IBM WebSphere Business Integration Server Express and Express Plus

Adapter for Web Services ユーザーズ・ガイド

アダプター・バージョン 3.4.x

IBM WebSphere Business Integration Server Express and Express Plus

Adapter for Web Services ユーザーズ・ガイド

アダプター・バージョン 3.4.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、 261 ページの『特記事項』に記載されている情報をお読みください。

本書は、IBM WebSphere Business Integration Adapter for Web Services (5724-H09) バージョン 3.4.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 Web Services User Guide
Adapter Version 3.4.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

目次

本書について

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

本書では、Adapter for Web Services のインストール、コネクター・プロパティーの構成、ビジネス・オブジェクトの開発、およびトラブルシューティングについて説明します。

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

本書の対象読者は、WebSphere の顧客、コンサルタント、開発者など、Adapter for Web Services をインプリメントする人々です。

本書を読むための前提条件

本書の中では、いろいろな前提条件が挙げられています。その多くは、Web サービスに関する情報が掲載された Web サイトや Web サービスのためのリソースが提供されている Web サイトへの参照です。また、WebSphere Business Integration Server Express システムのインプリメントに関する知識も必要です。最初に「システム・インプリメンテーション・ガイド」を読むことをお勧めします。この資料には、詳細な情報が記載された文献に関する相互参照が示されています。

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。

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

Eclipse テクノロジー

WebSphere Business Integration Server Express Adapter for Web Services には、Eclipse テクノロジーが組み込まれています。Eclipse は強力なソフトウェア開発ツール群と機能豊富なデスクトップ・アプリケーションを構成するための優れたオープン・ソース・フレームワークです。Eclipse のプラグイン・フレームワークを活用してデスクトップにテクノロジーを統合すると、テクノロジー・プロバイダーは自社製品を差異化し、価値を加えることに集中することができるため、時間と経費の節約になります。

Eclipse は、開発者のオープン・ソース・コミュニティーが構築した、複数言語、マルチプラットフォーム、複数ベンダーがサポートされる環境であり、Eclipse Foundation により使用料なしで提供されます。Eclipse は Java™ 言語で書かれており、広範囲に渡るプラグイン構成ツールキットと実例を含み、Windows®、Linux、QNX、および Macintosh OS X など、さまざまなデスクトップ・オペレーティング・システムでの配置が可能です。Eclipse および Eclipse Foundation の詳細については、http://eclipse.org をご覧ください。

表記上の規則

本書では、以下の規則を使用します。

`Courier フォント`	コマンド名、ファイル名、入力する情報、システムが画面に出力する情報などのリテラル値を示します。
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。
青のアウトライン	青のアウトラインは、マニュアルをオンラインで表示するときにのみ見られるもので、相互参照用のハイパーリンクを示します。アウトラインの内側をクリックすることにより、参照先オブジェクトにジャンプできます。
{ }	構文の記述行の場合、複数のオプションが中括弧で囲まれている場合、その中の 1 つのオプションのみを選択することが必要です。
[ ]	構文の記述行の場合、大括弧 [] で囲まれた部分は、オプショナル・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の繰り返しを示します。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示します。
< >	命名規則において不等号括弧で囲まれた部分は、名前を構成する複数の要素の 1 つを示します。 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。Linux および i5/OS システムの場合は、円記号はスラッシュ (/) に置き換えてください。すべての WebSphere Business Integration Server Express 製品のパス名は、システム上の製品がインストールされているディレクトリーを基準にした相対パス名です。
`%`*`text`*`%` および `$`*`text`*	パーセント記号で囲まれたテキストは、Windows の `text` システム変数またはユーザー変数の値を示します。Linux 環境ではこれを `$`*`text`* で表します。すなわち、*`text`* Linux 環境変数の値を示します。

ProductDir	WebSphere Business Integration Server Express Adapter 製品がインストールされているディレクトリーを表します。各プラットフォームのデフォルトは、以下のとおりです。 Windows: IBM¥WebSphereServer i5/OS: /QIBM/ProdData/WBIServer44/product Linux: /home/${username}/IBM/WebSphereServer
”	メニューから次のように項目を選択します。「ファイルの選択 (Choose File)」”「更新 (Update)」”「SGML リファレンス (SGML References)」

本リリースの新機能

リリース 3.4.x の新機能

2004 年 9 月に更新されました。アダプター・バージョン 3.4.x に対応する本書のリリースには、以下の新情報または訂正情報が記載されています。

本リリースでは、Windows プラットフォームの場合のみ、双方向スクリプト・データの処理に対するサポートが追加されています。

本リリースでは、以下のプラットフォームおよびプラットフォームの更新に対するサポートが追加されています。

- Microsoft Windows 2003 (Standard Edition または Enterprise Edition)
- Linux:
  RedHat Enterprise Linux WS/AS/ES3.0 Update 2、Intel (IA32)
  SuSE Linux ES 8.1 SP3、Intel (IA32)
  SuSE Linux ES 9.0、Intel (IA32)
- IBM i5/OS V5R3 および OS/400 V5R2
- カスタム・アダプターのコンパイル用の Java コンパイラー IBM JDK 1.4.2 for Windows 2000

ibmjsse.jar、xercesImpl.jar、および xmlParserAPIs.jar の JAR ファイルがインストール後のファイル構造から除外されました。

