IBM WebSphere Business Integration Server Express and Express Plus

Adapter for COM ユーザーズ・ガイド

アダプター・バージョン 1.2.x

IBM WebSphere Business Integration Server Express and Express Plus

Adapter for COM ユーザーズ・ガイド

アダプター・バージョン 1.2.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、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、WebSphere Business Integration Server Express Adapter for COM バージョン 1.2.x に適用されます。

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

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

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

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

(URL は、変更になる場合があります)

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

原 典： WebSphere Business Integration
Server Express and Express Plus
Adapter for COM User Guide
Adapter Version 1.2.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 2005.8

この文書では、平成明朝体™W3、平成明朝体™W7、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™W7、平成明朝体™W9、平成角ゴシック体™W3、
平成角ゴシック体™W5、平成角ゴシック体™W7

目次

本書について

製品 IBM(R) WebSphere Business Integration Server Express および IBM(R) WebSphere Business Integration Server Express Plus は、InterChange Server Express、関連する Toolset Express、CollaborationFoundation、およびソフトウェア統合アダプターのセットで構成されています。Toolset Express に含まれるツールは、ビジネス・オブジェクトの作成、変更、および管理に役立ちます。プリパッケージされている各種アダプターは、お客様の複数アプリケーションにまたがるビジネス・プロセスに応じて、いずれかを選べるようになっています。標準プロセス・テンプレート (CollaborationFoundation) によって、カスタマイズされたプロセスをすばやく作成することができます。

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

対象読者

本書は、WebSphere Business Integration Server Express 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 Server Express 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

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

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

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。
%text%	% 記号で囲まれたテキストは、Windows™ の `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 1.2.x の新機能

2004 年 6 月更新。Adapter for COM のバージョン 1.2.x では、編集の説明を加えてマニュアルが更新されています。

第 1 章 概要

Connector for COM は、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、Interchange Server Express 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。

アダプターの規格

このコネクターは 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

このコネクターは Windows 2003 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ー、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・エンティティー (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 Server Express 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ー・コンポーネント。
- **COM コンポーネント**。コネクターは、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。コネクター処理の際は、プロキシー・オブジェクトが、コネクターの 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ー、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。適切なコネクター・プロパティー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、コネクターの寿命に対応して持続します。作成される接続数は、PoolSize プロパティー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ー。
- **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ーを検査し、そのエンティティーの中からビジネス・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 Express 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・エンティティー。保管されるオブジェクトは、プロキシー・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。
- **プロキシー・クラス**。コネクター内の COM コンポーネント・クラスを表す Java クラス。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 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ーがビジネス・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 コネクターのアーキテクチャー

このセクションでは、コネクターのアーキテクチャーについて説明します。 5 ページの図 1 に上位で行われる要求処理を、 6 ページの図 2 にコネクター動作の詳細を示します。

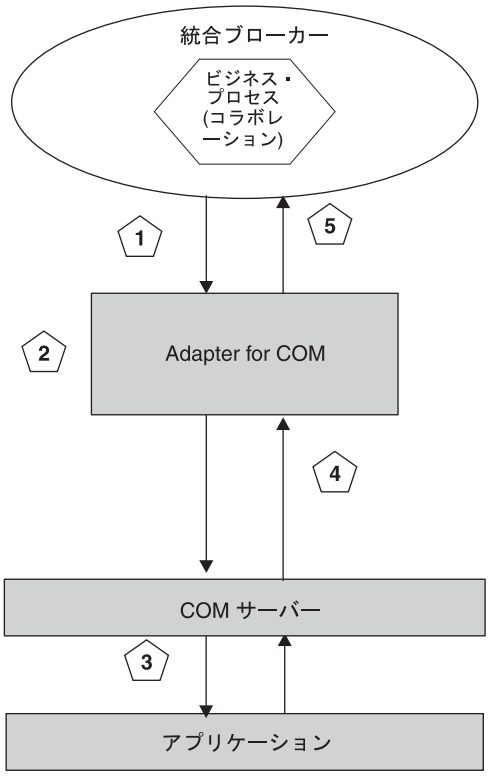


図 1. Connector for COM での要求処理

1. コネクターが、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ー・オブジェクト・インスタンスを作成します。プロキシー・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ー・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ー・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、プロキシー・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、プロキシー・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、 6 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。

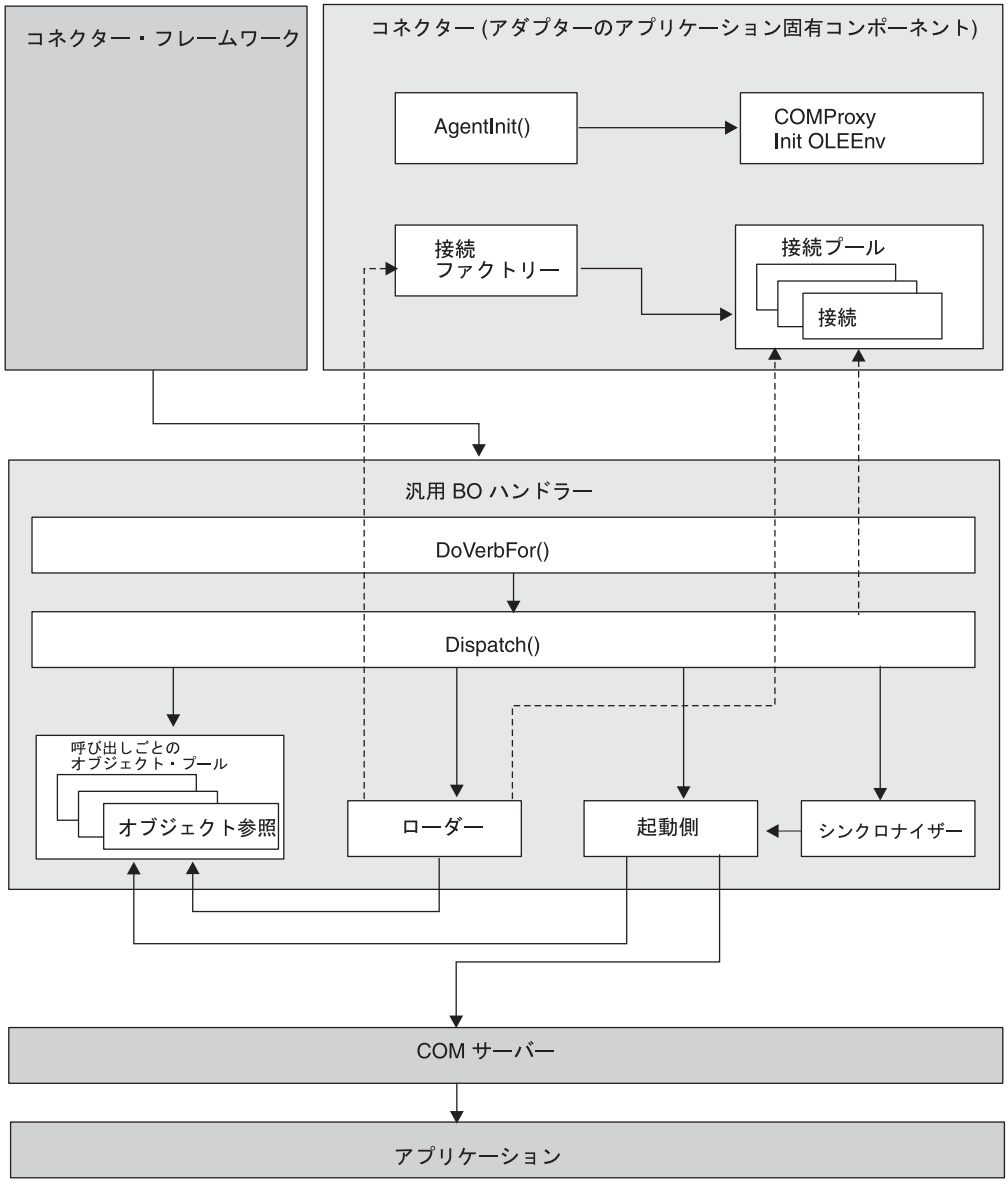


図 2. Connector for COM

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・プロパティー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・プロパティー、および以下の各シナリオでコネクター・プロパティー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ー』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` プロパティー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 クラス表記を使用して修飾されているもの) 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。
   - メソッドが通常のメソッドである場合は、起動側を呼び出します。引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

プロパティー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・プロパティーが設定されます)。

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、起動側は、以下を実行して、メソッドのパラメーターおよび引数を構成します。
   - 起動側は、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) の各プロパティーに値が保管 (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、34 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 Express 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、起動側、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 (7 ページ) および 7 (7 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.getAttrASIHashtable(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、コネクターで DCOM がサポートされます。リモート・サーバー名は、コネクター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、コネクターは、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

コネクターは国際化され、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、コネクターがデータを転送するとき、コネクター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。WebSphere Business Integration Server Express 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』
- 『コネクター・ファイルの構造』

本章では、コネクター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、InterChange Server Express をインストールし、統合ブローカーを始動するタスクについては、ブローカーに関する資料で説明しています。

Adapter for COM と関連ファイルのインストール

WebSphere Business Integration Server Express アダプター製品のインストールについては、「*WebSphere Business Integration Server Express インストール・ガイド (Windows 版)*」を参照してください。このガイドは、WebSphere Business Integration Server Express Adapters Infocenter (http://www.ibm.com/websphere/wbiserverexpress/infocenter) にあります。

コネクター・ファイルの構造

インストーラーは、コネクター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、コネクターを *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ーへのショートカットを追加します。*ProductDir* は、コネクターのインストール先ディレクトリーを指します。

16 ページの表 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¥start_COM_service.bat	コネクター・サービスの始動スクリプト。
¥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_COMProxy.dll	COM コンポーネントを起動するために使用される Java プロキシーのプロパティー、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_COMProxyGen.exe	プロキシー・クラスのプロキシー・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ー・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ー定義。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

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・オブジェクトのエレメントの「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 Express 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ーには、標準構成プロパティーとアダプター固有の構成プロパティーという 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 Express を使用して、これらのプロパティーの値を設定する必要があります。詳細については、 81 ページの『付録 B. Connector Configurator Express』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・プロパティー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ーへの変更は、即時に有効になります。他のコネクター・プロパティーを変更する場合は、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。プロパティーが

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) であるかを判別するには、System Manager の「コネクターのプロパティー」ウィンドウにある「更新メソッド」列を参照します。

標準コネクター・プロパティー

標準コネクター構成プロパティーでは、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ーの詳細については、 59 ページの『付録 A. コネクターの標準構成プロパティー』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ーについて、このコネクター固有の情報を以下の表に示します。

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

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

コネクター固有のプロパティー

コネクター固有の構成プロパティーは、コネクター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ーを使用すれば、コネクター内の静的情報やロジックを、再コーティングや再ビルドせずに変更できるようになります。

コネクター固有のプロパティーを構成するには、Connector Configurator Express を使用します。「**アプリケーション構成プロパティー**」タブをクリックして、構成プロパティーを追加または変更します。詳細については、 81 ページの『付録 B. Connector Configurator Express』を参照してください。

コネクター固有のプロパティーはすべてオプションであり、ユーザー固有のコネクター構成要件を基にした設定が可能です。コネクターは、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、コネクターのコネクター固有構成プロパティーと、その説明および指定可能な値を示します。**+** 文字は、プロパティー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、プロパティーの階層関係など、プロパティー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. コネクター固有の構成プロパティー

名前	指定可能な値	デフォルト値
+ Factory	なし	なし

表 2. コネクター固有の構成プロパティー (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、コネクター固有プロパティーの階層関係を示します。

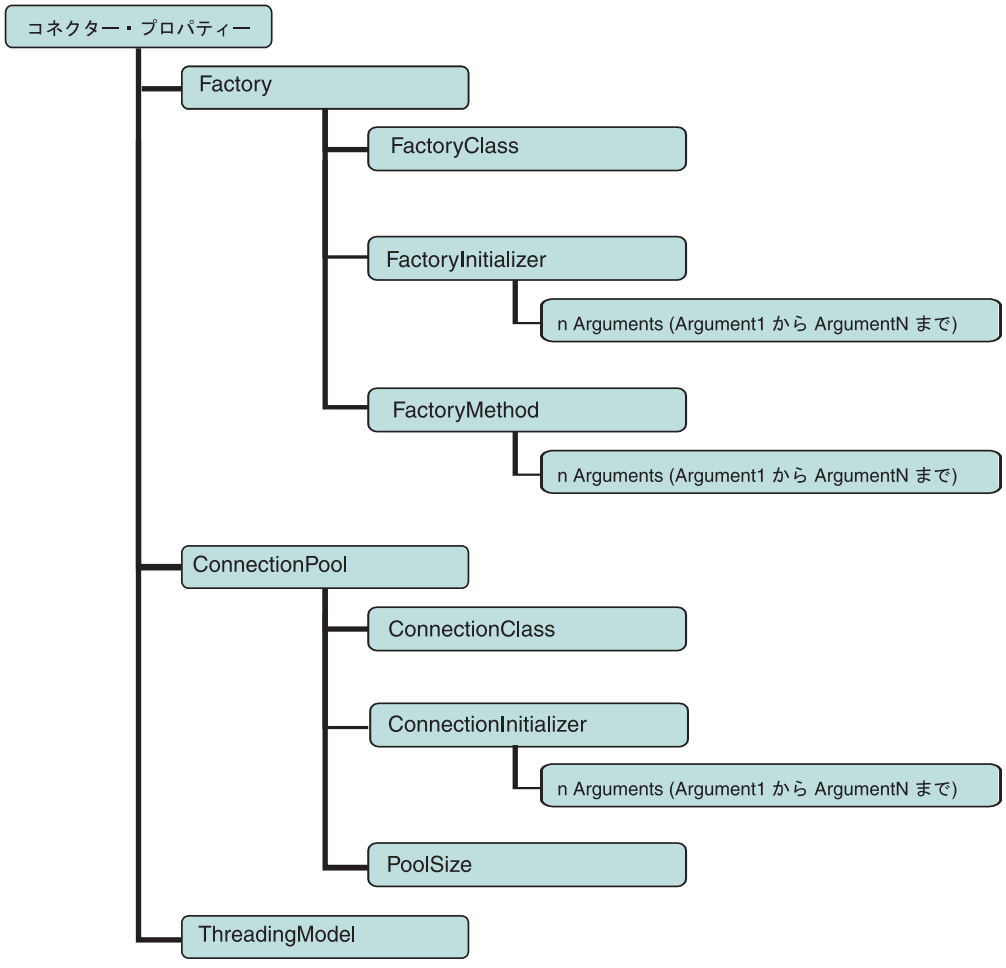


図 3. コネクター固有プロパティーの階層

Factory

ファクトリー・クラスの情報を表す階層プロパティー。

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、コネクターはファクトリー・プロキシー・オブジェクトと接続のインスタンスを生成します (コネクターの動作方法、ステップ 1 (6 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、コネクターの初期化時に作成されます (ステップ 1 (6 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、コネクターは、ファクトリー・プロキシー・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 (6 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ストリング値。この値は、暗号化ストリングの場合もあれば、非暗号化ストリングの場合もあります。

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。プロパティー名には、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。各引数の値は、暗号化ストリングまたは非暗号化ストリングになります。

ConnectionPool

接続クラスの情報を表すためのプロパティー。

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、コネクターはファクトリー・プロキシー・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 (6 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 (6 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 プロパティー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ストリング値。この値は、暗号化ストリングの場合もあれば、非暗号化ストリングの場合もあります。

PoolSize

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

ThreadingModel

コネクター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。このプロパティーを Apartment に設定すると、コネクターは 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、コネクターは単一スレッドになります。

このプロパティーを Free に設定すると、コネクターは 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: コネクターを Single-Threaded Apartment モード (プロパティー値を 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。

複数のコネクター・インスタンスの作成

コネクター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・コネクターの作成と同じです。以下に示すステップを実行することによって、コネクター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。

- コネクター・インスタンス用に新規ディレクトリーを作成します。
- 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。
- 新規コネクター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。

新規ディレクトリーの作成

新規ディレクトリーの作成: ProductDir¥connectors¥connectorInstance.

