドラー・メタオブジェクトの名前とクラス名に関する情報を、さまざまな方法で取得します。例えば次のようになります。

- WebSphere Business Integration Adapter for XML をデータ・ハンドラーを使用するように構成するには、DataHandlerConfigMO コネクター構成プロパティーを設定し、XML コネクターのビジネス・オブジェクトに MimeType 属性を設定します。
- WebSphere Business Integration Adapter for JText をデータ・ハンドラーを使用するように構成するには、ClassName、または DataHandlerConfigMO および MimeType 属性を JText 構成メタオブジェクトに設定します。

詳細については、 30 ページの『データ・ハンドラーを使用するようにコネクターを構成する方法』を参照してください。

ヒント: データ・ハンドラーを使用するようにコネクターを構成するときには、メタオブジェクト名のスペルおよび MIME タイプのスペルが正しいことを確認してください。

国際化データ・ハンドラー

国際化データ・ハンドラー は、特定のロケール向けにカスタマイズできるように作成されたデータ・ハンドラーです。*ロケール* は、エンド・ユーザーの特定の国、言語、または地域に固有のデータを処理する方法に関する情報をまとめた、ユーザー環境の要素です。ロケールは、通常はオペレーティング・システムの一部としてインストールされます。ロケール依存データを処理するデータ・ハンドラーを作成することを、データ・ハンドラーの*国際化対応* (I18N) と呼びます。また、特定ロケール向けに国際化データ・ハンドラーを作成することを、データ・ハンドラーの*ローカリゼーション* (L10N) と呼びます。

このセクションでは、国際化データ・ハンドラーに関する以下の情報について説明します。

- ロケールとは
- 国際化データ・ハンドラーの設計上の考慮事項

ロケールとは

ロケール は、ユーザー環境に関する以下の情報を提供します。

- 言語および国 (または地域) ごとの国/地域別情報
  - データ・フォーマット:
    - 日付: 曜日および月の名前とその省略名、および日付の構成 (日付の分離文字を含む) を定義します。
    - 数値: 3 桁ごとの区切り記号と小数点記号、およびこれらの記号を数値中のどこに配置するかを定義します。

- 時刻: 12 時間表記の標識 (AM および PM 標識など)、および時刻の構成を定義します。
- 通貨値: 数値と通貨記号、およびこれらの記号を通貨値の中のどこに配置するかを定義します。

– 照合順序: 特定の文字コード・セットおよび言語のデータをソートする方法です。
– ストリング処理には、文字 (大文字および小文字) の比較、サブストリング、連結などのタスクがあります。

• *文字エンコード:* 文字 (英字) を文字コード・セットの数値にマッピングします。例えば、ASCII 文字コード・セットでは文字「A」は 65 にエンコードされ、EBCIDIC 文字セットではこの文字は 43 にエンコードされます。*文字コード・セット* には、1 つ以上の言語のすべての文字のエンコード方式が含まれます。

ロケール名は次のような形式になります。

ll_TT.codeset

ここで、*ll* は 2 文字の言語コード (通常は小文字)、*TT* は 2 文字の国および地域コード (通常は大文字)、*codeset* は関連する文字コード・セットの名前を表します。ロケール名の *codeset* 部分は、たいていの場合はオプションです。ロケールは、通常はオペレーティング・システム・インストールの一部としてインストールされます。

国際化データ・ハンドラーの設計上の考慮事項

このセクションでは、データ・ハンドラーの国際化に関する以下の設計上の考慮事項について説明します。

- ロケール依存設計原則
- 文字エンコード設計原則

ロケール依存設計原則

国際化するには、データ・ハンドラーをロケール依存になるようにコード化する必要があります。つまり、データ・ハンドラーはロケールの設定を考慮して、そのロケールに適切なタスクを実行する必要があります。例えば、英語を使用するロケールの場合、データ・ハンドラーは英語のメッセージ・ファイルからエラー・メッセージを取得します。この製品でインストールされるデータ・ハンドラー・フレームワークは、国際化されています。開発したデータ・ハンドラーの国際化対応 (I18N) を完了する際は、そのデータ・ハンドラー・インプリメンテーションが国際化されていることを確認してください。

国際化データ・ハンドラーは、次のロケール依存設計原則に従う必要があります。

- すべてのエラー、状況、およびトレース・メッセージのテキストは、メッセージ・ファイルにおいてデータ・ハンドラー・インプリメンテーションから分離されていて、ロケールの言語に翻訳されている。
- データのソートまたは照合時に、ロケールの言語および国に対して適切な照合順序が使用される。
- ストリング処理 (比較、サブストリング、大文字小文字など) が、ロケールの言語の文字に対して適切である。

- 日付、数値、および時刻のフォーマットがロケールに対して適切である。

データ・ハンドラーは、直列化データ・アプリケーションとビジネス・オブジェクトの間の変換の際に、ロケール依存処理 (データ・フォーマット変換など) を実行する必要があります。データ・ハンドラーの環境に関連するロケールを追跡するため、DataHandler クラスには private ロケール変数があります。この変数はデータ・ハンドラーが稼働するオペレーティング・システムのロケールに初期化されています。データ・ハンドラー環境のロケール (この private locale 変数の値) へは、実行時に表 52 の accessor メソッドを介してアクセスできます。

表 52. データ・ハンドラー環境のロケールにアクセスするメソッド

データ・ハンドラー・クラス	メソッド
DataHandler	getLocale(), setLocale()

ビジネス・オブジェクトが作成されると、そのデータにロケールが関連付けられます。このロケールは、作成されたビジネス・オブジェクトのデータに適用されます。ビジネス・オブジェクト定義の名前やその属性には適用されません (これらは英語ロケール en_US に関連するコード・セットの文字でなければなりません)。ビジネス・オブジェクトを作成するため、データ・ハンドラーでは表 53 に示すメソッドを使用できます。これらのメソッドは、DataHandler クラスの private ロケール変数にアクセスできます。これらのメソッドはビジネス・オブジェクトを作成するときに、private DataHandler ロケール変数の示すロケールをこのビジネス・オブジェクトに関連付けます。

注: 表 53 のメソッドは、トップレベル・ビジネス・オブジェクトのロケールのみを設定します。ビジネス・オブジェクトに子ビジネス・オブジェクトが含まれている場合は、その子ビジネス・オブジェクトではロケールがシステム・デフォルトに設定されるため、有効なロケール値がない場合があります。

ビジネス・オブジェクトを作成して、そのデータにロケールを設定するには、表 53 のメソッドを使用します。private ロケール変数がビジネス・オブジェクトのデータに対して適切なロケールを示すようにするため、表 53 のメソッドを呼び出す*前*に、setLocale() メソッドを使用することができます。

表 53. ビジネス・オブジェクトにロケールを割り当てるメソッド

データ・ハンドラー・クラス	メソッド
DataHandler	getBO() - パブリック, getBOName()

文字エンコード設計原則

あるコード・セットを使用するロケーションから別のコード・セットを使用するロケーションにデータを転送する場合、データの意味が保持されるような文字変換形式を実行する必要があります。Java 仮想マシン (JVM) 内の Java ランタイム環境では、Unicode 文字セットでデータを表します。Unicode 文字セットは、ほとんどの既知の文字コード・セット (単一バイトおよびマルチバイトの両方) の文字のエンコードが含まれる汎用文字セットです。Unicode にはいくつかのエンコード・フォーマットがあります。ビジネス・インテグレーション・システム内で最も頻繁に使用されるエンコードを以下に示します。

- Universal multi-octet Character Set 2: UCS-2

UCS-2 エンコードは、2 バイト (オクテット) でエンコードされた Unicode 文字セットです。

- UCS Transformation Format、8 ビット形式: UTF-8

  UTF-8 エンコードは、UNIX 環境で Unicode 文字データを使用するために設計されたものです。UTF-8 ではすべての ASCII コード値 (0...127) がサポートされるので、ASCII コード値が別のコードに解釈されることはありません。各コード値は通常 1 バイト値、2 バイト値、または 3 バイト値で表します。

IBM WebSphere Business Integration システム のほとんどのコンポーネントは Java で書かれています。したがって、ほとんどのシステム・コンポーネント間のデータ転送では、データは Unicode コード・セットでエンコードされるため、文字変換の必要はありません。

データ・ハンドラーは Java で書かれたコンポーネントなので、Unicode コード・セットの直列化データを処理します。また通常は、データの入力ストリームのソースも Unicode で処理されます。したがってデータ・ハンドラーは、通常は直列化データの文字変換を実行する必要はありません。ただし、入力データまたは出力データ内に、システム・デフォルトとは異なる文字エンコードのバイト配列が含まれている場合は、データ・ハンドラーはその文字エンコードを提供する必要があります。

データ・ハンドラーの環境に関連する文字エンコードを追跡するため、`DataHandler` クラスには private 文字エンコード変数があります。この変数は、データ・ハンドラーが稼働するオペレーティング・システムのロケールに関連する文字エンコードに初期化されています。データ・ハンドラー環境の文字エンコード (この private 文字エンコード変数の値) へは、実行時に表 54 の accessor メソッドを介してアクセスできます。

表 54. データ・ハンドラーの文字エンコードを検索するメソッド

データ・ハンドラー・クラス	メソッド
`DataHandler`	getEncoding(), setEncoding()

第 9 章 データ・ハンドラーの基本クラス・メソッド

DataHandler クラスは、データ・ハンドラーの基本クラスです。このクラスは、com.crossworlds.DataHandlers パッケージに格納されています。カスタム・データ・ハンドラーを含むすべてのデータ・ハンドラーが、この抽象クラスを拡張する必要があります。

この章で説明するメソッドは、次の 3 つのカテゴリーに分けられます。

- インプリメントする必要がある抽象メソッド
- 必要に応じてオーバーライドできる提供されたインプリメンテーションを持つパブリック・メソッド
- コネクターまたはアクセス・クライアントが呼び出す静的メソッド

表 55 に DataHandler クラスのメソッドを示します。

表 55. DataHandler クラスのメンバー・メソッド

メンバー・メソッド	タイプ	説明	ページ
createHandler()	パブリック静的	データ・ハンドラーのインスタンスを作成します。	182
getBO() - 抽象	抽象	ビジネス・オブジェクトに、直列化入力データから抽出した値を取り込みます。	183
getBO() - パブリック	パブリック	直列化データをビジネス・オブジェクトに変換します。	185
getBOName()	パブリック	直列化データの内容に基づいてビジネス・オブジェクトの名前を取得します。	186
getBooleanOption()	パブリック	ブール・データが含まれている場合に、指定されたデータ・ハンドラー構成オプションの値を取得します。	187
getByteArrayFromBO()	パブリック	ビジネス・オブジェクトを直列化して、バイト配列にします。	188
getEncoding()	パブリック	データ・ハンドラーが使用している文字エンコードを検索します。	189
getLocale()	パブリック	データ・ハンドラーのロケールを検索します。	189
getOption()	パブリック	指定されたデータ・ハンドラー構成オプションの値 (ただし、設定されている場合のみ) を取得します。	190
getStreamFromBO()	抽象	ビジネス・オブジェクトを直列化して、InputStream オブジェクトにします。	191
getStringFromBO()	抽象	ビジネス・オブジェクトを直列化して、String オブジェクトにします。	192
setConfigMOName()	静的	DataHandler 基本クラスの静的プロパティーに、トップレベルのデータ・ハンドラー・メタオブジェクトの名前を設定します。	193
setEncoding()	パブリック	データ・ハンドラーが使用している文字エンコードを設定します。	193
setLocale()	パブリック	データ・ハンドラーのロケールを設定します。	194
setOption()	パブリック	指定されたデータ・ハンドラー構成オプションの値を設定します。	195

表 55. *DataHandler* クラスのメンバー・メソッド *(続き)*

メンバー・メソッド	タイプ	説明	ページ
traceWrite()	パブリック	データ・ハンドラーのトレース・メッセージを書き込むための適切なトレース書き込み機能を呼び出します。	195

createHandler()

データ・ハンドラーのインスタンスを作成します。

構文

```
public static DataHandler createHandler(String className,
    String mimeType, String BOPrefix);
```

パラメーター

className 作成するデータ・ハンドラー・インスタンスのクラス名。指定しないと、メソッドは *mimeType* 引き数を使用して、インスタンス化するデータ・ハンドラー・クラスを判別します。

mimeType 作成するデータ・ハンドラー・インスタンスの MIME タイプを指定します。指定しないと、メソッドは、提供される *className* 値を予期します。メタオブジェクトのキー。*BOPrefix* が指定されると、*mimeType* はキーの一部になります。

BOPrefix オプションのパラメーター。存在する場合は、*mimeType* と結合して、メタオブジェクトのキーを形成します。この引き数は、MIME サブタイプを指定するために使用できます。また、データ・ハンドラーの構成プロパティー BOPrefix を設定するために使用することもできます。

