

**IBM WebSphere Business Integration  
Adapters**



## **Adapter for COM ユーザーズ・ガイド**

*バージョン 1.1.x*



**IBM WebSphere Business Integration  
Adapters**



## **Adapter for COM ユーザーズ・ガイド**

*バージョン 1.1.x*

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、95 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 Adapter for COM バージョン 1.1.x、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：	IBM WebSphere Business Integration Adapters Adapter for COM User Guide Version 1.1.x
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 2004.1

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

© Copyright International Business Machines Corporation 2003. All rights reserved.

© Copyright IBM Japan 2004

---

# 目次

本書について	v
対象読者	v
本書の前提条件	v
関連文書	v
表記上の規則	vi
<b>本リリースでの新機能</b>	<b>vii</b>
リリース 1.1.x の新機能	vii
<b>第 1 章 概要</b>	<b>1</b>
アダプター環境	1
用語	3
COM コネクターのアーキテクチャー	5
ビジネス・オブジェクト要求	9
動詞の処理	9
カスタム・ビジネス・オブジェクト・ハンドラー	10
DCOM サポート	13
ロケール依存データの処理	14
<b>第 2 章 アダプターのインストール</b>	<b>15</b>
インストール作業の概要	15
コネクター・ファイルの構造	15
インストール後の作業	16
<b>第 3 章 アダプターの構成</b>	<b>17</b>
構成タスクの概要	17
コネクターの構成	17
複数のコネクター・インスタンスの作成	22
始動ファイルの構成	23
コネクターの始動	23
コネクターの停止	25
ログ・ファイルとトレース・ファイルの使用	25
<b>第 4 章 ビジネス・オブジェクトについて</b>	<b>27</b>
メタデータの定義	27
コネクター・ビジネス・オブジェクトの構造	28
属性のマッピング: COM、Java、およびビジネス・オブジェクト	34
ビジネス・オブジェクト・プロパティ例	36
ビジネス・オブジェクトの生成	40
<b>第 5 章 ビジネス・オブジェクトの作成および変更</b>	<b>41</b>
ODA for COM の概要	41
ビジネス・オブジェクト定義の生成	41
ビジネス・オブジェクト情報の指定	48
ビジネス・オブジェクト・ファイルのアップロード	54
<b>第 6 章 トラブルシューティングとエラー処理</b>	<b>55</b>
エラー処理	55
ロギング	57
トレース	57

<b>付録 A. コネクターの標準構成プロパティ</b> . . . . .	<b>59</b>
新規プロパティと削除されたプロパティ . . . . .	59
標準コネクタ・プロパティの構成 . . . . .	59
標準プロパティの要約 . . . . .	61
標準構成プロパティ . . . . .	65
<b>付録 B. Connector Configurator</b> . . . . .	<b>77</b>
Connector Configurator の概要 . . . . .	77
Connector Configurator の始動 . . . . .	78
System Manager からの Connector Configurator の実行 . . . . .	79
コネクタ固有のプロパティ・テンプレートの作成 . . . . .	79
新規構成ファイルの作成 . . . . .	82
既存ファイルの使用 . . . . .	83
構成ファイルの完成 . . . . .	84
構成ファイル・プロパティの設定 . . . . .	85
構成ファイルの保管 . . . . .	92
構成ファイルの変更 . . . . .	93
構成の完了 . . . . .	93
国際化環境における Connector Configurator の使用 . . . . .	94
<b>特記事項</b> . . . . .	<b>95</b>
プログラミング・インターフェース情報 . . . . .	96
商標 . . . . .	97

---

## 本書について

IBM<sup>(R)</sup> WebSphere<sup>(R)</sup> Business Integration Adapter ポートフォリオは、主要な e-business テクノロジー、エンタープライズ・アプリケーション、レガシー、およびメインフレーム・システムに統合コネクティビティを提供します。製品セットには、ビジネス・プロセスの統合に向けてコンポーネントをカスタマイズ、作成、および管理するためのツールとテンプレートが含まれています。

本書では、IBM WebSphere Business Integration adapter for COM のインストール、構成、ビジネス・オブジェクトの開発、およびトラブルシューティングについて説明します。

---

## 対象読者

本書は、WebSphere Business Integration システムをお客様のサイトでサポートおよび管理する、コンサルタント、開発者、およびシステム管理者を対象としています。

---

## 本書の前提条件

本書のユーザーは、WebSphere Business Integration システム、ビジネス・オブジェクトおよびコラボレーション開発、および COM テクノロジーに精通している必要があります。

---

## 関連文書

この製品に付属する資料の完全セットで、すべての WebSphere Business Integration Adapters のインストールに共通な機能とコンポーネントについて説明します。また、特定のコンポーネントに関する参考資料も含まれています。

以下のサイトから、関連資料をインストールすることができます。

- 一般的なアダプター情報が必要な場合、アダプターを WebSphere Message Broker (WebSphere MQ Integrator、WebSphere MQ Integrator Broker、WebSphere Business Integration Message Broker) とともに使用する場合、およびアダプターを WebSphere Application Server とともに使用する場合は、以下のサイトを参照してください。
  - <http://www.ibm.com/websphere/integration/wbiadapters/infocenter>
- アダプターを InterChange Server とともに使用する場合は、以下のサイトを参照してください。
  - <http://www.ibm.com/websphere/integration/wicsserver/infocenter>
  - <http://www.ibm.com/websphere/integration/wbicollaborations/infocenter>
- Message Broker (WebSphere MQ Integrator Broker、WebSphere MQ Integrator、および WebSphere Business Integration Message Broker) の詳細については、以下のサイトを参照してください。

- <http://www.ibm.com/software/integration/mqfamily/library/manualsa/>
- WebSphere Application Server の詳細については、以下を参照してください。
  - <http://www.ibm.com/software/webservers/appserv/library.html>

これらのサイトには、資料をダウンロード、インストールして参照するための簡単な説明が提供されています。

---

## 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、記述されたとおりの値を示します。
太字	初出語を示します。
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。
青いアウトライン	オンラインで表示したときのみに見られる青いアウトラインは、相互参照用のハイパーリンクです。アウトライン内をクリックすることにより、参照先オブジェクトに飛ぶことができます。
{ }	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指定できることを意味します。
< >	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・パスの規則として円記号 (¥) を使用します。すべての製品のパス名は、使用システムで製品がインストールされたディレクトリーを基準とした相対パス名です。
%テキスト%	% 記号で囲まれたテキストは、Windows™ text システム変数またはユーザー変数の値を示します。
ProductDir	製品がインストールされているディレクトリーを示します。



---

## 本リリースでの新機能

---

### リリース 1.1.x の新機能

2003 年 12 月更新。Adapter for COM のバージョン 1.1.x では、今回のリリースで追加された新機能は以下のとおりです。

- アダプターのインストール情報は、本書から移動しました。この情報の新たな入手先については、15 ページの『Adapter for COM と関連ファイルのインストール』を参照してください。
- 1.1.x バージョンから、Adapter for COM は Microsoft Windows NT 上ではサポートされなくなりました。



---

## 第 1 章 概要

- 『アダプター環境』
- 3 ページの『用語』
- 5 ページの『COM コネクターのアーキテクチャー』
- 9 ページの『ビジネス・オブジェクト要求』
- 9 ページの『動詞の処理』
- 10 ページの『カスタム・ビジネス・オブジェクト・ハンドラー』
- 13 ページの『DCOM サポート』
- 14 ページの『ロケール依存データの処理』

Connector for COM は、WebSphere Business Integration Adapter for COM のランタイム・コンポーネントです。COM Adapter には、コネクター、メッセージ・ファイナル、構成ツール、および Object Discovery Agent (ODA) が含まれます。コネクターを使用することにより、WebSphere 統合ブローカーは、COM サーバーで実行しているアプリケーション、つまり Common Object Model (COM) コンポーネントとビジネス・オブジェクトの交換が可能になります。

コネクターは、コネクター・フレームワークとアプリケーション固有のコンポーネントという 2 コンポーネントで構成されています。コネクター・フレームワークのコードはすべてのコネクターに共通なので、コネクター・フレームワークは、統合ブローカーとアプリケーション固有のコンポーネントとの仲介役の機能を果たします。アプリケーション固有のコンポーネントには、特定のテクノロジー (この場合は COM) またはアプリケーション用に調整されたコードが含まれています。コネクター・フレームワークは、統合ブローカーとアプリケーション固有のコンポーネントとの間で以下のようなサービスを提供します。

- ビジネス・オブジェクトの受信と送信
- 始動メッセージや管理メッセージの交換の管理

本書では、コネクター・フレームワークおよびアプリケーション固有のコンポーネントの両方について解説しています。ここでは、これらの両方のコンポーネントを「コネクター」と呼んでいます。

**注:** コネクター・コンポーネント は、コネクターの一部です。COM ソフトウェア・アーキテクチャーで使用されるバイナリー・ソフトウェア・コンポーネントの COM コンポーネント と混同しないでください。

---

## アダプター環境

アダプターをインストール、構成、および使用する前に、環境の要件を把握しておく必要があります。

- 2 ページの『ブローカーの互換性』
- 2 ページの『アダプターの規格』
- 2 ページの『アダプターのプラットフォーム』

- 3 ページの『アダプターの依存関係』

## ブローカーの互換性

アダプターが使用するアダプター・フレームワークは、アダプターと通信する統合ブローカーのバージョンとの互換性を備えている必要があります。Adapter for COM のバージョン 1.1.x は、以下のバージョンのアダプター・フレームワークおよび以下の統合ブローカーでサポートされています。

- アダプター・フレームワーク:
  - WebSphere Business Integration Adapter Framework、バージョン 2.10、2.2.0、2.3.0、2.3.1、2.4.0
- 統合ブローカー:
  - WebSphere InterChange Server、バージョン 4.1.1、4.2.0、4.2.1、4.2.2
  - WebSphere MQ Integrator、バージョン 2.1.0
  - WebSphere MQ Integrator Broker、バージョン 2.1.0
  - WebSphere Business Integration Message Broker 5.0
  - WebSphere Application Server Enterprise、バージョン 5.0.2 (WebSphere Studio Application Developer Integration Edition バージョン 5.0.1 と併用)

例外については、『リリース情報』を参照してください。

**注:** 統合ブローカーのインストールおよびその前提条件に関する説明については、以下の資料を参照してください。

- WebSphere InterChange Server (ICS) については、「システム・インストール・ガイド (UNIX 版)」または「システム・インストール・ガイド (Windows 版)」を参照してください。
- メッセージ・ブローカー (WebSphere MQ Integrator Broker、WebSphere MQ Integrator、および WebSphere Business Integration Message Broker) については、「WebSphere Message Brokers 使用アダプター・インプリメンテーション・ガイド」およびメッセージ・ブローカーのインストール資料を参照してください。) 一部の資料は次の Web サイトにあります。  
<http://www.ibm.com/software/integration/mqfamily/library/manualsa/>
- WebSphere Application Server については、「アダプター実装ガイド (WebSphere Application Server)」および次の資料を参照してください。  
<http://www.ibm.com/software/webservers/appserv/library.html>

## アダプターの規格

このアダプターは COM 2.0 仕様に従って記述されているため、この規格に従って設計されている COM アプリケーションと互換性があります。

COM について詳しくは、<http://www.microsoft.com/com/tech/com.asp> を参照してください。

## アダプターのプラットフォーム

このアダプターは Windows 2000 プラットフォームで稼働します。

## アダプターの依存関係

このコネクターには以下の依存関係があります。

### JDK ソフトウェア

Java Development Kit (JDK) バージョン 1.3.x は Adapter for COM をインストールする前提条件となります。このバージョンの JDK がインストール済みでない場合、WebSphere Business Integration Adapter Framework バージョン 2.4 ソフトウェア・パッケージ (Windows バージョンのみ) には、IBM JDK バージョン 1.3.1 の個別のインストールが用意されています。IBM JDK バージョン 1.3.1 は WebSphere Business Integration Adapter Framework のインストールの一部としてインストールされるのではないことに注意してください。JDK をインストールするためには、別にインストール作業が必要です。JDK を WebSphere Business Integration Adapter Framework からインストールする方法の詳細は、該当のソフトウェア・パッケージを参照してください。

### COMProxy

コネクターでは COMProxy を使用します。COMProxy は、Java プログラムを ActiveX オブジェクトと通信できるようにするインターフェース・ツールです。このツールを使用して、コネクターが COM コンポーネントを起動する際に必要とする Java プロキシ・オブジェクトを生成します。COM コンポーネントのプロパティ、構造、およびメソッドは、通常、タイプ・ライブラリー・ファイル (.tlb、.dll、.ole、.olb、または .exe) で定義します。Java Native Interface および COM テクノロジーを使用することにより、COMProxy が COM コンポーネントを Java オブジェクトのように扱うことができるようになります。

- コネクターの始動スクリプトを使用して、以下に示すパッケージ `com.ibm.adapters.utils.comproxy` 内の COMProxy クラスが、実行時にクラスパスに含まれることを確認します。
  - `ActiveXCanvas.class`
  - `COMconstants.class`
  - `ComException.class`
  - `Dispatch.class`
  - `JVariant.class`
  - `OleEnvironment.class`
- COM サーバーとインターフェースをとるため、COMProxy C++ ランタイム・ライブラリー (`BIA_COMProxy.dll`) がアダプターに組み込まれています。コネクターの始動スクリプトを使用して、この DLL がコネクターの `java.library.path` に組み込まれていることを確認します。
- コネクターは、Dispatch (OLE 自動) インターフェース・タイプをインプリメントする COM オブジェクトをサポートします。

---

## 用語

本書で使用する用語は、以下のとおりです。

- **ASI (アプリケーション固有情報)**。特定のアプリケーションまたはテクノロジー用に調整されたメタデータ。ASI は、ビジネス・オブジェクトの属性レベルおよびビジネス・オブジェクト・レベルの両方にあります。『動詞 ASI』も参照してください。
- **BO (ビジネス・オブジェクト)**。ビジネス・エンティティ (従業員など) およびデータ上のアクション (作成操作や更新操作) を表す一連の属性。WebSphere Business Integration システムのコンポーネントは、ビジネス・オブジェクトを使用して、情報を交換したり、アクションを起動したりします。
- **BO (ビジネス・オブジェクト) ハンドラー**。アプリケーションと対話するメソッドを含み、要求ビジネス・オブジェクトをアプリケーション操作に変換するコネクタ・コンポーネント。
- **COM コンポーネント**。コネクタは、ビジネス・オブジェクトと COM コンポーネント・オブジェクトの間で処理を実行して、COM サーバーと対話します。コネクタ処理の際は、プロキシー・オブジェクトが、コネクタの COM コンポーネント (COM アプリケーションの一部) を代理します。プロキシー は、COM コンポーネントを表す Java クラスです。
- **COMProxy**。Java プログラムを ActiveX オブジェクトと通信できるようにするインターフェース・ツールです。このツールを使用して、コネクタが COM コンポーネントを起動する際に必要とする Java プロキシー・オブジェクトを生成します。COM コンポーネントのプロパティ、構造、およびメソッドは、通常、タイプ・ライブラリー・ファイル (.tlb、.dll、.ole、.olb、または .exe) で定義します。Java Native Interface および COM テクノロジーを使用することにより、COMProxy が COM コンポーネントを Java オブジェクトのように扱うことができるようになります。
- **接続ファクトリー**。アプリケーションを参照する特殊なプロキシー・オブジェクト。適切なコネクタ・プロパティが設定されている場合、ファクトリー・オブジェクトは、接続プールに配置される接続を作成します。このファクトリー・オブジェクトは、コネクタの寿命に対応して持続します。作成される接続数は、PoolSize プロパティで指定される値によって異なります。
- **接続オブジェクト**。接続クラスのインスタンスである特殊なプロキシー・オブジェクト。接続とは、状態情報を含むアプリケーションを参照することです。アダプター・サイドにおける接続のインスタンスごとに、COM サイドに対応するオブジェクトがあります。接続の使用により、バッチとしてのインスタンス生成、任意の検索、接続プールへの送信、および他のスレッドによる再利用が可能です。
- **接続プール**。接続オブジェクトを保管および検索するために使用するリポジトリ。
- **外部キー**。一意的に子ビジネス・オブジェクトを識別する値を持つ単純属性。通常、この属性は、子の基本キー値を持つことで子ビジネス・オブジェクトとその親を識別します。Connector for COM は、外部キーを使用して、プール可能な接続オブジェクトを指定します。
- **ODA (Object Discovery Agent)**。アプリケーション内で指定されたエンティティを検査し、そのエンティティの中からビジネス・オブジェクト属性に対応するエレメントを「発見」することで、自動的にビジネス・オブジェクト定義を生成するツール。ODA は、アダプターをインストールすると、自動的にインスト

ールされます。Business Object Designer では、ODA にアクセスして対話式にやりとりするグラフィカル・ユーザー・インターフェースを提供しています。

- **呼び出しごとのオブジェクト・プール。**単一の doVerbFor メソッドを呼び出す際に、あるメソッドがら次のメソッドへ渡す必要のあるオブジェクトを保管するためのプログラマチック・エンティティ。保管されるオブジェクトは、プロキシ・オブジェクトの場合もあれば、単純属性の場合もあります。
- **プロキシ・クラス。**コネクタ内の COM コンポーネント・クラスを表す Java クラス。コネクタは、ビジネス・オブジェクトの ASI で指定されたプロキシ・クラス名のプロキシ・オブジェクト・インスタンスを作成します。
- **動詞 ASI (アプリケーション固有情報)。**動詞 ASI を使用し、特定の動詞について、その動詞がアクティブな場合にコネクタがビジネス・オブジェクトを処理する方法を指定します。動詞 ASI には、現在の要求ビジネス・オブジェクトを処理するために呼び出すメソッドの名前を含むことができます。

## COM コネクタのアーキテクチャー

このセクションでは、コネクタのアーキテクチャーについて説明します。図 1 に上位で行われる要求処理を、7 ページの図 2 にコネクタ動作の詳細を示します。

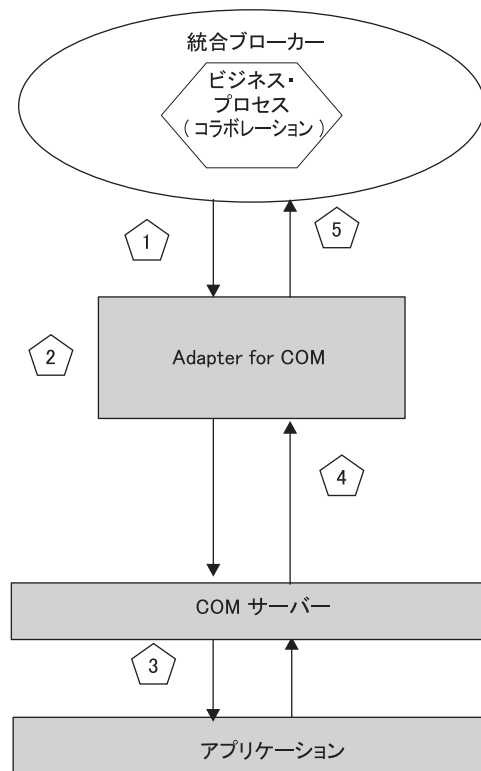


図 1. Connector for COM での要求処理

1. コネクタが、統合ブローカーからビジネス・オブジェクト要求を受け取ります。

2. コネクターが、ビジネス・オブジェクトのプロキシ・オブジェクト・インスタンスを作成します。プロキシ・オブジェクト・インスタンスは、コネクターが要求を送信する先の COM オブジェクトの代理として機能します。コネクターがプロキシ・オブジェクトを作成および処理する方法の詳細については、『コネクターの動作方法』を参照してください。
3. コネクターがプロキシ・オブジェクトを使用して、COM サーバーで実行されている対応する COM オブジェクトを呼び出し、データを COM アプリケーションに書き込むことによって、プロキシ・オブジェクトを処理します。
4. コネクターが COM サーバー・オブジェクトからデータを読み取りまたは取得して、プロキシ・オブジェクトを更新します。
5. アダプターが、元のオブジェクト要求が成功したのか、または失敗 (FAIL 状況) したのかを示すメッセージを統合ブローカーに戻します。要求が成功した場合は、コネクターも統合ブローカーに更新されたビジネス・オブジェクトを戻します。

## コネクターの動作方法

このセクションでは、7 ページの図 2 に示すように、コネクターの各部分がビジネス・オブジェクトを処理する方法について説明します。



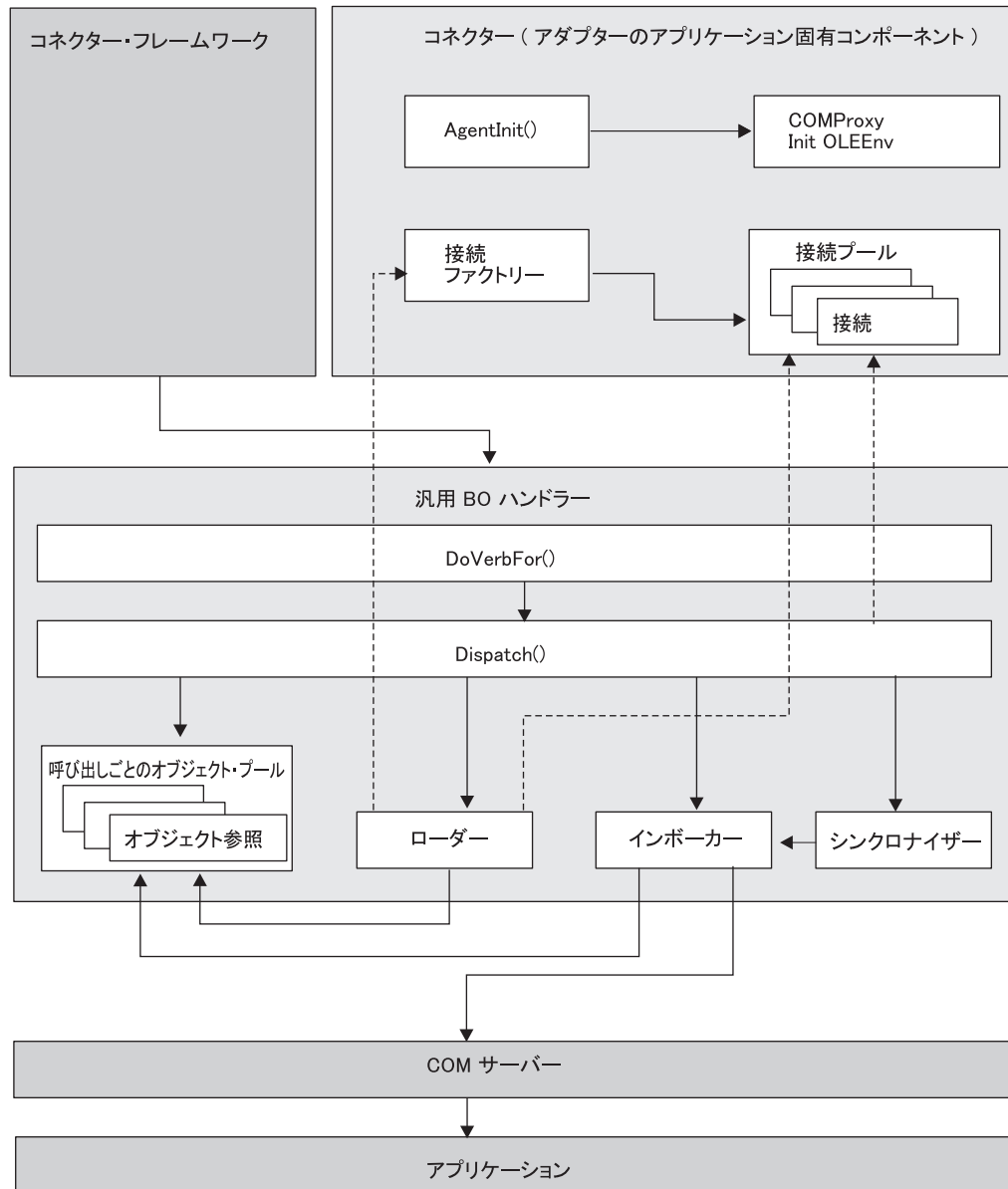


図 2. Connector for COM

1. コネクターの初回始動時に、コネクターのエージェント・クラスが、以下の初期化処理を実行します。
  - OLE 環境のインスタンスを生成します。
  - コネクター・プロパティの設定方法に応じて、以下のいずれかを実行します。コネクター・プロパティ、および以下の各シナリオでコネクター・プロパティがどのように作用するかについての詳細は、18 ページの『コネクター固有のプロパティ』を参照してください。
    - **シナリオ 1:** アプリケーションを参照する接続ファクトリー・オブジェクト・インスタンスの作成。ファクトリー・オブジェクトは、コネクターの寿命に応じて持続し、接続プールに配置される接続を作成します。作成される接続数は、コネクターの PoolSize プロパティで指定された値によって異なります。

- **シナリオ 2:** 接続プールに配置される接続オブジェクトのみの作成。接続数は、PoolSize プロパティーで指定した値によって異なります。このシナリオでは、ファクトリー・オブジェクトは作成しません。
  - **シナリオ 3:** ビジネス・オブジェクトによるメソッド呼び出しの対象となるファクトリー・プロキシー・オブジェクトの作成 (ファクトリー・クラスは BO のプロキシー・クラス ASI と一致)。このシナリオでは、接続は作成しません。
2. 統合ブローカーが、ビジネス・オブジェクト形式で、要求をコネクターに送信します。
  3. コネクターの BO ハンドラーがオブジェクトを受け取ります。
  4. BO ハンドラーの doVerbFor() メソッドが、BO の ASI を読み取ってプロキシー・クラス名を取得する Dispatch() メソッドを呼び出します。Dispatch() メソッドはプロキシー・クラス名を取得し、それをローダーに送信します。
  5. ローダーは、プロキシー・クラス名を使用してプロキシー・クラス (Mypackage.myclass などの有効な Java クラス表記が使用されているもの) をロードし、プロキシー・オブジェクト・インスタンスを作成して、そのインスタンスを呼び出しごとのオブジェクト・プールにロードします。ローダーは、オブジェクト確認のための検査を行います。検査の内容は以下のとおりです。
    - 接続であるか。接続の場合は、接続オブジェクトとして接続プールから検索します。
    - ファクトリー・オブジェクトであるか。ファクトリー・オブジェクトの場合は、静的オブジェクトとして接続ファクトリーから検索します。
  6. ディスパッチを実行して BO の動詞 ASI を読み通し、メソッドのリストを作成します。動詞 ASI は、属性名が配列されたリストです。各属性は、プロキシー・オブジェクトに対するメソッドを表します。つまり、動詞 ASI とは、メソッドのリストではなく、それぞれがプロキシー・オブジェクト・メソッドを表す値を持つ属性のリストです。
  7. 動詞 ASI リストの各メソッドについて、BO ハンドラーの InvokeMethods() メソッドは、InvokeMethod() を呼び出し、以下のいずれかを実行します。
    - メソッドが通常の方法である場合は、インボーカーを起動します。引き数が外部キーとしてマークされている場合は、呼び出しごとのオブジェクト・プールに引き数を保管します。属性が取り込まれていない場合は、use\_attribute\_value の属性 ASI を検査します。use\_attribute\_value の ASI が存在する場合は、呼び出しごとのオブジェクト・プールからオブジェクトのプルを試みます。
    - プロキシー・オブジェクトのすべての属性に対して、シンクロナイザー (BO ハンドラーのオブジェクト同期プロセス) のロード操作 (LoadFromProxy 関数) と保管操作 (WriteToProxy 関数) を呼び出します。呼び出される操作は、動詞 ASI の内容によって異なります。LoadFromProxy (ロード) および WriteToProxy (保管) は、動詞 ASI に組み込み可能な定義済み関数です。これらの関数は、ビジネス・オブジェクトの単純属性を COM コンポーネントのパブリック・プロパティーに同期させることを目的としています。
    - 特定の単純属性に対して、ロード操作または保管操作を呼び出します (LoadFromProxy を呼び出すと、プロキシー・プロパティーが取得され、BO