コネクターに、コネクター固有のメタオブジェクトがある場合、コネクター・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、次のディレクトリーを作成して、ファイルをそこに格納します。

ProductDir¥repository¥connectorInstance

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

InterChange Server Express サーバー名を startup.bat のパラメーターとして指定できます。例えば、start_COM.bat connName serverName などです。

ビジネス・オブジェクト定義の作成

各コネクター・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. 初期コネクター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 Express を使用してそれらのファイルをインポートします。初期コネクター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. 初期コネクターのファイルは、次のディレクトリーに入っていなければなりません。

   ```
   ProductDir¥repository¥initialConnectorInstance
   ```

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

コネクター定義の作成

Connector Configurator Express 内で、コネクター・インスタンスの構成ファイル (コネクター定義) 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. 初期コネクターの構成ファイル (コネクター定義) をコピーし、名前変更します。
2. 各コネクター・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、コネクター・プロパティー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『ビジネス・オブジェクト定義の作成』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。
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。

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。Windows システムでは、始動スクリプトはコネクターのランタイム・ディレクトリー (*ProductDir*¥connectors¥*connName*)にある必要があります。ここで、*connName* はコネクターを示します。

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
Windows	start_*connName*.bat

始動スクリプトを実行すると、デフォルトでは、*Productdir* にある構成ファイルが検索されます (以下のコマンドを参照)。ここは、ご使用の構成ファイルを格納する場所です。

注: アダプターが JMS トランスポートを使用している場合は、ローカル構成ファイルが必要です。

- **コネクターの始動:**
  - 「**開始**」メニューから、「**プログラム**」>「**IBM WebSphere Business Integration Server Express**」>「**アダプター**」>「**コネクター**」を選択します。デフォルトでは、プログラム名は「IBM WebSphere Business Integration Server Express」です。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、ご使用のコネクターへのデスクトップ・ショートカットを作成することもできます。
  - Windows コマンド行で次を入力: start_connName connName brokerName {-cconfigFile}.
  - 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、コネクターが始動します。

コネクターの停止

コネクターを停止する方法は、コネクターが始動された方法によって異なります。

- 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。
- 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 Express で構成してください。このツールの詳細については、81 ページの『付録 B. Connector Configurator Express』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 Express 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

Connector for COM は、メタデータ主導型です。WebSphere Business Integration Server Express システムでは、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 Express 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 Express 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、属性プロパティー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、「*ビジネス・オブジェクト開発ガイド*」を参照してください。

コネクター・ビジネス・オブジェクトの構造

コネクターは、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 コネクター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 でプロキシー・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、フラットなものと階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 コンテナー・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 コンテナー・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、常に引数のリストの末尾になります。

COM ID 名で使用される非英数字文字は下線 (_) のみです。非英数字文字は、対応するビジネス・オブジェクトで下線に解決されます。ただし、WebSphere Business Integration Server Express 形式では COM ID 名の先頭に下線を使用できないため、先頭に下線がある場合は、ODA により下線が文字列 `BBB_` に解決されます。

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、プロキシー・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、26 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
`proxy_class=` *`<nameOfProxyClass>`*	ビジネス・オブジェクトが表すプロキシー・クラスの名前。この ASI 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。この値は、有効な Java パッケージ表記 (`java.lang.Vector` など) を使用して指定する必要があります。
`auto_load_or_write=true`	ビジネス・オブジェクトが、引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、まずプロキシー・オブジェクトへの書き込み (WriteToProxy) を行ってから子オブジェクト内の引数に対応する引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ー・オブジェクトからの読み取り (LoadFromProxy) 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。

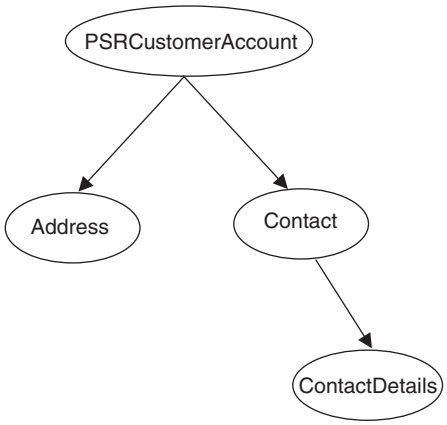


図 4. ビジネス・オブジェクト階層と動詞 ASI

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。

コネクター開発者が、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ー・オブジェクト属性の getter メソッドを呼び出して、プロキシー・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ー・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ー・オブジェクトに書き込みます。
LoadFromProxy (属性名なし)	プロキシー・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 からプロキシー・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詳しくは、9 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、コネクター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ー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、32 ページの表 9 に示します。

30 ページの表 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 プロキシーとビジネス・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、 32 ページの表 9 を参照してください。
MaxLength	使用しません。
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・プロパティーの設定を true にして指定します。この属性を使用するのは、コネクターではなく Business Object Designer Express 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、コネクターが確認しなければならないことを示します (ステップ 7 (7 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。 property=<*propertyName*>, type=<*typeName*> property は、COM オブジェクトのプロパティーの名前です。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ー名の取得に使用されます。 32 ページの表 9 を参照してください。 type は、単純属性の Java タイプです。 Java プロキシーとビジネス・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、 32 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	子がコンテナー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。

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
Is Required	使用しません。
AppSpecificInfo	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。 property=*propertyName*, use_attribute_value=<*(optional)BOName.AttributeName*>, type=<typeName> property は、COM オブジェクトのプロパティーの名前です。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ー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、そのような引数は、名前を持たない、いずれかの標準的な型の引数に過ぎないからです。 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 Express を使用して設定します。 type は、プロパティーの 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、 32 ページの表 9 を参照してください。
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	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。 method_name=<*nameOfMethod*>
Cardinality	1 を設定します。

メソッドには引数と戻り値があることに注意してください。引数と戻り値は、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、リストにして示します。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 プロキシー・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、 27 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、コネクター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	float	Float	VT_R4	Float	`float`
Float*	float[]	Float[]	VT_R4 | VT_BYREF	Float	`float_refererence`
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	boolean	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	byte	Byte	VT_UI1	Integer	byte
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、 32 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・プロパティー例

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。

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

コネクター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ー・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、 34 ページの『コネクター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。

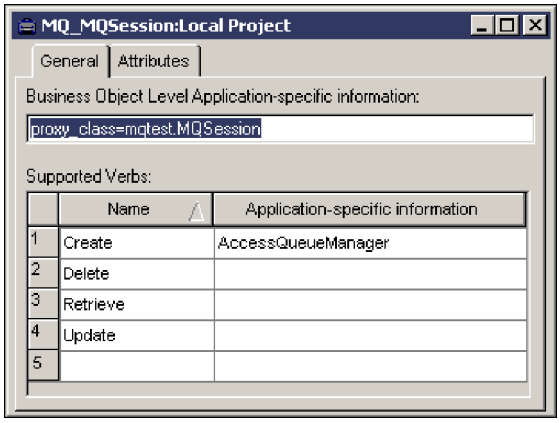


図 5. ビジネス・オブジェクト・レベルの ASI とサポートされる動詞

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 プロキシー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ー・オブジェクトを示しています。proxy_class ASI の詳細については、 27 ページの表 4 を参照してください。

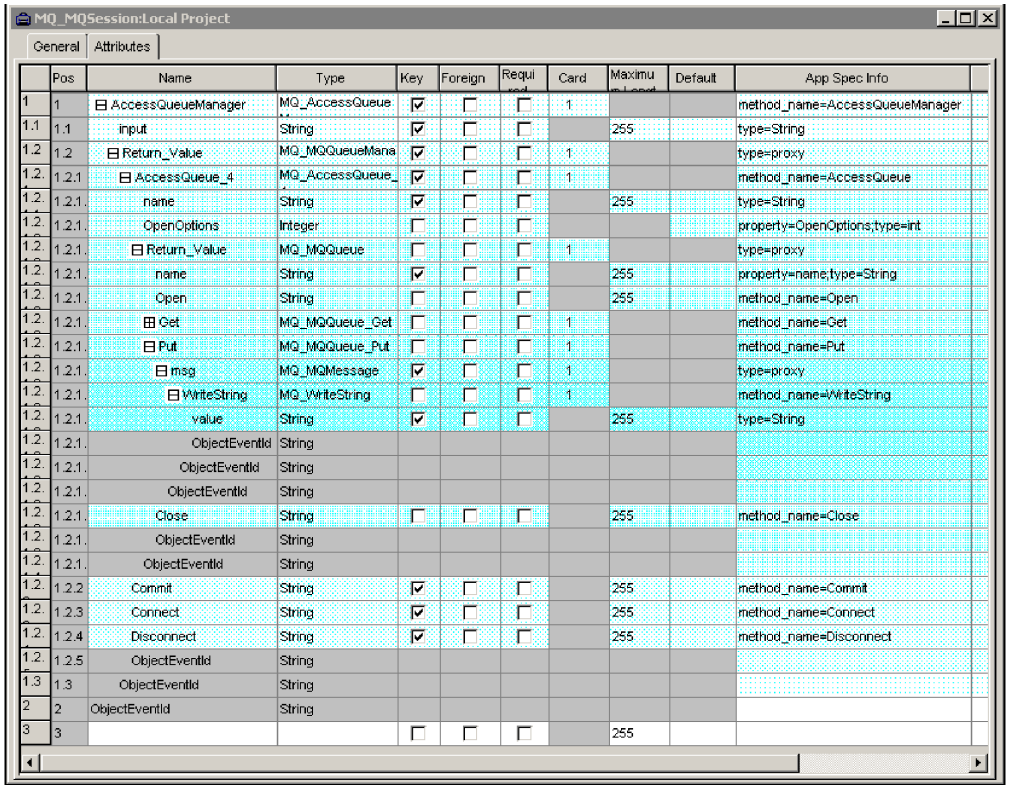


図 6. MQSession オブジェクトのビジネス・オブジェクト階層

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 ポインターを戻すメソッドです。このポインターは、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。

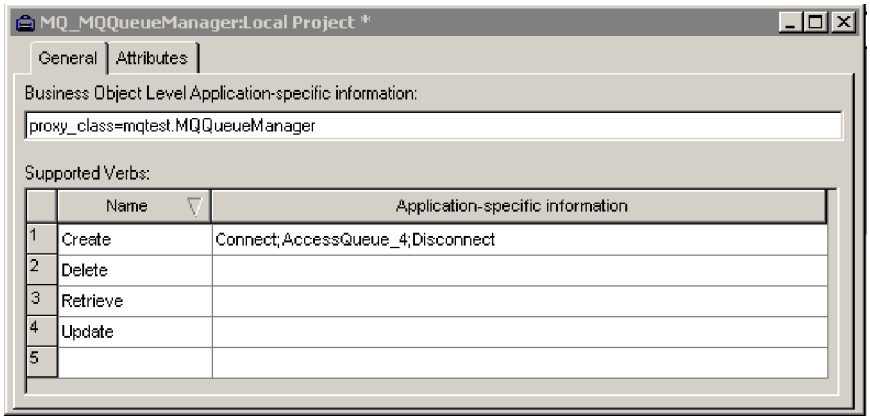


図 7. MQ_MQQueueManager のメソッド呼び出しシーケンス

図 7 では、ビジネス・オブジェクト MQ_MQQueueManager の動詞 Create のメソッド・シーケンスを示します。

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

36 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 というプロキシー・オブジェクトを戻します。このとき、コネクターは、AccessQueue_4 を実行した後、戻されたプロキシー・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。

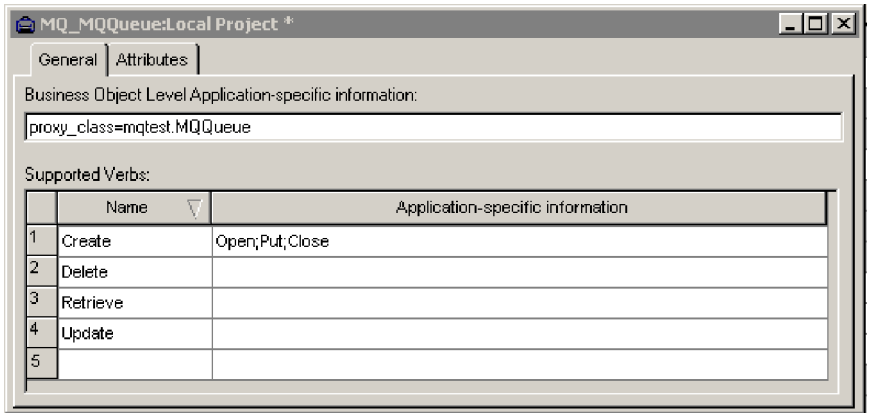


図 8. MQ_MQQueue のメソッド呼び出しシーケンス

図 8 は、AccessQueue_4 の子 (36 ページの図 6) である MQ_MQQueue のメソッド・シーケンスを示します。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 を引数とするので、コネクターは、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。

コネクター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 (36 ページの図 6) の親である MQ_MQQueueManager (37 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。