戻り値

データ・ハンドラーのインスタンス。

例外

Exception
メソッドがデータ・ハンドラーのインスタンスを生成できない場合にスローされます。

注記

このメソッドは、その *className*、*mimeType*、および *BOPrefix* パラメーターの値に基づいて、データ・ハンドラーのインスタンスを作成します。

- createHandler() メソッドがコネクターによって呼び出される場合、コネクターは *className* 値を指定できます。*className* が指定されると、createHandler() は、このクラス名のデータ・ハンドラーをインスタンス化します。
- *mimeType* が指定されると、createHandler() は、指定された MIME タイプに基づいて、データ・ハンドラーを作成します。

メソッドは、名前が *mimeType* パラメーターまたは *mimeType* と *BOPrefix* の組み合わせのいずれかの内容タイプに一致する属性について、トップレベルのデータ・ハンドラー・メタオブジェクトを検査します。一致する属性が検出された場合は、子メタオブジェクト内の ClassName 属性の値がクラス名として使用されます。

メソッドは、データ・ハンドラーのクラスのインスタンス化に成功すると、setupOptions() を呼び出して、データ・ハンドラー・インスタンスが使用できるように構成プロパティーをセットアップします。 createHandler() がデータ・ハンドラーをインスタンス化する方法の詳細については、14 ページの『データ・ハンドラーのインスタンス化』を参照してください。

例えば、MIME = "text/xml-application-xxx" の場合、メソッドは com.crossworlds.DataHandlers.text.xml_application_xxx クラスをロードします。

getBO() - 抽象

ビジネス・オブジェクトに、直列化入力データから抽出した値を取り込みます。

構文

```
public abstract void getBO(Reader serializedData,
        BusinessObjectInterface theBusObj, Object config);

public abstract BusinessObjectInterface getBO(Reader serializedData,
        Object config);
```

パラメーター

serializedData 直列化データにアクセスする Java Reader オブジェクト。

theBusObj データを取り込むビジネス・オブジェクト。

config データ・ハンドラーの追加の構成情報が含まれているオプションのオブジェクト。

戻り値

このメソッドの最初の形式には戻り値はありません。2 番目の形式はビジネス・オブジェクトを戻します。

注記

この getBO() メソッドは、データ・ハンドラーのためにストリングからビジネス・オブジェクトへの変換を実行する抽象メソッドです。つまりこのメソッドは、(Reader オブジェクトによってアクセスされる) 汎用直列化データをビジネス・オブジェクトに変換する方法を定義します。このメソッドには次の 2 つの形式があります。

- 最初の形式では、呼び出し元から渡される空のビジネス・オブジェクト *theBusObj* を取り込みます。
- 2 番目の形式では、ビジネス・オブジェクト・インスタンスを作成し、それを取り込みます。

注: Server Access Interface のコンテキストで呼び出されるデータ・ハンドラーは、2 番目の形式の getBO() メソッドのみ の機能を提供する必要があります。

要確認: getBO() メソッドは、デフォルト・インプリメンテーションのない抽象メソッドです。したがって、データ・ハンドラー・クラスはこのメソッドをインプリメントする必要があります。

Java Reader オブジェクトとして、直列化データを getBO() へ渡します。ただし、Reader は基本クラスであるため、実際には、Reader クラスのいくつかのサブクラスのうち 1 つのインスタンスを渡します。一部の Reader サブクラスは mark() 操作のインプリメンテーションを提供しますが、それ以外は提供しません。mark() 操作を使用すると、呼び出し元は、ストリーム内の特定の位置にマークを付け、後からその位置に戻ることができます。

注: Reader オブジェクトを XML データ・ハンドラーの getBO() メソッドに渡すには、Reader サブクラスが mark() メソッドをインプリメントすることが必要です。 Reader クラスの isMarkSupported() メソッドが、使用する Reader オブジェクトのサポートであるかどうかを判別するために、そのメソッドを呼び出すことができます。直列化データを StringReader オブジェクトとして渡すことをお勧めします。

データ・ハンドラーに、メタオブジェクトに含まれているよりも多くの構成情報を提供する必要がある場合は、*config* オプションを使用して、そのような情報が含まれたオブジェクトを渡すことができます。例えば、*config* は、ビジネス・オブジェクトからの XML 文書を作成するために使用されるスキーマの、テンプレート・ファイルまたは URL へのストリングである可能性があります。

config がビジネス・オブジェクト・タイプである場合は、getBO() メソッドをインプリメントして、setupOptions(*config*) を呼び出すことができます。setupOptions() メソッドは、DataHandler 基本クラスに定義されます。このメソッドは、ビジネス・オブジェクト内の属性名をプロパティー名として使用し、デフォルト値をそれらのプロパティーの値として使用します。データ・ハンドラーが使用できるように、オブジェクトの構成プロパティーの値を設定します。

抽象 getBO() メソッドをインプリメントすると、データ・ハンドラーを呼び出すコンポーネントは、表 56 に示すパブリック・ストリングからビジネス・オブジェクトへの変換メソッドのいずれかを呼び出すことができます。

表 56. パブリック・ストリングからビジネス・オブジェクトへの変換メソッド

パブリック・ストリングからビジネス・オブジェクトへの変換メソッド	説明
getBO(Object *serializedData*, Object *config*)	汎用 Object 内の直列化データをビジネス・オブジェクトに変換します。
getBO(String *serializedData*, Object *config*)	Object 内の直列化データをビジネス・オブジェクトに変換します。
getBO(InputStream *serializedData*, Object *config*)	InputStream 内の直列化データをビジネス・オブジェクトに変換します。
getBO(byte[] *serializedData*, Object *config*)	バイト配列内の直列化データをビジネス・オブジェクトに変換します。

参照

getBO() - パブリック

---

getBO() - パブリック

直列化データをビジネス・オブジェクトに変換します。

構文

```
public BusinessObjectInterface getBO(Object serializedData,
    Object config);

public BusinessObjectInterface getBO(String serializedData,
    Object config);

public BusinessObjectInterface getBO(InputStream serializedData,
    Object config);

public BusinessObjectInterface getBO(byte[] serializedData,
    Object config);

public void getBO(Object serializedData,
    BusinessObjectInterface theBusObj, Object config);
```

パラメーター

serializedData 直列化データへの参照。

theBusObj データを取り込むビジネス・オブジェクト。

config データ・ハンドラーの追加の構成情報が含まれているオプションのオブジェクト。

戻り値

最初の 4 つの形式は、入力 Object、String、InputStream、またはバイト配列オブジェクトからのデータを取り込むビジネス・オブジェクトを戻します。 5 番目の形式は、指定されたビジネス・オブジェクトに直列化データからのデータを取り込みます。

例外

Exception

メソッドが直列化データをビジネス・オブジェクトに変換できない場合にスローされます。

NotImplementedException

getBO() メソッドのパブリック・バージョンが実装されていない場合にスローされます。

注記

この getBO() メソッドは、ストリングからビジネス・オブジェクトへの変換を実行するパブリック・メソッドです。 DataHandler 基本クラスには、getBO() の抽象形式 (183 ページを参照) が含まれています。これは、データ・ハンドラー・クラスの一部としてインプリメントする必要があります。getBO() のパブリック・バージョンは、コンポーネント (コネクターやアクセス・クライアントなど) が

serializedData を Object、String、または InputStream オブジェクト、あるいはバイト配列として指定できるようにする、一連のユーティリティー・メソッドを定義します。ユーティリティー・メソッドは、指定された直列化データを Reader オブジェクトに変換し、次に抽象 getBO() メソッドの 1 つを呼び出して、Reader オブジェクトをビジネス・オブジェクトに変換します。

パブリック getBO() メソッドの形式は次のとおりです。

- 最初の形式は、*serializedData* オブジェクトのタイプに基づいて適切な getBO() メソッドを呼び出します。例えば、データのタイプが String である場合、メソッドは getBO(String *serializedData*, Object *config*) を呼び出します。
- 2 番目、3 番目、および 4 番目の形式は、String、InputStream、またはバイト配列オブジェクトから Reader オブジェクトを作成し、抽象 getBO() メソッドを呼び出して、Reader オブジェクトをビジネス・オブジェクトに変換します。
- 5 番目の形式は、呼び出し元がビジネス・オブジェクトを渡す時に getBO() 呼び出しを処理するもう 1 つのユーティリティーです。これは、次のタスクを実行します。
  - 渡される serializedData オブジェクトのタイプを判別する
  - 必要に応じて String または InputStream オブジェクトへオブジェクトを変換する
  - 抽象バージョンを呼び出して、ビジネス・オブジェクトおよびデータを引き渡す

config 引き数については、抽象形式の getBO() (183 ページ) の説明を参照してください。

参照

getBO() - 抽象

---

getBOName()

直列化データの内容に基づいてビジネス・オブジェクトの名前を取得します。

構文

```
public String getBOName(Reader serializedData);
public String getBOName(String serializedData);
public String getBOName(InputStream serializedData);
```

パラメーター

serializedData　メッセージが含まれている Reader オブジェクトに対する参照。

戻り値

ビジネス・オブジェクトの名前が含まれている String オブジェクトを戻します。NameHandlerClass 属性の値が存在しない場合、このメソッドは null を戻します。

例外

getBOName() の 2 番目と 3 番目の形式は以下の例外をスローします。

MalformedDataException
直列化データ (*serializedData*) のフォーマットが誤っている場合にスローされます。

NotImplementedException
ネーム・ハンドラーが実装されていない場合にスローされます。

注記

getBOName() メソッドは、ネーム・ハンドラーのインスタンスを作成して、直列化データからビジネス・オブジェクト定義の名前を抽出します。 NameHandlerClass メタオブジェクト属性の値に基づいてこのネーム・ハンドラーのインスタンスを作成します。ネーム・ハンドラーは、メッセージの内容に基づいて、ビジネス・オブジェクト名を作成します。

getBOName() メソッドの形式は次のとおりです。

- 最初の形式は、BOPrefix メタオブジェクト属性 (存在する場合のみ) および NameHandler クラスからの戻り値に基づいて、ビジネス・オブジェクト名を戻します。
- 2 番目の形式は、String から Reader オブジェクトを作成して、最初の形式を呼び出します。
- 3 番目の形式は、InputStream から Reader オブジェクトを作成して、最初の形式を呼び出します。

現在、次のような IBM 納入時データ・ハンドラーで、このメソッドを使用しています。

- XML データ・ハンドラー

  XML データ・ハンドラーのデフォルト・ネーム・ハンドラーは、基本クラス getBOName(Reader *data*) を呼び出します。データ・ハンドラーが要求を処理できない場合は、ビジネス・オブジェクトの基本名を抽出するために、<!DOCTYPE Name が使用されます。最終的な名前は次のようになります。

  ```
  BOPrefix + "_" + Name.getStreamFromBO()
  ```

独自のネーム・ハンドラーを作成するには、NameHandler 抽象基本クラスを拡張し、getBOName() をオーバーライドします。

詳細については、96 ページの『カスタム XML ネーム・ハンドラーの作成』を参照してください。

getBooleanOption()

ブール・データが含まれている場合に、指定されたデータ・ハンドラー構成オプションの値を取得します。

構文

```
public boolean getBooleanOption(String name);
```

パラメーター

name 構成オプションの名前。

戻り値

Boolean 型オプションの値を戻します。

注記

createHandler() メソッドは、データ・ハンドラーに関連付けられている子メタオブジェクトを使用して、その構成オプションを初期化します。オプションにブール値が含まれている限り、getBooleanOption() メソッドを使用すると、これらのオプションのうちいずれかの値を取得できます。

---

getByteArrayFromBO()

ビジネス・オブジェクトを直列化して、バイト配列にします。

構文

```
abstract byte[] getByteArrayFromBO(BusinessObjectInterface theBusObj,
      Object config);
```

パラメーター

theBusObj バイト配列に変換するビジネス・オブジェクト。

config データ・ハンドラーの追加の構成情報が含まれているオプションのオブジェクト。

戻り値

指定されたビジネス・オブジェクトを表す直列化データを含むバイト配列。

例外

Exception
メソッドがビジネス・オブジェクトを直列化データのバイト配列に変換できない場合にスローされます。

注記

getByteArrayFromBO() メソッドは、データ・ハンドラーのためにビジネス・オブジェクトからバイトへの変換を実行します。 *theBusObj* ビジネス・オブジェクト内のデータをバイト配列 (Java byte[] オブジェクト) へ変換します。

要確認: getByteArrayFromBO() メソッドは、デフォルト・インプリメンテーションのない抽象メソッドです。したがって、データ・ハンドラー・クラスはこのメソッドをインプリメントする必要があります。