プロパティに取得された値が設定されます。WriteToProxy を呼び出すと、BO の値を使用してプロキシー・プロパティが設定されます。

**注:** 動詞 ASI が空の場合、BO ハンドラーは、設定済みのパラメーターを指定して、BO に対するメソッドを検索し、呼び出します。1 つのメソッドのみがパラメーターを設定できます。それ以外、つまり複数のメソッドを設定すると、動詞 ASI が空であっても、コネクターはエラーをログに記録し、FAIL コードを戻します。

8. プロキシー・オブジェクトの各メソッドについて、インボーカーは、以下を実行して、メソッドのパラメーターおよび引き数を構成します。
  - インボーカーは、属性に (String などの単一データ型ではなく) BO 型を発見すると、アクティブな BO ハンドラーに対して再帰的に Dispatch() メソッドを呼び出します。
  - Dispatch() は、親メソッドがそのメソッド呼び出しを起動するために使用するプロキシー・オブジェクトを戻します。
  - シンクロナイザーと呼ばれる、BO ハンドラーの同期化処理によって WriteToProxy が起動され、COM コンポーネント (プロキシー・オブジェクト) の各プロパティに値が保管 (設定) されます。これにより、COM サーバーのデータが更新されます。保管される値は、COM コンポーネントが対応するビジネス・オブジェクトの対応する属性に基づきます。
9. COM サーバーから値が戻されると、LoadFromProxy 関数はプロキシー・オブジェクトの「getters」を呼び出し、プロキシー・オブジェクトから戻されたデータを BO にロードします。
10. コネクターが、ビジネス・オブジェクトを統合ブローカーに戻します。

---

## ビジネス・オブジェクト要求

ビジネス・オブジェクト要求は、統合ブローカーがビジネス・オブジェクトをコネクターに送信する際に処理されます。ビジネス・オブジェクトの唯一の要件は、対応する COM コンポーネント・オブジェクト (プロキシー・オブジェクトが代行) にマップされる必要があるということです。プロキシー・クラスは、コネクター内の COM コンポーネントを表す Java クラスです。コネクターは、ビジネス・オブジェクトの ASI で指定されたプロキシー・クラス名のプロキシー・オブジェクト・インスタンスを作成します。

---

## 動詞の処理

コネクターは、各ビジネス・オブジェクトの動詞を基にブローカーによってコネクターへに渡されるビジネス・オブジェクトを処理します。

コネクター・フレームワークはブローカーから要求を受け取ると、要求ビジネス・オブジェクトのビジネス・オブジェクト定義に関連するビジネス・オブジェクト・ハンドラー・クラスの doVerbFor() メソッドを呼び出します。doVerbFor() メソッドの役割は、要求ビジネス・オブジェクトのアクティブな動詞に基づいて、実行する動詞処理を決定することです。また、要求ビジネス・オブジェクトから情報を取得し、操作要求を作成してアプリケーションに送信するという役割もあります。

コネクター・フレームワークが要求ビジネス・オブジェクトを `doVerbFor()` に渡すと、このメソッドは、ビジネス・オブジェクト ASI を検索して、BO ハンドラーを起動します。起動された BO ハンドラーは、動詞 ASI を読み取り、それを一連の呼び出し可能関数に変換します。動詞 ASI は、対象の動詞について呼び出す必要があるメソッドが配列されたリストです。呼び出しが作成される順序は、オブジェクトの処理を成功させる上で重要です。

コネクターでは、単一の `doVerbFor()` 呼び出しで、複数のコンポーネントの処理がサポートされます。

コネクターが動詞の処理時に行う呼び出しのシーケンスの例については、36 ページの『ビジネス・オブジェクト・プロパティー例』および 50 ページの図 16 を参照してください。

動詞 ASI が空の場合、BO ハンドラーは、設定済みのパラメーターを指定してメソッドを検索し、呼び出します。設定できるのは 1 つのメソッドのみです。それ以外、つまり、複数のメソッドを設定すると、動詞 ASI が空であっても、コネクターはエラーをログに記録し、FAIL コードを戻します。エラー処理の詳細については、55 ページの『エラー処理』を参照してください。

コネクターは特定の動詞をサポートするものではありませんが、ユーザーは、ODA を使用して、カスタム動詞を構成することができます。既存の標準の動詞は、Create、Retrieve、Update、および Delete です。カスタム動詞には、Business Object Designer 内で実行する Object Discovery Agent (ODA) を使用して、どのようなセマンティクスでも与えることができます。ODA を使用して動詞にメソッド呼び出しシーケンスを割り当てる方法については、41 ページの『第 5 章 ビジネス・オブジェクトの作成および変更』を参照してください。

**注:** ユーザーは、動詞 ASI に 2 つの定義済み関数 (`LoadFromProxy` および `WriteToProxy`) を指定することができます。これらの関数は、ビジネス・オブジェクトの単純属性を COM コンポーネントのパブリック・プロパティーに同期させることを目的としています。

---

## カスタム・ビジネス・オブジェクト・ハンドラー

ビジネス・オブジェクトを作成する場合、BO の動詞 ASI に CBOH キーワードを指定すると、デフォルトの BO ハンドラーをオーバーライドできます。コネクターの実行時に、`doVerbFor()` メソッドを使用して、ビジネス・オブジェクト ASI を検索します。CBOH キーワードが検出されると、`doVerbFor()` によってカスタム BO ハンドラーが起動されます。

カスタム BO ハンドラーを使用すると、プロキシー・クラスに直接アクセスできます。そのため、ローダー、インボーカー、および接続プールからの接続の検索処理をバイパスできます。これらの処理については、5 (8 ページ) および 7 (8 ページ) で説明します。

コネクターでは、親レベルのビジネス・オブジェクトのみでカスタム BO ハンドラーをサポートします。カスタム BO ハンドラー作成の詳細については、「コネクター開発ガイド」を参照してください。

## カスタム BO ハンドラーの例

以下に示すカスタム BO ハンドラーの例では、ビジネス・オブジェクトが CBOH=comadaptertest.Lotus123BOHandler という動詞 ASI を使用して定義されているものと想定します。ビジネス・オブジェクトは、CellAddress=A1 という属性 ASI を持ちます。この場合の A1 は、ワークシートに表示される属性のアドレスです。カスタム BO ハンドラーは、属性 ASI によって指定されたセル・アドレスを使用して、ワークシートに属性を書き込みます。

```
package comadaptertest;

import com.crossworlds.cwconnectorapi.*;
import com.crossworlds.cwconnectorapi.exceptions.*;
import java.util.*;
import com.ibm.adapters.utils.comproxy.*;
import lotus123.*;

public class Lotus123BOHandler implements
CWCUSTOMBOHANDLERINTERFACE {

    public int doVerbForCustom(CWConnectorBusObj bo) throws
VerbProcessingFailedException {

        Application currentApplication;
        Document currentDocument;

        CWConnectorUtil.traceWrite(CWConnectorUtil.LEVEL4,
"Entering AdapterBOHandler.doVerbFor()");

        try {
            currentDocument = new Document();

            // Get the application object.
            currentApplication = new Application(currentDocument.
get_Parent());

            // Make the application visible.
            currentApplication.set_Visible(new Boolean("true"));
            currentDocument = new Document(currentApplication.
NewDocument());

        } catch (ComException e) {
            CWConnectorUtil.generateAndLogMsg(91000,
CWConnectorUtil.XRD_ERROR, CWConnectorUtil.
CONNECTOR_MESSAGE_FILE,
e.getMessage());

            CWConnectorExceptionObject vSub = new
CWConnectorExceptionObject();
            vSub.setMsg(e.getMessage());
            vSub.setStatus(CWConnectorConstant.
APPRESPONSETIMEOUT);

            throw new VerbProcessingFailedException(vSub);

        }

        //do verb processing on this business object
dispatch(bo, currentDocument);

        CWConnectorUtil.traceWrite(CWConnectorUtil.
LEVEL4, "Leaving AdapterBOHandler.doVerbFor()");

        return CWConnectorConstant.SUCCEED;
    } //doVerbFor

    private void dispatch(CWConnectorBusObj bo,
```

```

Document currentDocument) throws VerbProcessingFailedException {
    CWConnectorUtil.traceWrite(CWConnectorUtil.
LEVEL4, "Entering dispatch");

    CWConnectorUtil.traceWrite(CWConnectorUtil.
LEVEL3, "Processing business object" + bo.getName());

    try {
        //put this object out onto the spreadsheet.
        //Follow ASI for Cell addresses
        businessObjectToWorksheet(bo, currentDocument);
    } catch (ComException e) {
        CWConnectorUtil.generateAndLogMsg(90001,
CWConnectorUtil.XRD_ERROR, CWConnectorUtil.
CONNECTOR_MESSAGE_FILE, e.getMessage());
        CWConnectorExceptionObject vSub = new
CWConnectorExceptionObject();
        vSub.setMsg(e.getMessage());
        vSub.setStatus(CWConnectorConstant.
APPRESPONSETIMEOUT);
        throw new VerbProcessingFailedException(vSub);
    } catch (CWException e) {
        CWConnectorExceptionObject vSub = new
CWConnectorExceptionObject();
        vSub.setMsg(e.getMessage());
        vSub.setStatus(CWConnectorConstant.FAIL);
        throw new VerbProcessingFailedException(vSub);
    }
}

public static void businessObjectToWorksheet(CWConnectorBusObj
bo, Document currentDocument) throws CWException {
    String incomingAttribute = "";
    int attrCount = bo.getAttrCount() - 1; //ignore objeventID
    Ranges ranges;
    Range currentRange;

    ranges = new Ranges(currentDocument.get_Ranges());

    for (int i = 0; i < attrCount; i++) {
        try {
            if ((!bo.isObjectType(i)) && (!bo.isIgnore(i))) {
                if (bo.isBlank(i))
                    incomingAttribute = "";
                else
                    incomingAttribute =
bo.getStringValue(i);

                String CellAddress = getCellAddress(bo, i);
                currentRange = new Range(ranges.Item(new
String(CellAddress)));
                currentRange.set_Contents(incomingAttribute);

                if (CWConnectorUtil.isTraceEnabled
(CWConnectorUtil.LEVEL5)) {
                    CWConnectorUtil.traceWrite
(CWConnectorUtil.LEVEL5,
"Application datum from BO to application " + CellAddress + "=" +
incomingAttribute);
                }
            }
        } catch (AttributeNotFoundException e) {
            CWConnectorUtil.generateAndLogMsg(91012,

```

```

CWConnectorUtil.XRD_ERROR,
CWConnectorUtil.CONNECTOR_MESSAGE_FILE,
bo.getAttrName(i), bo.getName());
        throw e;
    } catch (WrongAttributeException e) {
        CWConnectorUtil.generateAndLogMsg(91013,
CWConnectorUtil.XRD_ERROR,
CWConnectorUtil.CONNECTOR_MESSAGE_FILE,
bo.getAttrName(i), bo.getName());
        throw e;
    }
    }
}
public static String getCellAddress(CWConnectorBusObj bo, int i)
throws CWException {
    String columnName = null;
    try {
        columnName = getNameFromASI
        (bo.getAttrASISet(i, ":"), "CellAddress");
    } catch (WrongASIFormatException e) {
        CWConnectorUtil.generateAndLogMsg(91014,
CWConnectorUtil.XRD_ERROR,
CWConnectorUtil.CONNECTOR_MESSAGE_FILE, bo.getAttrName(i),
"ColumnName");
        throw e;
    }
    return columnName;
}
private static String getNameFromASI(Hashtable asi, String fieldName)
throws CWException,
WrongASIFormatException {
    String resultName = (String) asi.get(fieldName);

    if (resultName == null || resultName.equals(""))
        throw new WrongASIFormatException();

    resultName = resultName.toUpperCase();

    CWConnectorUtil.traceWrite(CWConnectorUtil.LEVEL4, "Found " +
fieldName + " = " + resultName);
    return resultName;
}
}
}

```

---

## DCOM サポート

Distributed Component Object Model (DCOM) は、ソフトウェア・コンポーネントをネットワーク上で直接対話できるようにするプロトコルです。DCOM は COM を拡張して、異なるコンピューター (LAN、WAN、またはインターネット) 上のオブジェクト間で行われる通信をサポートします。DCOM を使用することにより、アプリケーションをさまざまなロケーションに分散できます。

別のロケーションでオブジェクトを作成する場合にリモート・サーバー名を指定できるようにすると、コネクタで DCOM がサポートされます。リモート・サーバー名は、コネクタが稼働しているマシンのシステム・レジストリー内、または DCOM クラス・ストア内の固定構成として指定できます。リモート・サーバーを登録することにより、コネクタは、DCOM コンポーネントへ透過的にアクセスできるようになります。

---

## ロケール依存データの処理

コネクタは国際化され、2 バイト文字セットをサポートする COM インターフェースを記述する 2 バイト文字セットの配信をサポートし、特定の言語でメッセージ・テキストを配信できるようになっています。ある文字コード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へ、コネクタがデータを転送するとき、コネクタはデータの意味を保存するため、文字変換を実行します。

Java 仮想マシン (JVM) 内部の Java ランタイム環境では、Unicode 文字コード・セットでデータを表現します。Unicode は、既知の文字コード・セットのほとんど (単一バイトおよびマルチバイトの両方) に対応するエンコード方式を含んでいます。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。そのため、統合コンポーネント間でデータを転送するときは、ほとんどの場合文字変換は必要ありません。



---

## 第 2 章 アダプターのインストール

- 『インストール作業の概要』
- 『Adapter for COM と関連ファイルのインストール』
- 『コネクタ・ファイルの構造』

本章では、コネクタのインストール方法について説明します。

---

### インストール作業の概要

Connector for COM をインストールするには、以下の作業を実行する必要があります。

#### アダプターの前提条件の確認

アダプターをインストールする前に、アダプターのインストールおよび実行に必要な環境の前提条件がすべてご使用のシステム上にあることを確認してください。詳細については、1 ページの『アダプター環境』を参照してください。

#### 統合ブローカーのインストール

統合ブローカーのインストール、つまり、WebSphere Business Integration システムをインストールし、ブローカーを始動するタスクについては、ご使用のブローカーに関する資料に説明されています。COM 用のコネクタがサポートするブローカーの詳細については、2 ページの『ブローカーの互換性』を参照してください。

ブローカーのインストールについての詳細は、ご使用のブローカーのインプリメンテーション・ガイドを参照してください。

#### Adapter for COM と関連ファイルのインストール

WebSphere Business Integration Adapter 製品のインストールについては、「*WebSphere Business Integration Adapters インストール・ガイド*」を参照してください。この資料は、次の Web サイトの WebSphere Business Integration Adapters Infocenter にあります。

<http://www.ibm.com/websphere/integration/wbiadapters/infocenter>

---

### コネクタ・ファイルの構造

インストーラーは、コネクタに関連付けられた標準ファイルをご使用のシステムにコピーします。

このユーティリティーは、コネクタを `ProductDir¥connectors¥COM` ディレクトリーにインストールして、「スタート」メニューにコネクタへのショートカットを追加します。`ProductDir` は、コネクタのインストール先ディレクトリーを指します。

表 1 に、コネクターが使用するファイル構造を示します。また、インストーラーでコネクターのインストールを選択すると自動的にインストールされるファイルも示します。

表 1. コネクターのファイル構造

ProductDir のサブディレクトリー	説明
¥connectors¥COM¥BIA_COM.jar	COM コネクターのみに使用されるクラスを含みます。
¥connectors¥COM¥start_COM.bat	汎用コネクターの始動スクリプト。
¥connectors¥COM¥ext¥	ODA によって生成される .jar ファイルを保管できるディレクトリー。このディレクトリーを保管先にする場合は、始動スクリプト (start_COM.bat) 内にこのディレクトリーを指定してください。
¥connectors¥COM¥BIA_COMPProxy.dll	COM コンポーネントを起動するために使用される Java プロキシのプロパティ、構造、およびメソッドを定義する COMProxy C++ ランタイム・ライブラリー。COM サーバーとインターフェースを取るコネクターを使用可能にします。
¥connectors¥messages¥BIA_COMConnector.txt	コネクターのメッセージ・ファイル。
¥ODA¥COM¥BIA_COMODA.jar	COM ODA。
¥ODA¥COM¥start_COMODA.bat	ODA 始動ファイル。
¥ODA¥COM¥BIA_COMPProxyGen.exe	プロキシ・クラスのプロキシ・オブジェクト・インスタンスを作成するためにコネクターが使用するプロキシ・クラスを生成します。
¥ODA¥messages¥BIA_COMODAAgent_de_DE.txt	ODA のメッセージ・ファイル (ドイツ語テキスト・ストリング)。
¥ODA¥messages¥BIA_COMODAAgent_en_US.txt	ODA のメッセージ・ファイル (米国英語テキスト・ストリング)。
¥ODA¥messages¥BIA_COMODAAgent_es_ES.txt	ODA のメッセージ・ファイル (スペイン語テキスト・ストリング)。
¥ODA¥messages¥BIA_COMODAAgent_fr_FR.txt	ODA のメッセージ・ファイル (フランス語テキスト・ストリング)。
¥ODA¥messages¥BIA_COMODAAgent_it_IT.txt	ODA のメッセージ・ファイル (イタリア語テキスト・ストリング)。
¥ODA¥messages¥BIA_COMODAAgent_ja_JP.txt	ODA のメッセージ・ファイル (日本語テキスト・ストリング)。
¥ODA¥messages¥BIA_COMODAAgent_ko_KR.txt	ODA のメッセージ・ファイル (韓国語テキスト・ストリング)。
¥ODA¥messages¥BIA_COMODAAgent_pt_BR.txt	ODA のメッセージ・ファイル (ポルトガル語 (ブラジル) テキスト・ストリング)。
¥ODA¥messages¥BIA_COMODAAgent_zh_CN.txt	ODA のメッセージ・ファイル (中国語 (簡体字) テキスト・ストリング)。
¥ODA¥messages¥BIA_COMODAAgent_zh_TW.txt	ODA のメッセージ・ファイル (中国語 (繁体字) テキスト・ストリング)。
¥repository¥COM¥BIA_CN_COM.txt	コネクターのリポジトリ定義。デフォルトの名前は、BIA_CN_COM.txt です。

注: すべての製品のパス名は、使用システムで製品がインストールされたディレクトリーを基準とした相対パス名です。

## インストール後の作業

アダプターをインストールした後、実行する前に構成を行う必要があります。詳細については、17 ページの『第 3 章 アダプターの構成』を参照してください。

---

## 第 3 章 アダプターの構成

- 『構成タスクの概要』
- 『コネクターの構成』
- 22 ページの『複数のコネクター・インスタンスの作成』
- 23 ページの『始動ファイルの構成』
- 23 ページの『コネクターの始動』
- 25 ページの『コネクターの停止』
- 25 ページの『ログ・ファイルとトレース・ファイルの使用』

---

### 構成タスクの概要

インストールが完了したら、始動する前に、このセクションで説明するコンポーネントを構成する必要があります。

### コネクターの構成

コネクターの構成とは、コネクターをセットアップして構成することです。詳細については、『コネクターの構成』を参照してください。

### ビジネス・オブジェクトの構成

ビジネス・オブジェクトは、ODA (Object Discovery Agent) を使用して構成します。ODA を使用すると、ビジネス・オブジェクト定義を生成できます。ビジネス・オブジェクト定義とは、ビジネス・オブジェクトのテンプレートです。ODA は、指定したアプリケーション・オブジェクトの検証、ビジネス・オブジェクト属性に対応するビジネス・オブジェクトの要素の「発見」、および情報を示すビジネス・オブジェクト定義の生成を実行します。Business Object Designer では、Object Discovery Agent にアクセスし、ODA と対話的に連動するグラフィカル・インターフェースを提供しています。

ODA の使用に関する詳細については、41 ページの『第 5 章 ビジネス・オブジェクトの作成および変更』を参照してください。

---

### コネクターの構成

コネクターの構成プロパティには、標準構成プロパティとアダプター固有の構成プロパティという 2 つのタイプがあります。アダプターを実行する前に、Connector Configurator を使用して、これらのプロパティの値を設定する必要があります。詳細については、77 ページの『付録 B. Connector Configurator』を参照してください。

コネクターは、始動時に構成値を取得します。実行時セッション中に、1 つ以上のコネクター・プロパティの値の変更が必要になることがあります。

AgentTraceLevel など一部のコネクター構成プロパティへの変更は、即時に有効になります。他のコネクター・プロパティを変更する場合は、変更後に、コネクター・コンポーネントの再始動またはシステムの再始動が必要です。プロパティが

動的 (変更が即時に有効化) であるか、または静的 (コネクタ・コンポーネントの再始動またはシステムの再始動が必要) であるかを判別するには、System Manager の「コネクタのプロパティ」ウィンドウにある「更新メソッド」列を参照します。

## 標準コネクタ・プロパティ

標準コネクタ構成プロパティでは、すべてのアダプターが使用する情報を提供します。これらのプロパティの詳細については、59 ページの『付録 A. コネクタの標準構成プロパティ』を参照してください。

付録にリストされている標準構成プロパティについて、このコネクタ固有の情報を以下の表に示します。

プロパティ	説明
DuplicateEvent Elimination	コネクタはこのプロパティを使用しません。
Locale	このコネクタは国際化対応されているため、このプロパティの値は変更することができます。
PollEndTime	コネクタはこのプロパティを使用しません。
PollFrequency	コネクタはこのプロパティを使用しません。
PollStartTime	コネクタはこのプロパティを使用しません。

コネクタを稼働させる前に、ApplicationName 構成プロパティの値を設定する必要があります。

## コネクタ固有のプロパティ

コネクタ固有の構成プロパティは、コネクタが実行時に必要とする情報を提供します。これらのプロパティを使用すれば、コネクタ内の静的情報やロジックを、再コーディングや再ビルドせずに変更できるようになります。

コネクタ固有のプロパティを構成するには、Connector Configurator を使用します。「アプリケーション構成プロパティ」タブをクリックして、構成プロパティを追加または変更します。詳細については、77 ページの『付録 B. Connector Configurator』を参照してください。

コネクタ固有のプロパティはすべてオプションであり、ユーザー固有のコネクタ構成要件を基にした設定が可能です。コネクタは、必要に応じて、ファクトリー・オブジェクトと接続の両方を作成することも、ファクトリー・オブジェクトのみ、または接続のみを作成することもできます。

表 2 に、コネクタのコネクタ固有構成プロパティと、その説明および指定可能な値を示します。+ 文字は、プロパティ階層内でのその項目の位置を示しています。19 ページの図 3 に示すような、プロパティの階層関係など、プロパティに関する詳細については、それに続くセクションを参照してください。

表 2. コネクタ固有の構成プロパティ

名前	指定可能な値	デフォルト値
+ Factory	なし	なし

表 2. コネクタ固有の構成プロパティ (続き)

名前	指定可能な値	デフォルト値
+ + FactoryClass	クラス名	なし
+ + FactoryInitializer	初期化指定子のメソッド名	なし
+ + + Arguments	暗号化ストリングまたは非暗号化ストリング	なし
+ + FactoryMethod	メソッド名	なし
+ + + Arguments	暗号化ストリングまたは非暗号化ストリング	なし
+ ConnectionPool	なし	なし
+ + ConnectionClass	クラス名	なし
+ + ConnectionInitializer	初期化指定子のメソッド名	なし
+ + + Arguments	暗号化ストリングまたは非暗号化ストリング	なし
+ + PoolSize	整数	0
+ ThreadingModel	Apartment、Free	Free

図 3 に、コネクタ固有プロパティの階層関係を示します。

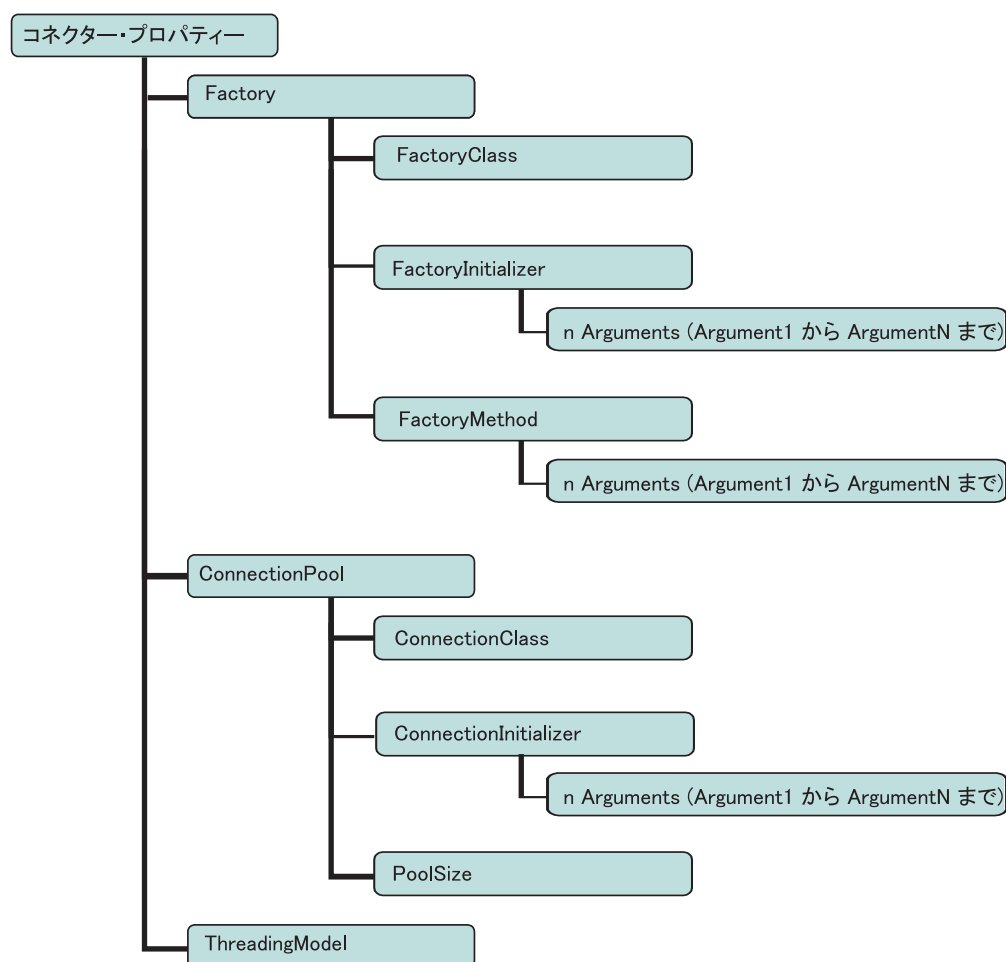


図 3. コネクタ固有プロパティの階層

## Factory

ファクトリー・クラスの情報を表す階層プロパティ。

### FactoryClass

ファクトリー・クラスの名前。

- FactoryClass、ConnectionClass、および FactoryMethod を指定すると、コネクタはファクトリー・プロキシ・オブジェクトと接続のインスタンスを生成します (コネクタの動作方法、ステップ 1 (7 ページ) のシナリオ 1 を参照してください)。
- FactoryClass を指定せずに、ConnectionClass を指定すると、特定の接続クラスおよびサイズを持つ接続プールが、コネクタの初期化時に作成されます (ステップ 1 (7 ページ) のシナリオ 2 を参照してください)。
- FactoryClass のみを指定すると、コネクタは、ファクトリー・プロキシ・オブジェクトのインスタンスを生成しますが (ステップ 1 (7 ページ) のシナリオ 3 を参照してください)、接続は使用しません。

### FactoryInitializer

FactoryClass に対する初期化指定子のメソッド名。このメソッドがコンストラクターとして機能することはありません。

### Arguments

FactoryInitializer メソッドのパラメーターを示す文字列値。この値は、暗号化文字列の場合もあれば、非暗号化文字列の場合もあります。

### FactoryMethod

FactoryClass に対する FactoryMethod のメソッド名。FactoryMethod を指定すると、接続プールは FactoryMethod を使用して接続を取得します。接続オブジェクトが作成されると、ConnectionInitializer が呼び出されます。接続の取得は、常に FactoryMethod を通じて行われるとは限りません。

### Arguments

FactoryMethod のパラメーターは、Factory に対する引き数 (Argument1、Argument2 など) である必要があります。また、適切な順序でリストされなければなりません。プロパティ名には、メソッドが実行するパラメーターの数に応じて、Argument1、Argument2 などの名前を指定します。各引き数の値は、暗号化文字列または非暗号化文字列になります。