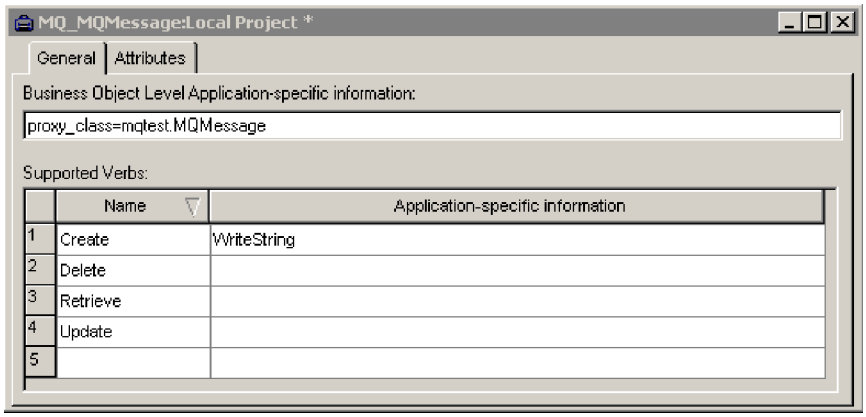


図 9. MQMessage のメソッド呼び出しシーケンス

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。コネクターでは、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。コネクターは、WriteString を呼び出した後、階層を「引き返し」、MQ_MQQueue (37 ページの図 8) の Close メソッドを実行し、MQ_MQQueueManager (37 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。コネクターは、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。コネクターは、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・リポジトリーのビジネス・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、コネクターでビジネス・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 Express 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

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・オブジェクトのエレメントの「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 Express 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。Object Discovery Agent によって、これらの定義の作成プロセスが自動化されます。ODA を使用して、ビジネス・オブジェクトおよび Connector Configurator Express を作成し、それらをサポートするコネクターを構成します。Connector Configurator Express について詳しくは、81 ページの『付録 B. Connector Configurator Express』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 Express 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 Express の起動方法および使用方法について詳しくは、「*ビジネス・オブジェクト開発ガイド*」を参照してください。