データ・ハンドラーに、メタオブジェクトに含まれているよりも多くの構成情報を提供する必要がある場合は、*config* オプションを使用して、そのような情報が含まれたオブジェクトを渡すことができます。例えば、*config* は、ビジネス・オブジェクトからの XML 文書を作成するために使用されるスキーマの、テンプレート・ファイルまたは URL へのストリングである可能性があります。

config がビジネス・オブジェクト・タイプである場合は、getByteArrayFromBO() メソッドをインプリメントして、setupOptions(*config*) を呼び出すことができます。 setupOptions() メソッドは、DataHandler 基本クラスに定義されます。このメソッドは、ビジネス・オブジェクト内の属性名をプロパティー名として使用し、デフォルト値をそれらのプロパティーの値として使用します。データ・ハンドラーが使用できるように、オブジェクトの構成プロパティーの値を設定します。

参照

getBO() - パブリック, getStreamFromBO(), getStringFromBO()

getEncoding()

データ・ハンドラーが使用している文字エンコードを検索します。

構文

```
public final String getEncoding();
```

パラメーター

なし。

戻り値

データ・ハンドラーの文字エンコードが含まれた String。

注記

getEncoding() メソッドは、データ・ハンドラーの文字エンコードを検索します。文字エンコードはロケールの一部で、ロケールはデータの国/地域別情報を言語、国 (または地域) ごとに定義します。このメソッドは、DataHandler クラスの private 文字エンコード変数の値を検索する accessor メソッドです。この文字エンコードは、データ・ハンドラーが処理する直列化データの文字エンコードを示します。

このメソッドは、データ・ハンドラーが文字変換などの文字エンコード処理を実行する必要がある場合に役立ちます。

参照

setEncoding()

getLocale()

データ・ハンドラーのロケールを検索します。

構文

```
public final Locale getLocale();
```

パラメーター

なし。

戻り値

データ・ハンドラーの環境のロケールを記述する Java Locale オブジェクト。

注記

getLocale() メソッドはデータ・ハンドラーのロケールを検索します。ロケールは、データの国/地域別情報を言語、国 (または地域)、および文字エンコードごとに定義します。このメソッドは、DataHandler クラスの private ロケール変数の値を検索する accessor メソッドです。デフォルトでは、データ・ハンドラーのロケールは、データ・ハンドラーが稼働するオペレーティング・システムのロケールになります。

このメソッドは、データ・ハンドラーがロケール依存処理を実行する必要がある場合に役立ちます。

参照

setLocale()

---

getOption()

指定されたデータ・ハンドラー構成オプションの値 (ただし、設定されている場合のみ) を取得します。

構文

```
public String getOption(String name);
```

パラメーター

name 構成オプションの名前。

戻り値

オプションの値が含まれている String オブジェクト。

注記

getOption() メソッドは、構成オプションの値を取得します。データ・ハンドラーに関連付けられた子メタオブジェクトが存在する場合、createHandler() メソッドは、これらのメタオブジェクト属性のデフォルト値を使用して、その構成オプションを初期化します。これらのいずれかのオプションの値を取得するには、getOption() メソッドを使用できます。

参照

setOption()

---

getStreamFromBO()

ビジネス・オブジェクトを直列化して、InputStream オブジェクトにします。

構文

```
abstract InputStream getStreamFromBO(BusinessObjectInterface theBusObj,
      Object config);
```

パラメーター

theBusObj ストリームに変換するビジネス・オブジェクト。

config データ・ハンドラーの追加の構成情報が含まれているオプションのオブジェクト。

戻り値

ビジネス・オブジェクトを表す直列化データが含まれている InputStream オブジェクト。

例外

Exception
メソッドがビジネス・オブジェクトを直列化データのストリームに変換できない場合にスローされます。

注記

getStreamFromBO() メソッドは、データ・ハンドラーのためにビジネス・オブジェクトからストリームへの変換を実行します。 *theBusObj* ビジネス・オブジェクト内のデータをストリーム (Java InputStream オブジェクト) へ変換します。

要確認: getStreamFromBO() メソッドは、デフォルト・インプリメンテーションのない抽象メソッドです。したがって、データ・ハンドラー・クラスはこのメソッドをインプリメントする必要があります。

データ・ハンドラーに、メタオブジェクトに含まれているよりも多くの構成情報を提供する必要がある場合は、config オプションを使用して、そのような情報が含まれたオブジェクトを渡すことができます。例えば、*config* は、ビジネス・オブジェクトからの XML 文書を作成するために使用されるスキーマの、テンプレート・ファイルまたは URL へのストリングである可能性があります。

config がビジネス・オブジェクト・タイプである場合は、getStreamFromBO() メソッドをインプリメントして、setupOptions(*config*) を呼び出すことができます。setupOptions() メソッドは、DataHandler 基本クラスに定義されます。このメソッドは、ビジネス・オブジェクト内の属性名をプロパティー名として使用し、デフォルト値をそれらのプロパティーの値として使用します。データ・ハンドラーが使用できるように、オブジェクトの構成プロパティーの値を設定します。

参照

getBO() - パブリック, getByteArrayFromBO(), getStringFromBO()

---

getStringFromBO()

ビジネス・オブジェクトを直列化して、String オブジェクトにします。

構文

```
abstract String getStringFromBO(BusinessObjectInterface theBusObj,
      Object config);
```

パラメーター

theBusObj String に変換するビジネス・オブジェクト。

config データ・ハンドラーの追加の構成情報が含まれているオプションのオブジェクト。

戻り値

直列化データを含む String オブジェクトは、ビジネス・オブジェクト内のデータを表します。

例外

Exception
メソッドがビジネス・オブジェクトを直列化データのストリングに変換できない場合にスローされます。

注記

getStringFromBO() メソッドは、データ・ハンドラーのためにビジネス・オブジェクトからストリングへの変換を実行します。 *theBusObj* ビジネス・オブジェクト内のデータを Java String オブジェクトへ変換します。

要確認: getStringFromBO() メソッドは、デフォルト・インプリメンテーションのない抽象メソッドです。したがって、データ・ハンドラー・クラスはこのメソッドをインプリメントする必要があります。

データ・ハンドラーに、メタオブジェクトに含まれているよりも多くの構成情報を提供する必要がある場合は、config オプションを使用して、そのような情報が含まれたオブジェクトを渡すことができます。例えば、*config* は、ビジネス・オブジェクトからの XML 文書を作成するために使用されるスキーマの、テンプレート・ファイルまたは URL へのストリングである可能性があります。

config がビジネス・オブジェクト・タイプである場合は、getStreamFromBO() メソッドをインプリメントして、setupOptions(*config*) を呼び出すことができます。setupOptions() メソッドは、DataHandler 基本クラスに定義されます。このメソッドは、ビジネス・オブジェクト内の属性名をプロパティー名として使用し、デフォルト値をそれらのプロパティーの値として使用します。データ・ハンドラーが使用できるように、オブジェクトの構成プロパティーの値を設定します。

参照

getBO() - パブリック, getByteArrayFromBO(), getStreamFromBO()

---

setConfigMOName()

DataHandler 基本クラスの静的プロパティーに、トップレベルのデータ・ハンドラー・メタオブジェクトの名前を設定します。

構文

```
public static void setConfigMOName(String name);
```

パラメーター

name データ・ハンドラー・メタオブジェクトの名前が含まれたストリング。

戻り値

なし。

例外

Exception
メソッドが指定されたトップレベルのデータ・ハンドラー・メタオブジェクトを設定した場合にスローされます。

注記

トップレベルのデータ・ハンドラー・メタオブジェクトは、サポートされている MIME タイプと、それらに関連付けられた子メタオブジェクトの名前を保持します。データ・ハンドラー・メタオブジェクトについては、25 ページの『データ・ハンドラーの構成』を参照してください。

---

setEncoding()

データ・ハンドラーが使用している文字エンコードを設定します。

構文

```
public final void setEncoding(String encodingName);
```

パラメーター

encodingName
データ・ハンドラーの文字エンコードとして割り当てる新規の値が含まれた String オブジェクト。

戻り値

なし。

注記

setEncoding() メソッドはデータ・ハンドラーの文字エンコードを設定します。文字エンコードはロケールの一部で、ロケールはデータの国/地域別情報を言語、国 (または地域) ごとに定義します。このメソッドは、DataHandler クラスの private 文字エンコード変数を設定する accessor メソッドです。この文字エンコードは、データ・ハンドラーが処理する直列化データの文字エンコードを示します。

このメソッドは、データ・ハンドラーが文字変換などの文字エンコード処理を実行する必要がある場合に役立ちます。

参照

getEncoding()

---

setLocale()

データ・ハンドラーのロケールを設定します。

構文

```
public final void setLocale(Locale localeObject);
```

パラメーター

localeObject
: データ・ハンドラーの環境に割り当てる新規ロケールが含まれた Java Locale オブジェクト。

戻り値

なし。

注記

setLocale() メソッドは、データ・ハンドラーのロケールを設定します。ロケールは、データの国/地域別情報を言語、国 (または地域)、および文字エンコードごとに定義します。このメソッドは、DataHandler クラスの private ロケール変数を設定する accessor メソッドです。このロケールは、データ・ハンドラーが受け取り、作成する直列化データのロケールを示します。

このメソッドは、データ・ハンドラーがそのロケールを変更する必要がある場合に役立ちます。例えば、ビジネス・オブジェクトに変換される直列化データに別のロケールを指定する場合に使用します。getBO() の呼び出しの前に setLocale() を呼び出すと、getBO() が作成するビジネス・オブジェクトにロケールを関連付けるときに使用するロケールが変更されます。

参照

getLocale()

setOption()

指定されたデータ・ハンドラー構成オプションの値を設定します。

構文

```
public void setOption(String name, String value);
```

パラメーター

name 構成オプションの名前。

value 構成オプションの値。

戻り値

なし。

注記

setOption() メソッドは、構成オプションに新しい値を割り当てます。データ・ハンドラーに関連付けられた子メタオブジェクトが存在する場合、createHandler() メソッドは、これらのメタオブジェクト属性のデフォルト値を使用して、その構成オプションを初期化します。データ・ハンドラーがコネクターのコンテキストで呼び出される場合、この子メタオブジェクトは、コネクター・プロセス・メモリーに入れられます。データ・ハンドラーがアクセス・クライアントのコンテキストで呼び出される場合、このメタオブジェクトは、InterChange Server Express プロセスのメモリーに入れられます。setOption() メソッドを使用すると、これらのいずれかのオプションの値をオーバーライドできます。

注: setOption() を使用して構成オプションを変更すると、メモリー内のメタオブジェクト属性の値が設定されます。これは、リポジトリー内の属性の値には 影響しません。

参照

getOption()

traceWrite()

データ・ハンドラーのトレース・メッセージを書き込むための適切なトレース書き込み機能を呼び出します。

構文

```
public void traceWrite(String message, int level);
```

パラメーター

message トレース・メッセージに使用するメッセージ・テキスト。

level メッセージのトレース・レベルを指定する整数。トレース・レベルは 0 から 5 で、0 はトレースを指定しません。5 は完全トレースを指定します。

戻り値

なし。

注記

`traceWrite()` メソッドは、データ・ハンドラーが実行されるコンテキストに応じて、適切なトレース書き込み機能を呼び出すラッパー・メソッドです。デフォルトのトレースはコネクター・トレースです。データ・ハンドラーが Server Access Interface のコンテキストで実行される場合、`traceWrite()` メソッドは、トレース書き込みメソッドを呼び出す前に、トレース・サブシステムを設定します。

付録. XML ODA の使用

この付録では、Object Discovery Agent (ODA) の 1 つであり、XML 文書のビジネス・オブジェクト定義を生成する XML ODA について説明します。XML 文書は、文書タイプ定義 (DTD) またはスキーマ文書によってそのスキーマを定義できるので、XML ODA ではこれらのいずれかのデータ・モデルを使用して、XML 文書に固有のビジネス・オブジェクト要件を明確にすることができます。

この章を構成するセクションは次のとおりです。

- 『インストールと使用法』
- 200 ページの『Business Object Designer Express での XML ODA の使用』
- 210 ページの『生成されるビジネス・オブジェクト定義の内容』
- 211 ページの『ビジネス・オブジェクト定義の情報の変更』

インストールと使用法

このセクションでは、以下について説明します。

- 『XML ODA のインストール』
- 198 ページの『XML ODA を使用する前に』
- 198 ページの『XML ODA の起動』
- 199 ページの『XML ODA の複数インスタンスの実行』
- 199 ページの『エラー・メッセージ・ファイルおよびトレース・メッセージ・ファイルでの作業』

XML ODA のインストール

XML ODA をインストールするには、IBM WebSphere インストーラーを使用します。 IBM WebSphere Business Integration Server Express インストーラーによるこの ODA のインストール方法については、「*WebSphere Business Integration Server Express インストール・ガイド*」を参照してください。

インストールが完了すると、以下のファイルがシステムの製品ディレクトリーにインストールされます。

- `ODA¥XML¥XMLODA.jar`
- `ODA¥messages¥XMLODAAgent.txt`
- `ODA¥messages¥XMLODAAgent_ll_TT.txt files`(言語 (*ll*) および国または地域 (*TT*) に固有のメッセージ・ファイル)
- `ODA¥XML¥start_XMLODA.bat` (Windows のみ)
- `ODA/XML/start_XMLODA.sh` (UNIX のみ)