### ConnectionPool

接続クラスの情報を表すためのプロパティ。

### ConnectionClass

プール可能な接続クラスの名前。

- ConnectionClass および FactoryClass を指定すると、コネクタはファクトリー・プロキシ・オブジェクトのインスタンスを生成します。また、接続プール・インスタンスが作成され、接続が保管されます。接続を作成するのは、ファクトリーです。(ステップ 1 (7 ページ) のシナリオ 1 を参照してください。)

- `FactoryClass` を指定せずに `ConnectionClass` を指定すると、コネクターの初期化時に、接続プール・インスタンスが作成され、接続が保管されます (ステップ 1 (7 ページ) のシナリオ 2 を参照してください)。このシナリオでは、接続を作成するのはファクトリーではありません。

プール・サイズ (接続数) は、`PoolSize` プロパティで指定した値に基づきます。

接続は、数種類の状態情報を使用してアプリケーションを参照するため、単独使用のサーバーに接続をプールする場合は、接続が参照するアプリケーションについて複数インスタンスが作成されるということに注意してください。各インスタンスは、単一 BO ハンドラー・スレッドに呼び出されます。

同様に、マルチ使用サーバーで接続をプールする場合は (サーバー・オブジェクトのインスタンスの 1 つが接続を確立するために再利用される場合あり)、ファクトリーおよびファクトリー・メソッド呼び出しをセットアップして、接続プールを作成する必要があります。この場合、各 BO ハンドラー・スレッドは、処理時に必要となる離散的接続を接続プールからプルします。

## ConnectionInitializer

プール可能な `ConnectionClass` 初期化指定子メソッドの名前。このメソッドがコンストラクターとして機能することはありません。

`ConnectionInitializer` は、接続を作成したのがファクトリーであるかどうかに関係なく、接続オブジェクトの作成後に呼び出されます。

## Arguments

初期化指定子のパラメーターを示す文字列値。この値は、暗号化文字列の場合もあれば、非暗号化文字列の場合もあります。

## PoolSize

接続プールのサイズを決定します。このプロパティは、`ConnectionClass` を指定する場合に必要です。

## ThreadingModel

コネクタがスレッド化モデルに `Multi-Threaded Apartment (MTA)` を使用するのか、または `Single Threaded Apartment (STA)` を使用するのかを指示します。このプロパティを `Apartment` に設定すると、コネクタは `STA` モード (非スレッド・セーフ) で稼働します。`STA` モードにすると、コネクタは単一スレッドになります。

このプロパティを `Free` に設定すると、コネクタは `MTA` モードで稼働します。`MTA` モード (スレッド・セーフ) の場合、マルチスレッド・クライアントは、オブジェクトに対して直接呼び出しを実行できます。デフォルト値は `Free` です (スレッド・セーフの `Multi-Threaded Apartment` モード)。

**注:** コネクタを `Single-Threaded Apartment` モード (プロパティ値を `Apartment` に設定) で実行する場合、接続プールおよびファクトリーは使用できません。

## 複数のコネクタ・インスタンスの作成

コネクタの複数のインスタンスを作成する作業は、いろいろな意味で、カスタム・コネクタの作成と同じです。以下に示すステップを実行することによって、コネクタの複数のインスタンスを作成して実行するように、ご使用のシステムを設定することができます。次のようにする必要があります。

- コネクタ・インスタンス用に新規ディレクトリを作成します。
- 必要なビジネス・オブジェクト定義が設定されていることを確認します。
- 新規コネクタ定義ファイルを作成します。
- 新規始動スクリプトを作成します。

## 新規ディレクトリの作成

それぞれのコネクタ・インスタンスごとにコネクタ・ディレクトリを作成する必要があります。このコネクタ・ディレクトリには、次の名前を付けなければなりません。

```
ProductDir¥connectors¥connectorInstance
```

ここで `connectorInstance` は、コネクタ・インスタンスを一意的に示します。

コネクタに、コネクタ固有のメタオブジェクトがある場合、コネクタ・インスタンス用のメタオブジェクトを作成する必要があります。メタオブジェクトをファイルとして保管する場合は、次のディレクトリを作成して、ファイルをそこに格納します。

```
ProductDir¥repository¥connectorInstance
```

## ビジネス・オブジェクト定義の作成

各コネクタ・インスタンスのビジネス・オブジェクト定義がプロジェクト内にまだ存在しない場合は、それらを作成する必要があります。

1. 初期コネクタに関連付けられているビジネス・オブジェクト定義を変更する必要がある場合は、適切なファイルをコピーし、**Business Object Designer** を使用してそれらのファイルをインポートします。初期コネクタの任意のファイルをコピーできます。変更を加えた場合は、名前を変更してください。
2. 初期コネクタのファイルは、次のディレクトリに入っていないければなりません。

```
ProductDir¥repository¥initialConnectorInstance
```

作成した追加ファイルは、`ProductDir¥repository` の適切な `connectorInstance` サブディレクトリ内に存在している必要があります。

## コネクタ定義の作成

**Connector Configurator** 内で、コネクタ・インスタンスの構成ファイル (コネクタ定義) を作成します。これを行うには、以下のステップを実行します。

1. 初期コネクタの構成ファイル (コネクタ定義) をコピーし、名前変更します。
2. 各コネクタ・インスタンスが、サポートされるビジネス・オブジェクト (および関連メタオブジェクト) を正しくリストしていることを確認します。
3. 必要に応じて、コネクタ・プロパティをカスタマイズします。



## 始動スクリプトの作成

始動スクリプトは以下のように作成します。

1. 初期コネクターの始動スクリプトをコピーし、コネクター・ディレクトリーの名前を含む名前を付けます。  
`dirname`
2. この始動スクリプトを、22 ページの『新規ディレクトリーの作成』で作成したコネクター・ディレクトリーに格納します。
3. 始動スクリプトのショートカットを作成します (Windows のみ)。
4. 初期コネクターのショートカット・テキストをコピーし、新規コネクター・インスタンスの名前に一致するように (コマンド行で) 初期コネクターの名前を変更します。

これで、ご使用の統合サーバー上でコネクターの両方のインスタンスを同時に実行することができます。

カスタム・コネクター作成の詳細については、「コネクター開発ガイド (C++ 用)」または「コネクター開発ガイド (Java 用)」を参照してください。

---

## 始動ファイルの構成

Connector for COM を始動する前に、始動ファイルを構成する必要があります。

Windows プラットフォーム用のコネクターを構成するには、以下の手順に従って `start_COM.bat` ファイルを変更する必要があります。

1. `start_COM.bat` ファイルを開きます。
2. 「SET JCLASSES...」で始まるセクションまでスクロールします。
3. JCLASSES 変数を、ODA を使用して作成した `.jar` ファイルを指すように編集します。例えば、ODA を使用して作成した `.jar` ファイルが `c:¥WebSphereAdapters¥connectors¥COM¥SampleLotus123.jar` である場合には、JCLASSES 変数を `JCLASSES=c:¥WebSphereAdapters¥connectors¥COM¥SampleLotus123.jar;%JCLASSES%` に設定します。

---

## コネクターの始動

コネクターは、**コネクター始動スクリプト**を使用して明示的に始動する必要があります。始動スクリプトは、次に示すようなコネクターのランタイム・ディレクトリーに存在していなければなりません。

`ProductDir¥connectors¥connName`

ここで、`connName` はコネクターを示します。始動スクリプトの名前は、表 3 に示すように、オペレーティング・システム・プラットフォームによって異なります。

表 3. コネクターの始動スクリプト

オペレーティング・システム	始動スクリプト
UNIX ベースのシステム	<code>connector_manager_connName</code>
Windows	<code>start_connName.bat</code>

コネクタ始動スクリプトは、以下に示すいずれかの方法で起動することができます。

- Windows システムで「スタート」メニューから。  
「プログラム」>「IBM WebSphere Business Integration Adapters」>「アダプター」>「コネクタ」を選択します。デフォルトでは、プログラム名は「IBM WebSphere Business Integration Adapters」となっています。ただし、これはカスタマイズすることができます。あるいは、ご使用のコネクタへのデスクトップ・ショートカットを作成することもできます。

- コマンド行から。

- Windows システム:

```
start_connName connName brokerName [-cconfigFile ]
```

- UNIX ベースのシステム:

```
connector_manager_connName -start
```

ここで、*connName* はコネクタの名前であり、*brokerName* は以下のご使用の統合ブローカーを表します。

- WebSphere InterChange Server の場合は、*brokerName* に ICS インスタンスの名前を指定します。
- WebSphere Message Brokers (WebSphere MQ Integrator、WebSphere MQ Integrator Broker、または WebSphere Business Integration Message Broker) または WebSphere Application Server の場合は、*brokerName* にブローカーを示すストリングを指定します。

**注:** Windows システム上の WebSphere Message Broker または WebSphere Application Server の場合は、*-c* オプションに続いてコネクタ構成ファイルの名前を指定しなければなりません。ICS の場合は、*-c* はオプションです。

- Adapter Monitor から (WebSphere Business Integration Adapters 製品のみ)。  
Adapter Monitor は System Manager 始動時に起動されます。  
このツールを使用して、コネクタのロード、アクティブ化、非アクティブ化、休止、シャットダウン、または削除を行うことができます。
- System Monitor から (WebSphere InterChange Server 製品のみ)。  
このツールを使用して、コネクタのロード、アクティブ化、非アクティブ化、休止、シャットダウン、または削除を行うことができます。
- Windows システムでは、Windows サービスとして始動するようにコネクタを構成することができます。この場合、Windows システムがブートしたとき (自動サービスの場合)、または Windows サービス・ウィンドウを通じてサービスを始動したとき (手動サービスの場合) に、コネクタが始動します。

コマンド行の始動オプションなどのコネクタの始動方法の詳細については、以下の資料のいずれかを参照してください。

- WebSphere InterChange Server については、「システム管理ガイド」を参照してください。
- WebSphere Message Brokers については、「WebSphere Message Brokers 使用アダプター・インプリメンテーション・ガイド」を参照してください。

- WebSphere Application Server については、「アダプター実装ガイド (WebSphere Application Server)」を参照してください。

---

## コネクターの停止

コネクターを停止する方法は、以下に示すように、コネクターが始動された方法によって異なります。

- コマンド行からコネクターを始動した場合は、コネクター始動スクリプトを用いて、以下の操作を実行します。
  - Windows システムでは、始動スクリプトを起動すると、そのコネクター用の別個の「コンソール」ウィンドウが作成されます。このウィンドウで、「Q」と入力して Enter キーを押すと、コネクターが停止します。
  - UNIX ベースのシステムでは、コネクターはバックグラウンドで実行されるため、別ウィンドウはありません。代わりに、次のコマンドを実行してコネクターを停止します。

```
connector_manager_connName -stop
```

ここで、*connName* はコネクターの名前です。

- Adapter Monitor から (WebSphere Business Integration Adapters 製品のみ)。  
Adapter Monitor は System Manager 始動時に起動されます。  
このツールを使用して、コネクターのロード、アクティブ化、非アクティブ化、休止、シャットダウン、または削除を行うことができます。
- System Monitor から (WebSphere InterChange Server 製品のみ)  
このツールを使用して、コネクターのロード、アクティブ化、非アクティブ化、休止、シャットダウン、または削除を行うことができます。
- Windows システムでは、Windows サービスとして始動するようにコネクターを構成することができます。この場合、Windows システムのシャットダウン時に、コネクターは停止します。

---

## ログ・ファイルとトレース・ファイルの使用

アダプターのコンポーネントは、複数レベルのメッセージ・ロギングとトレースを提供します。コネクターは、アダプター・フレームワークを使用してエラー・メッセージ、通知メッセージ、およびトレース・メッセージを記録します。エラー・メッセージと通知メッセージは、ログ・ファイルに記録されます。トレース・メッセージは、対応するトレース・レベル (0 から 5) とともにトレース・ファイルに記録されます。ロギングとトレース・レベルの詳細については、55 ページの『第 6 章 トラブルシューティングとエラー処理』を参照してください。

ログ・ファイルとトレース・ファイルの名前、およびトレース・レベルを Connector Configurator で構成してください。このツールの詳細については、77 ページの『付録 B. Connector Configurator』を参照してください。

ODA にはロギング機能がないことに注意してください。エラー・メッセージは、直接ユーザー・インターフェースに送られます。トレース・ファイルとトレース・レベルは Business Object Designer 内で構成します。構成の手順は 43 ページの『エ

ージェントの構成』に記載されています。ODA のトレース・レベルは、57 ページの『トレース』に定義されているコネクターのトレース・レベルと同じです。

---

## 第 4 章 ビジネス・オブジェクトについて

本章では、ビジネス・オブジェクトの構造、アダプターによるビジネス・オブジェクトの処理方法、およびビジネス・オブジェクトに関するアダプターの想定について説明します。

本章の内容は、次のとおりです。

- 『メタデータの定義』
- 28 ページの『コネクター・ビジネス・オブジェクトの構造』
- 34 ページの『属性のマッピング: COM、Java、およびビジネス・オブジェクト』
- 36 ページの『ビジネス・オブジェクト・プロパティ一例』
- 40 ページの『ビジネス・オブジェクトの生成』

---

### メタデータの定義

Connector for COM は、メタデータ主導型です。WebSphere Business Integration システムにおいて、メタデータは、COM アプリケーション・オブジェクトのデータ構造を記述するアプリケーション固有の情報として定義されています。メタデータは、ビジネス・オブジェクト定義を構成するために使用します。コネクターは、実行時にこの定義を使用して、ビジネス・オブジェクトを作成します。

コネクターのインストール後、コネクターを実行する前に、ビジネス・オブジェクト定義を作成する必要があります。コネクタ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た任意の名前を付けることができます。命名規則については詳しくは、「コンポーネント命名ガイド」を参照してください。

メタデータ主導型コネクターは、サポートする各ビジネス・オブジェクトを、ビジネス・オブジェクト定義にエンコードされているメタデータに基づいて処理します。これにより、コードに変更を加えなくても、コネクターは新規または変更されたビジネス・オブジェクト定義を処理できるようになります。新規オブジェクトを作成するには、Business Object Designer で Object Discovery Agent (ODA) を使用します。既存のオブジェクトを変更するには、(ODA を介さずに) 直接 Business Object Designer を使用します。

アプリケーション固有のメタデータには、ビジネス・オブジェクトの構造と、属性プロパティの設定が記述されています。各ビジネス・オブジェクトの実際のデータ値は、実行時にメッセージ・オブジェクトへ送られます。

コネクターは、サポートするビジネス・オブジェクトの構造、親ビジネス・オブジェクトと子ビジネス・オブジェクト間の関係、およびデータの形式を推測します。そのため、ビジネス・オブジェクトの構造を、対応する COM オブジェクトに対して定義した構造と正確に一致させる必要があります。構造が一致していないと、アダプターはビジネス・オブジェクトを正しく処理できません。

ビジネス・オブジェクト構造を変更する必要がある場合には、COM 内の対応するオブジェクトに対してその変更を行い、その変更をタイプ・ライブラリーにエクスポートして、ODA に入力します。

ビジネス・オブジェクト定義の変更について詳しくは、「WBIA ビジネス・オブジェクト開発ガイド」を参照してください。

---

## コネクタ・ビジネス・オブジェクトの構造

コネクタは、COM コンポーネントで使用するビジネス・オブジェクトを処理します。このセクションでは、COM コネクタで処理されるビジネス・オブジェクトの構造に関連した主要概念について説明します。

### 属性

ODA は、タイプ・ライブラリー・ファイル (.tlb、.dll、.olb、.ole、または .exe) に定義されている COM コンポーネントに含まれる属性ごとに、対応するビジネス・オブジェクト属性を生成します。タイプ・ライブラリー・ファイルにはインターフェースが含まれており、各インターフェースにはメソッドとプロパティーが存在します。タイプ・ライブラリー・ファイルは、ODA でプロキシ・オブジェクト定義がコンパイルされる際に使用されます。

COM クラス内の属性が単純属性ではなくコンポーネントである場合、BO 属性は、COM オブジェクト内の該当のコンポーネントと定義が一致する子オブジェクトにマップされます。

ビジネス・オブジェクトには、フラットなものと同層構造のものがあります。フラットなビジネス・オブジェクトには、単純属性、つまり、ストリングなどの単一値を表し、子ビジネス・オブジェクトを参照しない属性のみが含まれます。階層ビジネス・オブジェクトは単純属性、および属性値を含む子ビジネス・オブジェクトまたは子ビジネス・オブジェクトの配列を含みます。

カーディナリティー 1 コンテナ・オブジェクトまたは単一カーディナリティー関係は、親ビジネス・オブジェクトの属性に単一の子ビジネス・オブジェクトが含まれる場合に発生します。この場合、子ビジネス・オブジェクトは、レコードを 1 つのみ含むコレクションになります。属性タイプは、子ビジネス・オブジェクトになります。

カーディナリティー n コンテナ・オブジェクトまたは複数カーディナリティー関係は、親ビジネス・オブジェクトの属性が子ビジネス・オブジェクトの配列を含む場合に発生します。この場合、子ビジネス・オブジェクトは、複数のレコードを含むコレクションになります。属性タイプは、子ビジネス・オブジェクトの配列の属性タイプと同じになります。

### メソッド

ビジネス・オブジェクト内には、COM タイプ・ライブラリー・ファイルに定義されているメソッドごとに属性が作成されます。属性タイプは、メソッド・パラメーターを表す属性を含む子 BO です。子 BO の属性は、COM メソッドのパラメーターと同じ順序で表示されます。また、子 BO には Return\_Value 属性もあります。この属性は、引き数のリストの末尾に現れ、COM メソッド呼び出しの結果を示しま

す。これらの子 BO の属性は、メソッドのパラメーターや戻り値のタイプに応じて、単純タイプまたはオブジェクト・タイプ (複合タイプ) になります。戻り値が現れる場所は、常に引き数のリストの末尾になります。

プロパティまたはメソッド名に特殊文字が含まれる場合は、それに対応する属性名が WebSphere Business Integration スイートの形式に変更され、属性 ASI を使用してプロパティまたはメソッドの実際の名前が設定されます。

## アプリケーション固有の情報

アプリケーション固有の情報を使用して、ビジネス・オブジェクトの処理方法に関するアプリケーション固有の指示をコネクターに提供します。ビジネス・オブジェクト定義を拡張または変更する場合には、定義内のアプリケーション固有の情報と、コネクターが予期する構文とを必ず一致させる必要があります。

アプリケーション固有の情報は、名前と値のペアとして表され、個別のビジネス・オブジェクト属性や動詞に対して指定できるだけでなく、ビジネス・オブジェクト全体に対しても指定することができます。

### ビジネス・オブジェクト・レベルの ASI

オブジェクト・レベルの ASI では、ビジネス・オブジェクトの性質およびビジネス・オブジェクトに含まれるオブジェクトについての基本情報を提供します。表 4 に、プロキシ・オブジェクトを表すビジネス・オブジェクトのビジネス・オブジェクト・レベルの ASI を示します。

**注:** ASI 名は、メソッド、メソッド・パラメーター、およびメソッド戻り値を表すビジネス・オブジェクトとして認識されません。COM オブジェクトのメソッドとして作成されるビジネス・オブジェクト属性の詳細については、28 ページの『メソッド』を参照してください。

表 4. オブジェクト・レベル ASI

オブジェクト・レベル ASI	説明
<code>proxy_class=&lt;nameOfProxyClass&gt;</code>	ビジネス・オブジェクトが表すプロキシ・クラスの名前。この ASI を使用して、プロキシ・クラスをビジネス・オブジェクトにマップします。この値は、有効な Java パッケージ表記 ( <code>java.lang.Vector</code> など) を使用して指定する必要があります。
<code>auto_load_or_write=true</code>	ビジネス・オブジェクトが、引き数と戻り値の両方に使用されるレコード構造を表すことを示しています。この ASI は、アダプターに対し、まずプロキシ・オブジェクトへの書き込み ( <code>WriteToProxy</code> ) を行ってから子オブジェクト内の引き数に対応する引き数を使用して関数呼び出しを行い、その後関数呼び出しから値が戻されたらプロキシ・オブジェクトからの読み取り ( <code>LoadFromProxy</code> ) を実行するように指示します。

### 動詞 ASI

どのビジネス・オブジェクトも、動詞を含んでいます。この動詞を使用して、受信アプリケーションがビジネス・オブジェクト内のデータを処理する方法を記述します。

動詞 ASI には、一連の属性名が含まれています。各属性名は、汎用ビジネス・オブジェクト・ハンドラーによる呼び出しの対象となるメソッドが含まれます。呼び出し対象のメソッドは、ビジネス・オブジェクトかその親のいずれかに属するものです。通常は前者であり、この場合は、呼び出し対象のメソッドを、オブジェクトの動詞 ASI 内にそのまま指定します。例えば、コンポーネントにメソッド `IncrementCounter` が含まれている場合、このメソッドを呼び出すには、対応するビジネス・オブジェクトの動詞 ASI にこのメソッドをそのまま指定します。

呼び出し対象のメソッドがビジネス・オブジェクト階層の親に属する場合は、`PARENT` タグの付いたメソッド名をプレフィックス交換することにより、親を参照することができます。

例えば、図 4 は、`ContactDetails` が `Contact` の子オブジェクトであり、`Contact` は `PSRCustomerAccount` の子であるというビジネス・オブジェクト階層を示しています。

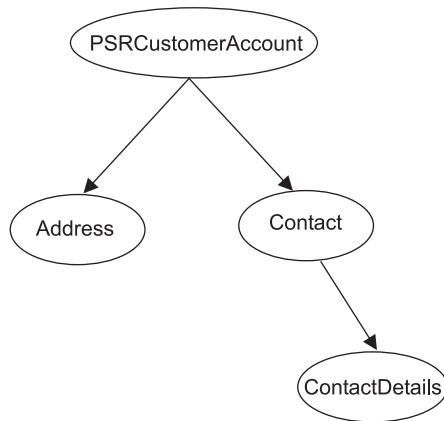


図 4. ビジネス・オブジェクト階層と動詞 ASI

`PSRCustomerAccount` に属するメソッドを `ContactDetails` ビジネス・オブジェクトで呼び出す場合、`ContactDetails` の動詞 ASI は、次のようなビジネス・オブジェクト階層を示します。

```
PARENT.PARENT.<methodName>
```

メソッドが、`Contact` ビジネス・オブジェクトに属する場合は、`ContactDetails` の動詞 ASI を次のように設定します。

```
PARENT.<methodName>
```

呼び出しが可能なのは、階層内の親オブジェクトに属するメソッドのみです。また、親ビジネス・オブジェクトから子のメソッドを呼び出すことはできません。

コネクタ開発者が、動詞に割り当てられる COM 操作を決定します。サポートされる動詞は、以下のとおりです。

- Create
- Delete
- Retrieve
- Update



以下のキーワードは、属性名の動詞 ASI シーケンスに使用します。

表 5. 動詞 ASI に使用可能なキーワード

キーワード	説明
LoadFromProxy= <attributeName>	指定されたプロキシ・オブジェクト属性の getter メソッドを呼び出して、プロキシ・オブジェクトからビジネス・オブジェクトに属性をロードします。
WriteToProxy = <attributeName>	指定されたプロキシ・オブジェクト属性の setter メソッドを呼び出して、ビジネス・オブジェクトの属性値を、そのビジネス・オブジェクトに対応するプロキシ・オブジェクトに書き込みます。
LoadFromProxy (属性名なし)	プロキシ・オブジェクトから現在の BO にロードされる非メソッド属性の getter メソッドをすべて呼び出します。
WriteToProxy (属性名なし)	現在の BO からプロキシ・オブジェクトに書き込まれる非メソッド属性の setter メソッドをすべて呼び出します。
CBOH=<custom BO handler className>	カスタム BO ハンドラーのクラス名。この場合、汎用 BO ハンドラーは使用しません。カスタム BO ハンドラーについては、10 ページの『カスタム・ビジネス・オブジェクト・ハンドラー』を参照してください。

どのオブジェクトでも、サポートされる 4 つの動詞 (Create、Retrieve、Delete、および Update) を指定することができ、各動詞のアクションとして n + 2 個のメソッドを割り当てることができます (n は、対応する COM コンポーネントに含まれるメソッドの数です)。2 個の追加のメソッドは、表 5 に定義されている、コネクタでサポートされるメソッド (LoadFromProxy および WriteToProxy) です。

## 属性レベル ASI

ビジネス・オブジェクトの属性レベルの ASI は、子ビジネス・オブジェクトを含む複合属性である場合も、単純属性である場合もあります。複合属性である場合、含まれている子がオブジェクトのプロパティとメソッドのどちらであるかによって ASI は異なります。元の COM タイプ・ライブラリーに含まれる各属性タイプが、ODA によって生成されたビジネス・オブジェクトにどのようにマップされるのかについては、34 ページの表 9 に示します。

表 6 に、単純属性の ASI を示します。単純属性は、必ず、子以外の値 (ブール値、ストリング値、整数値など) になります。

表 6. 単純属性の属性レベルの ASI

属性	説明
Name	ビジネス・オブジェクトのフィールド名です。
Type	ビジネス・オブジェクトのフィールド・タイプです。Java プロキシとビジネス・オブジェクト属性の型への COM コンポーネントの型のマッピングについては、34 ページの表 9 を参照してください。
MaxLength	使用しません。

表 6. 単純属性の属性レベルの ASI (続き)

属性	説明
IsKey	各ビジネス・オブジェクトは、少なくとも 1 つはキー属性を持つ必要があります。キー属性は、属性のキー・プロパティの設定を true にして指定します。この属性を使用するのは、コネクタではなく Business Object Designer であることに注意してください。
IsForeignKey	呼び出しごとのオブジェクト・プールにオブジェクトを保管する必要があるかどうかを、コネクタが確認しなければならないことを示します (ステップ 7 (8 ページ) を参照してください)。
IsRequired	使用しません。
AppSpecInfo	元の Java タイプを保持します。この属性は次のようにフォーマット設定します。  <code>property=&lt;propertyName&gt;, type=&lt;typeName&gt;</code>  property は、COM オブジェクトのプロパティの名前です。この名前と値のペアは、オリジナルの COM オブジェクトのプロパティ名の取得に使用されます。  type は、単純属性の Java タイプです。Java プロキシとビジネス・オブジェクト属性の型への COM コンポーネントの型のマッピングの詳細については、34 ページの表 9 を参照してください。  この属性は、ビジネス・オブジェクトである場合には proxy に設定する必要があります。単純属性の場合のように、属性がビジネス・オブジェクトにマップされておらず、属性の値を基に参照を行うことがない場合には、type=PlaceholderOnly と指定することができます。これにより、BO ハンドラーに対して、その属性を基に参照を行わないように、またその属性に値を取り込まないように、指示したことになります。このとき、その属性が外部キーに設定されている (IsForeignKey がチェックされている) 場合や、use_attribute_value が互換性のある値に設定されている場合には、その属性を引き続きマルチ呼び出しフローの一部として使用できます。
DefaultValue	使用しません。

表 7 に、メソッド以外を表す子オブジェクトが含まれる複合属性の ASI を示します。

表 7. メソッド以外を表す子オブジェクトが含まれる属性の属性レベルの ASI

属性	説明
type	含まれているオブジェクトの型です。型がビジネス・オブジェクトである場合には、proxy に設定します。
ContainedObjectVersion	使用しません。
Relationship	子がコンテナ属性であることを示します。Containment に設定します。
IsKey	使用しません。
IsForeignKey	使用しません。
Is Required	使用しません。

表7. メソッド以外を表す子オブジェクトが含まれる属性の属性レベルの ASI (続き)

属性	説明
AppSpecificInfo	<p>元の COM アプリケーション・フィールド名を保持します。この属性は次のようにフォーマット設定します。</p> <pre>property=propertyName, use_attribute_value=&lt;(optional)BOName.AttributeName&gt;, type=&lt;typeName&gt;</pre> <p>property は、COM オブジェクトのプロパティの名前です。この名前と値のペアは、元の COM オブジェクトのプロパティ名の取得に使用されます。属性に含まれているのがメソッドの引き数である場合には、property に値を設定しないでください。これは、そのような引き数は、名前を持たない、いずれかの標準的な型の引き数に過ぎないからです。</p> <p>use_attribute_value は、BOName.AttributeName のフォーマットで表したビジネス・オブジェクト名です。この ASI を設定すると、アダプターは、呼び出しごとのオブジェクト・プールから属性にアクセスします。この値は、ビジネス・オブジェクトの作成時に ODA で設定するのではなく、Business Object Designer を使用して設定します。</p> <p>type は、プロパティの Java タイプです。属性が単純属性ではない場合、つまりビジネス・オブジェクトを含んでいる場合には、proxy に設定する必要があります (対応する COM の型は IDispatch* です)。COM、Java、およびビジネス・オブジェクトの間での型のマッピングについては、34 ページの表9 を参照してください。</p>
Cardinality	1 を設定します。