ODA の始動

ODA は、メタデータ・リポジトリー (タイプ・ライブラリー・ファイル) が存在するリポジトリー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 およびコネクター .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 コンパイラー (javac.exe) のディレクトリーが 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 がディレクトリー 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%
```

または、システム PATH 変数にストリング c:¥jdk131_02¥bin を追加します。

Business Object Designer Express の実行

Business Object Designer Express 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。

ウィザード画面の使用中は、どの時点でも、「**戻る**」をクリックして前の画面に戻ったり、「**次へ**」をクリックして次の画面に進んだり、「**キャンセル**」をクリックして現在の画面を取り消してウィザードを終了することができます。

ウィザードの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 Express 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. [hostname:port] の付いている AGENTNAME (start_COMODA スクリプトで指定) を「**検索されたエージェント**」リストから選択し、「**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。)

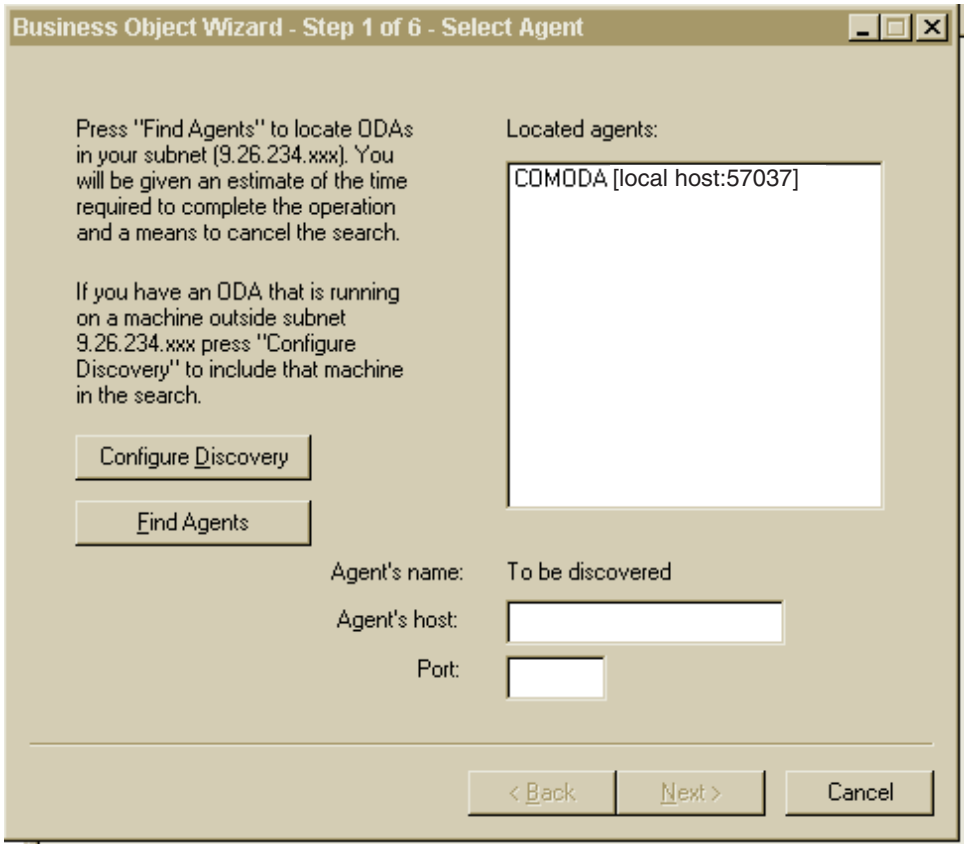


図 10. 「エージェントの選択」画面

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。

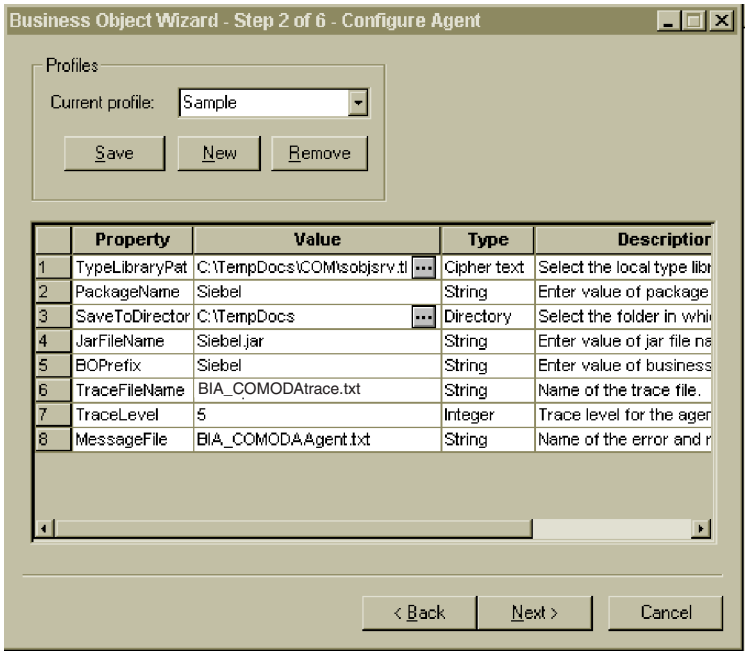


図 11. 「エージェントの構成」画面

表 10 に、この画面で設定するプロパティーを示します。

表 10. エージェント・プロパティーの構成

プロパティー名	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。ファイル・パス・リストを参照し、パス名を選択します。
PackageName	なし	String	(必須) COMProxy によって生成されるすべてのプロキシー・ファイルの保管先パッケージ。COMProxy は、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 プロキシー・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。ファイル・パス・リストを参照し、パス名を選択します。
JarFileName	なし	String	(必須) ODA によって生成されたプロキシー・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所は、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・プロパティーの構成 (続き)

プロパティー名	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。デフォルト値は BIA_COMODAtrace.txt です。
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 はエラー「メッセージ・ファイルが見つからないか、読み取れません (Cannot find or read message file)」を生成します。

「エージェントの構成」画面 (44 ページの図 11) 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、プロパティー・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。

1. 「エージェントの構成」画面 (44 ページの図 11) で、「プロファイル」グループ・ボックスの「**新規**」ボタンをクリックし、新規プロファイルを作成します。
2. 44 ページの表 10 で定義されているように、各プロパティーの値を入力します。

   注: 既存のプロファイルを使用する場合、プロパティー値はあらかじめ入力されていますが、必要に応じてその値を変更できます。
3. 「プロファイル」グループ・ボックスの「**保管**」ボタンをクリックし、新規または更新した値を保管します。

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。

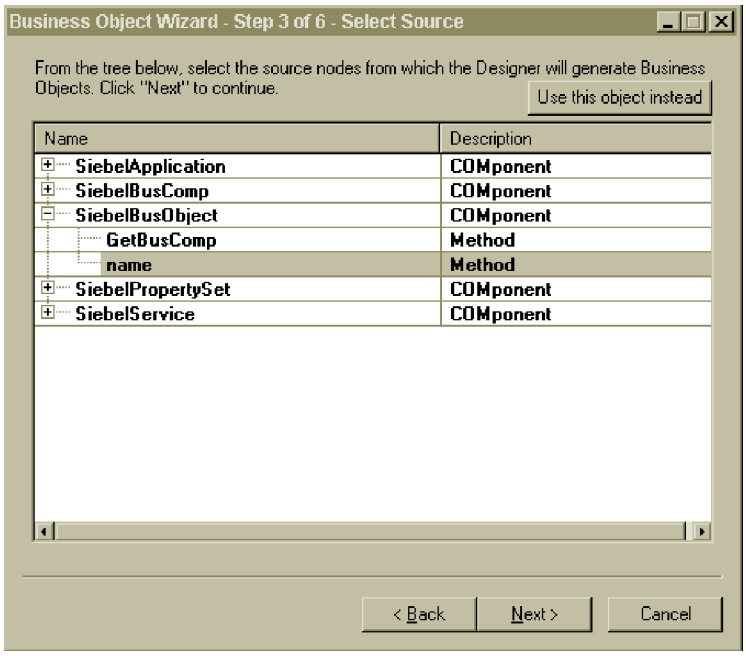


図 12. 「ソースの選択」画面

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**」をクリックして、ウィザードの次の画面に進みます。

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。

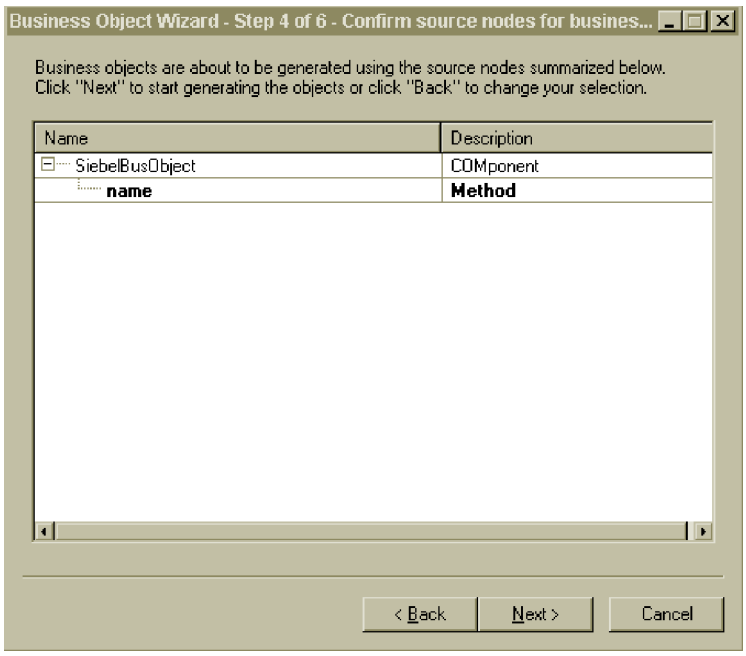


図 13. ソース・ノード画面の確認

「**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 プロパティー」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、32 ページの表 9 を参照してください。

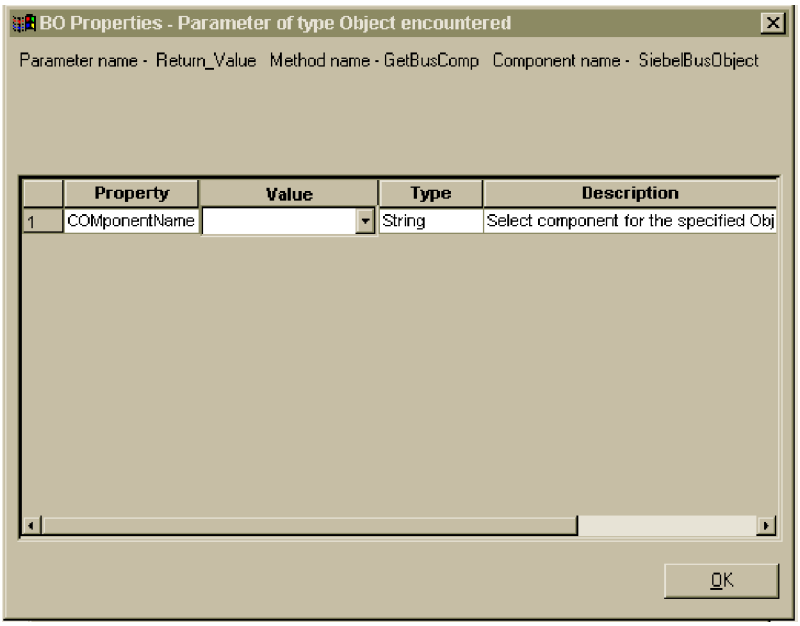


図 14. 「BO プロパティー」画面

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

オブジェクトの有効な動詞と、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 Express 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 コネクターのビジネス・オブジェクト構造について詳しくは、25 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

「BO プロパティー」画面 (図 14 参照) が完了したら、「BO プロパティー - コンポーネントの動詞を選択してください」画面が表示されます。49 ページの図 15 に、46 ページの図 12 で作成した `name` ビジネス・オブジェクトの場合のこの画面を示します。

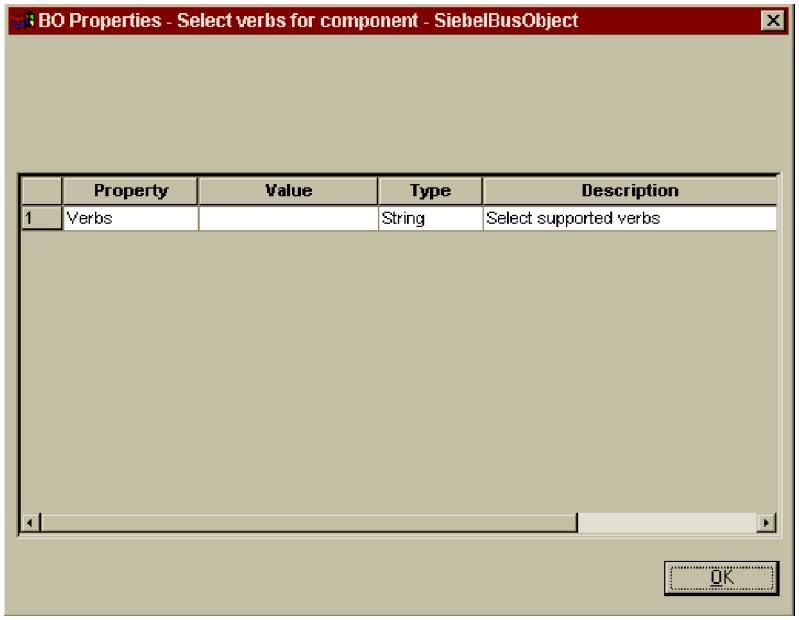
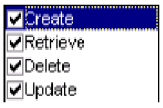


図 15. コンポーネントの動詞選択画面

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 を使用すると、サポートされる 4 つの動詞を指定できます (Create、Retrieve、Delete、Update)。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 Express を使用します。

COM コネクターのビジネス・オブジェクト動詞の詳細については、28 ページの『動詞 ASI』を参照してください。

1. Verbs プロパティーの「**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

ODA では、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、コネクターでサポートされるメソッド (`LoadFromProxy` および `WriteToProxy`) です。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。

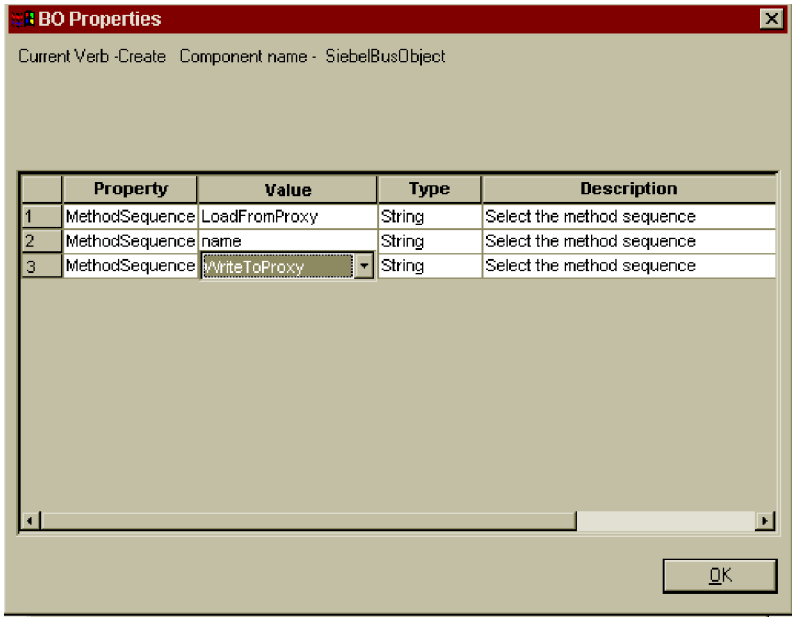


図 16. 動詞メソッド・シーケンスの設定

1. MethodSequence プロパティーの「**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 Express 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、 29 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。

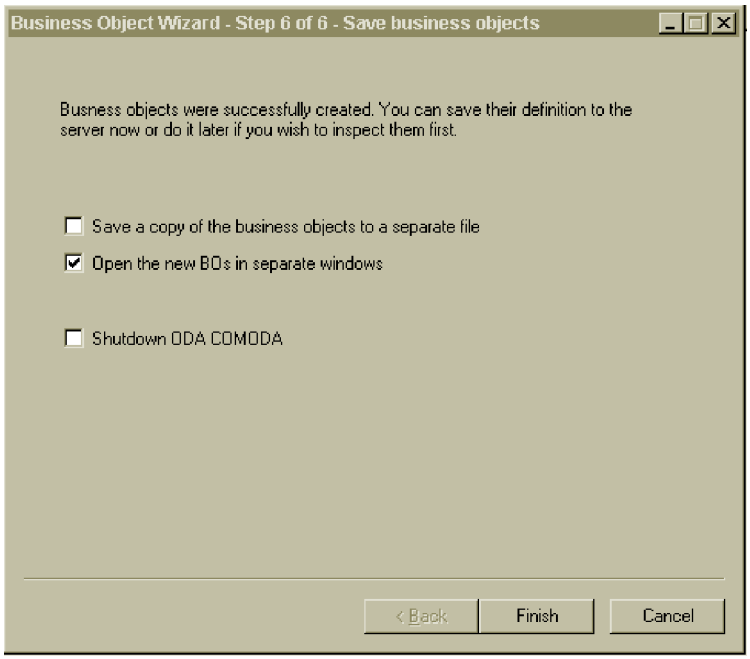


図 17. 「ビジネス・オブジェクトの保管」画面

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 Express 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するか、選択します。

Business Object Designer Express 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

Business Object Designer Express にはビジネス・オブジェクトの属性が表示されます (図 18 を参照)。COM コネクターの属性レベル ASI の詳細については、29 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。例えば、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。

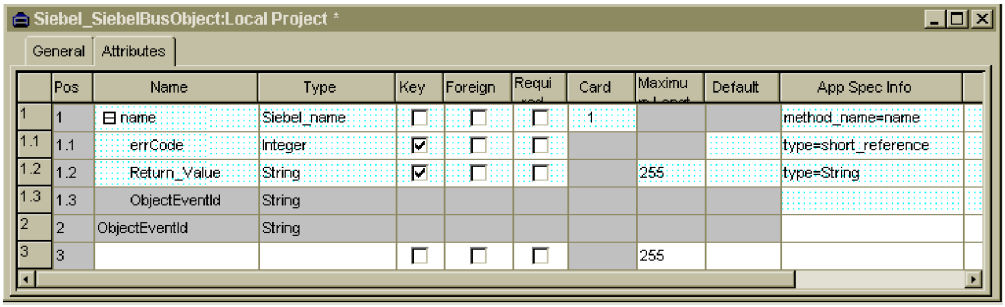


図 18. 属性 ASI の設定

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 Express 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 Express 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

ビジネス・オブジェクト・レベル ASI を表示および変更できます。ビジネス・オブジェクト・レベル ASI の詳細については、 27 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 ASI がブランクの場合、その動詞に対してメソッド・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)。

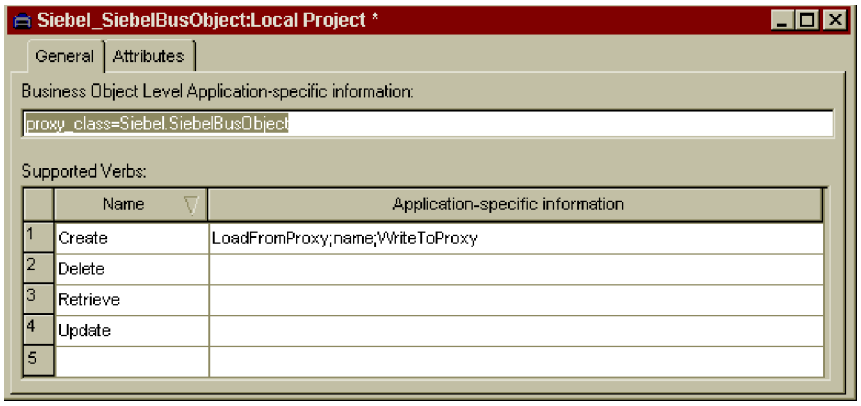


図 19. ビジネス・オブジェクト・レベル ASI の設定

この画面では、トップレベル・ビジネス・オブジェクトの ASI およびサポートされる動詞の ASI を変更できます。子オブジェクト・タイプの属性を含むビジネス・オブジェクトをオープンする場合は、「**一般**」タブにトップレベル・ビジネス・オブジェクトの ASI が表示され、変更することができます。子ビジネス・オブジェクトを別のウィンドウでオープンする場合は、その ASI は「**一般**」タブにトップレベル・オブジェクトの ASI として表示されます。

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。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ーにアップロードする必要がある場合、「システム・インプリメンテーション・ガイド」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

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ー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。

プロキシーおける ClassNotFound

ローダーがプロキシー・クラス名を受け取り、そのクラスのプロキシー・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ー・クラス名を受け取り、そのクラスのプロキシー・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 ClassNotFound

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ーで指定した 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。

ローダーまたは起動側における IllegaI 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、コネクターは例外を発生します。

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。

起動側における NoSuchMethodException

対応するプロキシー・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、コネクターは例外を発生します。

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。

起動側における 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、コネクターは例外を発生します。

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。

起動側におけるメソッド・オブジェクト内の無効な引数 (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、メソッドの引数が設定されていない場合、コネクターは例外を発生します。

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

プロキシー・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、コネクターは例外を発生します。

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。

コネクター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ー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詳しくは、17 ページの『コネクターの構成』のコネクター構成プロパティーを参照してください。

表 11 に、お勧めするコネクター・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	コネクター・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。 • 状況情報を提供します。 • 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。 • 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。
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。 • コネクター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。 • 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。 • 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。
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。 • 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、コネクターがビジネス・オブジェクト内で外部キーを検出したり、コネクターがビジネス・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。 • 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。

表 11. トレース・メッセージの内容 (続き)

レベル	説明
レベル 4	このトレース・メッセージ・レベルでは、以下を実行します。 • 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。 • 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。 • 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。
レベル 5	このトレース・メッセージ・レベルでは、以下を実行します。 • 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・コンフィギュレーター・プロパティーの値などがあります。 • 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。 • 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。 • 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、オブジェクト処理の開始前と完了後にビジネス・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)。

付録 A. コネクターの標準構成プロパティー

この付録では、WebSphere Business Integration Server Express アダプターのコネクター・コンポーネントの標準構成プロパティーについて説明します。説明は、InterChange Server Express が対象となります。

このコネクターに固有のプロパティーについては、本書の該当するセクションを参照してください。

新規プロパティー

以下の標準プロパティーは、本リリースで追加されました。

- AdapterHelpName
- ControllerEventSequencing
- jms.ListenerConcurrency
- jms.TransportOptimized
- TivoliTransactionMonitorPerformance

標準コネクター・プロパティーの概要

コネクターには 2 つのタイプの構成プロパティーがあります。

- 標準構成プロパティー。フレームワークが使用します。
- アプリケーション固有またはコネクター固有の構成プロパティー。エージェントが使用します。

これらのプロパティーは、アダプターのフレームワークおよびエージェントの実行時の振る舞いを決定します。

このセクションでは、Connector Configurator Express の始動方法について説明し、すべてのプロパティーに共通する特性について説明します。コネクター固有の構成プロパティーの詳細については、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 Express の始動

コネクター・プロパティーの構成は Connector Configurator Express から行います。Connector Configurator Express には、System Manager からアクセスします。Connector Configurator Express の使用法の詳細については、本書の Connector Configurator Express に関するセクションを参照してください。

Connector Configurator Express と System Manager は、Windows システム上でのみ動作します。

構成プロパティー値の概要

コネクターは、以下の順序に従ってプロパティーの値を決定します。

1. デフォルト
2. InterChange Server Express 統合ブローカー用のリポジトリー
3. ローカル構成ファイル
4. コマンド行

プロパティー・フィールドのデフォルトの長さは 255 文字です。STRING プロパティー・タイプの長さに制限はありません。INTEGER タイプの長さは、アダプターを実行しているサーバーによって決まります。

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・プロパティーの値を変更する場合は、プロパティー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。

プロパティーの更新特性 (すなわちコネクター・プロパティーへの変更を有効にする方法とタイミング) は、プロパティーの性質によって異なります。

標準コネクター・プロパティーには、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、新規の値が即時に有効になります。ただし、コネクターがスタンドアロン・モードの場合 (System Manager に依存しない) です。
- **エージェント再始動 (InterChange Server Express のみ)**
  コネクター・エージェントを停止して再始動しなければ、新規の値が有効になりません。
- **コンポーネント再始動**
  System Manager でコネクターを停止してから再始動しなければ、新規の値が有効になりません。エージェントまたはサーバー・プロセスを停止して再始動する必要はありません。
- **システム再始動**
  コネクター・エージェントおよびサーバーを停止して再始動しなければ、新規の値が有効になりません。

特定のプロパティーの更新方法を確認するには、「Connector Configurator Express」ウィンドウ内の「**更新メソッド**」列を参照するか、61 ページの表 12 の「更新メソッド」列を参照してください。

標準プロパティーが存在できる場所が 3 箇所あります。一部のプロパティーは複数の場所にあってもかまいません。

- **ReposController**
  このプロパティーはコネクター・コントローラー内にあり、その場所でのみ有効です。エージェント・サイドで値を変更した場合、コントローラーには影響しません。

- **ReposAgent**
  このプロパティーはエージェント内にあり、その場所でのみ有効です。プロパティーによっては、ローカル構成によってこの値をオーバーライドされることがあります。
- **LocalConfig**
  このプロパティーは、コネクターの構成ファイル内にあり、構成ファイルを通じてのみ機能することができます。コントローラーはこのプロパティーの値を変更することができず、システムが再配置されてコントローラーが明示的に更新されなければ、構成ファイルに加えられた変更を認識しません。

標準プロパティーの早見表

表 12 は、標準コネクター構成プロパティーの早見表です。すべてのコネクターでこれらのプロパティーすべてを必要とするわけではなく、プロパティー設定は異なる場合があります。

各プロパティーの説明については、表の次のセクションを参照してください。

注: 表 12 の注の欄で、「RepositoryDirectory が <REMOTE> に設定され」という句は、ブローカーが InterChange Server Express であることを示します。

表 12. 標準構成プロパティーの要約

プロパティー名	指定可能な値	デフォルト値	更新メソッド	注