注: 特に指定がない限り、本書ではディレクトリー・パスの記述に円記号 (¥) を使用します。UNIX システムの場合には、円記号 (¥) はスラッシュ (/) に置き換えてください。製品のすべてのパス名は、製品がシステムにインストールされているディレクトリーに対する相対パス名です。

XML ODA を使用する前に

XML ODA を実行する前に、XML ODA に必要なファイルがシステムに存在することを確認してください。特に、ご使用の製品ディレクトリーの bin サブディレクトリーに ODA 環境ファイルがインストールされていることを確認してください。

UNIX

ODA 環境ファイル (CWODAEnv.sh) が *ProductDir*/bin ディレクトリーにインストールされていることを確認してください。

Windows

ODA 環境ファイル (CWODAEnv.bat) が *ProductDir*¥bin ディレクトリーにインストールされていることを確認してください。

また、ODA を実行する始動スクリプトまたはバッチ・ファイルで変数が正しく設定されていることを確認してください。編集用で、シェル・ファイル (start_XMLODA.sh) またはバッチ・ファイル (start_XMLODA.bat) を開いて、表 57 に記載されている値が正しいことを確認してください。

表 57. シェルおよびバッチ・ファイルの構成変数

変数	説明	例
set AGENTNAME	ODA の名前	set AGENTNAME=XMLODA
set AGENT	ODA の jar ファイルの名前	**UNIX**: set AGENT = ${*ProductDir*}/ODA/XML/XMLODA.jar **WINDOWS**: set AGENT = %*ProductDir*%¥ODA¥XML¥XMLODA.jar
set AGENTCLASS	ODA の Java クラスの名前	set AGENTCLASS=com.crossworlds.oda.xml.XMLAgent

XML ODA をインストールして、シェル・ファイルまたはバッチ・ファイルの構成変数を設定したら (表 57 を参照)、以下の操作を実行してビジネス・オブジェクトを生成する必要があります。

1. XML ODA を起動します。
2. Business Object Designer Express を起動します。
3. Business Object Designer Express の GUI インターフェースであるビジネス・オブジェクト・ウィザードの 6 段階の処理を順に実行し、ODA を構成して実行します。

これらの段階については、以下のセクションで詳しく説明します。

XML ODA の起動

XML ODA は、ご使用のオペレーティング・システムに応じた始動スクリプトを使って起動できます。

UNIX

```
start_XMLODA.sh
```

Windows

```
start_XMLODA.bat
```

注: Windows インストーラーには、インストールする ODA を始動するためのショートカットが用意されています。このインストーラーを使用して XML ODA をインストールした場合、メニューの 「プログラム (Programs)」 > 「IBM WebSphere Business Integration Adapters」 > 「アダプター」 > 「Object Discovery Agent」に始動するためのショートカットがあります。

XML ODA を構成して実行するには、Business Object Designer Express を使用します。Business Object Designer Express が始動するビジネス・オブジェクト・ウィザードは、各スクリプト・ファイルまたはバッチ・ファイルの AGENTNAME 変数に指定された名前を使用して各 ODA を検索します。このコネクターのデフォルトの ODA 名は、XMLODA です。

XML ODA の複数インスタンスの実行

ローカル・ホスト・マシンまたはリモート・ホスト・マシンのいずれかで XML ODA の複数インスタンスを実行することができます。各インスタンスは、固有のポートで実行されます。Business Object Designer Express から ODA を起動する際に、このポート番号を指定できます。 201 ページの図 38 は、Business Object Designer Express のウィンドウで、実行する ODA を選択する様子を示したものです。

エラー・メッセージ・ファイルおよびトレース・メッセージ・ファイルでの作業

エラー・メッセージ・ファイルおよびトレース・メッセージ・ファイル (デフォルトは XMLODAAgent.txt) は、製品ディレクトリーのサブディレクトリーである ¥ODA¥messages に置かれています。これらのファイルには、次の命名規則が使用されます。

*AgentName*Agent.txt

ODA スクリプト・ファイルまたはバッチ・ファイルの複数のインスタンスを作成し、表現している ODA ごとに固有の名前を指定した場合には、ODA のインスタンスごとにメッセージ・ファイルを用意することができます。名前が異なる複数の ODA インスタンスが使用するメッセージ・ファイルを共通にすることもできます。有効なメッセージ・ファイルを指定する方法は、次の 2 通りです。

- ODA の名前を変更し、それに対応するメッセージ・ファイルを作成しない場合には、ODA 構成作業の一部として、Business Object Designer Express でメッセージ・ファイルの名前を変更する必要があります。Business Object Designer Express

はメッセージ・ファイルの名前を指定しますが、実際にファイルを作成するわけではありません。ODA 構成ファイルの一部として表示されたファイルが存在しない場合には、既存のファイルを指すように値を変更してください。

- 特定の ODA に対応する既存のメッセージ・ファイルをコピーし、必要に応じて変更することもできます。Business Object Designer Express は、各ファイルが命名規則に従って命名されることを前提としています。例えば、AGENTNAME 変数に XMLODA1 と指定されている場合、このツールでは、関連付けられるメッセージ・ファイルの名前を XMLODA1Agent.txt であると想定します。したがって、Business Object Designer Express が確認のために ODA 構成の一部としてファイル名を表示した場合、このファイル名は ODA 名が基本になっています。デフォルトのメッセージ・ファイルが正しく命名されていることを確認し、必要に応じて訂正してください。

要確認: ODA の構成時にメッセージ・ファイルの名前を正しく指定できなかった場合、ODA はメッセージを表示しないで動作します。メッセージ・ファイル名の指定の詳細については、202 ページの表 59 を参照してください。

構成の作業時に、以下の内容を指定します。

- XML ODA がエラー情報とトレース情報を書き込むファイルの名前
- トレース・レベル (0 から 5 の範囲)

表 58 で、各トレース・レベルの値を説明します。

表 58. トレース・レベル

トレース・レベル	説明
0	すべてのエラーを記録します。
1	メソッドのすべての開始メッセージおよび終了メッセージをトレースします。
2	ODA のプロパティーとそれらの値をトレースします。
3	すべてのビジネス・オブジェクトの名前をトレースします。
4	作成されたすべてのスレッドの詳細をトレースします。
5	• すべてのプロパティーの ODA 初期化値を示します。 • XML ODA が作成した各スレッドの詳細な状況をトレースします。 • ビジネス・オブジェクト定義ダンプをトレースします。

これらの値をどこで構成するかについては、202 ページの表 59 を参照してください。

Business Object Designer Express での XML ODA の使用

このセクションでは、XML ODA を使用してビジネス・オブジェクト定義を生成するために Business Object Designer Express を使用する方法について説明します。Business Object Designer Express の起動については、「*ビジネス・オブジェクト開発ガイド*」を参照してください。 Business Object Designer Express には、これらの各段階を順に示すビジネス・オブジェクト・ウィザードが用意されています。ODA の起動後、Business Object Designer Express を起動し、(ODA を構成して実行する)

ビジネス・オブジェクト・ウィザードへのアクセスを取得する必要があります。ビジネス・オブジェクト・ウィザードには、ODA を使用してビジネス・オブジェクト定義を生成するための 6 つの段階があります。

ODA の起動後、このウィザードを起動するには、次の手順を実行します。

1. Business Object Designer Express を開きます。
2. 「ファイル」メニューから、「ODA を使用して新規作成...」サブメニューを選択します。

   ビジネス・オブジェクト・ウィザードの最初のウィンドウ (「エージェントの選択」ウィンドウ) が表示されます。図 38 に、このウィンドウを示します。

ODA を選択、構成、および実行するには、以下の手順に従ってください。

1. 『ODA の選択』
2. 202 ページの『構成プロパティーの指定』
3. 204 ページの『ノードの展開と XML エレメントの選択』
4. 205 ページの『オブジェクトの選択の確認』
5. 206 ページの『ビジネス・オブジェクト定義の生成』およびオプションで 207 ページの『追加情報の入力』
6. 209 ページの『ビジネス・オブジェクト定義の保管』

ODA の選択

図 38 に、ビジネス・オブジェクト・ウィザードの 6 段階のウィザードの最初のダイアログ・ボックスを示します。このウィンドウでは、実行する ODA を選択します。

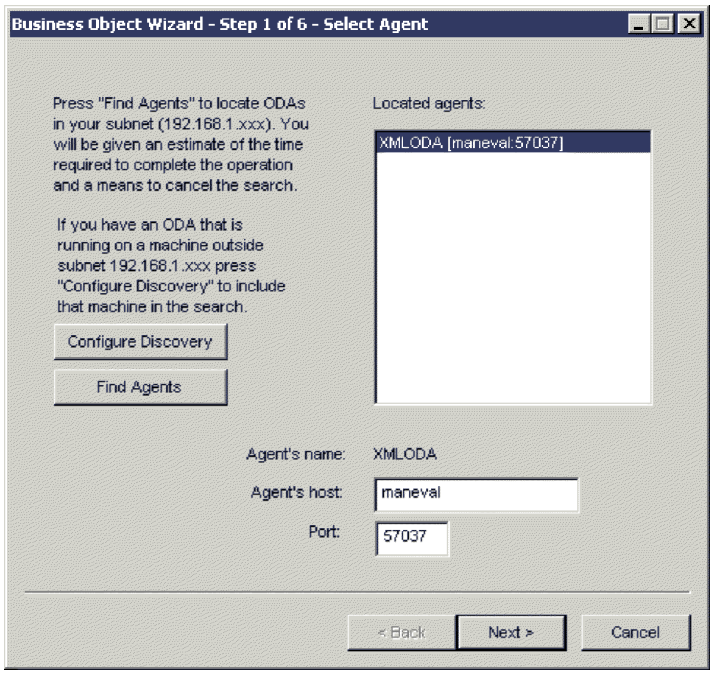

図 38. ODA の選択

ODA を選択するには、以下の手順を行います。

1. 「エージェントの検索」ボタンをクリックすることにより、登録済みまたは現在実行中の ODA のすべてを「検索されたエージェント」フィールドに表示します。あるいは、ホスト名とポート番号を使用して、ODA を検出することもできます。

   注: ビジネス・オブジェクト・ウィザードが目的の ODA を見つけられなかった場合は、ODA の設定を調べてください。

2. 表示リストから、目的の ODA を選択します。

   ビジネス・オブジェクト・ウィザードの「エージェント名」フィールドに、選択した ODA が表示されます。

構成プロパティーの指定

ビジネス・オブジェクト・ウィザードは、XML ODA と初めて通信するときに、図 39 に示すように一連の ODA 構成プロパティーを入力するよう求めてきます。

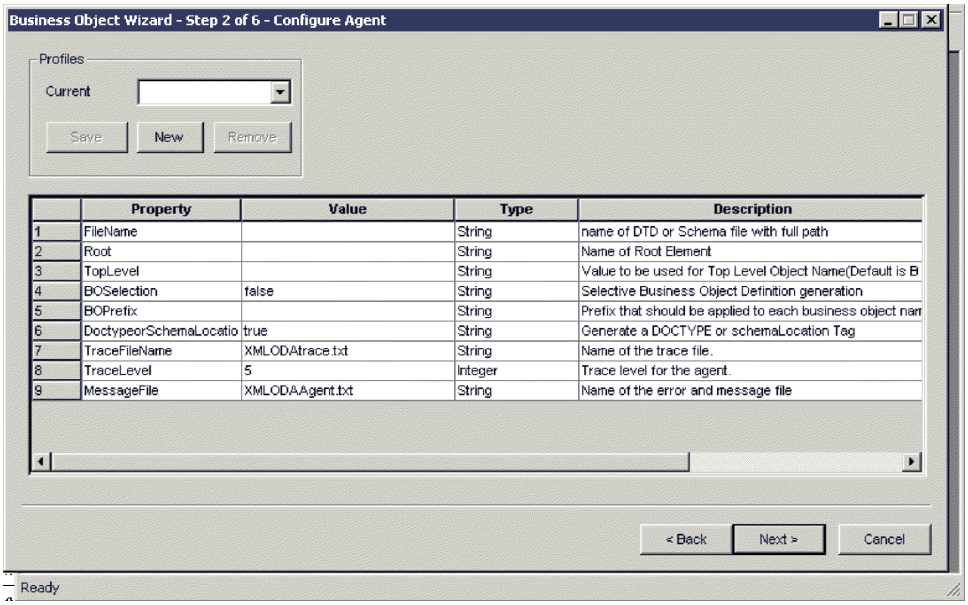

図 39. ODA 構成プロパティーの指定

XML ODA プロパティーの構成を表 59 に示します。

表 59. XML ODA 構成プロパティー

行番号	プロパティー名	プロパティー・タイプ	説明
1	FileName	String	DTD またはスキーマ文書の絶対パス名。 DTD ファイルには、.dtd という拡張子が必要です。スキーマ文書ファイルには、.xsd という拡張子が必要です。