表8 に、メソッドを表す子オブジェクトが含まれる複合属性の ASI を示します。

表8. メソッドを表す子オブジェクトが含まれる属性の属性レベルの ASI

属性	説明
Name	ビジネス・オブジェクトのフィールド名です。
type	ビジネス・オブジェクトです。
Relationship	子オブジェクトであることを示す Containment に設定します。
IsKey	属性名が UniqueName である場合は true に設定し、それ以外の場合は false に設定します。
IsForeignKey	false を設定します。
Is Required	false を設定します。
AppSpecificInfo	<p>外部 COM サーバーに対して行われるメソッド呼び出しの名前を表す、元の COM アプリケーションのフィールド名を指定します。この属性のフォーマットは次のとおりです。</p> <pre>method_name=&lt;nameOfMethod&gt;</pre>
Cardinality	1 を設定します。

メソッドには引き数と戻り値があることに注意してください。引き数と戻り値は、子オブジェクトを含む複合値の場合もあれば、単純値の場合もあります。

## 属性のマッピング: COM、Java、およびビジネス・オブジェクト

このセクションでは、タイプ・ライブラリーに定義されている COM の型と、それらの型に対応する Java 構造およびビジネス・オブジェクト属性を、リストにしています。子ビジネス・オブジェクト以外のビジネス・オブジェクト属性はすべて、データ型が String となります。ビジネス・オブジェクトにおいて、ASI は、属性の実際のデータ型を保持しており、Java プロキシ・オブジェクトに対してメソッドを呼び出す場合に使用されます。

ビジネス・オブジェクト ASI の詳細については、29 ページの『アプリケーション固有の情報』を参照してください。

**注:** COMProxy インターフェース・ツールがサポートしていない COM の型は、コネクタではサポートされません。

表9. オブジェクト・マッピング: COM、Java、およびビジネス・オブジェクト

COM の型	Java プリミティブ	Java ボックス化	COMProxy 内部	ビジネス・オブジェクト	属性 ASI の type の値
Float	浮動	Float	VT_R4	Float	float
Float*	float[]	Float[]	VT_R4   VT_BYREF	Float	float_reference
BSTR	プリミティブ型は存在しません。	String	VT_BSTR	String	String
BSTR*	プリミティブ型は存在しません。	String[]	VT_BSTR   VT_BYREF	String	String_reference
Int	int	Integer	VT_I4	Integer	int
int*	int[]	Int[]	VT_I4   VT_BYREF	Integer	int_reference
IDispatch*	プリミティブ型は存在しません。	Object	VT_DISPATCH	ビジネス・オブジェクト	proxy
IDispatch**	プリミティブ型は存在しません。	Object[]	VT_DISPATCH   VT_ARRAY	ビジネス・オブジェクト	ArrayOf_proxy
Short	short	Short	VT_I2	Integer	short
Short*	short[]	Short[]	VT_I2   VT_BYREF	Integer	short_reference
VARIANT	プリミティブ型は存在しません。	Object	VT_VARIANT	String	variant
VARIANT_BOOL	ブール値	Boolean	VT_BOOL	Boolean	boolean
VARIANT_BOOL*	boolean[]	Boolean[]	VT_BOOL   VT_BYREF	Boolean	boolean_reference
Long	int	Integer	VT_I4	Integer	int
Long*	int[]	Integer	VT_I4   VT_BYREF	Integer	int_reference

表9. オブジェクト・マッピング: COM、Java、およびビジネス・オブジェクト (続き)

COM の型	Java プリミティブ	Java ボックス化	COMProxy 内部	ビジネス・オブ ジェクト	属性 ASI の type の値
CURRENCY	long	Long	VT_CY	Integer	long
CURRENCY*	long[]	Long[]	VT_CY   VT_BYREF	Integer	long_reference
DATE	プリミティブ型 は存在しませ ん。	java.util.Date	VT_DATE	Date	Date
DATE*	プリミティブ型 は存在しませ ん。	Date[]	VT_DATE   VT_BYREF	Date	Date_reference
double	double	Double	VT_R8	Double	double
double*	double[]	Double[]	VT_R8   VT_BYREF	Double	double_reference
unsigned char	バイト	Byte	VT_UI1	Integer	バイト
unsigned char*	byte[]	Byte[]	VT_UI1   VT_BYREF	Integer	byte_reference
Decimal	プリミティブ型 は存在しませ ん。	サポートされ ていません。	サポートされてい ません。	サポートされて いません。	サポートされてい ません。
Decimal*	プリミティブ型 は存在しませ ん。	サポートされ ていません。	サポートされてい ません。	サポートされて いません。	サポートされてい ません。
hyper	プリミティブ型 は存在しませ ん。	サポートされ ていません。	サポートされてい ません。	サポートされて いません。	サポートされてい ません。
hyper*	プリミティブ型 は存在しませ ん。	サポートされ ていません。	サポートされてい ません。	サポートされて いません。	サポートされてい ません。
Small	プリミティブ型 は存在しませ ん。	サポートされ ていません。	サポートされてい ません。	サポートされて いません。	サポートされてい ません。
Small*	プリミティブ型 は存在しませ ん。	サポートされ ていません。	サポートされてい ません。	サポートされて いません。	サポートされてい ません。
SAFEARRAY(type)	type[]	Type[]	VT_ARRAY	カーディナリテ ィー n のビジネ ス・オブジェク トの子 (単純属性 を持つ)	ArrayOf_type
Enum	int	Integer	VT_INT	Integer	int

注: 属性の値を基に参照を行うことがない場合には、ASI type=PlaceholderOnly を使用する必要があります。これにより、アダプターに対して、その属性の値を取り込まないように指示したことになります。その属性が外部キーに設定されている場合 (IsForeignKey が true に設定されている場合) や、互換性のある

属性を参照する ASI use\_attribute\_value が指定されている場合には、その属性を引き続きマルチ呼び出しフローの一部として使用することができます。

## 配列型

array 型 (配列型) に関しては以下の点に注意してください。

- array 型を使用するには、ASI type=ArrayOf\_<value> を指定する必要があります (value は、34 ページの表 9 に示した属性 ASI の値の 1 つです)。例えば、type=ArrayOf\_int を使用すると、int 型変数の配列が指定されます。これらは、要素が含まれているカーディナリティー n のビジネス・オブジェクトにマップされます。
- Java のオブジェクト配列 (Object[]) については、ASI の type を ArrayOf\_proxy に設定します。プロキシー・オブジェクトの処理は、各配列要素に対して行われます。プロキシー配列が関数の引き数になっている場合には、メソッドが実行される前に、配列の各オブジェクトで動詞の処理が発生します。また、戻り値になっている場合には、メソッドが実行された後、配列の各オブジェクトで動詞の処理が発生します。
- サイズ指定された配列は、入力としては使用できますが、出力としては使用できません。
- SafeArray は、入力値としても、戻り値としてもサポートされます。

---

## ビジネス・オブジェクト・プロパティ例

このセクションでは、アダプターでのビジネス・オブジェクトの処理を具体的に示す例を紹介します。

- 『コネクタの呼び出し順序の例』
- 37 ページの『ビジネス・オブジェクトの例』

### コネクタの呼び出し順序の例

下記のサンプル・コードは、「Hello World」という簡単なメッセージを WebSphere MQ に書き込むコネクタ・コードです。

```
/******  
*Create the MQSession object and access the MQQueueManager and (local) MQQueue  
*****/  
MQSession MQSess = new MQSession();  
iDispatch = MQSess.AccessQueueManager("COMTest");  
MQQueueManager QMgr = new MQQueueManager(iDispatch);  
  
QMgr.Connect();  
MQQueue MyQueue=new MQQueue(QMgr.AccessQueue_4("SYSTEM.DEFAULT.LOCAL.QUEUE",17));  
  
MyQueue.Open();  
MQMessage PutMsg = new MQMessage(MQSess.AccessMessage());  
  
//write a string to the message  
PutMsg.WriteString("Hello World");  
  
//put the message on the queue  
MyQueue.Put(PutMsg);  
MyQueue.Close();  
QMgr.Disconnect();
```

## ビジネス・オブジェクトの例

以下の画面例は、36 ページの『コネクタの呼び出し順序の例』に示したコードが正常に機能するために必要なビジネス・オブジェクト構造とアプリケーション固有の情報を示しています。以下の画面例で説明するビジネス・オブジェクトは、Create 動詞を使用しています。

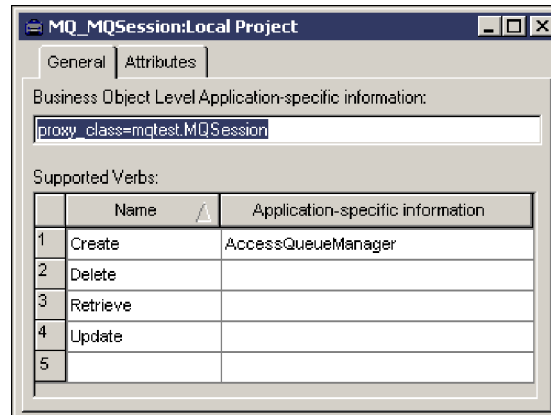


図 5. ビジネス・オブジェクト・レベルの ASI とサポートされる動詞

図 5 は、トップレベルのビジネス・オブジェクト MQ\_MQSession を示しています。このビジネス・オブジェクトは、メソッド・シーケンス内で最初に作成されるオブジェクトにあたります。Create 動詞の ASI には、AccessQueueManager と指定されています。これは、MQQueueManager オブジェクトへのアクセスを可能にする関数です。

AccessQueueManager は、Integer 型のオブジェクトを戻します。このオブジェクトが MQQueueManager のコンストラクターに渡されると、MQQueueManager プロキシのインスタンスが作成されます。ビジネス・オブジェクト・レベルの ASI にストリング proxy\_class=MQTest.MQSession が指定されていることに注意してください。これは、現在のビジネス・オブジェクトに対応する COM コンポーネントを表すプロキシ・オブジェクトを示しています。proxy\_class ASI の詳細については、29 ページの表 4 を参照してください。

	Pos	Name	Type	Key	Foreign	Required	Card	Maximum Length	Default	App Spec Info
1	1	AccessQueueManager	MQ_AccessQueue	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	1			method_name=AccessQueueManager
1.1	1.1	input	String	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>		255		type=String
1.2	1.2	Return_Value	MQ_MQQueueMan	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	1			type=proxy
1.2.1	1.2.1	AccessQueue_4	MQ_AccessQueue	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	1			method_name=AccessQueue
1.2.1	1.2.1	name	String	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>		255		type=String
1.2.1	1.2.1	OpenOptions	Integer	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>				property=OpenOptions;type=int
1.2.1	1.2.1	Return_Value	MQ_MQQueue	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	1			type=proxy
1.2.1	1.2.1	name	String	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>		255		property=name;type=String
1.2.1	1.2.1	Open	String	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>		255		method_name=Open
1.2.1	1.2.1	Get	MQ_MQQueue_Get	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	1			method_name=Get
1.2.1	1.2.1	Put	MQ_MQQueue_Put	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	1			method_name=Put
1.2.1	1.2.1	msg	MQ_MQMessage	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	1			type=proxy
1.2.1	1.2.1	WriteString	MQ_WriteString	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	1			method_name=WriteString
1.2.1	1.2.1	value	String	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>		255		type=String
1.2.1	1.2.1	ObjectEventId	String							
1.2.1	1.2.1	ObjectEventId	String							
1.2.1	1.2.1	ObjectEventId	String							
1.2.1	1.2.1	Close	String	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>		255		method_name=Close
1.2.1	1.2.1	ObjectEventId	String							
1.2.1	1.2.1	ObjectEventId	String							
1.2.1	1.2.1	ObjectEventId	String							
1.2.2	1.2.2	Commit	String	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>		255		method_name=Commit
1.2.3	1.2.3	Connect	String	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>		255		method_name=Connect
1.2.4	1.2.4	Disconnect	String	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>		255		method_name=Disconnect
1.2.5	1.2.5	ObjectEventId	String							
1.3	1.3	ObjectEventId	String							
2	2	ObjectEventId	String							
3	3			<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>		255		

図 6. MQSession オブジェクトのビジネス・オブジェクト階層

図 6 は、親オブジェクト (MQSession) から最下位の子までを含む、ビジネス・オブジェクト階層を示しています。AccessQueueManager は、iDispatch ポインタを戻すメソッドです。このポインタは、MQQueueManager (ビジネス・オブジェクトの Return\_Value 属性に指定されています) にマップされています。MQQueueManager は、図 6 に示すビジネス・オブジェクト構造に含まれる子オブジェクト MQ\_MQQueueManager によって表現される、プロキシー・オブジェクトです。

Business Object Level Application-specific information:	
proxy_class=mqtest.MQQueueManager	
Supported Verbs:	
Name	Application-specific information
1 Create	Connect,AccessQueue_4,Disconnect
2 Delete	
3 Retrieve	
4 Update	
5	

図 7. MQ\_MQQueueManager のメソッド呼び出しシーケンス



38 ページの図 7 は、MQ\_MQQueueManager のメソッド・シーケンスを示しています。このシーケンスは、次の 3 つのメソッドで構成されています。

1. Connect
2. AccessQueue\_4
3. Disconnect

38 ページの図 6 に示したように、BO 構造に含まれる AccessQueue\_4 は、MQ\_MQQueue というプロキシ・オブジェクトを戻します。このとき、コネクタは、AccessQueue\_4 を実行した後、戻されたプロキシ・オブジェクト (AccessQueue\_4 の子) を、MQ\_MQQueueManager の呼び出しシーケンスの 3 番目、つまり最後のメソッドである Disconnect を実行する前に処理します。

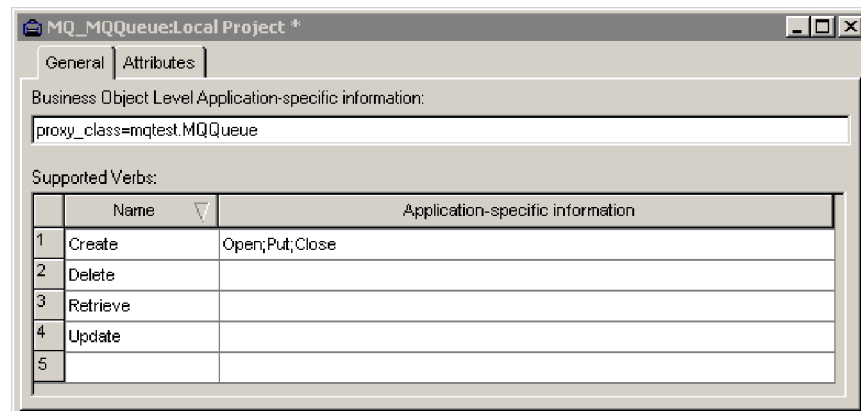


図 8. MQ\_MQQueue のメソッド呼び出しシーケンス

図 8 は、MQ\_MQQueue (AccessQueue\_4 の子) のメソッド・シーケンスを示しています。この呼び出しシーケンスは、次の 3 つのメソッドで構成されています。

1. Open
2. Put
3. Close

Put メソッドは MQMessage を引き数とするので、コネクタは、Put メソッドを実行する前に MQMessage オブジェクトを作成する (そして作成したオブジェクトに対してメソッドを実行する) 必要があります。コネクタの処理動作には再帰的な性質があります。つまり、Put は、AccessQueue\_4 の親である MQ\_MQQueueManager (38 ページの図 7) の Disconnect メソッドを実行する前に実行されます。

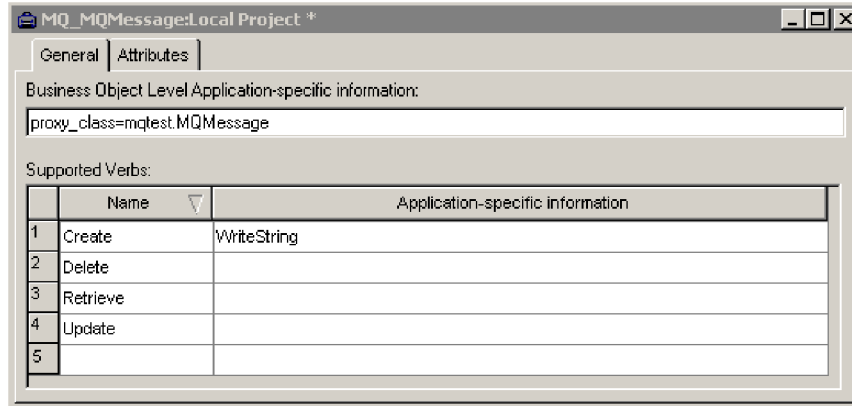


図9. MQMessage のメソッド呼び出しシーケンス

図9は、MQMessage のメソッド・シーケンスを示しています。このシーケンスは、単純ストリングを引数とする WriteString メソッドのみを呼び出します。この例では、引数は「Hello World」メッセージです。このメッセージが WebSphere MQ に書き込まれます。コネクタでは、その後、このセクションで説明した再帰的シーケンスに従ってメソッドの処理が続行されます。コネクタは、WriteString を呼び出した後、階層を「引き返し」、MQ\_MQQueue (39 ページの図8) の Close メソッドを実行し、MQ\_MQQueueManager (38 ページの図7) の Disconnect メソッドを実行します。

## ビジネス・オブジェクトの生成

実行時にイベントが発生するたびに、COM アプリケーションは、オブジェクト・レベルのデータとトランザクション・タイプに関する情報を含むメッセージ・オブジェクトを送信します。コネクタは、このデータを対応するビジネス・オブジェクト定義にマップして、アプリケーション固有のビジネス・オブジェクトを作成します。コネクタは、作成したビジネス・オブジェクトを統合ブローカーに送信して、処理します。また、統合ブローカーから戻されるビジネス・オブジェクトを受け取り、COM アプリケーションに渡します。

**注:** COM アプリケーションのオブジェクト・モデルが変更されたときは、ODA を使用して新しい定義を作成してください。統合ブローカー・リポジトリのビジネス・オブジェクト定義が COM アプリケーションによって送信されるデータと正確に一致しない場合は、コネクタでビジネス・オブジェクトを作成することはできないためトランザクションは失敗します。

Business Object Designer に用意されているグラフィカル・インターフェースを使用すると、実行時に使用するビジネス・オブジェクト定義を作成および変更できます。詳細については、41 ページの『第5章 ビジネス・オブジェクトの作成および変更』を参照してください。

---

## 第 5 章 ビジネス・オブジェクトの作成および変更

- 『ODA for COM の概要』
- 『ビジネス・オブジェクト定義の生成』
- 48 ページの『ビジネス・オブジェクト情報の指定』
- 54 ページの『ビジネス・オブジェクト・ファイルのアップロード』

---

### ODA for COM の概要

ODA (Object Discovery Agent) を使用すると、ビジネス・オブジェクト定義を生成できます。ビジネス・オブジェクト定義とは、ビジネス・オブジェクトのテンプレートです。ODA は、指定したアプリケーション・オブジェクトの検証、ビジネス・オブジェクト属性に対応するビジネス・オブジェクトの要素の「発見」、および情報を示すビジネス・オブジェクト定義の生成を実行します。Business Object Designer では、Object Discovery Agent にアクセスし、ODA と対話的に連動するグラフィカル・インターフェースを提供しています。

Object Discovery Agent (ODA) for COM は、COM タイプ・ライブラリー・ファイルに含まれているメタデータから、ビジネス・オブジェクト定義を生成します。Business Object Designer ウィザードを使用すると、ビジネス・オブジェクト定義の作成が自動的に処理されます。ODAを使用して、ビジネス・オブジェクトおよび Connector Configurator を作成し、それらをサポートするコネクタを構成します。Connector Configurator について詳しくは、77 ページの『付録 B. Connector Configurator』を参照してください。

---

### ビジネス・オブジェクト定義の生成

このセクションでは、Business Object Designer の COM ODA を使用して、ビジネス・オブジェクト定義を生成する方法について説明します。Business Object Designer の起動方法および使用方法について詳しくは、「WBIA ビジネス・オブジェクト開発ガイド」を参照してください。

#### ODA の始動

ODA は、メタデータ・リポジトリ (タイプ・ライブラリー・ファイル) が存在するリポジトリをマウントできるマシンであれば、どのマシンからでも start\_COMODA.bat 始動ファイルを使用して実行できます。この始動ファイルには、必須の COM およびコネクタ .jar ファイルをへのパスなどの始動パラメーターが含まれます。必須の .jar ファイルには、ODA を実行しているマシンからもアクセスできます。

ODA for COM のデフォルトの名前は、COMODA です。この名前は、始動スクリプトで AGENTNAME 変数の値を変更することにより、変更できます。

ODA を始動するには、以下のコマンドを実行します。

```
start_COMODA
```

この始動ファイルを使用するには、Java コンパイラ (javac.exe) のディレクトリが PATH 環境変数に含まれていなければならないことに注意してください。例えば、javac.exe がディレクトリ c:\jdk131\_02\bin にある場合には、次の行を start\_COMODA.bat に追加します。

```
set PATH=c:\jdk131_02\bin;%PATH%
```

## Business Object Designer の実行

Business Object Designer では、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ステップをガイドするウィザードを提供しています。ビジネス・オブジェクト定義を生成するステップは、以下のとおりです。

### エージェントの選択

エージェントを選択するには、以下のステップを実行します。

1. Business Object Designer を始動します。
2. 「ファイル」 > 「ODA を使用して新規作成」を選択します。「ビジネス・オブジェクト・ウィザード - ステップ 1/6 - エージェントの選択」画面が表示されます。
3. 「検索されたエージェント」リストで、(start\_COMODA スクリプトから) ODA/AGENTNAME を選択して、「次へ」をクリックします。(必要なエージェントが表示されない場合には、「エージェントの検索」をクリックしてください。)

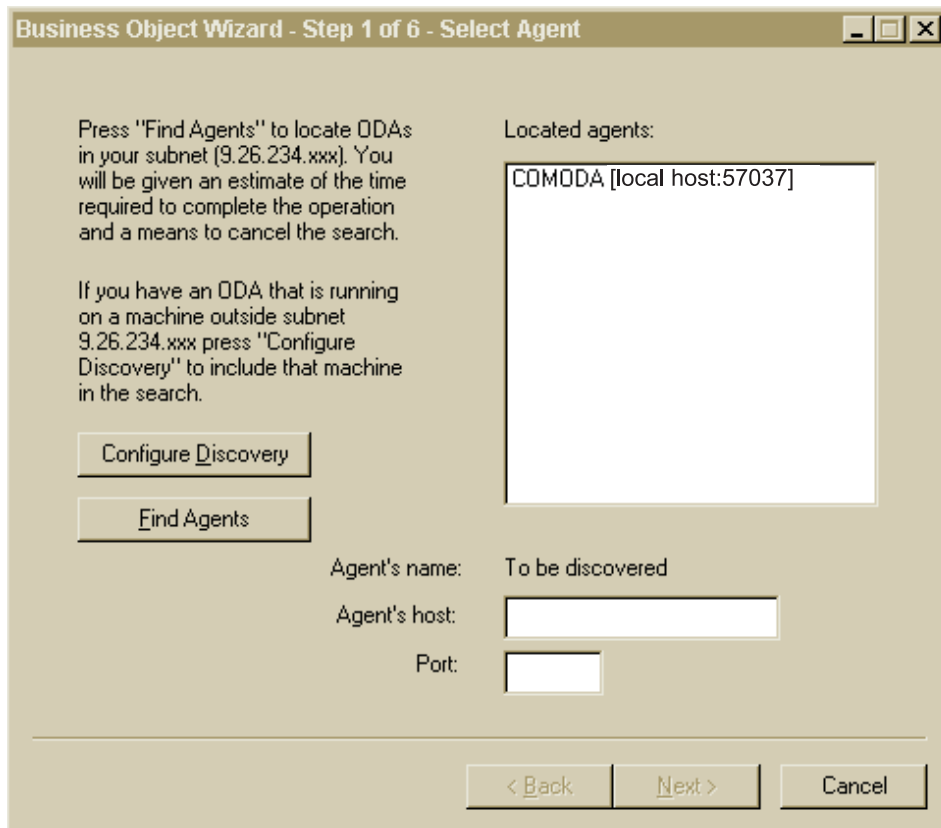


図 10. 「エージェントの選択」画面

### エージェントの構成

「エージェントの選択」画面の「次へ」をクリックすると、「ビジネス・オブジェクト・ウィザード - ステップ 2/6 - エージェントの構成」画面が表示されます。44 ページの図 11 は、サンプル値が指定されたエージェントの構成画面です。

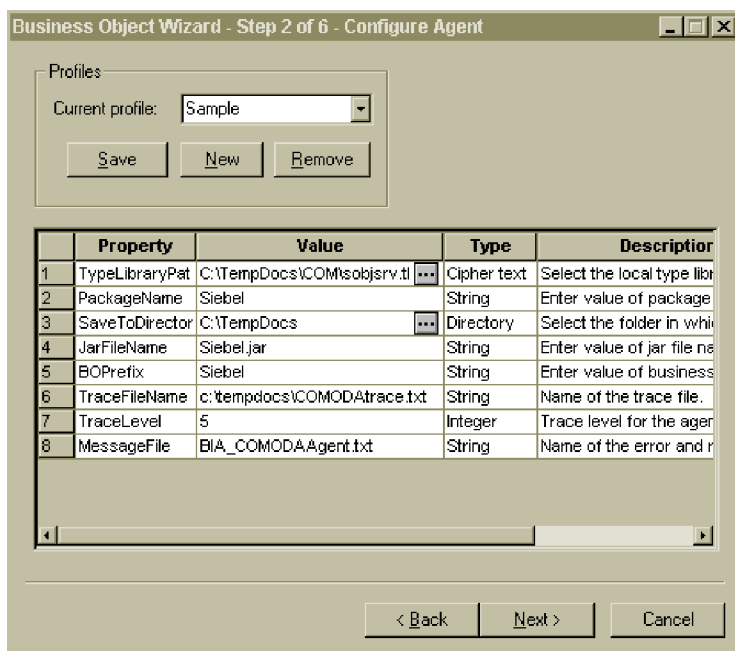


図 11. 「エージェントの構成」画面

表 10 に、この画面で設定するプロパティを示します。この画面で入力した値はすべて、プロファイルに保管できます。そのため、次回 ODA を実行する際、プロパティ・データを再入力しなくても、ドロップダウン・メニューからプロファイルを選択するだけで保管した値を再利用できます。それぞれが特定の値で構成された異なるセットを持つ複数のプロファイルを保管できます。

表 10. エージェント・プロパティの構成

プロパティ名	デフォルト値	タイプ	説明
TypeLibraryPath	なし	String	(必須) COM インターフェースが定義されているローカル・タイプ・ライブラリー・ファイル (.tlb、.dll、.ole、.olb、または .exe) へのパス。
PackageName	なし	String	(必須) COMProxy によって生成されるすべてのプロキシ・ファイルの保管先パッケージ。COMProxy は、コネクタが COM コンポーネントを起動する際に必要とする Java プロキシ・オブジェクトを生成します。
SaveToDirectory	なし	String	(必須) PackageName に指定したパッケージの保管先ディレクトリー。
JarFileName	なし	String	(必須) ODA によって生成されたプロキシ・クラスの保管先となる .jar ファイル。このファイルの格納先ディレクトリーの場合は、ODA によって自動的に判別されます。
BOPrefix	なし	String	ODA によって生成されたビジネス・オブジェクト名に追加するプレフィックス。

表 10. エージェント・プロパティの構成 (続き)