AdapterHelpName	有効な <Regional Setting> ディレクトリーを含む <ProductDir>¥bin¥Data¥App¥Help 内の有効なサブディレクトリーのいずれか	テンプレート名 (有効な場合) またはブランク・フィールド	コンポーネント再始動	サポートされる地域設定。chs_chn、cht_twn、deu_deu、esn_esp、fra_fra、ita_ita、jpn_jpn、kor_kor、ptb_bra、および enu_usa (デフォルト) を含む。
AdminInQueue	有効な JMS キュー名	*<CONNECTORNAME>*/ADMININQUEUE	コンポーネント再始動	このプロパティーは、DeliveryTransport の値が JMS の場合のみ有効です。
AdminOutQueue	有効な JMS キュー名	*<CONNECTORNAME>*/ADMINOUTQUEUE	コンポーネント再始動	このプロパティーは、DeliveryTransport の値が JMS の場合のみ有効です。
AgentConnections	1 から 4	1	コンポーネント再始動	このプロパティーは、DeliveryTransport の値が MQ または IDL で、RepositoryDirectory の値が <REMOTE> に設定され、BrokerType の値が ICS の場合のみ有効です。
AgentTraceLevel	0 から 5	0	ICS では動的、その他の場合はコンポーネント再始動	
ApplicationName	アプリケーション名	コネクターのアプリケーション名に指定された値	コンポーネント再始動	

表 12. 標準構成プロパティーの要約 (続き)

プロパティー名	指定可能な値	デフォルト値	更新メソッド	注
CharacterEncoding	サポートされる任意のコード。次のリストはその一部です。 ascii7、ascii8、SJIS、Cp949、GBK、Big5、Cp297、Cp273、Cp280、Cp284、Cp037、Cp437 。	ascii7	コンポーネント再始動	このプロパティーは、C++ コネクターでのみ有効です。
CommonEventInfrastructure	true または false	false	コンポーネント再始動	
CommonEventInfrastructureURL	URL ストリング。例えば、corbaloc:iiop:host:2809。	デフォルト値はありません。	コンポーネント再始動	このプロパティーは、CommonEvent Infrastructure の値が true の場合のみ有効です。
ConcurrentEventTriggeredFlows	1 から 32,767	1	コンポーネント再始動	このプロパティーは、RepositoryDirectory の値が <REMOTE> に設定されて、BrokerType の値が ICS の場合のみ有効です。
ContainerManagedEvents	ブランクまたは JMS	ブランク	コンポーネント再始動	このプロパティーは、DeliveryTransport の値が JMS の場合のみ有効です。
ControllerEventSequencing	true または false	true	動的	このプロパティーは、RepositoryDirectory の値が <REMOTE> に設定され、BrokerType の値が ICS の場合のみ有効です。
ControllerStoreAndForwardMode	true または false	true	動的	このプロパティーは、RepositoryDirectory の値が <REMOTE> に設定され、BrokerType の値が ICS の場合のみ有効です。
ControllerTraceLevel	0 から 5	0	動的	このプロパティーは、RepositoryDirectory の値が <REMOTE> に設定されて、BrokerType の値が ICS の場合のみ有効です。
DeliveryQueue	任意の有効な JMS キュー名	*<CONNECTORNAME>*/DELIVERYQUEUE	コンポーネント再始動	このプロパティーは、DeliveryTransport の値が JMS の場合のみ有効です。
DeliveryTransport	IDL または JMS	RepositoryDirectory の値が <REMOTE> の場合は IDL。それ以外の場合は JMS。	コンポーネント再始動	RepositoryDirectory の値が <REMOTE> ではない場合、このプロパティーの有効な値は JMS のみです。
DuplicateEventElimination	true または false	false	コンポーネント再始動	このプロパティーは、DeliveryTransport の値が JMS の場合のみ有効です。
EnableOidForFlowMonitoring	true または false	false	コンポーネント再始動	このプロパティーは、BrokerType の値が ICS の場合のみ有効です。
FaultQueue	任意の有効なキュー名	*<CONNECTORNAME>*/FAULTQUEUE	コンポーネント再始動	このプロパティーは、DeliveryTransport の値が JMS の場合のみ有効です。

表 12. 標準構成プロパティーの要約 (続き)

プロパティー名	指定可能な値	デフォルト値	更新メソッド	注
jms.FactoryClassName	CxCommon.Messaging.jms .IBMMQSeriesFactory、 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	このプロパティーは、DeliveryTransport の値が JMS の場合のみ有効です。
jms.ListenerConcurrency	1 から 32767	1	コンポーネント再始動	このプロパティーは、jms.TransportOptimized の値が true の場合のみ有効です。
jms.MessageBrokerName	jms.FactoryClassName の値が IBM の場合は、crossworlds.queue.manager を使用します。	crossworlds.queue. manager	コンポーネント再始動	このプロパティーは、DeliveryTransport の値が JMS の場合のみ有効です。
jms.NumConcurrent Requests	正整数	10	コンポーネント再始動	このプロパティーは、DeliveryTransport の値が JMS の場合のみ有効です。
jms.Password	任意の有効なパスワード		コンポーネント再始動	このプロパティーは、DeliveryTransport の値が JMS の場合のみ有効です。
jms.TransportOptimized	true または false	false	コンポーネント再始動	このプロパティーは、DeliveryTransport の値が JMS で、BrokerType の値が ICS の場合のみ有効です。
jms.UserName	任意の有効な名前		コンポーネント再始動	このプロパティーは、DeliveryTransport の値が JMS の場合のみ有効です。
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	このプロパティーは、RepositoryDirectory の値が <REMOTE> に設定され、BrokerType の値が ICS の場合のみ有効です。
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	このプロパティーは、RepositoryDirectory の値が <REMOTE> に設定され、BrokerType の値が ICS の場合のみ有効です。
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	このプロパティーは、RepositoryDirectory の値が <REMOTE> に設定され、BrokerType の値が ICS の場合のみ有効です。
ListenerConcurrency	1 から 100	1	コンポーネント再始動	このプロパティーは、DeliveryTransport の値が MQ の場合のみ有効です。
Locale	これは、サポートされるロケールの一部です。 en_US、ja_JP、ko_KR、 zh_CN、zh_TW、fr_FR、 de_DE、it_IT、es_ES、 pt_BR	en_US	コンポーネント再始動	

表 12. 標準構成プロパティーの要約 (続き)

プロパティー名	指定可能な値	デフォルト値	更新メソッド	注
LogAtInterchangeEnd	`true` または `false`	`false`	コンポーネント再始動	このプロパティーは、RepositoryDirectory の値が <REMOTE> に設定され、BrokerType の値が ICS の場合のみ有効です。
MaxEventCapacity	1 から `2147483647`	`2147483647`	動的	このプロパティーは、RepositoryDirectory の値が <REMOTE> に設定され、BrokerType の値が ICS の場合のみ有効です。
MessageFileName	有効なファイル名	`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	このプロパティーは、DuplicateEventElimination の値が `true` で、ContainerManagedEvents に値がない場合のみ有効です。
OADAutoRestartAgent	`true` または `false`	`false`	動的	このプロパティーは、RepositoryDirectory の値が <REMOTE> に設定され、BrokerType の値が ICS の場合のみ有効です。
OADMaxNumRetry	正整数	`1000`	動的	このプロパティーは、RepositoryDirectory の値が <REMOTE> に設定され、BrokerType の値が ICS の場合のみ有効です。
OADRetryTimeInterval	正整数 (単位: 分)	`10`	動的	このプロパティーは、RepositoryDirectory の値が <REMOTE> に設定され、BrokerType の値が ICS の場合のみ有効です。
PollEndTime	HH = 0 から 23 MM = 0 から 59	`HH:MM`	コンポーネント再始動	
PollFrequency	正整数 (単位: ミリ秒)	`10000`	ブローカーが ICS の場合は動的。そうでない場合は、コンポーネント再始動。	
PollQuantity	1 から `500`	`1`	エージェント再始動	このプロパティーは、ContainerManagedEvents の値が `JMS` の場合のみ有効です。
PollStartTime	HH = 0 から 23 MM = 0 から 59	`HH:MM`	コンポーネント再始動	
RepositoryDirectory	ブローカーが ICS の場合は <REMOTE>。それ以外の場合は任意の有効なローカル・ディレクトリー。	ICS の場合、値は <REMOTE> に設定されます。	エージェント再始動	

表 12. 標準構成プロパティーの要約 (続き)

プロパティー名	指定可能な値	デフォルト値	更新メソッド	注
RequestQueue	有効な JMS キュー名	*<CONNECTORNAME>* /REQUESTQUEUE	コンポーネント再始動	このプロパティーは、DeliveryTransport の値が JMS の場合のみ有効です。
ResponseQueue	有効な JMS キュー名	*<CONNECTORNAME>* /RESPONSEQUEUE	コンポーネント再始動	このプロパティーは、DeliveryTransport の値が JMS の場合のみ有効です。
RestartRetryCount	0 から 99	3	ICS の場合は動的、その他の場合はコンポーネント再始動	
RestartRetryInterval	1 から 2147483647 までの値 (分単位)。	1	ICS の場合は動的、その他の場合はコンポーネント再始動	
RHF2MessageDomain	mrm または xml	mrm	コンポーネント再始動	このプロパティーは、DeliveryTransport の値が JMS で、WireFormat の値が CwXML の場合のみ有効です。
SourceQueue	任意の有効な WebSphere MQ キュー名	*<CONNECTORNAME>* /SOURCEQUEUE	エージェント再始動	このプロパティーは、ContainerManagedEvents の値が JMS の場合のみ有効です。
SynchronousRequest Queue	任意の有効なキュー名	*<CONNECTORNAME>* /SYNCHRONOUSREQUEST QUEUE	コンポーネント再始動	このプロパティーは、DeliveryTransport の値が JMS の場合のみ有効です。
SynchronousRequest Timeout	0 から任意の数 (ミリ秒)	0	コンポーネント再始動	このプロパティーは、DeliveryTransport の値が JMS の場合のみ有効です。
SynchronousResponse Queue	任意の有効なキュー名	*<CONNECTORNAME>* /SYNCHRONOUSRESPONSE QUEUE	コンポーネント再始動	このプロパティーは、DeliveryTransport の値が JMS の場合のみ有効です。
TivoliMonitorTransaction Performance	true または false	false	コンポーネント再始動	
WireFormat	CwXML または CwBO	CwXML	エージェント再始動	RepositoryDirectory の値が <REMOTE> に設定されていない場合、このプロパティーの値は、CwXML でなければなりません。RepositoryDirectory の値が <REMOTE> に設定されている場合、値は CwBO でなければなりません。
WsifSynchronousRequest Timeout	0 から任意の数 (ミリ秒)	0	コンポーネント再始動	BrokerType の値が ICS の場合、このプロパティーは無効です。
XMLNameSpaceFormat	short または long	short	エージェント再始動	BrokerType の値が ICS の場合、このプロパティーは無効です。

標準プロパティー

このセクションでは、標準コネクター構成プロパティーについて説明します。

AdapterHelpName

AdapterHelpName プロパティーは、コネクター固有の全般ヘルプ・ファイルがあるディレクトリーの名前です。ディレクトリーは、*<ProductDir>*¥bin¥Data¥App¥Help 内に配置される必要があり、少なくとも言語ディレクトリー enu_usa が含まれていなければなりません。ロケールに応じて、その他のディレクトリーが含まれることがあります。

デフォルト値は、テンプレート名が有効であればテンプレート名、有効でなければブランクです。

AdminInQueue

AdminInQueue プロパティーは、統合ブローカーがコネクターへ管理メッセージを送信するときに使用するキューを指定します。

デフォルト値は *<CONNECTORNAME>*/ADMININQUEUE です。

AdminOutQueue

AdminOutQueue プロパティーは、コネクターが統合ブローカーへ管理メッセージを送信するときに使用するキューを指定します。

デフォルト値は *<CONNECTORNAME>*/ADMINOUTQUEUE です。

AgentConnections

AgentConnections プロパティーは、ORB (オブジェクト・リクエスト・ブローカー) が初期化するときに開かれる ORB 接続の数を制御します。

このプロパティーのデフォルト値は 1 です。

AgentTraceLevel

AgentTraceLevel プロパティーは、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を設定します。コネクターは、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。

デフォルト値は 0 です。

ApplicationName

ApplicationName プロパティーは、コネクター・アプリケーションの名前を一意的に識別します。この名前は、システム管理者が統合環境をモニターするために使用します。コネクターを実行する前に、このプロパティーに値を指定する必要があります。

デフォルトはコネクターの名前です。

BrokerType

BrokerType プロパティーは、使用している統合ブローカーのタイプを識別します。値は、ICS (InterChange Server Express) です。

CharacterEncoding

CharacterEncoding プロパティーは、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 ベースのコネクターでは、このプロパティーは使用しません。C++ ベースのコネクターでは、このプロパティーに ascii7 という値が使用されています。

デフォルトでは、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。サポートされる他の値をリストに追加するには、製品ディレクトリー (<*ProductDir*>) 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、本書の付録『Connector Configurator Express』を参照してください。

ConcurrentEventTriggeredFlows

ConcurrentEventTriggeredFlows プロパティーは、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されるビジネス・オブジェクトの数に設定します。例えば、このプロパティーの値を 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。

このプロパティーを 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、以下のプロパティーを構成する必要があります。

- Maximum number of concurrent events プロパティーの値を増加して、複数のスレッドを使用できるようにコラボレーションを構成する必要があります。
- 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を、複数の要求を並行して処理できるように構成する必要があります。

ConcurrentEventTriggeredFlows プロパティーは、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。

このプロパティーは、RepositoryDirectory プロパティーの値が <REMOTE> に設定されている場合のみ有効です。

デフォルト値は 1 です。

ContainerManagedEvents

ContainerManagedEvents プロパティーにより、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、1 つの JMS トランザクションとして宛先キューに配置されます。

このプロパティーを 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ーも設定する必要があります。

- PollQuantity = 1 から 500
- SourceQueue = /SOURCEQUEUE

また、MimeType および DHClass (データ・ハンドラー・クラス) 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。DataHandlerConfigMOName (オプションのメタオブジェクト名) を追加することもできます。これらのプロパティーの値を設定するには、Connector Configurator Express の「**データ・ハンドラー**」タブを使用します。

これらのプロパティーはアダプター固有ですが、以下に値の例をいくつか示します。

- MimeType = text¥xml
- DHClass = com.crossworlds.DataHandlers.text.xml
- DataHandlerConfigMOName = MO_DataHandler_Default

「**データ・ハンドラー**」 タブのこれらの値のフィールドは、ContainerManagedEvents プロパティーを JMS という値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。

ContainerManagedEvents プロパティーは、DeliveryTransport プロパティーの値が JMS に設定されている場合のみ有効です。

デフォルト値はありません。

ControllerEventSequencing

ControllerEventSequencing プロパティーは、コネクター・コントローラーでイベント順序付けを使用可能にします。

このプロパティーは、RepositoryDirectory プロパティーの値が <REMOTE> に設定されている場合 (BrokerType は ICS) のみ有効です。

デフォルト値は true です。

ControllerStoreAndForwardMode

ControllerStoreAndForwardMode プロパティーは、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コネクター・コントローラーが検出した後に、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 InterChange Server Express (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。

このプロパティーは、RepositoryDirectory プロパティーの値が <REMOTE> に設定されている場合 (BrokerType プロパティーの値が ICS) のみ有効です。

デフォルト値は `true` です。

ControllerTraceLevel

ControllerTraceLevel プロパティーは、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を設定します。

このプロパティーは、RepositoryDirectory プロパティーの値が <REMOTE> に設定されている場合のみ有効です。

デフォルト値は `0` です。

DeliveryQueue

DeliveryQueue プロパティーは、コネクターが統合ブローカーへビジネス・オブジェクトを送信するときに使用するキューを定義します。

このプロパティーは、DeliveryTransport プロパティーの値が `JMS` に設定されている場合のみ有効です。

デフォルト値は *<CONNECTORNAME>*`/DELIVERYQUEUE` です。

DeliveryTransport

DeliveryTransport プロパティーは、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。Java Messaging Service の場合、値は JMS です。

- RepositoryDirectory プロパティーの値が <REMOTE> に設定されている場合、DeliveryTransport プロパティーの値には `IDL` または `JMS` を指定することができ、デフォルトは `IDL` です。