表 59. XML ODA 構成プロパティー (続き)

行番号	プロパティー名	プロパティー・タイプ	説明
2	Root	String	root エレメントとして処理される予定の XML エレメント。 root エレメントを指定しない場合、XML ODA は、以下を前提事項として動作します。 • ODA が DTD の構文解析をする場合は、最初の XML エレメントが root として処理されます。 • ODA がスキーマ文書の構文解析をする場合は、最初のグローバル・エレメントが root として処理されます。
3	TopLevel	String	ODA が生成するトップレベル・ビジネス・オブジェクトに使用する名前。ODA は、トップレベル・ビジネス・オブジェクトの先頭にビジネス・オブジェクト・プレフィックスを付加します。このプレフィックスは BOPrefix プロパティーによって指定され、下線 (_) で区切られます。トップレベルの名前を*指定しない* と、ODA では、*BOPrefix_Root* (*BOPrefix* および *Root* は BOPrefix プロパティーおよび Root プロパティーの値) という名前をトップレベル・ビジネス・オブジェクトの名前として割り当てます。
4	BOSelection	String	ビジネス・オブジェクト定義の適用対象となるエレメントの名前を XML ODA によって選択できるかどうかを指定するブール値 (true または false)。 • このプロパティーを false に設定すると、ユーザーは XML ODA によって root エレメント以外のエレメントを選択できなくなり、root とそのすべての子エレメントのビジネス・オブジェクト定義が生成されるようになります。 • このプロパティーを true に設定すると、ユーザーは XML ODA によって すべてのエレメントを選択できるようになり、選択したエレメントのビジネス・オブジェクト定義のみが生成されるようになります。 デフォルトは false です。
5	BOPrefix	String	XML 文書の各ビジネス・オブジェクト定義の名前に対して ODA が適用するプレフィックス。ビジネス・オブジェクト・プレフィックスを指定しないと、ODA はビジネス・オブジェクト定義の名前の先頭にストリングを*付加しません*。
6	DoctypeorSchemaLocation	String	XML ODA が以下を対象として属性を生成するかどうかを指定するブール値 (true または false)。 • DTD の処理時: DOCTYPE タグ • スキーマ文書の処理時: schemaLocation 属性および xsi 属性 (XML スキーマ・インスタンス・ネーム・スペース内) デフォルトは true です。

表 59. XML ODA 構成プロパティー (続き)

行番号	プロパティー名	プロパティー・タイプ	説明
7	TraceFileName	String	XML ODA によるトレース情報の書き込み先ファイルの絶対パス名。このファイルが存在しない場合、XML ODA は指定のディレクトリーにこのファイルを作成します。このファイルがすでに存在する場合、XML ODA はこのファイルに追記します。 デフォルトでは、XML ODA により、XMLODAtrace.txt という名前のトレース・ファイルが製品ディレクトリーの ODA¥XML サブディレクトリーに作成されます。 トレース・ファイルに別の名前を指定する場合は、このプロパティーを使用してください。
8	TraceLevel	Integer	XML ODA に対して有効なトレースのレベル。有効な値は 0 から 5 です。このプロパティーのデフォルト値は 5 (フル・トレースが有効) です。詳細については、199 ページの『エラー・メッセージ・ファイルおよびトレース・メッセージ・ファイルでの作業』を参照してください。
9	MessageFile	String	エラー・ファイルおよびメッセージ・ファイルの絶対パス名。デフォルトでは、XML ODA により、XMLODAAgent.txt という名前のメッセージ・ファイルおよびエラー・ファイルが作成されます。 **要確認**: エラー・ファイルとメッセージ・ファイルは必ず製品ディレクトリーの ODA¥messages サブディレクトリーに置いてください。 このプロパティーは、既存のファイルを確認または指定するときに使用します。

要確認: Business Object Designer Express に表示されているデフォルト値が、存在しないファイルを指している場合には、メッセージ・ファイルの名前を訂正します。このダイアログ・ボックスから次に進んだときに名前が不正確な場合、Business Object Designer Express は ODA の起動元となったウィンドウにエラー・メッセージを表示します。このメッセージは、Business Object Designer Express ではポップアップしません。有効なメッセージ・ファイルを指定できないと、ODA はメッセージを表示せずに動作します。

これらのプロパティーは、XML ODA を使用するたびに再入力しなくても済むよう、指定のプロファイルに保存することができます。 ODA プロファイルの指定については、「*ビジネス・オブジェクト開発ガイド*」を参照してください。

ノードの展開と XML エレメントの選択

Business Object Designer Express は、前の段階で構成されたプロパティーを使用してツールを指定の XML スキーマ (DTD またはスキーマ文書) に接続します。接続後、Business Object Designer Express はツリーを表示します。このツリーのノードは、XML スキーマで定義されているすべての XML エレメントを表しています。

トップレベルの XML エレメントを展開すると、階層全体を表示できます。XML ODA は、XML エレメントごとに子ビジネス・オブジェクト定義を作成します。

図 40 に、このダイアログ・ボックスといくつかの XML エレメントが展開されている様子を示します。

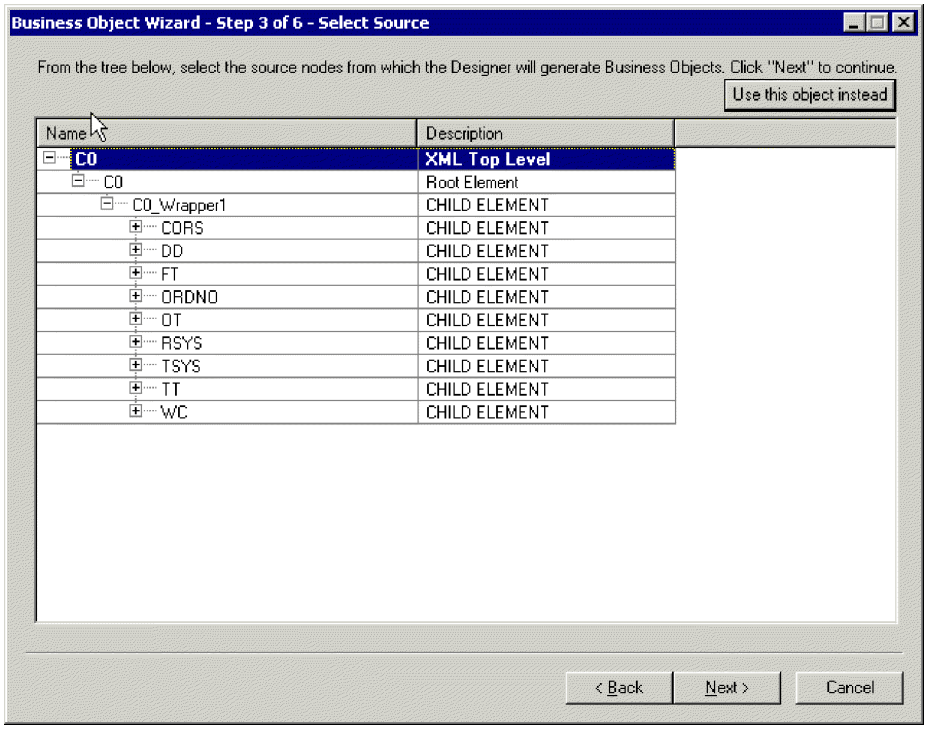

図 40. 展開されたノードが表示されている XML エレメントのツリー

必要な XML エレメントをすべて選択して、「次へ」をクリックします。

オブジェクトの選択の確認

生成済みのビジネス・オブジェクト定義を関連付ける予定の XML エレメントをすべて指定した後に Business Object Designer Express が表示するダイアログ・ボックスには、選択済みオブジェクトのみが表示されます。 206 ページの図 41 に、このダイアログ・ボックスを示します。

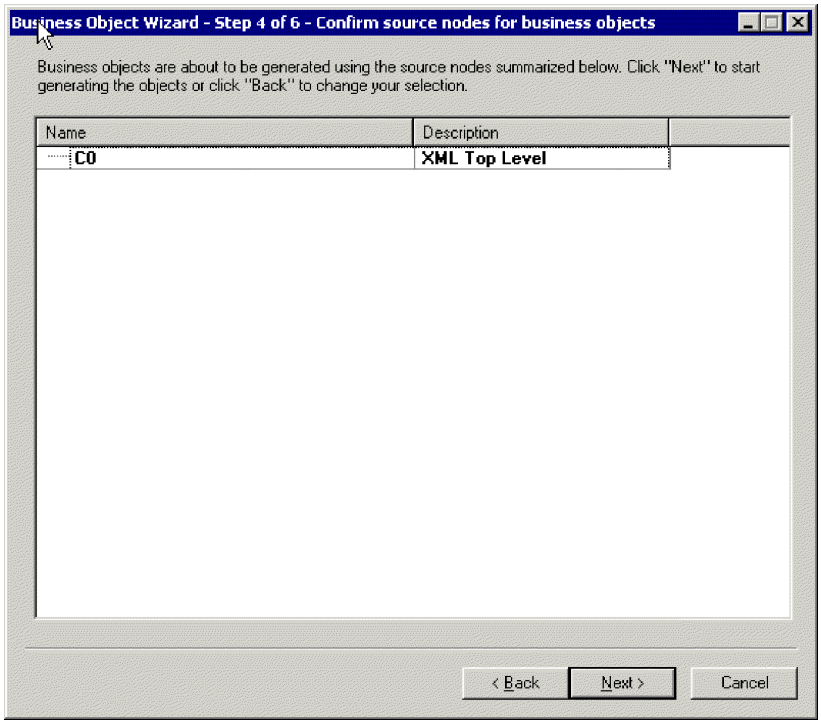

図 41. オブジェクトの選択の確認

このウィンドウには、以下のオプションがあります。

- 選択を確認するには、「次へ」をクリックします。
- 選択に誤りがあった場合には、「戻る」をクリックして直前のウィンドウに戻り、必要な変更を加えます。選択が正しい場合には、「次へ」をクリックします。

ビジネス・オブジェクト定義の生成

XML エレメントを選択後、次のダイアログ・ボックスでは、Business Object Designer Express がビジネス・オブジェクト定義を生成していることが通知されます。大量のコンポーネント・インターフェースを選択した場合は、この生成に時間がかかることがあります。

207 ページの図 42 に、このダイアログ・ボックスを示します。

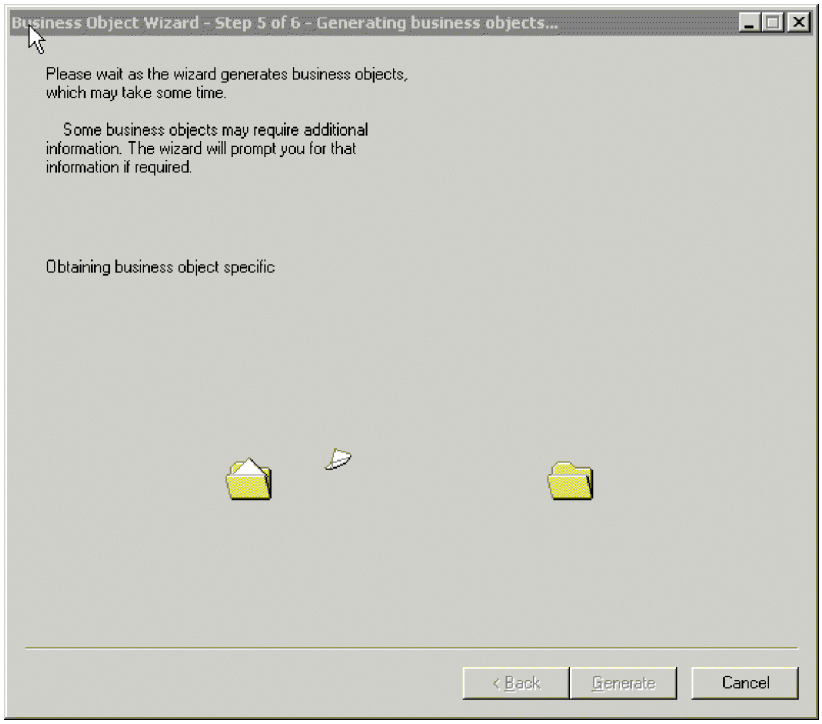

図 42. ビジネス・オブジェクト定義の生成

XML ODA は、次の情報からビジネス・オブジェクト定義の名前を生成します。

- `BOPrefix` ODA 構成プロパティーの値
- `TopLevel` ODA 構成プロパティーの値
- ビジネス・オブジェクト定義が表す XML エレメントの名前