プロパティ名	デフォルト値	タイプ	説明
TraceFileName	なし	String	トレース・メッセージ・ファイルの名前。 COMODAttrace.txt など。
TraceLevel	5	Integer	(必須) エージェントのトレース・レベル (0 から 5 まで)。トレース・レベルの詳細については、57 ページの『トレース』を参照してください。
MessageFile	なし	String	(必須) ODA によって表示されるすべてのメッセージを含むメッセージ・ファイルの名前。COM の場合、このファイルの名前は、BIA_COMODAAgent.txt になります。メッセージ・ファイルの名前を適切に指定しないと、ODA はメッセージなしで実行されます。

1. ODA を実行するのは今回が初めてである場合は、「プロファイル」グループ・ボックスの「新規」ボタンと「保管」ボタンを使用して、新しいプロファイルを作成します。次回以降は、既存のプロファイルを選択できます。
2. 44 ページの表 10 の定義に従って、各プロパティの名前、値、タイプ、および説明を入力します。

**注:** プロファイルを使用すると、プロパティ値が自動的に入力されます。ただし、これらの値は必要に応じて変更できます。

### ビジネス・オブジェクトの選択

46 ページの図 12 に示す「ビジネス・オブジェクト・ウィザード - ステップ 3/6 - ソースの選択」画面が表示されます。この画面には、COM タイプ・ライブラリー・ファイルに定義されているコンポーネントのリストが表示されます。この画面を使用して、ODA でのビジネス・オブジェクト定義の生成の対象になる COM コンポーネントを、任意の数選択します。

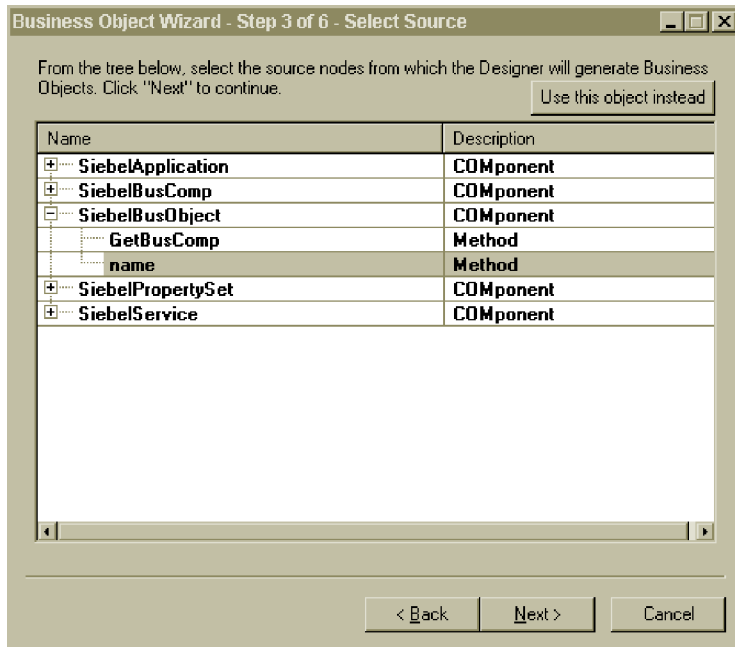


図 12. 「ソースの選択」画面

1. 必要に応じて、COM コンポーネントを展開し、そのコンポーネントのメソッドのリストを表示します。
2. 使用する COM オブジェクトを選択します。図 12 では、name メソッドが選択されています。
3. 「次へ」をクリックします。

### オブジェクト選択の確認

「ビジネス・オブジェクト・ウィザード - ステップ 4/6 - ビジネス・オブジェクトのソース・ノードの確認」画面が表示されます。この画面には、選択したオブジェクトが表示されます。



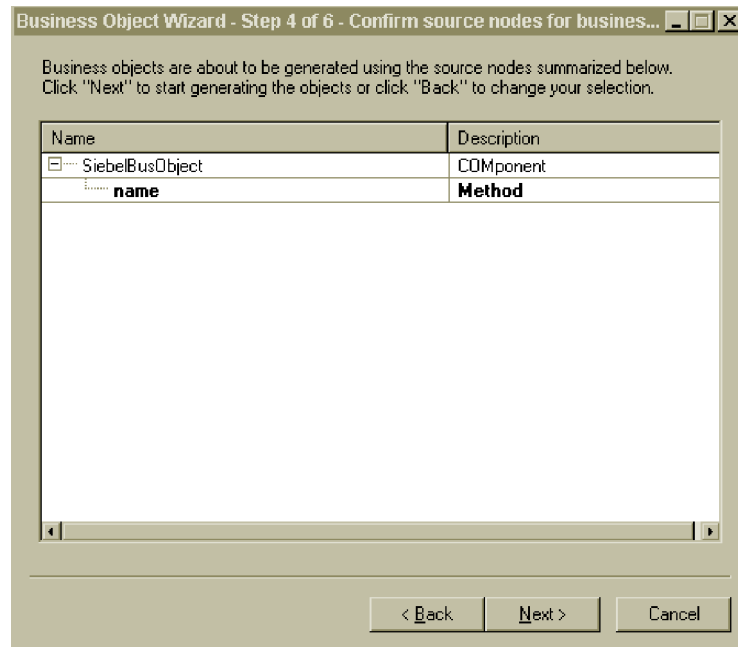


図 13. ソース・ノード画面の確認

「戻る」をクリックして変更を加えるか、「次へ」をクリックしてリストが正しいことを確認します。

「ビジネス・オブジェクト・ウィザード - ステップ 5/6 - ビジネス・オブジェクトの生成中...」画面が表示されます。画面には、ビジネス・オブジェクトが生成中であることを示すメッセージが表示されます。

パラメーターまたは戻り値が Java タイプ Object または Object[] であるメソッド (45 ページの『ビジネス・オブジェクトの選択』を参照してください) を選択されている場合、ODA は、48 ページの図 14 に示す「BO プロパティ」画面を表示します。この画面を使用して、該当のタイプのオブジェクトを COM コンポーネントまたは「String」にマップしてください。「値」列のドロップダウン・メニューには、現在のタイプ・ライブラリーに含まれるコンポーネントのみが表示されます。ビジネス・オブジェクト ASI への COM と Java の型のマッピングを示す詳細なリストについては、34 ページの表 9 を参照してください。

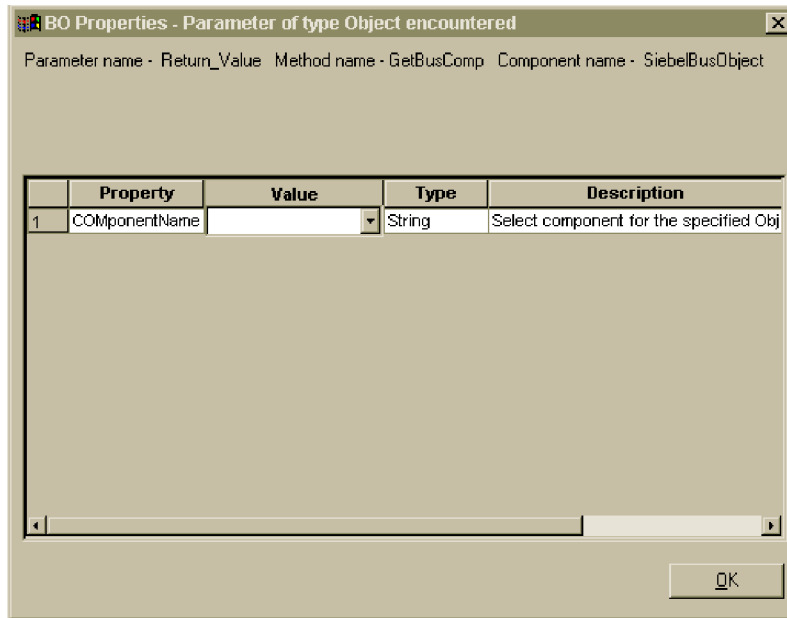


図 14. 「BO プロパティ」画面

ODA は、コンポーネント名、ODA が現在処理しているコンポーネントのメソッドの名前、およびデータ型が Object または Object[] のメソッド・パラメーターの名前を示す行を、この画面に示します。

## ビジネス・オブジェクト情報の指定

ビジネス・オブジェクトの作成が完了したら、そのオブジェクトにおいて有効な動詞を指定できます。また、そのオブジェクトの任意の動詞のメソッド・シーケンスを指定できます。このセクションでは、これらの情報を Business Object Designer と ODA を使用して指定する方法について説明します。情報のカテゴリーおよび COM コネクタのビジネス・オブジェクト構造について詳しくは、27 ページの『第 4 章 ビジネス・オブジェクトについて』を参照してください。

### 動詞の選択

ビジネス・オブジェクトの作成後、作成したオブジェクトを個別のウィンドウで開いたときに表示される Business Object Designer の最初の画面は、「BO プロパティ - コンポーネントの動詞の選択 (BO Properties - Select Verbs for component)」画面です。49 ページの図 15 は、46 ページの図 12 および 47 ページの図 13 のとおりに作成した name ビジネス・オブジェクトの場合にこの画面がどのように表示されるかを示しています。

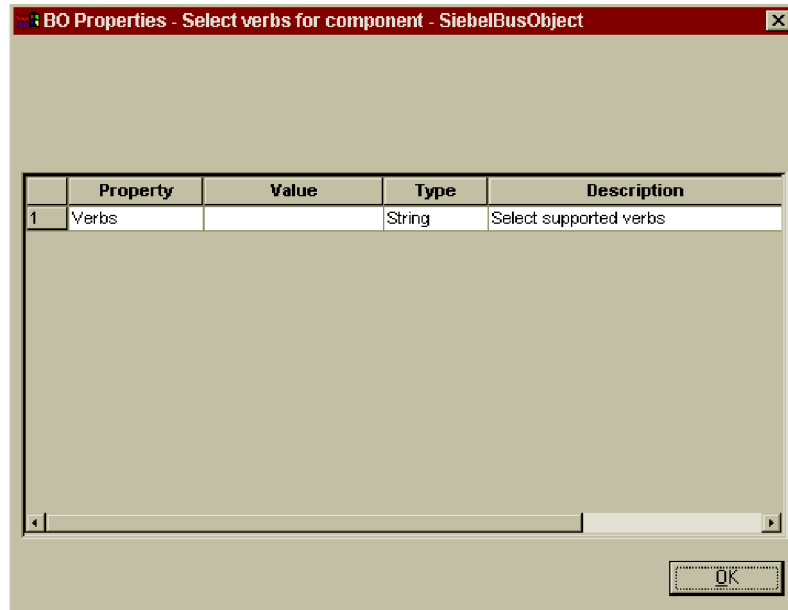
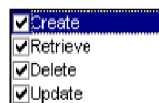


図 15. コンポーネントの動詞選択画面

この画面では、ビジネス・オブジェクトがサポートする動詞を指定します。ODA では、サポートされる 4 つの動詞 (Create、Retrieve、Delete、および Update) を指定することができ、各動詞のアクションとして  $n + 2$  個のメソッドを割り当てることができます ( $n$  は、対応する COM コンポーネントに含まれるメソッドの数です)。2 個の追加のメソッドは、コネクタでサポートされるメソッド (LoadFromProxy および WriteToProxy) です。サポートされる 4 つの動詞以外の追加の動詞を指定する場合や、ビジネス・オブジェクトを作成した後で動詞情報を編集する場合には、Business Object Designer を使用します。

COM コネクタのビジネス・オブジェクト動詞の詳細については、29 ページの『動詞 ASI』を参照してください。

1. Verbs プロパティの「値」リストから、ビジネス・オブジェクトでサポートする動詞を選択します。1 つ以上の動詞を選択してください。選択した動詞はいつでも選択解除できます。



2. 「OK」をクリックします。

## 動詞 ASI の指定

48 ページの『動詞の選択』のステップ 1 で選択した動詞ごとに、個別ウィンドウが表示されます。このウィンドウを使用して、動詞に対して実行するメソッド・シーケンス指定します。

図 16 は、46 ページの図 12 および 47 ページの図 13 のとおりに作成した name ビジネス・オブジェクトの Create 動詞の場合に、この画面がどのように表示されるかを示しています。

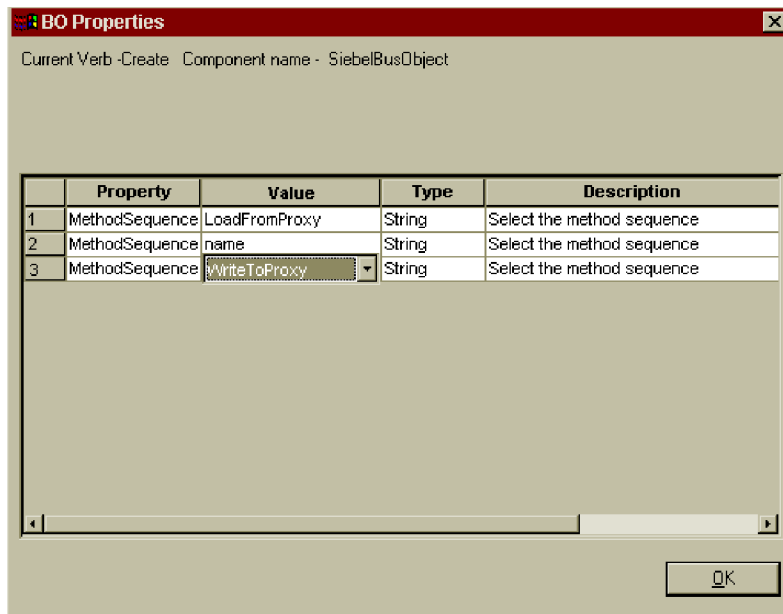


図 16. 動詞メソッド・シーケンスの設定

1. MethodSequence プロパティの「値」リストから、ビジネス・オブジェクトが最初に動詞に対して実行するメソッドを選択します。図 16 でのメソッド・シーケンスは、以下のとおりです。

- Create 動詞に対するメソッド・シーケンスで最初に実行されるメソッドは、LoadFromProxy です。
- シーケンスの 2 番目メソッドは、name です。
- シーケンスの 3 番目メソッドは、WriteToProxy です。

name メソッドは、Siebel ビジネス・オブジェクト・コンポーネントに用意されています (タイプ・ライブラリー・ファイルに定義されています)。

LoadFromProxy メソッドと WriteToProxy メソッドは、ODA が提供します。

動詞に対してメソッド・シーケンスを指定することにより、その動詞に関連した動詞 ASI を作成します。この動詞 ASI は、必要に応じて、後から Business Object Designer を使用して変更することができます。

2. 「OK」をクリックします。

COM 動詞 ASI がサポートするキーワードのリストについては、31 ページの表 5 を参照してください。

## ビジネス・オブジェクトを別のウィンドウで開く

「ビジネス・オブジェクト・ウィザード - ステップ 6/6 - ビジネス・オブジェクトの保管」画面が表示されます。

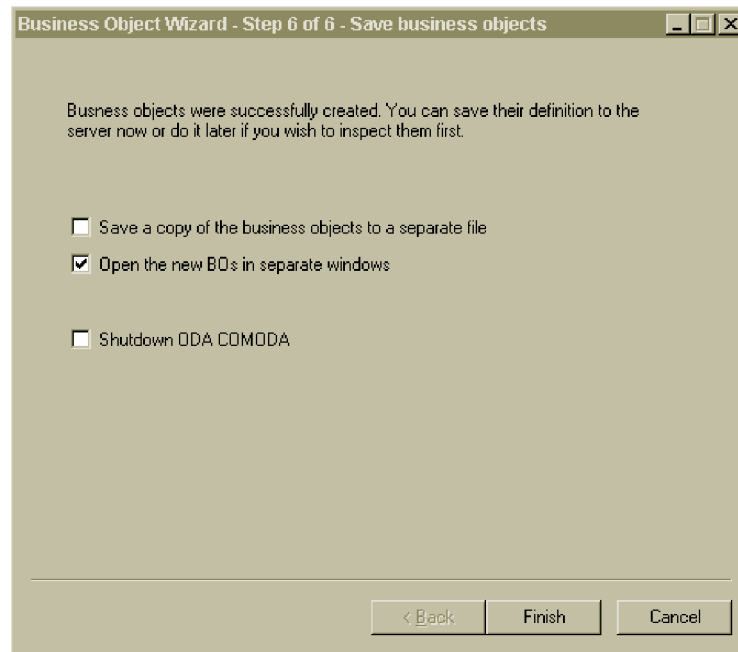


図 17. 「ビジネス・オブジェクトの保管」画面

新規作成したビジネス・オブジェクトは、必要に応じて Business Object Designer 内の別のウィンドウで開くことができます。また、トップレベルのビジネス・オブジェクトのキーを指定した後は、生成されたビジネス・オブジェクト定義をファイルに保存することもできます。

新規ビジネス・オブジェクトを別のウィンドウで開くには、以下の手順を実行します。

1. 「別のウィンドウで新規ビジネス・オブジェクトを開く」を選択します。
2. 「完了」をクリックします。各ビジネス・オブジェクトがそれぞれ別のウィンドウに表示され、作成したビジネス・オブジェクトおよびビジネス・オブジェクト動詞の ASI 情報をそれぞれのウィンドウで表示および設定することができます。詳細については、48 ページの『動詞の選択』および 49 ページの『動詞 ASI の指定』を参照してください。

ビジネス・オブジェクトをファイルに保管するには、以下の手順を実行します。この手順は、親レベルのビジネス・オブジェクトのキーを指定済みの場合に限り実行できます (52 ページの図 18 を参照してください)。

1. 「ビジネス・オブジェクトのコピーを個別のファイルに保管する」を選択します。ダイアログ・ボックスが表示されます。
2. 新しいビジネス・オブジェクト定義のコピーを保管するロケーションを入力します。

Business Object Designer が指定したロケーションにファイルを保管します。

ODA での作業を終了したら、「終了」をクリックする前に、「ODA COM ODA をシャットダウンする (Shutdown ODA COM ODA)」をチェックして、ODA をシャットダウンすることができます。

## 属性レベル ASI の指定

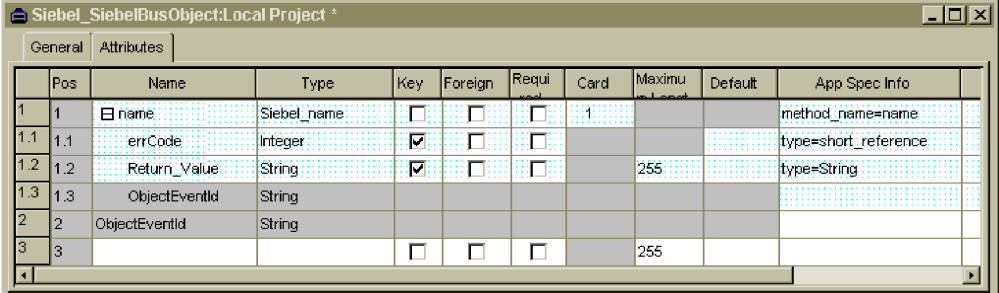
各動詞に対して実行するメソッド・シーケンスを指定して、動詞 ASI を定義すると、Business Object Designer にビジネス・オブジェクトの属性が表示されます。COM コネクタの属性レベル ASI の詳細については、31 ページの『属性レベル ASI』を参照してください。

属性は、「位置」列で定義した数値に従い、ビジネス・オブジェクト構造で表示される順序で「属性」タブにリストされます。単一 COM オブジェクト属性は単純属性として示され、その ASI には元の COM 属性名およびタイプが含まれます。

画面には、属性ごとに属性名、属性タイプ、および ASI 情報が表示されます。図 18 に、メソッド属性 ASI の例を示します。ビジネス・オブジェクトの name 属性には、元の COM コンポーネントのメソッドに属性をマップする ASI が含まれています。この例では、元のメソッドは、「アプリケーション固有の情報」列に method\_name=name ASI によって示されています。

また、name (子ビジネス・オブジェクト) には、以下の子オブジェクト属性があります。

- `errCode`。これは、COM タイプ・ライブラリーに含まれる元のメソッドのパラメーターです。この属性には `type` という ASI があり、これは COM タイプ・ライブラリー・ファイル内では `short_reference` に設定されています。この型は、ビジネス・オブジェクト内では `Integer` にマップされます。
- `return_value`。これは、name メソッドの戻り値を取得するために使用されています。COM タイプ・ライブラリー・ファイルでは、このメソッドは `BSTR` 型の戻り値を持つものとして定義されています。この型は、ビジネス・オブジェクト ASI では `String` に設定されます。COM タイプ・ライブラリー内の値を戻さないメソッドについては、ビジネス・オブジェクト属性のリストに `return_value` 属性が追加されません。



	Pos	Name	Type	Key	Foreign	Required	Card	Maximum Length	Default	App Spec Info
1	1	name	Siebel_name	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	1			method_name=name
1.1	1.1	errCode	Integer	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>				type=short_reference
1.2	1.2	Return_Value	String	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>		255		type=String
1.3	1.3	ObjectEventId	String	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>				
2	2	ObjectEventId	String	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>				
3	3			<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>		255		

図 18. 属性 ASI の設定

この画面では、親レベルのオブジェクトがキーであるかどうかを明示する必要があります (キーは、ビジネス・オブジェクトを ODA から別個のファイルに保管する際に必要になります)。また、この画面では、子オブジェクト・キーを必要に応じて設定したり、以下の情報を指定したりできます。

- ビジネス・オブジェクトを処理するコネクターに属性が必要かどうか。必要な場合は、「**必要**」チェック・ボックスにチェックマークを付けます。
- 属性の最大長が「**最大長**」列に表示される値と異なっているかどうか。
- 属性のデフォルト値。デフォルト値がある場合は、「**デフォルト**」列に値を入力します。

**注:** ビジネス・オブジェクトの作成や親レベルのキーの設定は ODA (Business Object Designer 内で実行) を使用して行うことができますが、外部キーの構成はこの方法で行ってはいけません。外部キーは非 ASI メタデータであるため、通常は、ODA を使用せずに構成する必要があります (Business Object Designer の「**ファイル**」 > 「**新規**」をクリックし、ODA を使用せずに新しいビジネス・オブジェクトを作成します)。

## ビジネス・オブジェクト・レベル ASI の指定

属性レベル ASI を指定すると、ビジネス・オブジェクト・レベル ASI を表示および変更できるようになります。ビジネス・オブジェクト・レベル ASI の詳細については、29 ページの『ビジネス・オブジェクト・レベルの ASI』を参照してください。

ビジネス・オブジェクト・レベル ASI は、「**一般**」タブにリストされます。「**ビジネス・オブジェクト・レベル・アプリケーション固有の情報**」フィールドに表示される ASI 値には、このビジネス・オブジェクトを表すプロキシー・クラスの名前が含まれます。コネクターはこの情報を使用して、プロキシー・クラスをビジネス・オブジェクトにマップします。

この画面には、ビジネス・オブジェクトがサポートする動詞もすべてリストされます。また、49 ページの『動詞 ASI の指定』で定義したような、動詞ごとの ASI も提供されます。動詞名がブランクの場合、その動詞のメソッド・シーケンスは実行されません。

54 ページの図 19 に、name ビジネス・オブジェクトのビジネス・オブジェクト・レベル ASI を示します。このビジネス・オブジェクトでは、メソッド・シーケンスが実行される動詞は **Create** だけです。この動詞には、次の図に示すメソッド・シーケンスを含む動詞 ASI が用意されています (このメソッド・シーケンスは、元々 50 ページの図 16 で設定したものです)。

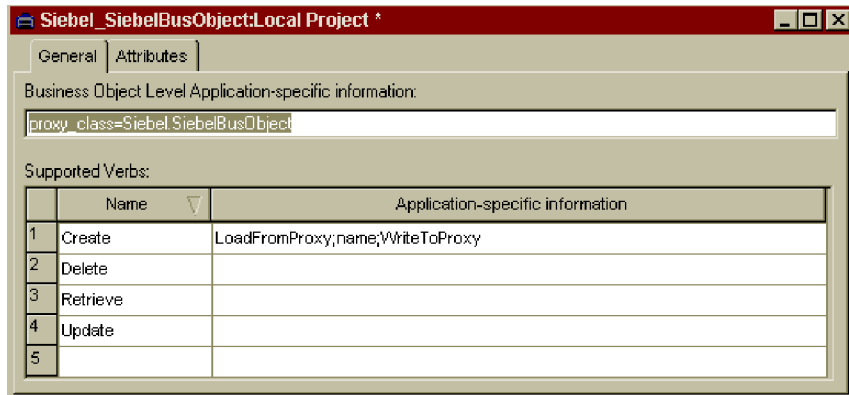


図 19. ビジネス・オブジェクト・レベル ASI の設定

この画面では、ビジネス・オブジェクトの ASI およびサポートされる動詞を変更できます。

## ビジネス・オブジェクト・ファイルのアップロード

新たに作成するビジネス・オブジェクト定義ファイルは、作成後に、統合ブローカーにアップロードする必要があります。この処理は、WebSphere InterChange Server、WebSphere MQ Integrator Broker、または WebSphere Application Server のどれを実行しているかによって異なります。

- **WebSphere InterChange Server** を実行している場合: ビジネス・オブジェクト定義ファイルをローカル・マシンに保管していて、ファイルをサーバーのリポジトリにアップロードする必要がある場合、InterChange Server のインプリメンテーションに関する文書を参照してください。
- **WebSphere MQ Integrator Broker** を実行している場合: ビジネス・オブジェクト定義を Business Object Designer から統合ブローカーにエクスポートする必要があります。詳細については、WebSphere MQ Integrator Broker のインプリメンテーションに関する文書を参照してください。
- **WebSphere Application Server** を実行している場合: 詳細については、WebSphere Application Server のインプリメンテーションに関する文書を参照してください。



---

## 第 6 章 トラブルシューティングとエラー処理

本章では、Adapter for COM によるエラーの処理方法について説明します。このアダプターによって、ロギング・メッセージおよびトレース・メッセージが生成されます。本章では、これらのメッセージと、トラブルシューティングのヒントについて説明します。本章の内容は、次のとおりです。

- 『エラー処理』
- 57 ページの『ロギング』
- 57 ページの『トレース』

---

### エラー処理

コネクターが生成するすべてのメッセージは、BIA\_COMConnector.txt という名前のメッセージ・ファイルに保管されます。(ファイル名は、LogFileName 標準コネクター構成プロパティによって決定されます。) 各メッセージには、メッセージ番号が付けられています。メッセージ番号は、以下のように、メッセージの次に表示されます。

```
Message number  
Message text
```

コネクターは、以下の各セクションで説明するような特定のエラーを処理します。

#### COMProxy によって生成される COM 例外

COMProxy インターフェース・ツールがさまざまなエラーを引き起こす場合があります。例えば、COM アプリケーションがダウンしたり、COM 呼び出しが失敗を戻したりすると、COMProxy ツールは例外をスローします。

コネクターは、エラーをログに記録して FAIL コードを戻すことにより、このような COMProxy 例外を処理します。COM 呼び出しの HRESULT は、COM 例外に含まれます。デバッグを補助するために、コネクターは HRESULT をログに記録し、VerbProcessingFailed 例外のメッセージ・フィールドに HRESULT を戻します。例外には、シーケンスにおける呼び出し失敗に関する情報も含まれます。

#### プロキシにおける ClassNotFound

ローダーがプロキシ・クラス名を受け取り、そのクラスのプロキシ・オブジェクトを作成しようとする際、クラスが見つからないと例外が発生します。コネクターはエラーをログに記録し、FAIL コードを戻します。ログには、見つからなかったクラス名が記載されます。

#### ローダーにおける InstantiationException

ローダーがプロキシ・クラス名を受け取り、そのクラスのプロキシ・オブジェクトを作成しようとする際、オブジェクト・インスタンスを作成できないと例外が発生します。コネクターはエラーをログに記録し、FAIL コードを戻します。ログには、インスタンス生成できなかったオブジェクトのクラス名が記載されます。

## ファクトリーまたは接続プールのセットアップ時の InstantiationException または ClassNotFoundException

以下のいずれかが生じると、致命的な例外が発生します。

- Agent Init() メソッドが、コネクターの構成プロパティで指定した Factory クラスまたは Connection クラスを検出できない。
- Agent Init() メソッドが、指定したクラスの Factory オブジェクトまたは Connection オブジェクトのインスタンスを生成できない。