- RepositoryDirectory プロパティーの値がローカル・ディレクトリーの場合、値に使用できるのは JMS のみです。

RepositoryDirectory プロパティーの値が IDL である場合、コネクターは、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。

デフォルト値は JMS です。

JMS

JMS トランスポート機構は、Java Messaging Service (JMS) を使用した、コネクターとクライアント・コネクター・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 プロパティーが Connector Configurator Express 内にリストされます。jms.MessageBrokerName プロパティーおよび jms.FactoryClassName プロパティーは、このトランスポートの必須プロパティーです。

InterChange Server Espress (ICS) が統合ブローカーである場合、以下の環境では、コネクターに JMS トランスポート機構を使用すると、メモリー制限が発生することもあります。

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 768 MB 未満である場合には、次の変数およびプロパティーを設定してください。

- 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 (<*ProductDir*>) の下の ¥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、「*WebSphere Business Integration Server Express インストール・ガイド (Windows 版)*」を参照してください。

DuplicateEventElimination

このプロパティーの値が true の場合、JMS 対応コネクターでは重複イベントがデリバリー・キューへデリバリーされないようにすることができます。この機能を使用するには、コネクター開発時に、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。

注: このプロパティーの値が true の場合、保証付きイベント・デリバリーを提供するには、MonitorQueue プロパティーを使用可能にする必要があります。

デフォルト値は false です。

EnableOidForFlowMonitoring

このプロパティーの値が true の場合、アダプター・ランタイムは、着信 ObjectEventID にフロー・モニターの外部キーのマークを付けます。

このプロパティーは、BrokerType プロパティーが ICS に設定されている場合のみ有効です。

デフォルト値は false です。

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 (および状況標識と問題説明) を FaultQueue プロパティーで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。

jms.FactoryClassName

jms.FactoryClassName プロパティーは、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。DeliveryTransport プロパティーの値が JMS に設定されている場合、このプロパティーを設定する必要があります。

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

jms.ListenerConcurrency

jms.ListenerConcurrency プロパティーは、JMS コントローラーの並行リスナーの数を指定します。コントローラー内部で、並行してメッセージを取り出して処理するスレッドの数を指定します。

このプロパティーは、jms.OptimizedTransport プロパティーの値が true の場合のみ有効です。

デフォルト値は 1 です。

jms.MessageBrokerName

jms.MessageBrokerName は、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として (DeliveryTransport プロパティーで) 指定する場合、このコネクター・プロパティーを設定する必要があります。

リモート・メッセージ・ブローカーに接続した場合、このプロパティーでは以下の値を指定する必要があります。
QueueMgrName:*Channel*:*HostName*:*PortNumber*
ここで、以下のように説明されます。
QueueMgrName は、キュー・マネージャー名です。

Channel は、クライアントが使用するチャネルです。
HostName は、キュー・マネージャーの配置先のマシン名です。
PortNumber は、キュー・マネージャーが listen に使用するポートの番号です。

例えば、次のようになります。
`jms.MessageBrokerName = WBIMB.Queue.Manager:CHANNEL1:RemoteMachine:1456`

デフォルト値は `crossworlds.queue.manager` です。ローカル・メッセージ・ブローカーに接続する場合は、デフォルト値を使用します。

jms.NumConcurrentRequests

jms.NumConcurrentRequests プロパティーは、コネクター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はブロックされ、処理を続行するには他のいずれかの要求が完了するのを待機する必要があります。

デフォルト値は `10` です。

jms.Password

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

デフォルト値はありません。

jms.TransportOptimized

jms.TransportOptimized プロパティーは、WIP (処理中の作業) が最適化されるかどうかを決定します。WIP を最適化するには、WebSphere MQ プロバイダーが必要です。最適化された WIP が作動するためには、メッセージング・プロバイダーが以下の操作を実行できなければなりません。

1. メッセージをキューから削除せずに読み取る。
2. メッセージ全体を受信側のメモリー空間に転送することなく、固有の ID を使用してメッセージを削除する。
3. 固有の ID を使用してメッセージを読み取る (リカバリーのために必要)。
4. 読み取られなかったイベントが現れるポイントを追跡する。

JMS API は、上記の条件 2 および 4 を満たさないため、最適化された WIP には使用できませんが、MQ Java API は 4 つの条件をすべて満たすため、最適化された WIP には必要です。

このプロパティーは、DeliveryTransport の値が `JMS` で、BrokerType の値が `ICS` の場合のみ有効です。

デフォルト値は `false` です。

jms.UserName

jms.UserName プロパティーは、JMS プロバイダーのユーザー名を指定します。このプロパティーの値はオプションです。

デフォルト値はありません。

JvmMaxHeapSize

JvmMaxHeapSize プロパティーは、エージェントの最大ヒープ・サイズ (メガバイト単位) を指定します。

このプロパティーは、RepositoryDirectory プロパティーの値が <REMOTE> に設定されている場合のみ有効です。

デフォルト値は 128M です。

JvmMaxNativeStackSize

JvmMaxNativeStackSize プロパティーは、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) を指定します。

このプロパティーは、RepositoryDirectory プロパティーの値が <REMOTE> に設定されている場合のみ有効です。

デフォルト値は 128K です。

JvmMinHeapSize

JvmMinHeapSize プロパティーは、エージェントの最小ヒープ・サイズ (メガバイト単位) を指定します。

このプロパティーは、RepositoryDirectory プロパティーの値が <REMOTE> に設定されている場合のみ有効です。

デフォルト値は 1M です。

ListenerConcurrency

ListenerConcurrency プロパティーは、統合ブローカーとして ICS を使用する場合の WebSphere MQ Listener でのマルチスレッド化をサポートしています。このプロパティーにより、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。

このプロパティーは、MQ トランスポートを使用するコネクターのみで有効です。DeliveryTransport プロパティーの値は `MQ` でなければなりません。

デフォルト値は 1 です。

Locale

Locale プロパティーは、言語コード、国または地域、および、オプションで関連した文字コード・セットを指定します。このプロパティーの値は、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。
ll は、2 文字の言語コード (小文字を使用) です。
TT は、2 文字の国または地域コード (大文字を使用) です。
codeset は、関連文字コード・セットの名前です (オプションの場合があります)。

デフォルトでは、サポートされるロケールの一部のみがリストされます。サポートされる他の値をリストに追加するには、<*ProductDir*>¥bin ディレクトリーにある ¥Data¥Std¥stdConnProps.xml ファイルを変更します。詳細については、本書の付録『Connector Configurator Express』を参照してください。

コネクターが国際化に対応していない場合、このプロパティーの有効な値は en_US のみです。特定のコネクターがグローバル化に対応しているかどうかを判別するには、そのアダプターのユーザーズ・ガイドを参照してください。

デフォルト値は en_US です。

LogAtInterchangeEnd

LogAtInterchangeEnd プロパティーは、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。

ログ宛先にログを記録すると、E 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 ファイルで MESSAGE_RECIPIENT の値として指定された宛先に対する E メール・メッセージが生成されます。例えば、LogAtInterChangeEnd の値を true に設定した場合にコネクター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、E メール・メッセージが送信されます。

このプロパティーは、RepositoryDirectory プロパティーの値が <REMOTE> に設定されている場合 (BrokerType の値が ICS) のみ有効です。

デフォルト値は false です。

MaxEventCapacity

MaxEventCapacity プロパティーは、コントローラー・バッファー内のイベントの最大数を指定します。このプロパティーは、フロー制御機能によって使用されます。

このプロパティーは、RepositoryDirectory プロパティーの値が <REMOTE> に設定されている場合 (BrokerType の値が ICS) のみ有効です。

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

MessageFileName プロパティーは、コネクター・メッセージ・ファイルの名前を指定します。メッセージ・ファイルの標準位置は、製品ディレクトリーの ¥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。

コネクター・メッセージ・ファイルが存在しない場合は、コネクターは 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: コネクターについて、コネクター独自のメッセージ・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ーズ・ガイドを参照してください。

デフォルト値は InterchangeSystem.txt です。

MonitorQueue

MonitorQueue プロパティーは、コネクター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を指定します。

このプロパティーは、DeliveryTransport プロパティーの値が JMS で、DuplicateEventElimination の値が true の場合のみ有効です。

デフォルト値は *<CONNECTORNAME>*/MONITORQUEUE です。

OADAutoRestartAgent

OADAutoRestartAgent プロパティーは、コネクター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、WebSphere 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ーを再始動したり、System Monitor からリモート・コネクター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、このプロパティーを true に設定する必要があります。WebSphere MQ-triggered OAD 機能の構成方法については、「*WebSphere Business Integration Server Express インストール・ガイド (Windows 版)*」を参照してください。

このプロパティーは、RepositoryDirectory プロパティーの値が <REMOTE> に設定されている場合 (BrokerType の値が ICS) のみ有効です。

デフォルト値は false です。

OADMaxNumRetry

OADMaxNumRetry プロパティーは、異常シャットダウンの後で WebSphere MQ によりトリガーされる Object Activation Daemon (OAD) がコネクター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。このプロパティーを有効にするには、OADAutoRestartAgent プロパティーを true に設定する必要があります。

このプロパティーは、RepositoryDirectory プロパティーの値が <REMOTE> に設定されている場合 (BrokerType の値が ICS) のみ有効です。

デフォルト値は 1000 です。

OADRetryTimeInterval

OADRetryTimeInterval プロパティーは、WebSphere MQ によりトリガーされる Object Activation Daemon (OAD) の再試行時間間隔の分数を指定します。コネクター・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、コネクター・コントローラーはコネクター・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 プロパティーで指定された回数だけ繰り返します。このプロパティーを有効にするには、OADAutoRestartAgent プロパティーを `true` に設定する必要があります。

このプロパティーは、RepositoryDirectory プロパティーの値が <REMOTE> に設定されている場合 (BrokerType の値が ICS) のみ有効です。

デフォルト値は `10` です。

PollEndTime

PollEndTime プロパティーは、イベント・キューのポーリングを停止する時刻を指定します。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、値を含まない `HH:MM` であるため、この値は必ず変更する必要があります。

アダプター・ランタイムが以下を検出した場合:

- PollStartTime が設定され PollEndTime が設定されていない、または
- PollEndTime が設定され PollStartTime が設定されていない

PollFrequency プロパティー用に構成されている値を使用してポーリングします。

PollFrequency

PollFrequency プロパティーは、あるポーリング・アクションの終了から次のポーリング・アクションの開始までの時間をミリ秒単位で指定します。これはポーリング・アクション間の間隔ではありません。この論理を次に説明します。

- ポーリングし、PollQuantity プロパティーの値により指定される数のオブジェクトを取得します。
- これらのオブジェクトを処理します。一部のコネクターでは、これは個別のスレッドで部分的に実行されます。これにより、次のポーリング・アクションまで処理が非同期に実行されます。
- PollFrequency プロパティーで指定された間隔にわたって遅延します。
- このサイクルを繰り返します。

このプロパティーでは、以下の値が有効です。

- ポーリング・アクション間のミリ秒数 (正整数)。
- ワード `no`。コネクター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。
- ワード `key`。コネクターは、コネクターのコマンド・プロンプト・ウィンドウで文字 `p` が入力されたときに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。

デフォルト値は 10000 です。

重要: 一部のコネクターでは、このプロパティーの使用が制限されています。このようなコネクターが存在する場合には、アダプターのインストールと構成に関する章で制約事項が説明されています。

PollQuantity

PollQuantity プロパティーは、コネクター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ー固有のポーリング数設定プロパティーがある場合、標準プロパティーの値は、このコネクター固有のプロパティーの設定値によりオーバーライドされます。

このプロパティーは、DeliveryTransport プロパティーの値が JMS であり、ContainerManagedEvents プロパティーに値がある場合のみ有効です。

E メール・メッセージもイベントと見なされます。コネクターは、E メールに関するポーリングを受けたときには次のように動作します。

- 一度ポーリングされると、コネクターはメッセージの本文を検出し、それを添付ファイルとして読み取ります。本文の MIME タイプにはデータ・ハンドラーが指定されていないので、コネクターはメッセージを無視します。
- コネクターは最初の BO 添付ファイルを処理します。この MIME タイプには対応するデータ・ハンドラーがあるので、コネクターはビジネス・オブジェクトを Visual Test Connector に送信します。
- 二度目にポーリングされると、コネクターは 2 番目の BO 添付ファイルを処理します。この MIME タイプには対応するデータ・ハンドラーがあるので、コネクターはビジネス・オブジェクトを Visual Test Connector に送信します。
- それが受け入れられると、3 番目の BO 添付ファイルが送信されます。

PollStartTime

PollStartTime プロパティーは、イベント・キューのポーリングを開始する時刻を指定します。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、値を含まない HH:MM であるため、この値は必ず変更する必要があります。

アダプター・ランタイムが以下を検出した場合:

- PollStartTime が設定され PollEndTime が設定されていない、または
- PollEndTime が設定され PollStartTime が設定されていない

PollFrequency プロパティー用に構成されている値を使用してポーリングします。

RepositoryDirectory

RepositoryDirectory プロパティーは、コネクターが XML スキーマ文書を読み取るリポジトリーの場所です。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保管されています。

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 Express リポジトリー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、この値はデフォルトで <*ProductDir*>¥repository に設定されます。ただし、これには任意の有効なディレクトリー名を設定することができます。

RequestQueue

RequestQueue プロパティーは、統合ブローカーがコネクターへビジネス・オブジェクトを送信するときに使用するキューを指定します。

このプロパティーは、DeliveryTransport プロパティーの値が JMS の場合のみ有効です。

デフォルト値は *<CONNECTORNAME>*/REQUESTQUEUE です。

ResponseQueue

ResponseQueue プロパティーは、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 InterChange Server Express (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。

このプロパティーは、DeliveryTransport プロパティーの値が JMS の場合のみ有効です。

デフォルト値は *<CONNECTORNAME>*/RESPONSEQUEUE です。

RestartRetryCount

RestartRetryCount プロパティーは、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に接続された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クライアント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

RestartRetryInterval プロパティーは、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にリンクされた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クライアント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。

プロパティーに使用可能な値の範囲は 1 から 2147483647 です。

デフォルト値は 1 です。

RHF2MessageDomain

RHF2MessageDomain プロパティーにより、JMS ヘッダーのドメイン名フィールドの値を構成できます。JMS トランスポートを介してデータを WebSphere Message Broker に送信するときに、アダプター・フレームワークにより JMS ヘッダー情報、ドメイン名、および固定値 mrm が書き込まれます。構成可能ドメイン名によって、WebSphere Message Broker がメッセージ・データを処理する方法を追跡できます。

ヘッダーの例を示します。