XML ODA は、これらの値を下線 (_) 文字で区切ります。したがって、生成される名前は次のようなフォーマットになります。

`BOPrefix_TopLevel_XMLelement`

追加情報の入力

XML ODA には動詞に関する追加情報が必要なため、Business Object Designer Express は「BO プロパティー」ウィンドウを表示して、情報の入力を求めてきます。 208 ページの図 43 に、このダイアログ・ボックスを示します。

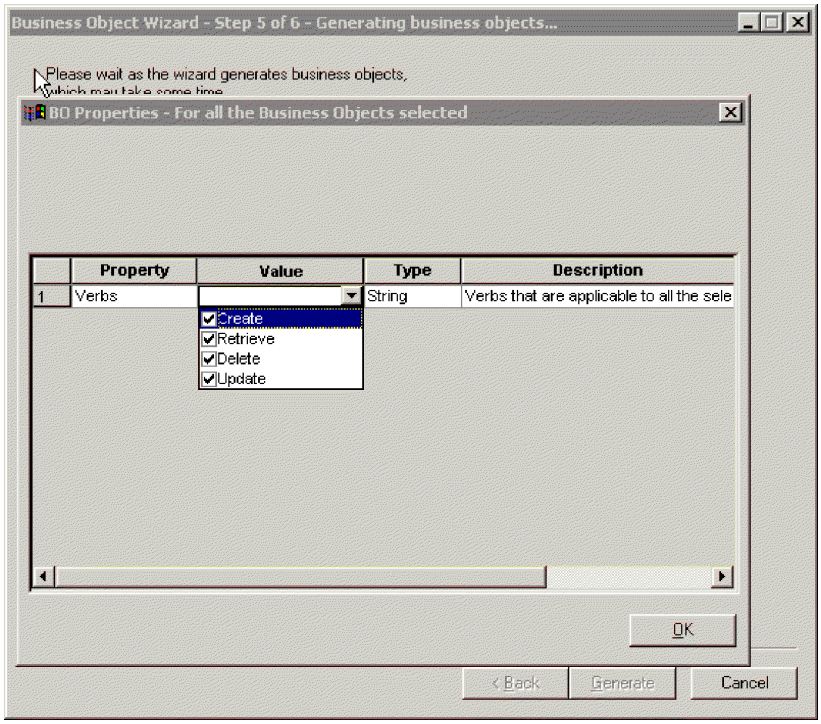

図 43. 追加情報の入力 - 動詞

「BO プロパティー」ウィンドウでは、動詞の情報を入力または変更します。「*値*」フィールドの内側をクリックして、ポップアップ・メニューから 1 つ以上の動詞を選択します。ポップアップ・メニューに表示されるのは、対象のビジネス・オブジェクトによってサポートされる動詞です。

注: 「BO プロパティー」ダイアログ・ボックスのフィールドに複数の値が設定されている場合、このダイアログ・ボックスを初めて表示したときは、このフィールドに何も表示されません。フィールドの内側をクリックすると、その値のドロップダウン・リストが表示されます。

XML 文書内に `anyAttribute` エレメントを含むスキーマ文書がある場合は、図 44 に示すように、追加の「BO プロパティー」ウィンドウが表示されます。

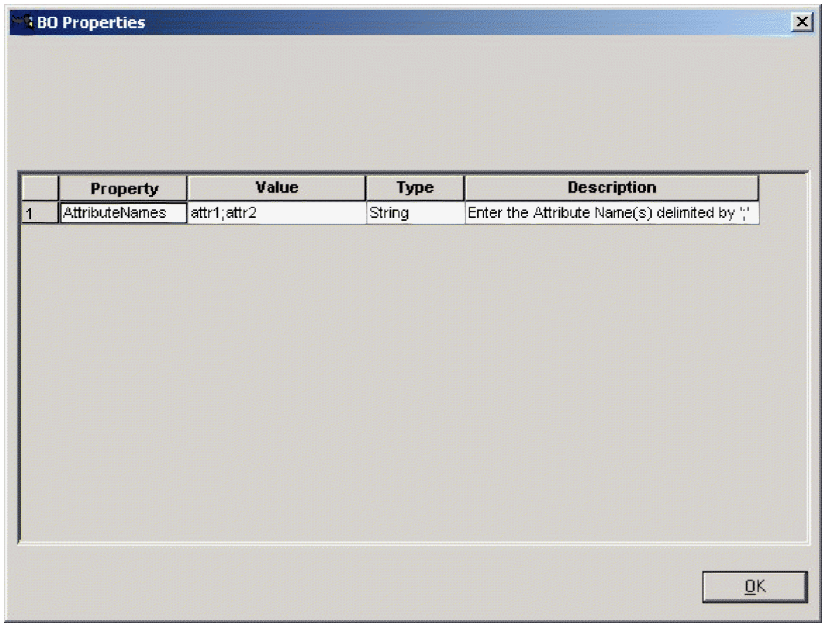

図 44. 追加情報の入力 - 属性名

この「BO プロパティー」ウィンドウでは、XML ODA により作成するビジネス・オブジェクトの属性名を入力します。各属性の間はセミコロン (;) で区切ってください。`anyAttribute` の詳細については、86 ページの『サポートされるスキーマ文書構造』を参照してください。

ビジネス・オブジェクト定義の保管

「BO プロパティー」ダイアログ・ボックスで必要なすべての情報を指定し、「OK」をクリックすると、Business Object Designer Express にウィザードの最終ダイアログ・ボックスが表示されます。このダイアログ・ボックスでは、以下のいずれかの操作を実行できます。

- ビジネス・オブジェクト定義をサーバーに保管する (InterChange Server が統合ブローカーの場合)。
- ビジネス・オブジェクト定義をファイルに保管する (統合ブローカーは任意)。
- 編集のため Business Object Designer Express でビジネス・オブジェクト定義を開く。

変更方法の詳細については、「*ビジネス・オブジェクト開発ガイド*」を参照してください。

210 ページの図 45 に、このダイアログ・ボックスを示します。

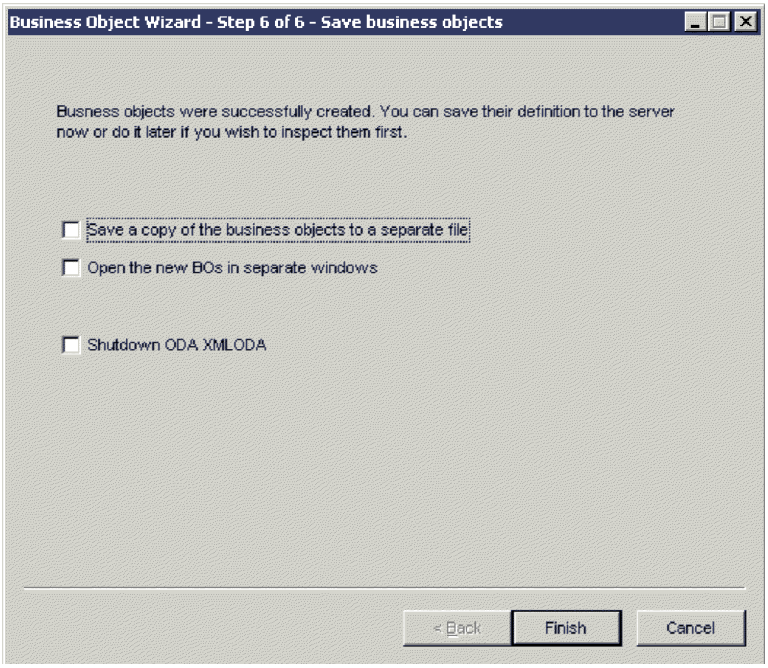

図 45. ビジネス・オブジェクト定義の保管

生成されるビジネス・オブジェクト定義の内容

XML ODA が生成する属性、動詞、アプリケーション固有情報などのビジネス・オブジェクト定義は、次に示すセクションに説明されています。

ビジネス・オブジェクト定義のコンポーネント	詳細
属性	38 ページの『ビジネス・オブジェクトの構造』 39 ページの『ビジネス・オブジェクトの属性プロパティー』
アプリケーション固有情報	42 ページの『アプリケーション固有情報』
動詞	43 ページの『ビジネス・オブジェクト動詞』

注: 以前のバージョンの XML ODA では、ObjectEventId 属性をキーとして指定する親ビジネス・オブジェクト定義を生成していました。Business Object Designer Express では、ビジネス・オブジェクト定義で ObjectEventId をキー属性に指定することはできなくなりました。このため、XML ODA では、この振る舞いを避けるために特別なステップの実行が必要になります。この新しい振る舞いでは、生成されたビジネス・オブジェクト定義をユーザーが変更して、キー属性を指定する必要があります。詳細については、 40 ページの『Key 属性および Foreign Key 属性のプロパティー』を参照してください。

ビジネス・オブジェクト定義の情報の変更

XML ODA によって作成されたビジネス・オブジェクト定義に登録されている情報は、変更が必要になる場合があります。例えば、不要な属性を手動で削除したり、属性のアプリケーション固有情報に必要なタグを追加する必要があります。ビジネス・オブジェクト定義を調べたり変更したりするには、Business Object Designer Express またはテキスト・エディターを使用します。修正後の定義をリポジトリーに再ロードするには、Business Object Designer Express を使用します。 InterChange Server (ICS) が統合ブローカーの場合は、`repos_copy` コマンドを使用して定義をリポジトリーにロードできます。WebSphere MQ Integrator Broker が統合ブローカーの場合は、システム・コマンドを使用して該当するファイルをリポジトリー・ディレクトリーにコピーできます。

特記事項

本書に記載の製品、サービス、または機能が日本においては提供されていない場合があります。日本で利用可能な製品、サービス、および機能については、日本 IBM の営業担当員にお尋ねください。本書で IBM 製品、プログラム、またはサービスに言及していても、その IBM 製品、プログラム、またはサービスのみが使用可能であることを意味するものではありません。これらに代えて、IBM の知的所有権を侵害することのない、機能的に同等の製品、プログラム、またはサービスを使用することができます。ただし、IBM 以外の製品とプログラムの操作またはサービスの評価および検証は、お客様の責任で行っていただきます。

IBM は、本書に記載されている内容に関して特許権 (特許出願中のものを含む) を保有している場合があります。本書の提供は、お客様にこれらの特許権について実施権を許諾することを意味するものではありません。実施権についてのお問い合わせは、書面にて下記宛先にお送りください。

〒106-0032
東京都港区六本木 3-2-31
IBM World Trade Asia Corporation
Licensing

以下の保証は、国または地域の法律に沿わない場合は、適用されません。

IBM およびその直接または間接の子会社は、本書を特定物として現存するままの状態で提供し、商品性の保証、特定目的適合性の保証および法律上の瑕疵担保責任を含むすべての明示もしくは黙示の保証責任を負わないものとします。国または地域によっては、法律の強行規定により、保証責任の制限が禁じられる場合、強行規定の制限を受けるものとします。

この情報には、技術的に不適切な記述や誤植を含む場合があります。本書は定期的に見直され、必要な変更は本書の次版に組み込まれます。IBM は予告なしに、随時、この文書に記載されている製品またはプログラムに対して、改良または変更を行うことがあります。

本書において IBM 以外の Web サイトに言及している場合がありますが、便宜のため記載しただけであり、決してそれらの Web サイトを推奨するものではありません。それらの Web サイトにある資料は、この IBM 製品の資料の一部ではありません。それらの Web サイトは、お客様の責任でご使用ください。

IBM は、お客様が提供するいかなる情報も、お客様に対してなんら義務も負うことのない、自ら適切と信ずる方法で、使用もしくは配布することができるものとします。

本プログラムのライセンス保持者で、(i) 独自に作成したプログラムとその他のプログラム（本プログラムを含む）との間での情報交換、および (ii) 交換された情報の相互利用を可能にすることを目的として、本プログラムに関する情報を必要とする方は、下記に連絡してください。

IBM Burlingame Laboratory Director

IBM Burlingame Laboratory
577 Airport Blvd., Suite 800
Burlingame, CA 94010
U.S.A

本プログラムに関する上記の情報は、適切な使用条件の下で使用することができますが、有償の場合もあります。

本書で説明されているライセンス・プログラムまたはその他のライセンス資料は、IBM 所定のプログラム契約の契約条項、IBM プログラムのご使用条件、またはそれと同等の条項に基づいて、IBM より提供されます。

この文書に含まれるいかなるパフォーマンス・データも、管理環境下で決定されたものです。そのため、他の操作環境で得られた結果は、異なる可能性があります。一部の測定が、開発レベルのシステムで行われた可能性がありますが、その測定値が、一般に利用可能なシステムのものと同じである保証はありません。さらに、一部の測定値が、推定値である可能性があります。実際の結果は、異なる可能性があります。お客様は、お客様の特定の環境に適したデータを確かめる必要があります。

IBM 以外の製品に関する情報は、その製品の供給者、出版物、もしくはその他の公に利用可能なソースから入手したものです。IBM は、それらの製品のテストは行っておりません。したがって、他社製品に関する実行性、互換性、またはその他の要求については確証できません。IBM 以外の製品の性能に関する質問は、それらの製品の供給者にお願いします。