コネクターはエラーをログに記録し、APP\_RESPONSE\_TIMEOUT コードを戻します。

## ローダーまたはインボーカーにおける Illegal AccessException

コードが無効であったり、COMProxy ツールがメソッドに対して不適切にアクセス(パブリックまたはプライベート) したりすると、コネクターは例外が発生します。

コネクターはエラーをログに記録し、FAIL コードを戻します。

## インボーカーにおける NoSuchMethodException

対応するプロキシ・オブジェクトに存在しないビジネス・オブジェクトに対してメソッドが指定されると、コネクターは例外が発生します。

コネクターはエラーをログに記録し、FAIL コードを戻します。

## インボーカーにおける InvocationTargetException

COM アプリケーション (コネクターがビジネス・オブジェクトを交換する対象) で例外が発生すると、コネクターは例外が発生します。

コネクターはエラーをログに記録し、FAIL コードを戻します。

## インボーカーにおけるメソッド・オブジェクト内の無効な引き数 (CXIgnore)

ビジネス・オブジェクトの動詞 ASI にメソッドは含まれているものの、メソッドの引き数が設定されていない場合、コネクターは例外が発生します。

コネクターはエラーをログに記録し、FAIL コードを戻します。

## キャストの失敗または属性タイプの間違い

プロキシ・オブジェクト・メソッドが、ビジネス・オブジェクトで指定したものと異なるデータ型を実行または戻すと、コネクターは例外が発生します。

コネクターはエラーをログに記録し、FAIL コードを戻します。

## 無効な動詞 ASI

コネクターに渡されるビジネス・オブジェクトの動詞 ASI が、不適切にフォーマット設定されていたり、不適切な構文を使用していたりすると、コネクターは例外を

発生します。例えば、適切なメソッド・シーケンスを含まない動詞 ASIや、アクティブな動詞対して CBOH (カスタム BO ハンドラー) を指定する子ビジネス・オブジェクトなどです。

コネクタはエラーをログに記録し、FAIL コードを戻します。

---

## ロギング

55 ページの『エラー処理』に説明されたすべてのエラーをメッセージ・ファイル (BIA\_COMConnector.txt) から読み取る必要があります。

---

## トレース

トレースはオプションのデバッグ機能であり、この機能をオンにするとコネクタの動作を密着して追跡できます。トレース・メッセージは、デフォルトでは STDOUT に書き込まれます。トレース・メッセージの構成については、17 ページの『コネクタの構成』のコネクタ構成プロパティを参照してください。トレースに関する情報、トレースを有効化して設定する方法については、「コネクタ開発ガイド」を参照してください。

表 11 に、お勧めするコネクタ・トレース・メッセージの内容をレベル別に示します。

表 11. トレース・メッセージの内容

レベル	説明
レベル 0	コネクタ・バージョンを示すトレース・メッセージには、このレベルを使用します。このレベルで実行されるトレースは他にありません。
レベル 1	このトレース・メッセージ・レベルでは、以下を実行します。 <ul style="list-style-type: none"><li>• 状況情報を提供します。</li><li>• 処理対象の各ビジネス・オブジェクトのキー情報を提供します。</li><li>• ポーリング・スレッドが入力キューで新規メッセージを検出するたびに記録を取ります。</li></ul>
レベル 2	このトレース・メッセージ・レベルでは、以下を実行します。 <ul style="list-style-type: none"><li>• コネクタが処理する各オブジェクトで使用する BO ハンドラーを識別します。</li><li>• ビジネス・オブジェクトが統合ブローカーに送られるたびにログに記録します。</li><li>• 要求ビジネス・オブジェクトを受け取るたびに指示を出します。</li></ul>
レベル 3	このトレース・メッセージ・レベルでは、以下を実行します。 <ul style="list-style-type: none"><li>• 処理対象の外部キーを識別します (該当する場合)。このようなメッセージは、コネクタがビジネス・オブジェクト内で外部キーを検出したり、コネクタがビジネス・オブジェクト内に外部キーを設定したりすると表示されます。</li><li>• ビジネス・オブジェクト処理に関連付けます。例えば、ビジネス・オブジェクト間の一致の検出や、子ビジネス・オブジェクトの配列におけるビジネス・オブジェクトの検出などです。</li></ul>

表 II. トレース・メッセージの内容 (続き)

レベル	説明
レベル 4	<p>このトレース・メッセージ・レベルでは、以下を実行します。</p> <ul style="list-style-type: none"> <li>• アプリケーション固有の情報を識別します。例えば、ビジネス・オブジェクト内のアプリケーション固有情報フィールドを処理するメソッドによって戻される値などです。</li> <li>• コネクターがいつ関数を呼び出したか、または終了したかを識別します。このようなメッセージは、コネクターの処理フローをトレースするときに役立ちます。</li> <li>• スレッド固有の処理を記録します。例えば、コネクターが複数のスレッドを作成した場合、新しいスレッドが作成されるたびにメッセージをログに記録します。</li> </ul>
レベル 5	<p>このトレース・メッセージ・レベルでは、以下を実行します。</p> <ul style="list-style-type: none"> <li>• コネクターの初期化を指示します。このメッセージ・タイプには、例えば、ブローカーから検索された各コネクター構成プロパティの値などがあります。</li> <li>• コネクターが作成した各スレッドの、実行中の詳細状況を提供します。</li> <li>• このアプリケーションで実行されるステートメントを表示します。該当する場合、宛先アプリケーションで実行されるすべてのステートメント、および置換されるすべての変数の値がコネクター・ログ・ファイルに記述されます。</li> <li>• ビジネス・オブジェクト・ダンプを記録します。コネクターは、オブジェクト処理を開始する前と後にビジネス・オブジェクトのテキスト表記を出力します (処理開始前はコネクターがコラボレーションから受け取るオブジェクトが出力され、処理後はコネクターがコラボレーションに戻すビジネス・オブジェクトが出力されます)。</li> </ul>

---

## 付録 A. コネクターの標準構成プロパティ

この付録では、WebSphere Business Integration Adapter のコネクタ・コンポーネントの標準構成プロパティについて説明します。この付録の内容は、以下の統合ブローカーで実行されるコネクタを対象としています。

- WebSphere InterChange Server (ICS)
- WebSphere Message Brokers (WebSphere MQ Integrator (WMQI)、WebSphere MQ Integrator Broker (WMQIB)、および WebSphere Business Integration Message Broker (WBIMB))
- WebSphere Application Server (WAS)

コネクタによっては、一部の標準プロパティが使用されないことがあります。Connector Configurator から統合ブローカーを選択するときには、そのブローカーで実行されるアダプターについて構成する必要のある標準プロパティのリストが表示されます。

コネクタ固有のプロパティの詳細については、該当するアダプターのユーザーズ・ガイドを参照してください。

**注:** 本書では、ディレクトリー・パスの規則として円記号 (¥) を使用します。UNIX システムを使用している場合は、円記号をスラッシュ (/) に置き換えてください。また、各オペレーティング・システムの規則に従ってください。

---

## 新規プロパティと削除されたプロパティ

以下の標準プロパティは、本リリースで追加されました。

### 新規プロパティ

- XMLNamespaceFormat

### 削除されたプロパティ

- RestartCount
- RHF2MessageDomain

---

## 標準コネクタ・プロパティの構成

アダプター・コネクタには、以下の 2 種類の構成プロパティがあります。

- 標準構成プロパティ
- コネクタ固有の構成プロパティ

このセクションでは、標準の構成プロパティについて説明します。コネクタ固有の構成プロパティの詳細については、該当するアダプターのユーザーズ・ガイドを参照してください。

## Connector Configurator の使用

Connector Configurator からコネクタ・プロパティを構成します。Connector Configurator には、System Manager からアクセスします。Connector Configurator の使用についての詳細は、付録の『Connector Configurator』を参照してください。

**注:** Connector Configurator と System Manager は、Windows システム上でのみ動作します。コネクタを UNIX システム上で運用している場合、これらのツールがインストールされた Windows マシンが必要です。UNIX 上で動作するコネクタのコネクタ・プロパティを設定する場合は、Windows マシン上で System Manager を起動し、UNIX の統合ブローカーに接続してから、コネクタ用の Connector Configurator を開く必要があります。

## プロパティ値の設定と更新

プロパティ・フィールドのデフォルトの長さは 255 文字です。

コネクタは、以下の順序に従って、プロパティの値を決定します (この場合、最も番号の大きい項目が他の項目の値をオーバーライドします)。

1. デフォルト
2. リポジトリ (統合ブローカーとして WebSphere InterChange Server を使用する場合のみ)
3. ローカル構成ファイル
4. コマンド行

コネクタは、始動時に構成値を取得します。実行時セッション中に 1 つ以上のコネクタ・プロパティの値を変更する場合は、プロパティの**更新メソッド**によって、変更を有効にする方法が決定されます。標準のコネクタ・プロパティには、以下の 4 種類の更新メソッドがあります。

### • 動的

変更を System Manager に保管すると、変更が即時に有効になります。コネクタが System Manager から独立してスタンドアロン・モードで稼働している場合 (例えば、いずれかの WebSphere Message Brokers と連携している場合) は、構成ファイルでのみプロパティを変更できます。この場合、動的更新は実行できません。

### • コンポーネント再始動

System Manager でコネクタを停止してから再始動しなければ、変更が有効になりません。アプリケーション固有コンポーネントまたは統合ブローカーを停止、再始動する必要はありません。

### • サーバー再始動

アプリケーション固有のコンポーネントおよび統合ブローカーを停止して再始動しなければ、変更が有効になりません。

### • エージェント再始動 (ICS のみ)

アプリケーション固有のコンポーネントを停止して再始動しなければ、変更が有効になりません。

特定のプロパティの更新方法を確認するには、「Connector Configurator」ウィンドウ内の「更新メソッド」列を参照するか、次に示すプロパティの要約の表の「更新メソッド」列を参照してください。

## 標準プロパティの要約

表 12 は、標準コネクタ構成プロパティの早見表です。標準プロパティの依存関係は RepositoryDirectory に基づいているため、コネクタによっては使用されないプロパティがあり、使用する統合ブローカーによってプロパティの設定が異なる可能性があります。

コネクタを実行する前に、これらのプロパティの一部の値を設定する必要があります。各プロパティの詳細については、次のセクションを参照してください。

表 12. 標準構成プロパティの要約

プロパティ名	指定可能な値	デフォルト値	更新メソッド	注
AdminInQueue	有効な JMS キュー名	CONNECTORNAME /ADMININQUEUE	コンポーネント再始動	Delivery Transport は JMS
AdminOutQueue	有効な JMS キュー名	CONNECTORNAME /ADMINOUTQUEUE	コンポーネント再始動	Delivery Transport は JMS
AgentConnections	1 から 4	1	コンポーネント再始動	Delivery Transport は MQ および IDL: Repository Directory は <REMOTE>
AgentTraceLevel	0 から 5	0	動的	
ApplicationName	アプリケーション名	コネクタ・アプリケーション名として指定された値	コンポーネント再始動	
BrokerType	ICS、WMQI、WAS			
CharacterEncoding	ascii7、ascii8、SJIS、Cp949、GBK、Big5、Cp297、Cp273、Cp280、Cp284、Cp037、Cp437 注: これは、サポートされる値の一部です。	ascii7	コンポーネント再始動	
ConcurrentEventTriggeredFlows	1 から 32,767	1	コンポーネント再始動	Repository Directory は <REMOTE>
ContainerManagedEvents	値なしまたは JMS	値なし	コンポーネント再始動	Delivery Transport は JMS
ControllerStoreAndForwardMode	true または false	True	動的	Repository Directory は <REMOTE>
ControllerTraceLevel	0 から 5	0	動的	Repository Directory は <REMOTE>
DeliveryQueue		CONNECTORNAME /DELIVERYQUEUE	コンポーネント再始動	JMS トランスポートのみ
DeliveryTransport	MQ、IDL、または JMS	JMS	コンポーネント再始動	Repository Directory がローカルの場合は、値は JMS のみ

表 12. 標準構成プロパティの要約 (続き)

プロパティ名	指定可能な値	デフォルト値	更新メソッド	注
DuplicateEventElimination	True または False	False	コンポーネント再始動	JMS トランスポートのみ、Container Managed Events は <NONE> でなければならない
FaultQueue		CONNECTORNAME/FAULTQUEUE	コンポーネント再始動	JMS トランスポートのみ
jms.FactoryClassName	CxCommon.Messaging.jms.IBMMQSeriesFactory または CxCommon.Messaging.jms.SonicMQFactory または任意の Java クラス名	CxCommon.Messaging.jms.IBMMQSeriesFactory	コンポーネント再始動	JMS トランスポートのみ
jms.MessageBrokerName	FactoryClassName が IBM の場合は crossworlds.queue.manager を使用。FactoryClassName が Sonic の場合 localhost:2506 を使用。	crossworlds.queue.manager	コンポーネント再始動	JMS トランスポートのみ
jms.NumConcurrentRequests	正整数	10	コンポーネント再始動	JMS トランスポートのみ
jms.Password	任意の有効なパスワード		コンポーネント再始動	JMS トランスポートのみ
jms.UserName	任意の有効な名前		コンポーネント再始動	JMS トランスポートのみ
JvmMaxHeapSize	ヒープ・サイズ (メガバイト単位)	128m	コンポーネント再始動	Repository Directory は <REMOTE>
JvmMaxNativeStackSize	スタックのサイズ (キロバイト単位)	128k	コンポーネント再始動	Repository Directory は <REMOTE>
JvmMinHeapSize	ヒープ・サイズ (メガバイト単位)	1m	コンポーネント再始動	Repository Directory は <REMOTE>
ListenerConcurrency	1 から 100	1	コンポーネント再始動	Delivery Transport は MQ でなければならない
Locale	en_US、ja_JP、ko_KR、zh_CN、zh_TW、fr_FR、de_DE、it_IT、es_ES、pt_BR 注: これは、サポートされるロケールの一部です。	en_US	コンポーネント再始動	
LogAtInterchangeEnd	True または False	False	コンポーネント再始動	Repository Directory は <REMOTE> でなければならない



表 12. 標準構成プロパティの要約 (続き)

プロパティ名	指定可能な値	デフォルト値	更新メソッド	注
MaxEventCapacity	1 から 2147483647	2147483647	動的	Repository Directory は <REMOTE> でなければならぬ
MessageFileName	パスまたはファイル名	InterchangeSystem.txt	コンポーネント再始動	
MonitorQueue	任意の有効なキュー名	CONNECTORNAME/MONITORQUEUE	コンポーネント再始動	JMS トランスポートのみ: DuplicateEvent Elimination は True でなければならぬ
OADAutoRestartAgent	True または False	False	動的	Repository Directory は <REMOTE> でなければならぬ
OADMaxNumRetry	正数	1000	動的	Repository Directory は <REMOTE> でなければならぬ
OADRetryTimeInterval	正数 (単位: 分)	10	動的	Repository Directory は <REMOTE> でなければならぬ
PollEndTime	HH:MM	HH:MM	コンポーネント再始動	
PollFrequency	正整数 (単位: ミリ秒)  no (ポーリングを使用不可にする)  key (コネクタのコマンド・プロンプト・ウィンドウで文字 p が入力された場合にのみポーリングする)	10000	動的	
PollQuantity	1 から 500	1	エージェント再始動	JMS トランスポートのみ: Container Managed Events を指定
PollStartTime	HH:MM(HH は 0 から 23、MM は 0 から 59)	HH:MM	コンポーネント再始動	

表 12. 標準構成プロパティの要約 (続き)

プロパティ名	指定可能な値	デフォルト値	更新メソッド	注
RepositoryDirectory	メタデータ・リポジトリの場所		エージェント再始動	ICS の場合は <REMOTE> に設定する。 WebSphere MQ Message Brokers および WAS の場合: C:\crossworlds¥repository に設定する
RequestQueue	有効な JMS キュー名	CONNECTORNAME/REQUESTQUEUE	コンポーネント再始動	Delivery Transport は JMS
ResponseQueue	有効な JMS キュー名	CONNECTORNAME/RESPONSEQUEUE	コンポーネント再始動	Delivery Transport が JMS の場合: Repository Directory が <REMOTE> の場合のみ必要
RestartRetryCount	0 から 99	3	動的	
RestartRetryInterval	適切な正数 (単位: 分): 1 から 2147483547	1	動的	
SourceQueue	有効な WebSphere MQ 名	CONNECTORNAME/SOURCEQUEUE	エージェント再始動	Delivery Transport が JMS であり、かつ Container Managed Events が指定されている場合のみ
SynchronousRequestQueue		CONNECTORNAME/ SYNCHRONOUSREQUESTQUEUE	コンポーネント再始動	Delivery Transport は JMS
SynchronousRequestTimeout	0 以上の任意の数値 (ミリ秒)	0	コンポーネント再始動	Delivery Transport は JMS
SynchronousResponseQueue		CONNECTORNAME/ SYNCHRONOUSRESPONSEQUEUE	コンポーネント再始動	Delivery Transport は JMS
WireFormat	CwXML、CwBO	CwXML	エージェント再始動	Repository Directory が <REMOTE> でない場合は CwXML。 Repository Directory が <REMOTE> であれば CwBO
WsifSynchronousRequest Timeout	0 以上の任意の数値 (ミリ秒)	0	コンポーネント再始動	WAS のみ
XMLNamespaceFormat	short、long	short	エージェント再始動	WebSphere MQ Message Brokers および WAS のみ

---

## 標準構成プロパティ

このセクションでは、各標準コネクタ構成プロパティの定義を示します。

### AdminInQueue

統合ブローカーからコネクタへ管理メッセージが送信される際に使用されるキューです。

デフォルト値は `CONNECTORNAME/ADMININQUEUE` です。

### AdminOutQueue

コネクタから統合ブローカーへ管理メッセージが送信される際に使用されるキューです。

デフォルト値は `CONNECTORNAME/ADMINOUTQUEUE` です。

### AgentConnections

`RepositoryDirectory` が `<REMOTE>` の場合のみ適用可能です。

`AgentConnections` プロパティは、`orb.init[]` により開かれる ORB 接続の数を制御します。

デフォルトでは、このプロパティの値は 1 に設定されます。このデフォルト値を変更する必要はありません。

### AgentTraceLevel

アプリケーション固有のコンポーネントのトレース・メッセージのレベルです。デフォルト値は 0 です。コネクタは、設定されたトレース・レベル以下の該当するトレース・メッセージをすべてデリバリーします。

### ApplicationName

コネクタのアプリケーションを一意的に特定する名前です。この名前は、システム管理者が WebSphere Business Integration システム環境をモニターするために使用されます。コネクタを実行する前に、このプロパティに値を指定する必要があります。

### BrokerType

使用する統合ブローカー・タイプを指定します。オプションは ICS、WebSphere Message Brokers (WMQI、WMQIB または WBIMB) または WAS です。

### CharacterEncoding

文字 (アルファベットの文字、数値表現、句読記号など) から数値へのマッピングに使用する文字コード・セットを指定します。

**注:** Java ベースのコネクタでは、このプロパティは使用しません。C++ ベースのコネクタでは、現在、このプロパティに `ascii7` という値が使用されています。

デフォルトでは、ドロップ・リストには、サポートされる文字エンコードの一部のみが表示されます。ドロップ・リストに、サポートされる他の値を追加するには、製品ディレクトリーにある `¥Data¥Std¥stdConnProps.xml` ファイルを手動で変更する必要があります。詳細については、Connector Configurator に関する付録を参照してください。

## ConcurrentEventTriggeredFlows

RepositoryDirectory が <REMOTE> の場合のみ適用可能です。

コネクターがイベントのデリバリー時に並行処理できるビジネス・オブジェクトの数を決定します。この属性の値を、並行してマップおよびデリバリーできるビジネス・オブジェクトの数に設定します。例えば、この属性の値を 5 に設定すると、5 個のビジネス・オブジェクトが並行して処理されます。デフォルト値は 1 です。

このプロパティを 1 よりも大きい値に設定すると、ソース・アプリケーションのコネクターが、複数のイベント・ビジネス・オブジェクトを同時にマップして、複数のコラボレーション・インスタンスにそれらのビジネス・オブジェクトを同時にデリバリーすることができます。これにより、統合ブローカーへのビジネス・オブジェクトのデリバリーにかかる時間、特にビジネス・オブジェクトが複雑なマップを使用している場合のデリバリー時間が短縮されます。ビジネス・オブジェクトのコラボレーションに到達する速度を増大させると、システム全体のパフォーマンスを向上させることができます。

ソース・アプリケーションから宛先アプリケーションまでのフロー全体に並行処理を実装するには、次のようにする必要があります。

- **Maximum number of concurrent events** プロパティの値を増加して、コラボレーションが複数のスレッドを使用できるように構成します。
- 宛先アプリケーションのアプリケーション固有コンポーネントが複数の要求を並行して実行できることを確認します。つまり、このコンポーネントがマルチスレッド化されているか、またはコネクター・エージェント並列処理を使用でき、複数プロセスに対応するよう構成されている必要があります。Parallel Process Degree 構成プロパティに、1 より大きい値を設定します。

ConcurrentEventTriggeredFlows プロパティは、順次に実行される単一スレッド処理であるコネクターのポーリングでは無効です。

## ContainerManagedEvents

このプロパティにより、JMS イベント・ストアを使用する JMS 対応コネクターが、保証付きイベント・デリバリーを提供できるようになります。保証付きイベント・デリバリーでは、イベントはソース・キューから除去され、単一 JMS トランザクションとして宛先キューに配置されます。

デフォルト値は No value です。

ContainerManagedEvents を JMS に設定した場合には、保証付きイベント・デリバリーを使用できるように次のプロパティも構成する必要があります。

- PollQuantity = 1 から 500
- SourceQueue = CONNECTORNAME/SOURCEQUEUE

また、MimeType、DHClass、および DataHandlerConfigMOName (オプション) プロパティーを設定したデータ・ハンドラーも構成する必要があります。これらのプロパティーの値を設定するには、Connector Configurator の「データ・ハンドラー」タブを使用します。「データ・ハンドラー」タブの値のフィールドは、ContainerManagedEvents を JMS に設定した場合にのみ表示されます。

**注:** ContainerManagedEvents を JMS に設定した場合、コネクターはその pollForEvents() メソッドを呼び出さなくなるため、そのメソッドの機能は使用できなくなります。

このプロパティーは、DeliveryTransport プロパティーが値 JMS に設定されている場合にのみ表示されます。

## ControllerStoreAndForwardMode

RepositoryDirectory が <REMOTE> の場合のみ適用可能です。

宛先側のアプリケーション固有のコンポーネントが使用不可であることをコネクター・コントローラーが検出した場合に、コネクター・コントローラーが実行する動作を設定します。

このプロパティーを true に設定した場合、イベントが ICS に到達したときに宛先側のアプリケーション固有のコンポーネントが使用不可であれば、コネクター・コントローラーはそのアプリケーション固有のコンポーネントへの要求をブロックします。アプリケーション固有のコンポーネントが作動可能になると、コネクター・コントローラーはアプリケーション固有のコンポーネントにその要求を転送します。

ただし、コネクター・コントローラーが宛先側のアプリケーション固有のコンポーネントにサービス呼び出し要求を転送した後でこのコンポーネントが使用不可になった場合、コネクター・コントローラーはその要求を失敗させます。

このプロパティーを false に設定した場合、コネクター・コントローラーは、宛先側のアプリケーション固有のコンポーネントが使用不可であることを検出すると、ただちにすべてのサービス呼び出し要求を失敗させます。

デフォルト値は true です。

## ControllerTraceLevel

RepositoryDirectory が <REMOTE> の場合のみ適用可能です。

コネクター・コントローラーのトレース・メッセージのレベルです。デフォルト値は 0 です。

## DeliveryQueue

DeliveryTransport が JMS の場合のみ適用されます。

コネクターから統合ブローカーへビジネス・オブジェクトが送信されるときに使用されるキューです。

デフォルト値は CONNECTORNAME/DELIVERYQUEUE です。

## DeliveryTransport

イベントのデリバリーのためのトランスポート機構を指定します。指定可能な値は、WebSphere MQ の MQ、CORBA IIOP の IDL、Java Messaging Service の JMS です。

- ICS がブローカー・タイプの場合は、DeliveryTransport プロパティの指定可能な値は MQ、IDL、または JMS であり、デフォルトは IDL になります。
- RepositoryDirectory がローカル・ディレクトリーの場合は、指定可能な値は JMS のみです。

DeliveryTransport プロパティに指定されている値が、MQ または IDL である場合、コネクタは、CORBA IIOP を使用してサービス呼び出し要求と管理メッセージを送信します。

### WebSphere MQ および IDL

イベントのデリバリー・トランスポートには、IDL ではなく WebSphere MQ を使用してください (1 種類の製品だけを使用する必要がある場合を除きます)。

WebSphere MQ が IDL よりも優れている点は以下のとおりです。

- 非同期 (ASYNC) 通信:  
WebSphere MQ を使用すると、アプリケーション固有のコンポーネントは、サーバーが利用不能である場合でも、イベントをポーリングして永続的に格納することができます。
- サーバー・サイド・パフォーマンス:  
WebSphere MQ を使用すると、サーバー・サイドのパフォーマンスが向上します。最適化モードでは、WebSphere MQ はイベントへのポインターのみをリポジトリ・データベースに格納するので、実際のイベントは WebSphere MQ キュー内に残ります。これにより、サイズが大きい可能性のあるイベントをリポジトリ・データベースに書き込む必要がありません。
- エージェント・サイド・パフォーマンス:  
WebSphere MQ を使用すると、アプリケーション固有のコンポーネント側のパフォーマンスが向上します。WebSphere MQ を使用すると、コネクタのポーリング・スレッドは、イベントを選出した後、コネクタのキューにそのイベントを入れ、次のイベントを選出します。この方法は IDL よりも高速で、IDL の場合、コネクタのポーリング・スレッドは、イベントを選出した後、ネットワーク経由でサーバー・プロセスにアクセスしてそのイベントをリポジトリ・データベースに永続的に格納してから、次のイベントを選出する必要があります。

### JMS

Java Messaging Service (JMS) を使用しての、コネクタとクライアント・コネクタ・フレームワークとの間の通信を可能にします。

JMS をデリバリー・トランスポートとして選択した場合は、`jms.MessageBrokerName`、`jms.FactoryClassName`、`jms.Password`、`jms.UserName` などの追加の JMS プロパティが Connector Configurator 内に表示されます。このうち最初の 2 つは、このトランスポートの必須プロパティです。

**重要:** 以下の環境では、コネクタに JMS トランスポート機構を使用すると、メモリ制限が発生することもあります。

- AIX 5.0
- WebSphere MQ 5.3.0.1
- ICS が統合ブローカーの場合

この環境では、WebSphere MQ クライアント内でメモリーが使用されるため、(サーバー側の) コネクター・コントローラーと (クライアント側の) コネクターの両方を始動するのは困難な場合があります。ご使用のシステムのプロセス・ヒープ・サイズが 768M 未満である場合には、次のように設定することをお勧めします。

- CWSHaredEnv.sh スクリプト内で LDR\_CNTRL 環境変数を設定する。

このスクリプトは、製品ディレクトリー配下の %bin ディレクトリーにあります。テキスト・エディターを使用して、CWSHaredEnv.sh スクリプトの最初の行として次の行を追加します。

```
export LDR_CNTRL=MAXDATA=0x30000000
```

この行は、ヒープ・メモリーの使用量を最大 768 MB (3 セグメント \* 256 MB) に制限します。プロセス・メモリーがこの制限値を超えると、ページ・スワッピングが発生し、システムのパフォーマンスに悪影響を与える場合があります。

- IPCCBaseAddress プロパティーの値を 11 または 12 に設定する。このプロパティーの詳細については、「システム・インストール・ガイド (UNIX 版)」を参照してください。

## DuplicateEventElimination

このプロパティーを true に設定すると、JMS 対応コネクターによるデリバリー・キューへの重複イベントのデリバリーが防止されます。この機能を使用するには、コネクターに対し、アプリケーション固有のコード内でビジネス・オブジェクトの **ObjectEventId** 属性として一意のイベント ID が設定されている必要があります。これはコネクター開発時に設定されます。