```
<mcd><Msd>mrm</Msd><Set>3</Set><Type>
Retek_POPhyDesc</Type><Fmt>CwXML</Fmt></mcd>
```

BrokerType の値が ICS の場合、このプロパティーは無効です。また、このプロパティーは、DeliveryTransport プロパティーの値が JMS で、WireFormat プロパティーの値が CwXML の場合のみ有効です。

可能な値は、mrm および xml です。デフォルト値は mrm です。

SourceQueue

SourceQueue プロパティーは、JMS イベント・ストアを使用する JMS 対応コネクター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ー・フレームワーク用に、JMS ソース・キューを指定します。詳細については、68 ページの『ContainerManagedEvents』を参照してください。

このプロパティーは、DeliveryTransport の値が JMS であり、ContainerManagedEvents の値が指定されている場合のみ有効です。

デフォルト値は *<CONNECTORNAME>*/SOURCEQUEUE です。

SynchronousRequestQueue

SynchronousRequestQueue プロパティーは、同期応答を要求する要求メッセージを、コネクター・フレームワークからブローカーにデリバリーします。このキューは、コネクター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、コネクター・フレームワークは、同期要求キューにメッセージを送信し、同期応答キューでブローカーからの応答を待機します。コネクター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。

このプロパティーは、DeliveryTransport の値が JMS の場合のみ有効です。

デフォルト値は *<CONNECTORNAME>*/SYNCHRONOUSREQUESTQUEUE です。

SynchronousRequestTimeout

SynchronousRequestTimeout プロパティーは、コネクターが同期要求への応答を待機する時間をミリ秒単位で指定します。コネクターは、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 (およびエラー・メッセージ) を障害キューに移動します。

このプロパティーは、DeliveryTransport の値が JMS の場合のみ有効です。

デフォルト値は 0 です。

SynchronousResponseQueue

SynchronousResponseQueue プロパティーは、同期要求に対する応答メッセージを、ブローカーからコネクター・フレームワークにデリバリーします。このキューは、コネクターが同期実行を使用する場合にのみ必要です。

このプロパティーは、DeliveryTransport の値が JMS の場合のみ有効です。

デフォルト値は *<CONNECTORNAME>*/SYNCHRONOUSRESPONSEQUEUE です。

TivoliMonitorTransactionPerformance

TivoliMonitorTransactionPerformance プロパティーは、IBM Tivoli Monitoring for Transaction Performance (ITMTP) を実行時に起動するかどうかを指定します。

デフォルト値は false です。

WireFormat

WireFormat プロパティーは、トランスポートでのメッセージ・フォーマットを指定します。

- RepositoryDirectory プロパティーの値がローカル・ディレクトリーの場合、値は CwXML です。
- RepositoryDirectory プロパティーの値がリモート・ディレクトリーの場合、値は CwBO です。

付録 B. Connector Configurator Express

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

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

- コネクターを構成するためのコネクター固有のプロパティー・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ーを設定する

この付録では、次のトピックについて説明します。

Connector Configurator Express の概要

Connector Configurator Express によって、InterChange Server Express 統合ブローカーで使用するアダプターのコネクター・コンポーネントを構成することができます。

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

- コネクターを構成するための**コネクター固有のプロパティー・テンプレート**を作成する。
- **コネクター構成ファイル**を作成します。インストールするコネクター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ーを設定する。
  場合によっては、コネクター・テンプレートでプロパティー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、InterChange Server Express 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。

コネクター構成プロパティーには、標準の構成プロパティー (すべてのコネクターがもつプロパティー) と、コネクター固有のプロパティー (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ーで必要なプロパティー) とが含まれます。

標準プロパティーはすべてのコネクターにより使用されるので、標準プロパティー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 Express により標準プロパティーがこの構成ファイルに挿入されます。ただし、Connector Configurator Express で各標準プロパティーの値を設定する必要があります。

標準プロパティーの範囲は、ブローカーと構成によって異なる可能性があります。特定のプロパティー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ーがあります。Connector Configurator Express の「標準のプロパティー」ウィンドウには、特定の構成で設定可能なプロパティーが表示されます。

ただし**コネクター固有プロパティー**の場合は、最初にプロパティーを定義し、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ー固有プロパティー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、83 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。

Connector Configurator Express の始動

以下の 2 種類のモードで Connector Configurator Express 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、System Manager を実行せずに Connector Configurator Express を実行し、コネクター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**」>「**すべてのプログラム**」から、「**IBM WebSphere Business Integration Express**」>「**Toolset Express**」>「**開発**」>「**Connector Configurator Express**」をクリックします。
- 「**ファイル**」>「**新規**」>「**コネクター構成**」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。

Connector Configurator Express 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方法が便利です (89 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 Express を実行できます。

Connector Configurator Express 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、「**コネクター**」を強調表示します。
3. System Manager メニュー・バーから、「**ツール**」>「**Connector Configurator Express**」をクリックします。「Connector Configurator Express」ウィンドウが開き、「**新規コネクター**」ダイアログ・ボックスが表示されます。

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。

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。

- 「System Manager」ウィンドウの「コネクター」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 Express 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。
- Connector Configurator Express で「**ファイル**」>「**開く**」を選択します。プロジェクトまたはプロジェクトが保管されているディレクトリーからコネクター構成ファイルを選択します。
- 「標準のプロパティー」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ーを確認します。

コネクター固有のプロパティー・テンプレートの作成

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

コネクター固有プロパティーのテンプレートを新規に作成するか、または既存のコネクター定義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、83 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、新しい名前でこのテンプレートを保管します。既存のテンプレートは ¥ProductDir¥bin¥Data¥App ディレクトリーにあります。

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ーを作成し、プロパティーの一般特性および値を定義し、プロパティー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、新規コネクター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。

Connector Configurator Express でテンプレートを作成するには、以下のステップを実行します。

1. 「**ファイル**」>「**新規**」>「**コネクター固有プロパティー・テンプレート**」をクリックします。
2. 「**コネクター固有プロパティー・テンプレート**」 ダイアログ・ボックスが表示されます。
   - 「**新規テンプレート名を入力してください**」の下の「**名前**」フィールドに、新規テンプレート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。
   - テンプレートに含まれているコネクター固有のプロパティー定義を調べるには、「**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ー定義のリストが「**テンプレートのプレビュー**」表示に表示されます。

3. テンプレートを作成するときには、ご使用のコネクターに必要なプロパティー定義に類似したプロパティー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。ご使用のコネクターで使用するコネクター固有のプロパティー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。
   - 既存のテンプレートを変更する場合には、「**変更する既存のテンプレートを選択してください: 検索テンプレート**」の下の「**テンプレート名**」テーブルのリストから、テンプレート名を選択します。
   - このテーブルには、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。テンプレートを検索することもできます。

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、「**プロパティー: コネクター固有プロパティー・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ーの「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:**
  プロパティー・タイプ
  プロパティー・サブタイプ
  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

「**プロパティー・タイプ**」がストリングの場合、「**プロパティー・サブタイプ**」を選択できます。これは、構成ファイルの保管時に構文検査を提供するオプションの値です。デフォルトはブランク・スペースで、プロパティーのサブタイプが指定されていないことを意味します。

プロパティー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、プロパティーの最大長、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. 「**プロパティーを編集**」表示でプロパティーの名前を選択します。
3. 「**最大長**」および「**最大複数値**」のフィールドに値を入力します。

新規プロパティー値を作成するには、以下のステップを実行します。

1. 「値」列見出しの左側の正方形を右マウス・ボタンでクリックします。
2. ポップアップ・メニューから「**追加**」を選択して、「プロパティー値」ダイアログ・ボックスを表示します。ダイアログ・ボックスでは、プロパティー・タイプに応じて、値を入力するか、または値と範囲の両方を入力することができます。

3. 新規プロパティー値を入力し、「OK」をクリックします。右側の「**値**」パネルに値が表示されます。

「**値**」パネルには、3 つの列からなるテーブルが表示されます。

「**値**」の列には、「**プロパティー値**」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、「**プロパティー値**」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: コネクター固有プロパティー・テンプレート**」ダイアログ・ボックスが表示されます。

依存プロパティーは、別のプロパティー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ーです。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。プロパティーを依存プロパティーとして指定し、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ー**」表示で、依存プロパティーとして指定するプロパティーを選択します。
2. 「**プロパティーを選択**」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ーを選択します。
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、依存プロパティー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ー**」表示で依存プロパティーを強調表示させて矢印をクリックし、「**依存プロパティー**」表示に移動させます。
6. 「**完了**」をクリックします。Connector Configurator Express により、XML 文書として入力した情報が、Connector Configurator Express 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。

パス名の設定

パス名の設定の一般的な規則のいくつかを以下に示します。

- Windows でのファイル名の最大長は 255 文字です。
- Windows では、絶対パス名は [Drive:][Directory]¥filename の形式に従う必要があります。例えば、C:¥WebSphereAdapters¥bin¥Data¥Std¥StdConnProps.xml のようにします。
- キュー名では、先頭または途中にスペースを使用することはできません。

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、構成ファイルの名前を指定し、統合ブローカーを選択する必要があります。

ファイルの拡張検証のために、オペレーティング・システムも選択します。ツールバーには「**ターゲット・システム**」というドロップ・リストがあり、ここで、プロパティーの拡張検証用のターゲット・オペレーティング・システムを選択できます。Windows で使用可能なオプションは、拡張検証なしです (拡張検証をオフに切り替えます)。始動時のデフォルトは「Windows」です。

Connector Configurator Express を始動するには、以下のステップを実行します。

- 「System Manager」ウィンドウで、「**ツール**」メニューから「**Connector Configurator Express**」を選択します。Connector Configurator Express が開きます。
- スタンドアロン・モードで、Connector Configurator Express を起動します。

構成ファイルの拡張検証用のオペレーティング・システムを設定するには、以下のステップを実行します。

- メニュー・バーの「**ターゲット・システム:**」ドロップ・リストをプルダウンします。
- 使用中のオペレーティング・システムを選択します。

次に、「**ファイル**」>「**新規**」>「**コネクター構成**」を選択します。「新規コネクター」ウィンドウで、新規コネクターの名前を入力します。

また、統合ブローカーも選択する必要があります。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ーが決まります。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。

- 「**Integration Broker**」フィールドで、ICS 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。

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. メニュー・バーの「**ターゲット・システム:**」ドロップ・リストを使用して、構成ファイルの拡張検証用のオペレーティング・システムを設定します (前述の『新規構成ファイルの作成』を参照してください)。
2. 「**ファイル**」>「**新規**」>「**コネクター構成**」をクリックします。
3. 以下のフィールドを含む「**新規コネクター**」ダイアログ・ボックス表示されます。
   - **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 Express 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」をクリックします。
   - **「コネクター固有プロパティー・テンプレート」を選択します。**

     ご使用のコネクター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、「**プロパティー・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ー固有のプロパティーが表示されます。

     使用するテンプレートを選択し、「**OK**」をクリックします。
4. 構成しているコネクター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ー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。
5. 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・コネクターを保管**」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ーにナビゲートし、「**保管**」をクリックします。Connector Configurator Express 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ー・パスおよび名前は、コネクターの始動ファイルで指定するコネクター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。

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

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。

- コネクター定義ファイル。これは、特定のコネクターのプロパティーと、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。コネクターの配布パッケージの ¥repository ディレクトリー内には、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、JMS コネクターの場合は CN_JMS.txt です)。
- ICS リポジトリー・ファイル。コネクターの以前の ICS インプリメンテーションで使用した定義は、そのコネクターの構成で使用されたリポジトリー・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。
- コネクター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、コネクターのコネクター固有プロパティーのほとんど、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、コネクター構成ファイルは、ファイルを開いて、プロパティー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ーを構成するには、Connector Configurator Express でそのファイルを開き、構成を修正してから、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 Express 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・コネクターを開く**」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 リポジトリー (*.in、*.out)

     ICS 環境でのコネクターの構成にリポジトリー・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。リポジトリー・ファイルに複数のコネクター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 (*.*)

     コネクターのアダプター・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、適切なコネクター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ー構成を開くには、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 Express 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ーを開くと、「Connector Configurator Express」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ー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、コネクター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 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コネクター・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。表に、「**プロパティー名**」、「**値**」、「**タイプ**」、「**サブタイプ**」 (「タイプ」がストリングである場合)、「**説明**」、および 「**更新メソッド**」が表示されます。
3. ここでファイルを保管するか、または 93 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。
4. 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。

   複数のコネクター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、コネクター構成をすべて 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。

   構成ファイルを作成する前に、プロパティーの拡張検証用のターゲット・オペレーティング・システムを選択することができる「**ターゲット・システム**」ドロップ・リストを使用します。

   Connector Configurator Express では、ファイルを保管する前に、必須の標準プロパティーすべてに値が設定されているかどうかが確認されます。必須の標準プロパティーに値が設定されていない場合、Connector Configurator Express 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、そのプロパティーの値を指定する必要があります。