本書には、日常の業務処理で用いられるデータや報告書の例が含まれています。より具体性を与えるために、それらの例には、個人、企業、ブランド、あるいは製品などの名前が含まれている場合があります。これらの名称はすべて架空のものであり、名称や住所が類似する企業が実在しているとしても、それは偶然にすぎません。

IBM の将来の方向または意向に関する記述については、予告なしに変更または撤回される場合があり、単に目標を示しているものです。

著作権使用許諾

本書には、様々なオペレーティング・プラットフォームでのプログラミング手法を例示するサンプル・アプリケーション・プログラムがソース言語で掲載されています。お客様は、サンプル・プログラムが書かれているオペレーティング・プラットフォームのアプリケーション・プログラミング・インターフェースに準拠したアプリケーション・プログラムの開発、使用、販売、配布を目的として、いかなる形式においても、IBM に対価を支払うことなくこれを複製し、改変し、配布することができます。このサンプル・プログラムは、あらゆる条件下における完全なテストを経ていません。従って IBM は、これらのサンプル・プログラムについて信頼性、利便性もしくは機能性があることをほのめかしたり、保証することはできません。

プログラミング・インターフェース情報

プログラミング・インターフェース情報は、プログラムを使用してアプリケーション・ソフトウェアを作成する際に役立ちます。

一般使用プログラミング・インターフェースにより、お客様はこのプログラム・ツール・サービスを含むアプリケーション・ソフトウェアを書くことができます。

ただし、この情報には、診断、修正、および調整情報が含まれている場合があります。診断、修正、調整情報は、お客様のアプリケーション・ソフトウェアのデバッグ支援のために提供されています。

警告: 診断、修正、調整情報は、変更される場合がありますので、プログラミング・インターフェースとしては使用しないでください。

商標

以下は、IBM Corporation の商標です。

IBM
IBM ロゴ
AIX
CrossWorlds
DB2
DB2 Universal Database
Lotus
Lotus Domino
Lotus Notes
MQIntegrator
MQSeries
Tivoli
WebSphere

Microsoft、Windows、Windows NT および Windows ロゴは、Microsoft Corporation の米国およびその他の国における商標です。

MMX および Pentium は、Intel Corporation の米国およびその他の国における商標です。

Java およびすべての Java 関連の商標およびロゴは、Sun Microsystems, Inc. の米国およびその他の国における商標または登録商標です。

他の会社名、製品名およびサービス名等はそれぞれ各社の商標です。

索引

日本語, 数字, 英字, 特殊文字の順に配列されています。なお, 濁音と半濁音は清音と同等に扱われています。

[ア行]

アクセス・クライアント 4, 10
  トップレベルのメタオブジェクト 26
アプリケーション固有情報
  サイズ制限 42
  属性 54, 78, 93
  ビジネス・オブジェクト 50, 71
  Delimited データ・ハンドラー 133
  FixedWidth データ・ハンドラー 123
  NameValue データ・ハンドラー 143
  Request-Response データ・ハンドラーおよび 108
  XML データ・ハンドラー 50
エスケープ文字 131, 136
エスケープ・ストリング 132
エンティティー・リゾルバー 36, 95, 98
応答データ・ハンドラー 99, 116

[カ行]

開発過程 151, 153
カスタム・データ・ハンドラー 96, 151, 177
  開発過程 151
  スタブ・ファイルの使用 156
  設計 155
  ネーム・ハンドラー 117, 171
  ビジネス・オブジェクトのセットアップ 176
  必須メソッド 157
  メソッドのインプリメント 156
  メタオブジェクト 155, 174
  ロケーション 6
  getBO() の例 157
  getStreamFromBO() の例 170
  getStringFromBO() の例 165
  JAR ファイルへの追加 173
コネクター 3
  構成 30, 177
  データ・ハンドラーのインスタンス化 19
  データ・ハンドラーの使用 7
  トップレベルのメタオブジェクト 26, 28
子メタオブジェクト 6, 14, 29, 174
  Delimited データ・ハンドラー 130
  FixedWidth データ・ハンドラー 121
  NameValue データ・ハンドラー 140
  Request-Response データ・ハンドラー 112
  XML データ・ハンドラー 43

[サ行]

スキーマ文書 61, 89
  エレメント 51, 64, 79, 81, 84, 93
  エンティティー・リゾルバー 36
  コメント 84, 93
  混合ビジネス・オブジェクト 65
  サンプル 62
  出現インディケーター 70
  処理命令 85, 93
  スキーマ・ロケーション 36, 63, 85, 202, 203
  属性 83, 84, 93
  属性プロパティー 70
  ターゲット・ネーム・スペース 72
  単純エレメント 81
  単純タイプ 81
  通常のビジネス・オブジェクト 64
  デフォルト・ネーム・スペース 77
  ネーム・スペース 72
  ビジネス・オブジェクト定義の作成 86, 89, 200, 210
  ビジネス・オブジェクト定義の要件 61
  ビジネス・オブジェクトの構造 61
  必須エレメント 88
  必要なビジネス・オブジェクト定義 38, 62
  複合タイプ 64, 65, 79, 81, 88
  ラッパー・ビジネス・オブジェクト 67, 76
  all グループ 67, 87
  any エレメント 87
  anyAttribute エレメント 87, 208
  attributeFormDefault 77, 84
  choice グループ 67, 87
  elementFormDefault 77, 80
  form 属性 80, 83
  import エレメント 74, 88
  include エレメント 87
  root エレメント 35, 38, 62, 63, 91, 203
  schema エレメント 62
  sequence グループ 64, 65, 87
  targetNamespace 72
  use 属性 71
  xmlns 属性 76
属性
  単純 80, 81
  複合 80
  無視 164
属性プロパティー
  カーディナリティー 49, 50, 70
  Cardinality 123, 132, 142
  Delimited データ・ハンドラー 132
  FixedWidth データ・ハンドラー 123
  Foreign Key 40, 41, 50, 71

属性プロパティー *(続き)*
Key 40, 41, 50, 71
MaxLength 123
Name 39, 108, 109, 110, 123, 132, 142
NameValue データ・ハンドラー 141
Required 41, 49, 70
Type 39, 123, 132, 142
XML データ・ハンドラー 39, 49, 70

[タ行]

データ変換
ストリームへの 191
ストリングへの 192
バイト配列への 188
ビジネス・オブジェクトから 8, 12, 163, 188, 191, 192
ビジネス・オブジェクトへの 9, 13, 157, 183, 185
データ・ハンドラー 3
インスタンス化 9, 10, 12, 13, 14, 22, 182
開発過程 151
概要 23
カスタマイズ 6, 96, 117, 151, 177
基本 5
基本クラス 156, 181
クラス 15, 156
クラスの識別 14
構成 17, 25, 32, 184, 189, 191, 192
構成オプション 187, 190, 195
国際化 177, 181
コネクター 7
コンテキスト 7
コンパイル・スクリプト 173
サンプル 6, 153
属性の無視 158, 164
特殊 5
トレース 195
パッケージ 17, 156
メタデータ駆動 22
文字エンコード 189, 193
呼び出しトリガー・フローで 7
ロケーション 5
ロケール 179, 189, 194
IBM 提供 5, 17
Server Access Interface による 10
動詞 (XML での保存) 43, 94
トップレベルのメタオブジェクト 6, 14, 19, 175, 193
トレース・メッセージ 195

[ナ行]

ネーム・ハンドラー 171, 187
Request-Response データ・ハンドラーおよび 101, 117
XML データ・ハンドラー 35, 95, 96

[ハ行]

ビジネス・オブジェクト
混合 51, 65
作成 183
属性の無視 158, 164
通常 51, 64
データ・ハンドラー 176
取り込み 10, 13, 183
名前 96
名前の取得 35, 101, 186
ビジネス・オブジェクトからの変換 8, 12, 163, 188, 191, 192
区切りデータ 135
固定幅ストリング 125
名前と値のペア 145
XML 文書 94
ビジネス・オブジェクトへの変換 9, 13, 157, 185
区切りデータ 134
固定幅 ストリング 124
名前と値のペア 144
入力フォーマット 115
XML 文書 92
プレフィックス 16, 182
戻り 183, 185
要件
Delimited データ・ハンドラー 131
FixedWidth データ・ハンドラー 122
NameValue データ・ハンドラー 141
Request-Response データ・ハンドラー 107
XML データ・ハンドラー 47, 61
ラッパー 51, 66, 67, 76
ロケール 179
ビジネス・オブジェクト定義
親 52, 66, 67
混合 51, 65
作成 89, 110, 200, 210
スキーマ文書 61, 89
スキーマ文書から 86
スキーマ文書の要件 47, 61
通常 51, 64
トップレベル
Request-Response データ・ハンドラー 101, 108, 111
XML データ・ハンドラー 38, 48, 61, 62, 90, 92
名前 207
ラッパー 51, 66, 67, 76
DTD から 60
DTD と 47, 61
root エレメント 38, 48, 63, 86, 91
文書タイプ定義 (DTD) 47, 61
エレメント 55, 58, 93
エンティティー・リゾルバー 36, 44
外部 60
コメント 59, 93
混合ビジネス・オブジェクト 51
サポートされる構造 60, 86

文書タイプ定義 (DTD) *(続き)*
サンプル 48
条件セクション 61
処理命令 59, 93
属性 56, 58, 93
属性プロパティー 39, 49
通常のビジネス・オブジェクト 51
動詞の保持 43
ネーム・スペース 61
パス 44
ビジネス・オブジェクト定義の作成 60, 89, 200, 210
ビジネス・オブジェクト定義の要件 47
ビジネス・オブジェクト定義への変換 60
ビジネス・オブジェクトの構造 38, 48
必要なビジネス・オブジェクト定義 38, 48
ラッパー・ビジネス・オブジェクト 51
ロケーション 202
ANY ディレクティブ 60
ATTLIST フラグメント 50, 71
CDATA セクション 57, 59, 93
DOCTYPE 宣言 48, 58, 61, 93, 96
ELEMENT フラグメント 50
FIXED 属性 48, 96
PCDATA エレメント 51, 56, 57, 93
root エレメント 35, 38, 48, 91, 203

[マ行]

メタオブジェクト 14, 25, 30
アクセス・クライアントによる使用 26
カスタム・データ・ハンドラー 155, 174
子 6, 14, 16, 17, 29
構造 6, 14
コネクター 177
コネクターによる使用 28
作成 174
使用 155
セットアップ 176
トップレベル 6, 14, 16, 17, 19, 26, 175, 193
名前の設定 19, 21, 193
ロード 176
MIME タイプ属性 6, 7, 25
Server Access Interface による使用 26
文字エンコード 178, 189, 193

[ヤ行]

要求データ・ハンドラー 99, 115

[ラ行]

ロケール 177, 178, 189, 194

A

Adapter 開発キット (ADK) 153
Alignment データ・ハンドラー構成プロパティー 121

B

BOCountSize データ・ハンドラー構成プロパティー 121
BONameSize データ・ハンドラー構成プロパティー 121, 126
BOPrefix XML ODA プロパティー 203, 207
BOPrefix データ・ハンドラー構成プロパティー 43, 112
要求/応答ネーム・ハンドラーおよび 101
createHandler() 18, 182
getBOName() 172, 187
XML ネーム・ハンドラー 35, 95, 96
BOSelection XML ODA プロパティー 203
BOVerbSize データ・ハンドラー構成プロパティー 121, 126

C

Cardinality 属性プロパティー
Delimited データ・ハンドラー 132
FixedWidth データ・ハンドラー 123
NameValue データ・ハンドラー 142
XML データ・ハンドラー 49, 50, 70
ClassName データ・ハンドラー構成プロパティー 16, 17, 31, 174
Delimited データ・ハンドラー 131
FixedWidth データ・ハンドラー 121
NameValue データ・ハンドラー 140
Request-Response データ・ハンドラー 113
XML データ・ハンドラー 44
createHandler() メソッド 15, 20, 22, 31, 182
クラスの検索 15, 173
クラス名による 15, 17, 156
コネクターによる呼び出し 9, 10
MIME タイプによる 15, 17, 155
Server Access Interface による呼び出し 12, 13
CustDataHandler.jar ファイル 6, 15, 17, 173
CwDataHandler.jar ファイル 5, 15, 17, 25, 173
CwXMLDataHandler.jar ファイル 6, 15, 25
CxBlank 属性値
Delimited データ・ハンドラー 131, 133, 136
FixedWidth データ・ハンドラー 121
NameValue データ・ハンドラー 140, 143, 146
XML データ・ハンドラー 42, 94
CxBlank データ・ハンドラー構成プロパティー
Delimited データ・ハンドラー 131, 133, 136, 146
FixedWidth データ・ハンドラー 121, 126
NameValue データ・ハンドラー 140, 143
CxBlankValue データ・ハンドラー構成プロパティー (使用すべきでない) 140
CxIgnore 属性値
Delimited データ・ハンドラー 132, 136
FixedWidth データ・ハンドラー 122, 126
NameValue データ・ハンドラー 142, 146