このプロパティーは、false に設定することもできます。

**注:** DuplicateEventElimination を true に設定する際は、MonitorQueue プロパティーを構成して保証付きイベント・デリバリーを使用可能にする必要があります。

## FaultQueue

コネクターでメッセージを処理中にエラーが発生すると、コネクターは、そのメッセージを状況表示および問題説明とともにこのプロパティーに指定されているキューに移動します。

デフォルト値は CONNECTORNAME/FAULTQUEUE です。

## JvmMaxHeapSize

エージェントの最大ヒープ・サイズ (メガバイト単位)。このプロパティーは、RepositoryDirectory の値が <REMOTE> の場合にのみ適用されます。

デフォルト値は 128M です。

## JvmMaxNativeStackSize

エージェントの最大ネイティブ・スタック・サイズ (キロバイト単位)。このプロパティは、RepositoryDirectory の値が <REMOTE> の場合にのみ適用されます。

デフォルト値は 128K です。

## JvmMinHeapSize

エージェントの最小ヒープ・サイズ (メガバイト単位)。このプロパティは、RepositoryDirectory の値が <REMOTE> の場合にのみ適用されます。

デフォルト値は 1M です。

## jms.FactoryClassName

JMS プロバイダーのためにインスタンスを生成するクラス名を指定します。JMS をデリバリー・トランスポート機構 (DeliveryTransport) として選択する際は、このコネクタ・プロパティを必ず設定してください。

デフォルト値は CxCommon.Messaging.jms.IBMMQSeriesFactory です。

## jms.MessageBrokerName

JMS プロバイダーのために使用するブローカー名を指定します。JMS をデリバリー・トランスポート機構 (DeliveryTransport) として選択する際は、このコネクタ・プロパティを必ず設定してください。

デフォルト値は crossworlds.queue.manager です。

## jms.NumConcurrentRequests

コネクタに対して同時に送信することができる並行サービス呼び出し要求の数 (最大値) を指定します。この最大値に達した場合、新規のサービス呼び出し要求はブロックされ、既存のいずれかの要求が完了した後で処理されます。

デフォルト値は 10 です。

## jms.Password

JMS プロバイダーのためのパスワードを指定します。このプロパティの値はオプションです。

デフォルトはありません。

## jms.UserName

JMS プロバイダーのためのユーザー名を指定します。このプロパティの値はオプションです。

デフォルトはありません。



## ListenerConcurrency

このプロパティは、統合ブローカーとして ICS を使用する場合の MQ Listener でのマルチスレッド化をサポートしています。このプロパティにより、データベースへの複数イベントの書き込み操作をバッチ処理できるので、システム・パフォーマンスが向上します。デフォルト値は 1 です。

このプロパティは、MQ トランスポートを使用するコネクタにのみ適用されます。DeliveryTransport プロパティには MQ を設定してください。

## Locale

言語コード、国または地域、および、希望する場合には、関連した文字コード・セットを指定します。このプロパティの値は、データの照合やソート順、日付と時刻の形式、通貨記号などの国/地域別情報を決定します。

ロケール名は、次の書式で指定します。

`ll_TT.codeset`

ここで、以下のように説明されます。

<code>ll</code>	2 文字の言語コード (普通は小文字)
<code>TT</code>	2 文字の国または地域コード (普通は大文字)
<code>codeset</code>	関連文字コード・セットの名前。名前のこの部分は、通常、オプションです。

デフォルトでは、ドロップ・リストには、サポートされるロケールの一部のみが表示されます。ドロップ・リストに、サポートされる他の値を追加するには、製品ディレクトリーにある `¥Data¥Std¥stdConnProps.xml` ファイルを手動で変更する必要があります。詳細については、Connector Configurator に関する付録を参照してください。

デフォルト値は `en_US` です。コネクタが国際化されていない場合、このプロパティの有効な値は `en_US` のみです。特定のコネクタが国際化されているかどうかを判別するには、以下の Web サイトにあるコネクタのバージョン・リストを参照してください。

<http://www.ibm.com/software/websphere/wbiadapters/infocenter>、または  
<http://www.ibm.com/websphere/integration/wicsserver/infocenter>

## LogAtInterchangeEnd

RepositoryDirectory が <REMOTE> の場合のみ適用可能です。

統合ブローカーのログ宛先にエラーを記録するかどうかを指定します。ブローカーのログ宛先にログを記録すると、電子メール通知もオンになります。これにより、エラーまたは致命的エラーが発生すると、InterchangeSystem.cfg ファイルに指定された MESSAGE\_RECIPIENT に対する電子メール・メッセージが生成されます。

例えば、LogAtInterChangeEnd を true に設定した場合にコネクタからアプリケーションへの接続が失われると、指定されたメッセージ宛先に、電子メール・メッセージが送信されます。デフォルト値は false です。

## MaxEventCapacity

コントローラー・バッファ内のイベントの最大数。このプロパティはフロー制御が使用し、RepositoryDirectory プロパティの値が <REMOTE> の場合にのみ適用されます。

値は 1 から 2147483647 の間の正整数です。デフォルト値は 2147483647 です。

## MessageFileName

コネクタ・メッセージ・ファイルの名前です。メッセージ・ファイルの標準位置は %connectors%messages です。メッセージ・ファイルが標準位置に格納されていない場合は、メッセージ・ファイル名を絶対パスで指定します。

コネクタ・メッセージ・ファイルが存在しない場合は、コネクタは InterchangeSystem.txt をメッセージ・ファイルとして使用します。このファイルは、製品ディレクトリーに格納されています。

**注:** 特定のコネクタについて、コネクタ独自のメッセージ・ファイルがあるかどうかを判別するには、該当するアダプターのユーザズ・ガイドを参照してください。

## MonitorQueue

コネクタが重複イベントをモニターするために使用する論理キューです。このプロパティは、DeliveryTransport プロパティ値が JMS であり、かつ DuplicateEventElimination が TRUE に設定されている場合にのみ使用されます。

デフォルト値は CONNECTORNAME/MONITORQUEUE です。

## OADAutoRestartAgent

RepositoryDirectory が <REMOTE> の場合のみ有効です。

コネクタが自動再始動およびリモート再始動機能を使用するかどうかを指定します。この機能では、MQ により起動される Object Activation Daemon (OAD) を使用して、異常シャットダウン後にコネクタを再始動したり、System Monitor からリモート・コネクタを始動したりします。

自動再始動機能およびリモート再始動機能を使用可能にするには、このプロパティを true に設定する必要があります。MQ により起動される OAD 機能の構成方法については、「システム・インストール・ガイド (Windows 版)」または「システム・インストール・ガイド (UNIX 版)」を参照してください。

デフォルト値は false です。

## OADMaxNumRetry

RepositoryDirectory が <REMOTE> の場合のみ有効です。

異常シャットダウンの後で MQ により起動される OAD がコネクタの再始動を自動的に試行する回数の最大数を指定します。このプロパティを有効にするためには、OADAutoRestartAgent プロパティを true に設定する必要があります。

デフォルト値は 1000 です。

## OADRetryTimeInterval

RepositoryDirectory が <REMOTE> の場合のみ有効です。

MQ により起動される OAD の再試行時間間隔の分数を指定します。コネクタ・エージェントがこの再試行時間間隔内に再始動しない場合は、コネクタ・コントローラーはコネクタ・エージェントを再び再始動するように OAD に要求します。OAD はこの再試行プロセスを OADMaxNumRetry プロパティで指定された回数だけ繰り返します。このプロパティを有効にするためには、OADAutoRestartAgent プロパティを true に設定する必要があります。

デフォルト値は 10 です。

## PollEndTime

イベント・キューのポーリングを停止する時刻です。形式は HH:MM です。ここで、HH は 0 から 23 時を表し、MM は 0 から 59 分を表します。

このプロパティには必ず有効な値を指定してください。デフォルト値は HH:MM ですが、この値は必ず変更する必要があります。

## PollFrequency

ポーリング・アクション間の時間の長さです。PollFrequency は以下の値のいずれかに設定します。

- ポーリング・アクション間のミリ秒数。
- ワード key。コネクタは、コネクタのコマンド・プロンプト・ウィンドウで文字 p が入力されたときのみポーリングを実行します。このワードは小文字で入力します。
- ワード no。コネクタはポーリングを実行しません。このワードは小文字で入力します。

デフォルト値は 10000 です。

**重要:** 一部のコネクタでは、このプロパティの使用が制限されています。このプロパティが使用されるかどうかを特定のコネクタについて判断するには、該当するアダプター・ガイドのインストールと構成についての章を参照してください。

## PollQuantity

コネクタがアプリケーションからポーリングする項目の数を指定します。アダプターにコネクタ固有のポーリング数設定プロパティがある場合、標準プロパティの値は、このコネクタ固有のプロパティの設定値によりオーバーライドされます。

## PollStartTime

イベント・キューのポーリングを開始する時刻です。形式は HH:MM です。ここで、HH は 0 から 23 時を表し、MM は 0 から 59 分を表します。

このプロパティーには必ず有効な値を指定してください。デフォルト値は HH:MM ですが、この値は必ず変更する必要があります。

## RequestQueue

統合ブローカーが、ビジネス・オブジェクトをコネクターに送信するときに使用されるキューです。

デフォルト値は CONNECTOR/REQUESTQUEUE です。

## RepositoryDirectory

コネクターが XML スキーマ文書を読み取るリポジトリの場所です。この XML スキーマ文書には、ビジネス・オブジェクト定義のメタデータが含まれています。

統合ブローカーが ICS の場合はこの値を <REMOTE> に設定する必要があります。これは、コネクターが InterChange Server リポジトリからこの情報を取得するためです。

統合ブローカーが WebSphere Message Broker または WAS の場合は、この値を <local directory> に設定する必要があります。

## ResponseQueue

DeliveryTransport が JMS の場合のみ適用可能で、RepositoryDirectory が <REMOTE> の場合のみ必須です。

JMS 応答キューを指定します。JMS 応答キューは、応答メッセージをコネクター・フレームワークから統合ブローカーへデリバリーします。統合ブローカーが ICS の場合、サーバーは要求を送信し、JMS 応答キューの応答メッセージを待ちます。

## RestartRetryCount

コネクターによるコネクター自体の再始動の試行回数を指定します。このプロパティーを並列コネクターに対して使用する場合、コネクターのマスター側のアプリケーション固有のコンポーネントがスレーブ側のアプリケーション固有のコンポーネントの再始動を試行する回数が指定されます。

デフォルト値は 3 です。

## RestartRetryInterval

コネクターによるコネクター自体の再始動の試行間隔を分単位で指定します。このプロパティーを並列コネクターに対して使用する場合、コネクターのマスター側のアプリケーション固有のコンポーネントがスレーブ側のアプリケーション固有のコンポーネントの再始動を試行する間隔が指定されます。指定可能な値の範囲は 1 から 2147483647 です。

デフォルト値は 1 です。

## SourceQueue

DeliveryTransport が JMS で、ContainerManagedEvents が指定されている場合のみ適用されます。

JMS イベント・ストアを使用する JMS 対応コネクタでの保証付きイベント・デリバリーをサポートするコネクタ・フレームワークに、JMS ソース・キューを指定します。詳細については、66 ページの『ContainerManagedEvents』を参照してください。

デフォルト値は `CONNECTOR/SOURCEQUEUE` です。

## SynchronousRequestQueue

`DeliveryTransport` が JMS の場合のみ適用されます。

同期応答を要求する要求メッセージを、コネクタ・フレームワークからブローカーに配信します。このキューは、コネクタが同期実行を使用する場合にのみ必要です。同期実行の場合、コネクタ・フレームワークは、`SynchronousRequestQueue` にメッセージを送信し、`SynchronousResponseQueue` でブローカーから戻される応答を待機します。コネクタに送信される応答メッセージには、元のメッセージの ID を指定する 相関 ID が含まれています。

デフォルトは `CONNECTORNAME/SYNCHRONOUSREQUESTQUEUE` です。

## SynchronousResponseQueue

`DeliveryTransport` が JMS の場合のみ適用されます。

同期要求に対する応答として送信される応答メッセージを、ブローカーからコネクタ・フレームワークに配信します。このキューは、コネクタが同期実行を使用する場合にのみ必要です。

デフォルトは `CONNECTORNAME/SYNCHRONOUSRESPONSEQUEUE` です。

## SynchronousRequestTimeout

`DeliveryTransport` が JMS の場合のみ適用されます。

コネクタが同期要求への応答を待機する時間を分単位で指定します。コネクタは、指定された時間内に応答を受信できなかった場合、元の同期要求メッセージをエラー・メッセージとともに障害キューに移動します。

デフォルト値は 0 です。

## WireFormat

トランスポートのメッセージ・フォーマットです。

- `RepositoryDirectory` がローカル・ディレクトリーの場合は、設定は `CwXML` になります。
- `RepositoryDirectory` の値が `<REMOTE>` の場合には、設定値は `CwBO` です。

## WsifSynchronousRequest Timeout

WAS 統合ブローカーでのみ使用されます。

コネクタが同期要求への応答を待機する時間を分単位で指定します。コネクタは、指定された時間内に応答を受信できなかった場合、元の同期要求メッセージをエラー・メッセージとともに障害キューに移動します。

デフォルト値は 0 です。

## **XMLNameSpaceFormat**

WebSphere Message Brokers および WAS 統合ブローカーでのみ使用されます。

ビジネス・オブジェクト定義の XML 形式でネーム・スペースを short と long のどちらにするかをユーザーが指定できるようにするための、強力なプロパティです。

デフォルト値は short です。

---

## 付録 B. Connector Configurator

この付録では、Connector Configurator を使用してアダプターの構成プロパティ値を設定する方法について説明します。

Connector Configurator を使用して次の作業を行います。

- コネクタを構成するためのコネクタ固有のプロパティ・テンプレートを作成する
- 構成ファイルを作成する
- 構成ファイル内のプロパティを設定する

注:

本書では、ディレクトリー・パスの規則として円記号 (¥) を使用します。UNIX システムを使用している場合は、円記号をスラッシュ (/) に置き換えてください。また、各オペレーティング・システムの規則に従ってください。

この付録では、次のトピックについて説明します。

- 77 ページの『Connector Configurator の概要』
- 78 ページの『Connector Configurator の始動』
- 79 ページの『コネクタ固有のプロパティ・テンプレートの作成』
- 82 ページの『新規構成ファイルの作成』
- 85 ページの『構成ファイル・プロパティの設定』
- 94 ページの『国際化環境における Connector Configurator の使用』

---

### Connector Configurator の概要

Connector Configurator では、次の統合ブローカーで使用するアダプターのコネクタ・コンポーネントを構成できます。

- WebSphere InterChange Server (ICS)
- WebSphere MQ Integrator (WMQI)、WebSphere MQ Integrator Broker (WMQIB)、および WebSphere Business Integration Message Broker (WBIMB)。これらをまとめて MQ Message Brokers と呼びます。
- WebSphere Application Server (WAS)

Connector Configurator を使用して次の作業を行います。

- コネクタを構成するためのコネクタ固有のプロパティ・テンプレートを作成する。
- **コネクタ構成ファイル**を作成します。インストールするコネクタごとに構成ファイルを 1 つ作成する必要があります。
- 構成ファイル内のプロパティを設定する。  
場合によっては、コネクタ・テンプレートでプロパティに対して設定されているデフォルト値を変更する必要があります。また、サポートされるビジネス・オブジェクト定義と、ICS の場合はコラボレーションとともに使用するマップを

指定し、必要に応じてメッセージング、ロギング、トレース、およびデータ・ハンドラー・パラメーターを指定する必要があります。

Connector Configurator の実行モードと使用する構成ファイルのタイプは、実行する統合ブローカーによって異なります。例えば、使用している統合ブローカーが WMQI の場合、Connector Configurator を System Manager から実行するのではなく、直接実行します (78 ページの『スタンドアロン・モードでの Connector Configurator の実行』を参照)。

コネクタ構成プロパティには、標準の構成プロパティ (すべてのコネクタにもつプロパティ) と、コネクタ固有のプロパティ (特定のアプリケーションまたはテクノロジーのためにコネクタに必要なプロパティ) とが含まれます。

**標準プロパティ**はすべてのコネクタにより使用されるので、標準プロパティを新規に定義する必要はありません。ファイルを作成すると、Connector Configurator により標準プロパティがこの構成ファイルに挿入されます。ただし、Connector Configurator で各標準プロパティの値を設定する必要があります。

標準プロパティの範囲は、ブローカーと構成によって異なる可能性があります。特定のプロパティに特定の値が設定されている場合にのみ使用できるプロパティがあります。Connector Configurator の「標準のプロパティ」ウィンドウには、特定の構成で設定可能なプロパティが表示されます。

ただし**コネクタ固有プロパティ**の場合は、最初にプロパティを定義し、その値を設定する必要があります。このため、特定のアダプターのコネクタ固有プロパティのテンプレートを作成します。システム内で既にテンプレートが作成されている場合には、作成されているテンプレートを使用します。システム内でまだテンプレートが作成されていない場合には、79 ページの『新規テンプレートの作成』のステップに従い、テンプレートを新規に作成します。

**注:** Connector Configurator は、Windows 環境内でのみ実行されます。UNIX 環境でコネクタを実行する場合には、Windows で Connector Configurator を使用して構成ファイルを変更し、このファイルを UNIX 環境へコピーします。

---

## Connector Configurator の始動

以下の 2 種類のモードで Connector Configurator を開始および実行できます。

- スタンドアロン・モードで個別に実行
- System Manager から

### スタンドアロン・モードでの Connector Configurator の実行

どのブローカーを実行している場合にも、Connector Configurator を個別に実行し、コネクタ構成ファイルを編集できます。

これを行うには、以下のステップを実行します。

- 「スタート」>「プログラム」から、「**IBM WebSphere InterChange Server**」>「**IBM WebSphere Business Integration Toolset**」>「**開発**」>「**Connector Configurator**」をクリックします。
- 「ファイル」>「新規」>「構成ファイル」を選択します。



- 「システム接続: Integration Broker」の隣のプルダウン・メニューをクリックします。使用しているブローカーに応じて、ICS、WebSphere Message Brokers、または WAS を選択します。

Connector Configurator を個別に実行して構成ファイルを生成してから、System Manager に接続してこの構成ファイルを System Manager プロジェクトに保存することもできます (84 ページの『構成ファイルの完成』を参照)。

---

## System Manager からの Connector Configurator の実行

System Manager から Connector Configurator を実行できます。

Connector Configurator を実行するには、以下のステップを実行します。

1. System Manager を開きます。
2. 「System Manager」ウィンドウで、「統合コンポーネント・ライブラリー」アイコンを展開し、「コネクタ」を強調表示します。
3. System Manager メニュー・バーから、「ツール」>「Connector Configurator」をクリックします。「Connector Configurator」ウィンドウが開き、「新規コネクタ」ダイアログ・ボックスが表示されます。
4. 「システム接続: Integration Broker」の隣のプルダウン・メニューをクリックします。使用しているブローカーに応じて、ICS、WebSphere Message Brokers、または WAS を選択します。

既存の構成ファイルを編集するには、以下のステップを実行します。

1. 「System Manager」ウィンドウの「コネクタ」フォルダーでいずれかの構成ファイルを選択し、右クリックします。Connector Configurator が開き、この構成ファイルの統合ブローカー・タイプおよびファイル名が上部に表示されます。
2. 「標準のプロパティ」タブをクリックし、この構成ファイルに含まれているプロパティを確認します。

---

## コネクタ固有のプロパティ・テンプレートの作成

コネクタの構成ファイルを作成するには、コネクタ固有プロパティのテンプレートとシステム提供の標準プロパティが必要です。

コネクタ固有プロパティのテンプレートを新規に作成するか、または既存のファイルをテンプレートとして使用します。

- テンプレートの新規作成については、79 ページの『新規テンプレートの作成』を参照してください。
- 既存のファイルを使用する場合には、既存のテンプレートを変更し、新しい名前でのこのテンプレートを保管します。

## 新規テンプレートの作成

このセクションでは、テンプレートでプロパティを作成し、プロパティの一般特性および値を定義し、プロパティ間の依存関係を指定する方法について説明します。次にそのテンプレートを保管し、新規コネクタ構成ファイルを作成するためのベースとして使用します。

テンプレートは以下のように作成します。

1. 「ファイル」>「新規」>「コネクタ固有プロパティ・テンプレート」をクリックします。
2. 以下のフィールドを含む「コネクタ固有プロパティ・テンプレート」ダイアログ・ボックスが表示されます。
  - 「テンプレート」、「名前」  
このテンプレートが使用されるコネクタ (またはコネクタのタイプ) を表す固有の名前を入力します。テンプレートから新規構成ファイルを作成するためのダイアログ・ボックスを開くと、この名前が再度表示されます。
  - 「旧テンプレート」、「変更する既存のテンプレートを選択してください」  
「テンプレート名」表示に、現在使用可能なすべてのテンプレートの名前が表示されます。
  - テンプレートに含まれているコネクタ固有のプロパティ定義を調べるには、「テンプレート名」表示でそのテンプレートの名前を選択します。そのテンプレートに含まれているプロパティ定義のリストが「テンプレートのプレビュー」表示に表示されます。テンプレートを作成するときには、ご使用のコネクタに必要なプロパティ定義に類似したプロパティ定義が含まれている既存のテンプレートを使用できます。
3. 「テンプレート名」表示からテンプレートを選択し、その名前を「名前の検索」フィールドに入力し (または「テンプレート名」で自分の選択項目を強調表示し)、「次へ」をクリックします。

ご使用のコネクタで使用するコネクタ固有のプロパティが表示されるテンプレートが見つからない場合は、自分で作成する必要があります。

### 一般特性の指定

「次へ」をクリックしてテンプレートを選択すると、「プロパティ: コネクタ固有プロパティ・テンプレート」ダイアログ・ボックスが表示されます。このダイアログ・ボックスには、定義済みプロパティの「一般」特性のタブと「値」の制限のタブがあります。「一般」表示には以下のフィールドがあります。

- **一般:**
  - プロパティ・タイプ
  - 更新されたメソッド
  - 説明
- **フラグ**
  - 標準フラグ
- **カスタム・フラグ**
  - フラグ

プロパティの一般特性の選択を終えたら、「値」タブをクリックします。

### 値の指定

「値」タブを使用すると、プロパティの最大長、最大複数値、デフォルト値、または値の範囲を設定できます。編集可能な値も許可されます。これを行うには、以下のステップを実行します。

1. 「値」タブをクリックします。「一般」のパネルに代わって「値」の表示パネルが表示されます。
2. 「プロパティを編集」表示でプロパティの名前を選択します。
3. 「最大長」および「最大複数値」のフィールドで、変更を行います。次のステップで説明するように、プロパティの「プロパティ値」ダイアログ・ボックスを開かない限り、そのプロパティの変更内容は受け入れられませんので、注意してください。
4. 値テーブルの左上の隅にあるボックスを右マウス・ボタンでクリックしてから、「追加」をクリックします。「プロパティ値」ダイアログ・ボックスが表示されます。このダイアログ・ボックスではプロパティのタイプに応じて、値だけを入力できる場合と、値と範囲の両方を入力できる場合があります。適切な値または範囲を入力し、「OK」をクリックします。
5. 「値」パネルが最新表示され、「最大長」および「最大複数値」で行った変更が表示されます。以下のような 3 つの列があるテーブルが表示されます。
 

「値」の列には、「プロパティ値」ダイアログ・ボックスで入力した値と、以前に作成した値が表示されます。

「デフォルト値」の列では、値のいずれかをデフォルトとして指定することができます。

「値の範囲」の列には、「プロパティ値」ダイアログ・ボックスで入力した範囲が表示されます。

値が作成されて、グリッドに表示されると、そのテーブルの表示内から編集できるようになります。テーブルにある既存の値の変更を行うには、その行の行番号をクリックして行全体を選択します。次に「値」フィールドを右マウス・ボタンでクリックし、「値の編集 (Edit Value)」をクリックします。

## 依存関係の設定

「一般」タブと「値」タブで変更を行ったら、「次へ」をクリックします。「依存関係: コネクタ固有プロパティ・テンプレート」ダイアログ・ボックスが表示されます。

依存プロパティは、別のプロパティの値が特定の条件に合致する場合にのみ、テンプレートに組み込まれて、構成ファイルで使用されるプロパティです。例えば、テンプレートに PollQuantity が表示されるのは、トランスポート機構が JMS であり、DuplicateEventElimination が True に設定されている場合のみです。プロパティを依存プロパティとして指定し、依存する条件を設定するには、以下のステップを実行します。

1. 「使用可能なプロパティ」表示で、依存プロパティとして指定するプロパティを選択します。
2. 「プロパティを選択」フィールドで、ドロップダウン・メニューを使用して、条件値を持たせるプロパティを選択します。
3. 「条件演算子」フィールドで以下のいずれかを選択します。
  - == (等しい)
  - != (等しくない)
  - > (より大)
  - < (より小)

>= (より大か等しい)

<= (より小か等しい)

4. 「条件値」フィールドで、依存プロパティをテンプレートに組み込むために必要な値を入力します。
5. 「使用可能なプロパティ」表示で依存プロパティを強調表示させて矢印をクリックし、「依存プロパティ」表示に移動させます。
6. 「完了」をクリックします。Connector Configurator により、XML 文書として入力した情報が、Connector Configurator がインストールされている %bin ディレクトリーの %data%app の下に保管されます。

---

## 新規構成ファイルの作成

構成ファイルを新規に作成するには、最初に統合ブローカーを選択します。選択したブローカーによって、構成ファイルに記述されるプロパティが決まります。

ブローカーを選択するには、以下のステップを実行します。

- Connector Configurator のホーム・メニューで、「ファイル」>「新規」>「コネクター構成」をクリックします。「新規コネクター」ダイアログ・ボックスが表示されます。
- 「Integration Broker」フィールドで、ICS 接続、WebSphere Message Brokers 接続、WAS 接続のいずれかを選択します。
- この章で後述する説明に従って「新規コネクター」ウィンドウの残りのフィールドに入力します。

また、以下の作業も実行できます。

- 「System Manager」ウィンドウで「コネクター」フォルダーを右クリックし、「新規コネクターの作成」を選択します。Connector Configurator が開き、「新規コネクター」ダイアログ・ボックスが表示されます。

## コネクター固有のテンプレートからの構成ファイルの作成

コネクター固有のテンプレートを作成すると、テンプレートを使用して構成ファイルを作成できます。

1. 「ファイル」>「新規」>「コネクター構成」をクリックします。
2. 以下のフィールドを含む「新規コネクター」ダイアログ・ボックス表示されず。

- **名前**

コネクターの名前を入力します。名前では大文字と小文字が区別されます。入力する名前は、システムにインストールされているコネクターのファイル名に対応した一意の名前でなければなりません。

**重要:** Connector Configurator では、入力された名前のスペルはチェックされません。名前が正しいことを確認してください。

- **システム接続**

ICS 接続、WebSphere Message Brokers 接続、WAS のいずれかをクリックします。

- 「コネクタ固有プロパティ・テンプレート」を選択します。

ご使用のコネクタ用に設計したテンプレートの名前を入力します。「テンプレート名」表示に、使用可能なテンプレートが表示されます。「テンプレート名」表示で名前を選択すると、「プロパティ・テンプレートのプレビュー」表示に、そのテンプレートで定義されているコネクタ固有のプロパティが表示されます。

使用するテンプレートを選択し、「OK」をクリックします。

3. 構成しているコネクタの構成画面が表示されます。タイトル・バーに統合ブローカーとコネクタの名前が表示されます。ここですべてのフィールドに値を入力して定義を完了するか、ファイルを保管して後でフィールドに値を入力するかを選択できます。
4. ファイルを保管するには、「ファイル」>「保管」>「ファイルに」をクリックするか、「ファイル」>「保管」>「プロジェクトに」をクリックします。プロジェクトに保管するには、System Manager が実行中でなければなりません。ファイルとして保管する場合は、「ファイル・コネクタを保管」ダイアログ・ボックスが表示されます。\*.cfg をファイル・タイプとして選択し、「ファイル名」フィールド内に名前が正しいスペル (大文字と小文字の区別を含む) で表示されていることを確認してから、ファイルを保管するディレクトリにナビゲートし、「保管」をクリックします。Connector Configurator のメッセージ・パネルの状況表示に、構成ファイルが正常に作成されたことが示されます。