「**ターゲット・システム**」ドロップ・リストから「Windows」を選択することによって拡張検証機能を使用する場合、システムはタイプだけでなくプロパティー・サブタイプを検証し、検証に失敗した場合は警告メッセージを表示します。

構成ファイル・プロパティーの設定

新規のコネクター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ー構成ファイルを開くときには、Connector Configurator Express 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ーに対応する複数のタブがあります。

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

- 標準プロパティー
- コネクター固有のプロパティー
- 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ーの場合に該当する)

注: JMS メッセージングを使用するコネクターの場合は、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ーが表示される場合があります。

InterChange Server Express で実行されているコネクターの場合、以下のプロパティー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
- セキュリティー

重要: Connector Configurator Express 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ー値も設定可能です。ただし、標準のプロパティーおよびコネクター固有プロパティー、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。

標準プロパティーとコネクター固有プロパティーの違いは、以下のとおりです。

- コネクターの標準プロパティーは、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用されます。すべてのコネクターが同じ標準プロパティーのセットを使用します。これらのプロパティーの説明は、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ーは、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ーがあります。これらのプロパティーには、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ーおよび推奨値が記述されています。

「**標準プロパティー**」と「**コネクター固有プロパティー**」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、標準のプロパティーを表します。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ー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ーを表します。これらのプロパティーは、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ーの除去も可能です。
- 「値」フィールドは構成できます。
- プロパティーごとに「**更新メソッド**」フィールドが表示されます。これは、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。この設定を構成することはできません。

標準コネクター・プロパティーの設定

標準のプロパティー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。

   注: プロパティーの「タイプ」が「ストリング」である場合、「サブタイプ」列にサブタイプ値が含まれている場合があります。このサブタイプは、プロパティーの拡張検証に使用されます。
3. 標準のプロパティー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 Express 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 Express 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。「**標準のプロパティー**」(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。

特定の標準プロパティーに関する詳細を参照するには、「標準のプロパティー」タブ付きシート内のそのプロパティーの「説明」列内の項目を左マウス・ボタンでクリックします。全般ヘルプをインストール済みの場合は、右側に矢印ボタンが表示されます。ボタンをクリックすると、「ヘルプ」ウィンドウが開き、標準プロパティーの詳細が表示されます。

注: ホット・ボタンが表示されない場合、そのプロパティーについては全般ヘルプが見つかっていません。

インストール済みの場合、全般ヘルプ・ファイルは
<*ProductDir*>¥bin¥Data¥Std¥Help¥<*RegionalSetting*>¥ にあります。

コネクター固有の構成プロパティーの設定

コネクター固有の構成プロパティーの場合、プロパティー名の追加または変更、値の構成、プロパティーの削除、およびプロパティーの暗号化が可能です。プロパティー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。プロパティーを追加するときは「**追加**」をクリックします。子プロパティーを追加するには、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. プロパティーまたは子プロパティーの値を入力します。

   注: プロパティーの「タイプ」が「ストリング」である場合、「サブタイプ」ドロップ・リストからサブタイプを選択できます。このサブタイプは、プロパティーの拡張検証に使用されます。
3. プロパティーを暗号化するには、「**暗号化**」ボックスを選択します。
4. 特定のプロパティーに関する詳細を参照するには、そのプロパティーの「説明」列内の項目を左マウス・ボタンでクリックします。全般ヘルプをインストール済みの場合は、ホット・ボタンが表示されます。ホット・ボタンをクリックすると、「ヘルプ」ウィンドウが開き、標準プロパティーの詳細が表示されます。

   注: ホット・ボタンが表示されない場合、そのプロパティーについては全般ヘルプが見つかっていません。
5. 91 ページの『標準コネクター・プロパティーの設定』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。

全般ヘルプ・ファイルがインストール済みで、AdapterHelpName プロパティーがブランクである場合、Connector Configurator Express は、
<*ProductDir*>¥bin¥Data¥App¥Help¥<*RegionalSetting*>¥ にあるアダプター固有の全般ヘルプ・ファイルを指します。それ以外の場合、Connector Configurator Express は、<*ProductDir*>¥bin¥Data¥App¥Help¥<*AdapterHelpName*>¥<*RegionalSetting*>¥にあるアダプター固有の全般ヘルプ・ファイルを指します。標準プロパティーについての付録で説明されている AdapterHelpName プロパティーを参照してください。

各プロパティーごとに表示される「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ー・プロパティー名を変更すると、コネクターに障害が発生する可能性があります。コネクター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、特定のプロパティー名が必要である場合があります。

コネクター・プロパティーの暗号化

「コネクター固有プロパティー」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ー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。

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

プロパティーに複数の値がある場合には、プロパティーの最初の値に「**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、そのプロパティーのすべての値が暗号化されます。プロパティーの複数の値を暗号化解除するには、そのプロパティーの最初の値の「**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、すべての複数値が暗号化解除されます。

更新メソッド

標準プロパティーについての付録である 59 ページの『付録 A. コネクターの標準構成プロパティー』内の『標準コネクター・プロパティーの概要』の下の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

コネクターで使用するビジネス・オブジェクトを指定するには、Connector Configurator Express 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。

注: コネクターによっては、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。

ご使用のブローカーが InterChange Server Express の場合

ビジネス・オブジェクト定義がコネクター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ー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・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 Express」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ー定義が、System Manager の ICL (Integration Component Library) 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 Express」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、コネクター定義が変更され、削除されたビジネス・オブジェクトはコネクターのこのインプリメンテ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 Express」ウィンドウでは、「エージェント・サポート」を選択しても問題ないかどうかの検証は行われません。

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。

関連付けられたマップ

各コネクターは、ビジネス・オブジェクト定義とそれらに関連付けられたマップのうち現在 InterChange Server Express でアクティブであるものを示す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ライブ・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に、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 Express」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。

- **明示的バインディング**

  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。InterChange Server Express は、ブート時、各コネクターのサポートされるビジネス・オブジェクトのそれぞれにマップを自動的にバインドしようとします。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 Express」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 InterChange Server Express 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。

セキュリティー

Connector Configurator Express 内の「**セキュリティー**」タブを使用して、メッセージにさまざまなプライバシー・レベルを設定することができます。DeliveryTransport プロパティーが JMS に設定されている場合のみ、この機能を使用できます。

デフォルトでは、「プライバシー」はオフになっています。使用可能にするには、「**プライバシー**」ボックスにチェック・マークを付けます。

「**鍵ストア・ターゲット・システムの絶対パス名**」は、以下の値です。

Windows の場合:
```
<ProductDir>¥connectors¥security¥<connectorname>.jks
```

このパスおよびファイルは、コネクターを始動するシステム、すなわちターゲット・システム上に存在していなければなりません。

ターゲット・システムが現在実行中のシステムである場合のみ、右側の「参照」ボタンを使用できます。「**プライバシー**」が使用可能であり、メニュー・バーの「**ターゲット・システム**」が Windows に設定されている場合を除き、これはグレーアウトされています。

「**メッセージのプライバシー・レベル**」は、3 つのメッセージ・カテゴリー (全メッセージ、全管理メッセージ、および全ビジネス・オブジェクト・メッセージ) で以下のように設定されます。

- “”: がデフォルトです。メッセージ・カテゴリーにプライバシー・レベルが設定されていない場合に使用します。
- none。デフォルトと同じではありません。メッセージ・カテゴリーにプライバシー・レベルなしと故意に設定する場合にこれを使用します。
- integrity
- privacy
- integrity_plus_privacy

「**鍵の保守**」機能によって、サーバーおよびアダプターの公開鍵を生成、インポート、およびエクスポートすることができます。

- 「**鍵の生成**」を選択すると、鍵を生成する keytool のデフォルトを含む「鍵の生成」ダイアログ・ボックスが表示されます。
- 「セキュリティー」タブの「**鍵ストア・ターゲット・システムの絶対パス名**」で入力した値が、鍵ストア値のデフォルトになります。
- 「OK」を選択すると、記入項目が検証され、鍵証明書が生成され、「Connector Configurator Express」ログ・ウィンドウに出力が送られます。

証明書をアダプター鍵ストアにインポートする前に、サーバー鍵ストアからエクスポートする必要があります。「**アダプター公開鍵のエクスポート**」を選択すると、「アダプター公開鍵のエクスポート」ダイアログ・ボックスが表示されます。

- エクスポート証明書のデフォルトは、ファイル拡張子が <filename>.cer であることを除き、鍵ストアと同じ値です。

「**サーバー公開鍵のインポート**」を選択すると、「サーバー公開鍵のインポート」ダイアログ・ボックスが表示されます。

- インポート証明書のデフォルトは、<*ProductDir*>¥bin¥ics.cer になります (システムにファイルが存在する場合)。
- インポート証明書関連はサーバー名でなければなりません。サーバーが登録されていれば、ドロップ・リストからそれを選択することができます。

DeliveryTransport の値が IDL の場合のみ、「**アダプター・アクセス制御**」機能が使用可能です。デフォルトでは、アダプターはゲスト ID を使用してログインします。「**ゲスト ID の使用**」ボックスにチェック・マークが付けられていない場合は、「**アダプター ID**」および「**アダプター・パスワード**」フィールドが使用可能です。

トレース/ログ・ファイル値の設定

コネクター構成ファイルまたはコネクター定義ファイルを開くと、Connector Configurator Express 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。これらの値は、Connector Configurator Express 内で変更できます。

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 表示に書き込みます。

     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ーの「**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・メッセージを指定し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、指定する格納場所に移動し、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『コネクター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。

構成ファイルの保管

コネクターの構成が完了したら、コネクター構成ファイルを保管します。Connector Configurator Express 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 Express のタイトル・バーには、InterChange Server Express が現在使用しているブローカー・モード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、統合コンポーネント・ライブラリーに *.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。デフォルトでは、このファイルは ¥WebSphereAdapters¥bin¥Data¥App に保管されます。

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、「システム・インプリメンテーション・ガイド」を参照してください。

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 Express 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・ウィンドウ内の利用可能なタブおよびフィールド選択がただちに変更され、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。

グローバル化環境における Connector Configurator Express の使用

Connector Configurator Express はグローバル化されており、構成ファイルと統合ブローカーの間での文字変換を処理できます。Connector Configurator Express 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 Express 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ーのドロップ・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 プロパティー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

本書は米国 IBM が提供する製品およびサービスについて作成したものです。本書に記載の製品、サービス、または機能が日本においては提供されていない場合があります。日本で利用可能な製品、サービス、および機能については、日本 IBM の営業担当員にお尋ねください。本書で IBM 製品、プログラム、またはサービスに言及していても、その IBM 製品、プログラム、またはサービスのみが使用可能であることを意味するものではありません。これらに代えて、IBM の知的所有権を侵害することのない、機能的に同等の製品、プログラム、またはサービスを使用することができます。ただし、IBM 以外の製品とプログラムの操作またはサービスの評価および検証は、お客様の責任で行っていただきます。IBM は、本書に記載されている内容に関して特許権 (特許出願中のものを含む) を保有している場合があります。本書の提供は、お客様にこれらの特許権について実施権を許諾することを意味するものではありません。実施権についてのお問い合わせは、書面にて下記宛先にお送りください。

〒106-0032
東京都港区六本木 3-2-31
IBM World Trade Asia Corporation
Licensing

以下の保証は、国または地域の法律に沿わない場合は、適用されません。IBM およびその直接または間接の子会社は、本書を特定物として現存するままの状態で提供し、商品性の保証、特定目的適合性の保証および法律上の瑕疵担保責任を含むすべての明示もしくは黙示の保証責任を負わないものとします。国または地域によっては、法律の強行規定により、保証責任の制限が禁じられる場合、強行規定の制限を受けるものとします。この情報には、技術的に不適切な記述や誤植を含む場合があります。本書は定期的に見直され、必要な変更は本書の次版に組み込まれます。IBM は予告なしに、随時、この文書に記載されている製品またはプログラムに対して、改良または変更を行うことがあります。本書において IBM 以外の Web サイトに言及している場合がありますが、便宜のため記載しただけであり、決してそれらの Web サイトを推奨するものではありません。それらの Web サイトにある資料は、この IBM 製品の資料の一部ではありません。それらの Web サイトは、お客様の責任でご使用ください。IBM は、お客様が提供するいかなる情報も、お客様に対してなんら義務も負うことのない、自ら適切と信ずる方法で、使用もしくは配布することができるものとします。本プログラムのライセンス保持者で、(i) 独自に作成したプログラムとその他のプログラム (本プログラムを含む) との間での情報交換、および (ii) 交換された情報の相互利用を可能にすることを目的として、本プログラムに関する情報を必要とする方は、下記に連絡してください。

IBM Corporation
577 Airport Blvd., Suite 800
Burlingame, CA 94010
U.S.A

本プログラムに関する上記の情報は、適切な使用条件の下で使用することができますが、有償の場合もあります。本書で説明されているライセンス・プログラムまたはその他のライセンス資料は、IBM 所定のプログラム契約の契約条項、IBM プログラムのご使用条件、またはそれと同等の条項に基づいて、IBM より提供されます。この文書に含まれるいかなるパフォーマンス・データも、管理環境下で決定されたものです。そのため、他の操作環境で得られた結果は、異なる可能性があります。一部の測定が、開発レベルのシステムで行われた可能性がありますが、その測定値が、一般に利用可能なシステムのものと同じである保証はありません。さらに、一部の測定値が、推定値である可能性があります。実際の結果は、異なる可能性があります。お客様は、お客様の特定の環境に適したデータを確かめる必要があります。IBM 以外の製品に関する情報は、その製品の供給者、出版物、もしくはその他の公に利用可能なソースから入手したものです。IBM は、それらの製品のテストは行っておりません。したがって、他社製品に関する実行性、互換性、またはその他の要求については確証できません。IBM 以外の製品の性能に関する質問は、それらの製品の供給者にお願いします。IBM の将来の方向または意向に関する記述については、予告なしに変更または撤回される場合があり、単に目標を示しているものです。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。著作権使用許諾: 本書には、様々なオペレーティング・プラットフォームでのプログラミング手法を例示するサンプル・アプリケーション・プログラムがソース言語で掲載されています。お客様は、サンプル・プログラムが書かれているオペレーティング・プラットフォームのアプリケーション・プログラミング・インターフェースに準拠したアプリケーション・プログラムの開発、使用、販売、配布を目的として、いかなる形式においても、IBM に対価を支払うことなくこれを複製し、改変し、配布することができます。このサンプル・プログラムは、あらゆる条件下における完全なテストを経ていません。従って IBM は、これらのサンプル・プログラムについて信頼性、利便性もしくは機能性があることをほのめかしたり、保証することはできません。この情報をソフトコピーでご覧になっている場合は、写真やカラーの図表は表示されない場合があります。

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
CICS
CrossWorlds
DB2
DB2 Universal Database
i5/OS
IMS
Informix
iSeries
Lotus
Lotus Domino
Lotus Notes
MQIntegrator
MQSeries
MVS
OS/400
Passport Advantage
SupportPac
WebSphere
z/OS
Microsoft、Windows、Windows NT および Windows ロゴは、Microsoft Corporation の米国およびその他の国における商標です。
MMX および Pentium は、Intel Corporation の米国およびその他の国における商標です。
Java およびすべての Java 関連の商標およびロゴは、Sun Microsystems, Inc. の米国およびその他の国における商標または登録商標です。
Linux は、Linus Torvalds の米国およびその他の国における商標です。
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。

WebSphere Business Integration Server Express and Express Plus 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 Server Express バージョン 4.4、および WebSphere Business Integration Server Express Plus バージョン 4.4

Printed in Japan

日本アイ・ビー・エム株式会社
〒106-8711　東京都港区六本木3-2-12