CxIgnore 属性値 *(続き)*
XML データ・ハンドラー 41, 94
CxIgnore データ・ハンドラー構成プロパティー
Delimited データ・ハンドラー 131, 132, 136
FixedWidth データ・ハンドラー 122, 126
NameValue データ・ハンドラー 140, 142, 146

D

Data Handler API 154
DataHandler クラス 154, 181, 196
拡張 156
抽象メソッド 157
パッケージ 181
createHandler() 182
getBOName() 186
getBooleanOption() 187
getBO() (抽象) 183
getBO() (パブリック) 185
getByteArrayFromBO() 188
getEncoding() 189
getLocale() 189
getOption() 190
getStreamFromBO() 191
getStringFromBO() 192
setConfigMOName() 193
setEncoding() 193
setLocale() 194
setOption() 195
traceWrite() 195
DataHandler パッケージ 181
DefaultEscapeBehavior データ・ハンドラー構成プロパティー 44, 58
DefaultVerb データ・ハンドラー構成プロパティー
NameValue データ・ハンドラー 140
Delimited データ・ハンドラー 129, 139
アプリケーション固有情報 133
エスケープ・ストリング 132
概要 129
既存のビジネス・オブジェクトを用いて 133
機能 129
区切り文字 132
構成 130
子メタオブジェクト 130
サンプル・ファイル 154
処理 130
ストリングのビジネス・オブジェクトへの変換 135
ストリングの要件 135
ストリングの例 136
ビジネス・オブジェクトの構造 132
ビジネス・オブジェクトのストリングへの変換 134
ビジネス・オブジェクトの属性プロパティー 132
ビジネス・オブジェクトの要件 131
ClassName 131
CxBlank 131
CxIgnore 131
Delimited データ・ハンドラー *(続き)*
Delimiter 131
DummyKey 131
Escape 131
OmitObjectEventId 131
Delimiter データ・ハンドラー構成プロパティー 130, 131, 132, 135, 136
DoctypeorSchemaLocation XML ODA プロパティー 48, 63, 86, 203
DTDPath データ・ハンドラー構成プロパティー 36, 44
DummyKey データ・ハンドラー構成プロパティー
Delimited データ・ハンドラー 131
FixedWidth データ・ハンドラー 122
NameValue データ・ハンドラー 141
XML データ・ハンドラー 44

E

EntityResolver データ・ハンドラー構成プロパティー 36, 44, 95, 98
Escape データ・ハンドラー構成プロパティー 130, 131, 136
E-Business Development Kit (EDK) 156

F

FileName XML ODA プロパティー 87, 202
FixedWidth データ・ハンドラー 119, 129
アプリケーション固有情報 123
埋め込み文字 120
概要 119
既存のビジネス・オブジェクトを用いて 123
機能 119
桁そろえ値 120
構成 120
子メタオブジェクト 121
サンプル・ファイル 154
処理 120
ストリングのビジネス・オブジェクトへの変換 125
ストリングの要件 125
ビジネス・オブジェクトの構造 123
ビジネス・オブジェクトのストリングへの変換 124
ビジネス・オブジェクトの属性プロパティー 123
ビジネス・オブジェクトの要件 122
Alignment 121
BOCountSize 121
BONameSize 121
BOVerbSize 121
ClassName 121
CxBlank 121
CxIgnore 122
DummyKey 122
MaxLength 属性プロパティー 119, 130
OmitObjectEventId 122
PadCharacter 122
Truncation 122, 125

Foreign Key 属性プロパティー
XML データ・ハンドラー 40, 41, 50, 71

G

getBOName() メソッド 170, 172, 179, 186
getBooleanOption() メソッド 171, 187
getBO() メソッド 10, 13, 157
抽象 157, 183
パブリック 170, 179, 185
getByteArrayFromBO() メソッド 157, 163, 188
getEncoding() メソッド 180, 189
getLocale() メソッド 179, 189
getOption() メソッド 171, 190
getStreamFromBO() メソッド 157, 163, 170, 191
getStringFromBO() メソッド 157, 163, 165, 192

I

IgnoreUndefinedAttributes データ・ハンドラー構成プロパティー 44
IgnoreUndefinedElements データ・ハンドラー構成プロパティー 44
InitialBufferSize データ・ハンドラー構成プロパティー 44

J

Java Connector Development Kit (JCDK) 154, 156

K

Key 属性プロパティー
XML データ・ハンドラー 40, 41, 50, 71

M

makeDataHandler.bat コンパイル・スクリプト 154, 173
make_datahandler コンパイル・スクリプト 173
MaxLength 属性プロパティー
Delimited データ・ハンドラー 130
FixedWidth データ・ハンドラー 119, 123, 126
MessageFile XML ODA プロパティー 204
MIME タイプ 15, 20, 26, 182
サブタイプ 16, 30, 45
命名上の制約 30, 175
text/delimited 5, 129
text/fixedwidth 5, 119
text/namevalue 5, 139
text/requestresponse 5, 101
text/xml 6, 33
MO_DataHandler_Default メタオブジェクト 20, 28
MO_DataHandler_DefaultDelimitedConfig メタオブジェクト 30, 131
MO_DataHandler_DefaultFixedWidthConfig メタオブジェクト 30, 121
MO_DataHandler_DefaultNameValueConfig メタオブジェクト 30, 140
MO_DataHandler_DefaultRequestResponseConfig メタオブジェクト 30, 112
MO_DataHandler_DefaultXMLConfig メタオブジェクト 30, 43
MO_Server_DataHandler メタオブジェクト 21, 26

N

Name 属性プロパティー
Delimited データ・ハンドラー 132
FixedWidth データ・ハンドラー 123
NameValue データ・ハンドラー 142
Request-Response データ・ハンドラーおよび 108, 109, 110
XML データ・ハンドラー 39
NameHandler クラス 171
NameHandlerClass データ・ハンドラー構成プロパティー 173, 187
Request-Response データ・ハンドラー 101, 113, 117
XML データ・ハンドラー 35, 45, 96
NameValue データ・ハンドラー 139, 148
アプリケーション固有情報 143
概要 139
既存のビジネス・オブジェクトを用いて 144
構成 140
子メタオブジェクト 140
サンプル・ファイル 146, 154
処理 140
ストリングのビジネス・オブジェクトへの変換 145
ストリングの要件 145
ビジネス・オブジェクトの構造 141
ビジネス・オブジェクトのストリングへの変換 144
ビジネス・オブジェクトの属性プロパティー 141
ビジネス・オブジェクトの要件 141
ClassName 140
CxBlank 140
CxIgnore 140
DefaultVerb 140
DummyKey 141
SkipCxIgnore 141
ValidateAttrCount 141

O

ObjectEventId 属性 89, 123, 131, 132, 141
OmitObjectEventId データ・ハンドラー構成プロパティー
Delimited データ・ハンドラー 131
FixedWidth データ・ハンドラー 122

P

PadCharacter データ・ハンドラー構成プロパティー 121, 122
Parser データ・ハンドラー構成プロパティー 35, 45, 95

R

Reader オブジェクト 184
RequestDataHandlerMimeType データ・ハンドラー構成プロパティー 113, 116
Request-Response データ・ハンドラー 99, 117
  応答ビジネス・オブジェクト 110, 111
  概要 99
  カスタマイズ 117
  構成 111
  子メタオブジェクト 112
  コンポーネント 101
  処理 106
  ストリングのビジネス・オブジェクトへの変換 116
  トップレベル・ビジネス・オブジェクト 101, 108, 111
  ネーム・ハンドラー 101, 117
  ビジネス・オブジェクト定義の作成 110
  ビジネス・オブジェクトの構造 107
  ビジネス・オブジェクトの入力フォーマットへの変換 115
  ビジネス・オブジェクトの要件 107
  要求ビジネス・オブジェクト 109, 111
  BOPrefix 112
  ClassName 113
  NameHandlerClass 113
  RequestDataHandlerMimeType 113
  ResponseDataHandlerMimeType 113
Required 属性プロパティー
  XML データ・ハンドラー 41, 49, 70
ResponseDataHandlerMimeType データ・ハンドラー構成プロパティー 113, 116
Root XML ODA プロパティー 38, 203

S

SAX パーサー 35, 45, 95
Server Access Interface
  データ・ハンドラーのインスタンス化 20
  データ・ハンドラーの使用 10
  トップレベルのメタオブジェクト 26
  getBO() 184
  IcreateBusinessObjectFrom() 13, 21
  ItoExternalForm() 12, 21
setConfigMOName() メソッド 19, 193
setEncoding() メソッド 180, 193
setLocale() メソッド 179, 194
setOption() メソッド 171, 195
setupOptions() メソッド 18, 183, 184, 189, 191, 192
SkipCxIgnore データ・ハンドラー構成プロパティー 141, 143
StubDataHandler.java ファイル 154, 156, 157

T

TopLevel XML ODA プロパティー 203, 207
TraceFileName XML ODA プロパティー 204
TraceLevel XML ODA プロパティー 204
traceWrite() メソッド 171, 195
Truncation データ・ハンドラー構成プロパティー 120, 122, 125
Type 属性プロパティー
  Delimited データ・ハンドラー 132
  FixedWidth データ・ハンドラー 123
  NameValue データ・ハンドラー 142
  XML データ・ハンドラー 39
TypeSubstitution XML ODA プロパティー 88

U

UseNewLine データ・ハンドラー構成プロパティー 45

V

ValidateAttrCount データ・ハンドラー構成プロパティー 141, 145, 146, 147
Validation データ・ハンドラー構成プロパティー 35, 45

X

XML Object Discovery Agent (ODA) 197, 211
  インストール 197
  起動 198
  構成プロパティー 202
  複数インスタンス 199
  プロパティー 202
  BOPrefix 203
  BOSelection 203
  DoctypeorSchemaLocation 86, 203
  FileName 87, 202
  MessageFile 204
  Root 203
  TopLevel 203
  TraceFileName 204
  TraceLevel 204
  TypeSubstitution 88
XML スキーマ・インスタンス・ネーム・スペース 72, 85, 203
XML スキーマ・ネーム・スペース 72
XML データ・ハンドラー 33, 98
  アプリケーション固有情報 50
    エスケープ 44, 58
    attr_fd 79, 83, 84
    attr_name 55, 56, 79
    attr_ns 79
    cw_mo_label 92, 94, 96, 163
    elem_fd 79, 80
    elem_name 55, 79, 81
    elem_ns 79
    escape=true 55, 58, 79, 94
    notag 55, 57, 81, 82, 92, 93
    type=attribute 55, 56, 57, 79, 83, 93
    type=attr_name 83
    type=cdata 55, 57, 59, 93

XML データ・ハンドラー *(続き)*
アプリケーション固有情報 *(続き)*
type=comment 55, 59, 79, 84, 93
type=defaultNS 73
type=doctype 55, 58, 93
type=MIXED 51, 66
type=pcdata 53, 55, 56, 57, 79, 81, 93
type=pi 55, 60, 79, 85
type=xmlns 73
type=xsinoNSlocation 79, 86
type=xsischemalocation 79, 86
xsinoNSlocation 93
xsinoschemalocation 93
エスケープ処理 44, 58, 94
エンティティー・リゾルバー 36, 98
概要 33
カスタマイズ 96
構成 43
子メタオブジェクト 43
コンポーネント 34
処理 33
属性レベルのアプリケーション固有情報 54, 78
動詞変換 43, 94
ネーム・ハンドラー 35, 95, 96, 187
ビジネス・オブジェクトの構造 38, 48, 61
ビジネス・オブジェクトの属性プロパティー 39, 49, 70
ビジネス・オブジェクトの変換 92
ビジネス・オブジェクト・レベルのアプリケーション固有情報 50, 71
ロケーション 6
BOPrefix 43
ClassName 44
DefaultEscapeBehavior 44
DTDPath 44
DummyKey 44
EntityResolver 44
IgnoreUndefinedAttributes 44
IgnoreUndefinedElements 44
InitialBufferSize 44
NameHandlerClass 45
Parser 45
Reader オブジェクトの制約事項 184
SAX パーサー 35
UseNewLine 45
Validation 45
XML 文書の変換 94
XML 文書 33
エスケープ処理 58, 84
外部参照 36, 95
構文解析 95
コメント 59, 84, 93
処理命令 59
属性 38, 56, 83
ビジネス・オブジェクトへの変換 94
要件 94
CDATA セクション 57, 59, 93
XML 文書 *(続き)*
noNamespaceSchemaLocation 85, 93
prolog 58, 60, 61
root エレメント 203
schemaLocation 63, 85, 93

Printed in Japan

日本アイ・ビー・エム株式会社
〒106-8711　東京都港区六本木3-2-12