**重要:** ここで設定するディレクトリ・パスおよび名前は、コネクタの始動ファイルで指定するコネクタ構成ファイルのパスおよび名前に一致している必要があります。

5. この章で後述する手順に従って、「Connector Configurator」ウィンドウの各タブにあるフィールドに値を入力し、コネクタ定義を完了します。

---

## 既存ファイルの使用

使用可能な既存ファイルは、以下の 1 つまたは複数の形式になります。

- コネクタ定義ファイル。  
コネクタ定義ファイルは、特定のコネクタのプロパティと、適用可能なデフォルト値がリストされたテキスト・ファイルです。コネクタの配布パッケージの `¥repository` ディレクトリ内には、このようなファイルが格納されていることがあります (通常、このファイルの拡張子は `.txt` です。例えば、XML コネクタの場合は `CN_XML.txt` です)。
- ICS リポジトリ・ファイル。  
コネクタの以前の ICS インプリメンテーションで使用した定義は、そのコネクタの構成で使用されたりポジトリ・ファイルで使用可能になります。そのようなファイルの拡張子は、通常 `.in` または `.out` です。
- コネクタの以前の構成ファイル。  
これらのファイルの拡張子は、通常 `*.cfg` です。

これらのいずれのファイル・ソースにも、コネクタのコネクタ固有プロパティのほとんど、あるいはすべてが含まれますが、この章内の後で説明するように、コネクタ構成ファイルは、ファイルを開いて、プロパティを設定しない限り完成しません。

既存ファイルを使用してコネクタを構成するには、Connector Configurator でそのファイルを開き、構成を修正し、そのファイルを再度保管する必要があります。

以下のステップを実行して、ディレクトリーから \*.txt、\*.cfg、または \*.in ファイルを開きます。

1. Connector Configurator 内で、「ファイル」>「開く」>「ファイルから」をクリックします。
2. 「ファイル・コネクタを開く」ダイアログ・ボックス内で、以下のいずれかのファイル・タイプを選択して、使用可能なファイルを調べます。
  - 構成 (\*.cfg)
  - ICS リポジトリ (\*.in、\*.out)  
ICS 環境でのコネクタの構成にリポジトリ・ファイルが使用された場合には、このオプションを選択します。リポジトリ・ファイルに複数のコネクタ定義が含まれている場合は、ファイルを開くとすべての定義が表示されます。
  - すべてのファイル (\*.\*)  
コネクタのアダプター・パッケージに \*.txt ファイルが付属していた場合、または別の拡張子で定義ファイルが使用可能である場合は、このオプションを選択します。
3. ディレクトリー表示内で、適切なコネクタ定義ファイルへ移動し、ファイルを選択し、「開く」をクリックします。

System Manager プロジェクトからコネクタ構成を開くには、以下のステップを実行します。

1. System Manager を始動します。System Manager が開始されている場合にのみ、構成を System Manager から開いたり、System Manager に保管することができます。
2. Connector Configurator を始動します。
3. 「ファイル」>「開く」>「プロジェクトから」をクリックします。

---

## 構成ファイルの完成

構成ファイルを開くか、プロジェクトからコネクタを開くと、「Connector Configurator」ウィンドウに構成画面が表示されます。この画面には、現在の属性と値が表示されます。

構成画面のタイトルには、ファイル内で指定された統合ブローカーとコネクタの名前が表示されます。正しいブローカーが設定されていることを確認してください。正しいブローカーが設定されていない場合、コネクタを構成する前にブローカー値を変更してください。これを行うには、以下のステップを実行します。

1. 「標準のプロパティー」タブで、BrokerType プロパティーの値フィールドを選択します。ドロップダウン・メニューで、値 ICS、WMQI、または WAS を選択します。
2. 選択したブローカーに関連付けられているプロパティーが「標準のプロパティー」タブに表示されます。ここでファイルを保管するか、または 87 ページの

『サポートされるビジネス・オブジェクト定義の指定』の説明に従い残りの構成フィールドに値を入力することができます。

- 構成が完了したら、「ファイル」>「保管」>「プロジェクトに」を選択するか、または「ファイル」>「保管」>「ファイルに」を選択します。

ファイルに保管する場合は、\*.cfg を拡張子として選択し、ファイルの正しい格納場所を選択して、「保管」をクリックします。

複数のコネクタ構成を開いている場合、構成をすべてファイルに保管するには「すべてファイルに保管」を選択し、コネクタ構成をすべて System Manager プロジェクトに保管するには「すべてプロジェクトに保管」をクリックします。

Connector Configurator では、ファイルを保管する前に、必須の標準プロパティすべてに値が設定されているかどうかを確認されます。必須の標準プロパティに値が設定されていない場合、Connector Configurator は、検証が失敗したというメッセージを表示します。構成ファイルを保管するには、そのプロパティの値を指定する必要があります。

---

## 構成ファイル・プロパティの設定

新規のコネクタ構成ファイルを作成して名前を付けるとき、または既存のコネクタ構成ファイルを開くときには、Connector Configurator によって構成画面が表示されます。構成画面には、必要な構成値のカテゴリに対応する複数のタブがあります。

Connector Configurator では、すべてのブローカーで実行されているコネクタで、以下のカテゴリのプロパティに値が設定されている必要があります。

- 標準のプロパティ
- コネクタ固有のプロパティ
- サポートされるビジネス・オブジェクト
- トレース/ログ・ファイルの値
- データ・ハンドラー (保証付きイベント・デリバリーで JMS メッセージングを使用するコネクタの場合に該当する)

**注:** JMS メッセージングを使用するコネクタの場合は、データをビジネス・オブジェクトに変換するデータ・ハンドラーの構成に関して追加のカテゴリが表示される場合があります。

**ICS** で実行されているコネクタの場合、以下のプロパティの値も設定されている必要があります。

- 関連付けられたマップ
- リソース
- メッセージング (該当する場合)

**重要:** Connector Configurator では、英語文字セットまたは英語以外の文字セットのいずれのプロパティ値も設定可能です。ただし、標準のプロパティおよびコネクタ固有プロパティ、およびサポートされるビジネス・オブジェクトの名前では、英語文字セットのみを使用する必要があります。

標準プロパティとコネクタ固有プロパティの違いは、以下のとおりです。

- コネクターの標準プロパティは、コネクターのアプリケーション固有のコンポーネントとブローカー・コンポーネントの両方によって共有されます。すべてのコネクターが同じ標準プロパティのセットを使用します。これらのプロパティの説明は、各アダプター・ガイドの付録 A にあります。変更できるのはこれらの値の一部のみです。
- アプリケーション固有のプロパティは、コネクターのアプリケーション固有コンポーネント (アプリケーションと直接対話するコンポーネント) のみに適用されます。各コネクターには、そのコネクターのアプリケーションだけで使用されるアプリケーション固有のプロパティがあります。これらのプロパティには、デフォルト値が用意されているものもあれば、そうでないものもあります。また、一部のデフォルト値は変更することができます。各アダプター・ガイドのインストールおよび構成の章に、アプリケーション固有のプロパティおよび推奨値が記述されています。

「標準プロパティ」と「コネクター固有プロパティ」のフィールドは、どのフィールドが構成可能であるかを示すために色分けされています。

- 背景がグレーのフィールドは、標準のプロパティを表します。値を変更することはできますが、名前の変更およびプロパティの除去はできません。
- 背景が白のフィールドは、アプリケーション固有のプロパティを表します。これらのプロパティは、アプリケーションまたはコネクターの特定のニーズによって異なります。値の変更も、これらのプロパティの除去も可能です。
- 「値」フィールドは構成できます。
- 「更新メソッド」フィールドは通知用であり、構成できません。このフィールドは、値が変更されたプロパティをアクティブにするために必要なアクションを示します。

## 標準コネクター・プロパティの設定

標準のプロパティの値を変更するには、以下の手順を実行します。

1. 値を設定するフィールド内でクリックします。
2. 値を入力するか、ドロップダウン・メニューが表示された場合にはメニューから値を選択します。
3. 標準のプロパティの値をすべて入力後、以下のいずれかを実行することができます。
  - 変更内容を破棄し、元の値を保持したままで Connector Configurator を終了するには、「ファイル」>「終了」をクリックし (またはウィンドウを閉じ)、変更内容を保管するかどうかを確認するプロンプトが出されたら「いいえ」をクリックします。
  - Connector Configurator 内の他のカテゴリーの値を入力するには、そのカテゴリーのタブを選択します。「標準のプロパティ」(またはその他のカテゴリー) で入力した値は、次のカテゴリーに移動しても保持されます。ウィンドウを閉じると、すべてのカテゴリーで入力した値を一括して保管するかまたは破棄するかを確認するプロンプトが出されます。
  - 修正した値を保管するには、「ファイル」>「終了」をクリックし (またはウィンドウを閉じ)、変更内容を保管するかどうかを確認するプロンプトが出されたら「はい」をクリックします。「ファイル」メニューまたはツールバーから「保管」>「ファイルに」をクリックする方法もあります。



## アプリケーション固有の構成プロパティの設定

アプリケーション固有の構成プロパティの場合、プロパティ名の追加または変更、値の構成、プロパティの削除、およびプロパティの暗号化が可能です。プロパティのデフォルトの長さは 255 文字です。

1. グリッドの左上端の部分で右マウス・ボタンをクリックします。ポップアップ・メニュー・バーが表示されます。プロパティを追加するときは「追加」をクリックします。子プロパティを追加するには、親の行番号で右マウス・ボタンをクリックし、「子を追加」をクリックします。
2. プロパティまたは子プロパティの値を入力します。
3. プロパティを暗号化するには、「暗号化」ボックスを選択します。
4. 86 ページの『標準コネクタ・プロパティの設定』の説明に従い、変更内容を保管するかまたは破棄するかを選択します。

各プロパティごとに表示される「更新メソッド」は、変更された値をアクティブにするためにコンポーネントまたはエージェントの再始動が必要かどうかを示します。

**重要:** 事前設定のアプリケーション固有のコネクタ・プロパティ名を変更すると、コネクタに障害が発生する可能性があります。コネクタをアプリケーションに接続したり正常に実行したりするために、特定のプロパティ名が必要である場合があります。

### コネクタ・プロパティの暗号化

「プロパティを編集」ウィンドウの「暗号化」チェック・ボックスにチェックマークを付けると、アプリケーション固有のプロパティを暗号化することができます。値の暗号化を解除するには、「暗号化」チェック・ボックスをクリックしてチェックマークを外し、「検証」ダイアログ・ボックスに正しい値を入力し、「OK」をクリックします。入力された値が正しい場合は、暗号化解除された値が表示されます。

各プロパティとそのデフォルト値のリストおよび説明は、各コネクタのアダプター・ユーザーズ・ガイドにあります。

プロパティに複数の値がある場合には、プロパティの最初の値に「暗号化」チェック・ボックスが表示されます。「暗号化」を選択すると、そのプロパティのすべての値が暗号化されます。プロパティの複数の値を暗号化解除するには、そのプロパティの最初の値の「暗号化」チェック・ボックスをクリックしてチェックマークを外してから、「検証」ダイアログ・ボックスで新規の値を入力します。入力値が一致すれば、すべての複数值が暗号化解除されます。

### 更新メソッド

付録『コネクタの標準構成プロパティ』の 60 ページの『プロパティ値の設定と更新』にある更新メソッドの説明を参照してください。

## サポートされるビジネス・オブジェクト定義の指定

コネクタで使用するビジネス・オブジェクトを指定するには、Connector Configurator の「サポートされているビジネス・オブジェクト」タブを使用します。

汎用ビジネス・オブジェクトと、アプリケーション固有のビジネス・オブジェクトの両方を指定する必要があり、またそれらのビジネス・オブジェクト間のマップの関連を指定することが必要です。

サポートされているビジネス・オブジェクトを指定するには、指定するビジネス・オブジェクトとそのマップがシステム内に存在している必要があります。

- ビジネス・オブジェクト定義とマップ定義が System Manager プロジェクトに保管されている必要があります。
- 統合ブローカーとして WebSphere Message Broker を使用している場合、ビジネス・オブジェクト定義と MQ メッセージ・セット・ファイルが存在している必要があります。

**注:** コネクタによっては、アプリケーションでイベント通知や (メタオブジェクトを使用した) 追加の構成を実行するために、特定のビジネス・オブジェクトをサポートされているものとして指定することが必要な場合もあります。詳細は、「コネクタ開発ガイド (C++ 用)」または「コネクタ開発ガイド (Java 用)」を参照してください。

## ご使用のブローカーが ICS の場合

ビジネス・オブジェクト定義がコネクタでサポートされることを指定する場合や、既存のビジネス・オブジェクト定義のサポート設定を変更する場合は、「サポートされているビジネス・オブジェクト」タブをクリックし、以下のフィールドを使用してください。

**ビジネス・オブジェクト名:** ビジネス・オブジェクト定義がコネクタによってサポートされることを指定するには、System Manager を実行し、以下の手順を実行します。

1. 「ビジネス・オブジェクト名」リストで空のフィールドをクリックします。  
System Manager プロジェクトに存在するすべてのビジネス・オブジェクト定義を示すドロップダウン・リストが表示されます。
2. 追加するビジネス・オブジェクトをクリックします。
3. ビジネス・オブジェクトの「エージェント・サポート」(以下で説明) を設定します。
4. 「Connector Configurator」ウィンドウの「ファイル」メニューで、「プロジェクトに保管」をクリックします。追加したビジネス・オブジェクト定義に指定されたサポートを含む、変更されたコネクタ定義が、System Manager のプロジェクトに保管されます。

サポートされるリストからビジネス・オブジェクトを削除する場合は、以下の手順を実行します。

1. ビジネス・オブジェクト・フィールドを選択するため、そのビジネス・オブジェクトの左側の番号をクリックします。
2. 「Connector Configurator」ウィンドウの「編集」メニューから、「行を削除」をクリックします。リスト表示からビジネス・オブジェクトが除去されます。
3. 「ファイル」メニューから、「プロジェクトの保管」をクリックします。

サポートされるリストからビジネス・オブジェクトを削除すると、コネクタ定義が変更され、削除されたビジネス・オブジェクトはコネクタのこのインプリメン

ーションで使用不可になります。コネクターのコードに影響したり、そのビジネス・オブジェクト定義そのものが System Manager から削除されることはありません。

**エージェント・サポート:** ビジネス・オブジェクトがエージェント・サポートを備えている場合、システムは、コネクター・エージェントを介してアプリケーションにデータを配布する際にそのビジネス・オブジェクトの使用を試みます。

一般に、コネクターのアプリケーション固有ビジネス・オブジェクトは、そのコネクターのエージェントによってサポートされますが、汎用ビジネス・オブジェクトはサポートされません。

ビジネス・オブジェクトがコネクター・エージェントによってサポートされるよう指定するには、「エージェント・サポート」ボックスにチェックマークを付けます。「Connector Configurator」ウィンドウでは「エージェント・サポート」の選択の妥当性は検査されません。

**最大トランザクション・レベル:** コネクターの最大トランザクション・レベルは、そのコネクターがサポートする最大のトランザクション・レベルです。

ほとんどのコネクターの場合、選択可能な項目は「最大限の努力」のみです。

トランザクション・レベルの変更を有効にするには、サーバーを再始動する必要があります。

### ご使用のブローカーが WebSphere Message Broker の場合

MQ メッセージ・セット・ファイル (\*.set ファイル) には、メッセージ・セット ID が記述されています。この ID は、コネクターがサポートするビジネス・オブジェクトを指定するときに、Connector Configurator で必要となります。MQ メッセージ・セット・ファイルの作成の詳細については、「*WebSphere MQ Integrator Broker* 用インプリメンテーション・ガイド」を参照してください。

ビジネス・オブジェクト定義をシステムに追加するごとに、Connector Configurator を使用して、コネクターがサポートするビジネス・オブジェクトを指定する必要があります。

サポートされるビジネス・オブジェクトを指定するには、以下の手順を実行します。

1. 「サポートされているビジネス・オブジェクト」タブを選択して、System Manager が実行中の場合は、ドロップ・リストを使用してビジネス・オブジェクト名を指定します。
2. System Manager が実行中でない場合は、「ビジネス・オブジェクト名」を入力します。
3. 「メッセージ・セット ID (オプション)」を入力します。

### ご使用のブローカーが WAS の場合

使用するブローカー・タイプとして WebSphere Application Server を選択した場合、Connector Configurator にメッセージ・セット ID は必要ありません。「サポートされるビジネス・オブジェクト」タブには、サポートされるビジネス・オブジェクトの「ビジネス・オブジェクト名」列のみが表示されます。

スタンドアロン・モードで作業している (System Manager に接続していない) 場合、手動でビジネス・オブジェクト名を入力する必要があります。

System Manager を実行している場合、「サポートされているビジネス・オブジェクト」タブの「ビジネス・オブジェクト名」列の下にある空のボックスを選択できます。コンボ・ボックスが表示され、コネクターが属する統合コンポーネント・ライブラリー・プロジェクトから選択可能なビジネス・オブジェクトのリストが示されます。このリストから必要なビジネス・オブジェクトを選択します。

## 関連付けられたマップ (ICS のみ)

各コネクターは、現在 WebSphere InterChange Server でアクティブなビジネス・オブジェクト定義、およびそれらの関連付けられたマップのリストをサポートします。このリストは、「関連付けられたマップ」タブを選択すると表示されます。

ビジネス・オブジェクトのリストには、エージェントでサポートされるアプリケーション固有のビジネス・オブジェクトと、コントローラーがサブスクリプション・コラボレーションに送信する、対応する汎用オブジェクトが含まれます。マップの関連によって、アプリケーション固有のビジネス・オブジェクトを汎用ビジネス・オブジェクトに変換したり、汎用ビジネス・オブジェクトをアプリケーション固有のビジネス・オブジェクトに変換したりするとき、どのマップを使用するかが決定されます。

特定のソースおよび宛先ビジネス・オブジェクトについて一意的に定義されたマップを使用する場合、表示を開くと、マップは常にそれらの該当するビジネス・オブジェクトに関連付けられます。ユーザーがそれらを変更する必要はありません (変更できません)。

サポートされるビジネス・オブジェクトで使用可能なマップが複数ある場合は、そのビジネス・オブジェクトを、使用する必要のあるマップに明示的にバインドすることが必要になります。

「関連付けられたマップ」タブには以下のフィールドが表示されます。

- **ビジネス・オブジェクト名**

これらは、「サポートされているビジネス・オブジェクト」タブで指定した、このコネクターでサポートされるビジネス・オブジェクトです。「サポートされているビジネス・オブジェクト」タブでビジネス・オブジェクトを追加指定した場合、その内容は、「Connector Configurator」ウィンドウの「ファイル」メニューから「プロジェクトに保管」を選択して、変更を保管した後に、このリストに反映されます。

- **関連付けられたマップ**

この表示には、コネクターの、サポートされるビジネス・オブジェクトでの使用のためにシステムにインストールされたすべてのマップが示されます。各マップのソース・ビジネス・オブジェクトは、「ビジネス・オブジェクト名」表示でマップ名の左側に表示されます。

- **明示的**

場合によっては、関連付けられたマップを明示的にバインドすることが必要になります。

明示的バインディングが必要なのは、特定のサポートされるビジネス・オブジェクトに複数のマップが存在する場合のみです。ICS は、ブート時、各コネクタでサポートされるそれぞれのビジネス・オブジェクトにマップを自動的にバインドしようとしています。複数のマップでその入力データとして同一のビジネス・オブジェクトが使用されている場合、サーバーは、他のマップのスーパーセットである 1 つのマップを見つけて、バインドしようとしています。

他のマップのスーパーセットであるマップがないと、サーバーは、ビジネス・オブジェクトを単一のマップにバインドすることができないため、バインディングを明示的に設定することが必要になります。

以下の手順を実行して、マップを明示的にバインドします。

1. 「明示的 (Explicit)」列で、バインドするマップのチェック・ボックスにチェックマークを付けます。
2. ビジネス・オブジェクトに関連付けるマップを選択します。
3. 「Connector Configurator」ウィンドウの「ファイル」メニューで、「プロジェクトに保管」をクリックします。
4. プロジェクトを ICS に配置します。
5. 変更を有効にするため、サーバーをリブートします。

## リソース (ICS)

「リソース」タブでは、コネクタ・エージェントが、コネクタ・エージェント並列処理を使用して同時に複数のプロセスを処理するかどうか、またどの程度処理するかを決定する値を設定できます。

すべてのコネクタがこの機能をサポートしているわけではありません。複数のプロセスを使用するよりも複数のスレッドを使用する方が通常は効率的であるため、Java でマルチスレッドとして設計されたコネクタ・エージェントを実行している場合、この機能を使用することはお勧めできません。

## メッセージング (ICS)

メッセージング・プロパティは、DeliveryTransport 標準プロパティの値として MQ を設定し、ブローカー・タイプとして ICS を設定した場合にのみ、使用可能です。これらのプロパティは、コネクタによるキューの使用方法に影響しません。

## トレース/ログ・ファイル値の設定

コネクタ構成ファイルまたはコネクタ定義ファイルを開くと、Connector Configurator は、そのファイルのログおよびトレースの値をデフォルト値として使用します。Connector Configurator 内でこれらの値を変更できます。

ログとトレースの値を変更するには、以下の手順を実行します。

1. 「トレース/ログ・ファイル」タブをクリックします。
2. ログとトレースのどちらでも、以下のいずれかまたは両方へのメッセージの書き込みを選択できます。
  - コンソールに (STDOUT):  
ログ・メッセージまたはトレース・メッセージを STDOUT ディスプレイに書き込みます。

**注:** STDOUT オプションは、Windows プラットフォームで実行しているコネクタの「トレース/ログ・ファイル」タブでのみ使用できます。

- ファイルに:  
ログ・メッセージまたはトレース・メッセージを指定されたファイルに書き込みます。ファイルを指定するには、ディレクトリー・ボタン (省略符号) をクリックし、指定する格納場所へ移動し、ファイル名を指定し、「保管」をクリックします。ログ・メッセージまたはトレース・メッセージは、指定した場所の指定したファイルに書き込まれます。

**注:** ログ・ファイルとトレース・ファイルはどちらも単純なテキスト・ファイルです。任意のファイル拡張子を使用してこれらのファイル名を設定できます。ただし、トレース・ファイルの場合、拡張子として `.trc` ではなく `.trace` を使用することをお勧めします。これは、システム内に存在する可能性がある他のファイルとの混同を避けるためです。ログ・ファイルの場合、通常使用されるファイル拡張子は `.log` および `.txt` です。

## データ・ハンドラー

データ・ハンドラー・セクションの構成が使用可能となるのは、`DeliveryTransport` の値に `JMS` を、また `ContainerManagedEvents` の値に `JMS` を指定した場合のみです。すべてのアダプターでデータ・ハンドラーを使用できるわけではありません。

これらのプロパティーに使用する値については、付録 A の『コネクタの標準構成プロパティー』の `ContainerManagedEvents` の下の説明を参照してください。その他の詳細は、「コネクタ開発ガイド (C++ 用)」または「コネクタ開発ガイド (Java 用)」を参照してください。

---

## 構成ファイルの保管

コネクタの構成が完了したら、コネクタ構成ファイルを保管します。Connector Configurator では、構成中に選択したブローカー・モードでファイルを保管します。Connector Configurator のタイトル・バーには現在のブローカー・モード (ICS、WMQI、または WAS) が常に表示されます。

ファイルは XML 文書として保管されます。XML 文書は次の 3 通りの方法で保管できます。

- System Manager から、InterChange Server プロジェクトに `*.con` 拡張子付きファイルとして保管します。
- 指定したディレクトリーに保管します。
- スタンドアロン・モードで、ディレクトリー・フォルダーに `*.cfg` 拡張子付きファイルとして保管します。

統合コンポーネント・ライブラリーに構成を保管している場合は、それを配置する必要があります。

- 統合ブローカーとして ICS を使用している場合は、構成を統合コンポーネント・ライブラリーからユーザーの InterChange Server プロジェクトにコピーし、System Manager を使用してそのユーザー・プロジェクトを配置します。

- 統合ブローカーとして WMQI を使用している場合は、構成ファイルを統合コンポーネント・ライブラリーからユーザーの Message Broker プロジェクトにコピーし、System Manager を使用して Integrator Broker に配置します。
- 統合ブローカーとして WBIMB を使用している場合は、構成ファイルを統合コンポーネント・ライブラリーからユーザーの Message Broker プロジェクトにコピーし、System Manager を使用して Message Broker ワークスペースに配置します。
- 統合ブローカーとして WAS を使用している場合は、構成を統合コンポーネント・ライブラリーからユーザーの WAS プロジェクトにコピーし、System Manager を使用してそのユーザー・プロジェクトを配置します。

System Manager でのプロジェクトの使用法、および配置の詳細については、以下のインプリメンテーション・ガイドを参照してください。

- ICS: 「*WebSphere InterChange Server インプリメンテーション・ガイド*」
- WebSphere Message Brokers: 「*WebSphere Message Brokers 使用アダプター・インプリメンテーション・ガイド*」
- WAS: 「*アダプター実装ガイド (WebSphere Application Server)*」

---

## 構成ファイルの変更

既存の構成ファイルの統合ブローカー設定を変更できます。これにより、他のブローカーで使用する構成ファイルを新規に作成するときに、このファイルをテンプレートとして使用できます。

**注:** 統合ブローカーを切り替える場合には、ブローカー・モード・プロパティーと同様に他の構成プロパティーも変更する必要があります。

既存の構成ファイルでのブローカーの選択を変更するには、以下の手順を実行します (オプション)。

- Connector Configurator で既存の構成ファイルを開きます。
- 「標準のプロパティー」タブを選択します。
- 「標準のプロパティー」タブの「**BrokerType**」フィールドで、ご使用のブローカーに合った値を選択します。

現行値を変更すると、プロパティー画面の利用可能なタブおよびフィールド選択がただちに更新され、選択した新規ブローカーに適したタブとフィールドのみが表示されます。

---

## 構成の完了

コネクターの構成ファイルを作成し、そのファイルを変更した後で、コネクターの始動時にコネクターが構成ファイルの位置を特定できるかどうかを確認してください。

これを行うには、コネクターが使用する始動ファイルを開き、コネクター構成ファイルに使用されている格納場所とファイル名が、ファイルに対して指定した名前およびファイルを格納したディレクトリーまたはパスと正確に一致しているかどうかを検証します。

## 国際化環境における Connector Configurator の使用

Connector Configurator は国際化され、構成ファイルと統合ブローカー間の文字変換を処理できます。Connector Configurator では、ネイティブなエンコード方式を使用しています。構成ファイルに書き込む場合は UTF-8 エンコード方式を使用します。

Connector Configurator は、以下の場所で英語以外の文字をサポートします。

- すべての値のフィールド
- ログ・ファイルおよびトレース・ファイル・パス (「**トレース/ログ・ファイル**」タブで指定)

CharacterEncoding および Locale 標準構成プロパティのドロップ・リストに表示されるのは、サポートされる値の一部のみです。ドロップ・リストに、サポートされる他の値を追加するには、製品ディレクトリーの `¥Data¥Std¥stdConnProps.xml` ファイルを手動で変更する必要があります。

例えば、Locale プロパティの値のリストにロケール `en_GB` を追加するには、`stdConnProps.xml` ファイルを開き、以下に太文字で示した行を追加してください。

```
<Property name="Locale"
isRequired="true"
updateMethod="component restart">
  <ValidType>String</ValidType>
  <ValidValues>
    <Value>ja_JP</Value>
    <Value>ko_KR</Value>
    <Value>zh_CN</Value>
    <Value>zh_TW</Value>
    <Value>fr_FR</Value>
    <Value>de_DE</Value>
    <Value>it_IT</Value>
    <Value>es_ES</Value>
    <Value>pt_BR</Value>
    <Value>en_US</Value>
    <Value>en_GB</Value>
    <DefaultValue>en_US</DefaultValue>
  </ValidValues>
</Property>
```



---

## 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を提供し、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せんが、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認できません。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をお願いします。

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ます、単に目標を示しているものです。

---

## 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。

汎用プログラミング・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  
Domino  
Lotus  
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、Pentium および ProShare 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。

Adapter for COM には、Eclipse Project (<http://www.eclipse.org/>) により開発されたソフトウェアが含まれています。



WebSphere Business Integration Adapter Framework V2.4.0







Printed in Japan