本リリースでは、次の新バージョンの API または API の要件をサポートし、以下に示す API が除外されました。

- Apache SOAP API により必要: Java Activation Framework 1.0.2 (activation.jar) および JavaMail™ API 1.3.1 (mail.jar)
- WSDL4J 1.4
- IBM JSSE 1.0.3
- XML4J 4.3.0
- 除外された API: Xerces Java パーサー

第 1 章 コネクターの概要

このコネクターは、WebSphere Business Integration Server Express Adapter for Web Services のランタイム・コンポーネントです。コネクターを使用すると、各企業は、自社の組織内部や取引先で使用するための Web サービスを集約、公開、および利用することができます。この資料に説明されているコネクターやそのほかのコンポーネントは、Simple Object Access Protocol (SOAP) メッセージの本文に記述されているビジネス・オブジェクトを交換するために必要な機能を提供します。

Adapter for Web Services は、 Web Services-Interoperability Organization (WS-I) に準拠して、SOAP 1.1 および 1.2 をサポートします。このアダプターは、WebSphere Business Integration Server Express のビジネス・プロセスを、コラボレーションの形で Web サービスとして公開します。一連の自己完結型の動的アプリケーションを提供します。これらのアプリケーションは、ネットワークを介して記述、パブリッシュ、呼び出しを行うことによって、革新的な製品、プロセス、およびバリュー・チェーンを作成することができます。このアダプターは、構成可能性の拡張により、外部にホストがある Web サービスと通信することができます。

Adapter for Web Services では、アダプター内からイベントおよび要求処理を行うための本格的な双方向言語サポートを実現します。イベント処理を同期式または非同期に行うことができ、アダプター内のリスナーによって、HTTP、HTTPS、および JMS トランスポート上の SOAP をサポートします。Web Services Gateway 製品との統合により、アダプターに代わって外部サービスを公開し、呼び出します。Web Services Description Language (WSDL) Object Discovery Agent (ODA) により、ビジネス・オブジェクトの生成と配置が容易になっています。ODA では、ローカル・ファイルから WSDL を検索したり、リモート URL や Universal Description Discovery and Integration (UDDI) レジストリーに接続したりすることができます。

この章では、Adapter for Web Services のインプリメントに使用されるスコープ、コンポーネント、設計ツール、およびアーキテクチャーについて説明します。また、本書で述べられている Web サービス・コンポーネントをインストールおよび構成するために行う必要のあるタスクについても概説します。コンポーネントのインストールおよび構成については、 14 ページの『インストール、構成、および設計のチェックリスト』を参照してください。

注: Adapter for Web Services は標準 Adapter Framework API を実装しています。このため、アダプターは Framework がサポートする任意の統合ブローカーと連動して動作します。ただし、アダプターが提供する機能は、厳密には InterChange Server Express 統合ブローカーをサポートするように設計されてい

ます。したがって、System Manager で「Web サービスとして公開」オプションを選択した場合、これは他の統合ブローカーではなく InterChange Server Express を意味します。

Adapter for Web Services の環境

アダプターをインストール、構成、および使用する前に、アダプターの環境要件を理解しておく必要があります。

- 『ソフトウェア前提条件』
- 『アダプターのプラットフォーム』
- 『規格および API』
- 4 ページの『ロケール依存データ』

ソフトウェア前提条件

Connector for Web Services をインストールするには、その前に以下の前提事項およびソフトウェア要件をよく確認してください。

- コネクターやそのほかのコンポーネントの設計は、SOAP 1.1 および 1.2 向けに公開されている仕様が基本になります。
- SOAP/JMS Web サービスを使用する場合は、ユーザー独自の JMS および JNDI インプリメンテーションをインストールする必要があります。
- HTTPS/SSL を使用する場合は、鍵ストアおよびトラストストアを作成するために、ユーザー独自のサード・パーティー・ソフトウェアが必要になります。

アダプターのプラットフォーム

アダプターでは、ブローカーのほかに、次のいずれかのオペレーティング・システムが必要です。

- Microsoft Windows 2003 (Standard Edition または Enterprise Edition)
- Linux:
  RedHat Enterprise Linux WS/AS/ES 3.0 with Update 2、Intel (IA32)
  SuSE Linux ES 8.1 SP3、Intel (IA32)
  SuSE Linux ES 9.0、Intel (IA32)
- IBM i5/OS V5R3 および OS/400 V5R2

  注: 特に明記しない限り、i5/OS は OS/400 および i5/OS を指します。
- すべてのオペレーティング・システム環境で、カスタム・アダプターのコンパイル用の Java コンパイラー (IBM JDK 1.4.2 for Windows 2000) が必要です。

規格および API

Adapter for Web Services (コネクター、WSDL ODA、および SOAP データ・ハンドラー) は、2003 年 8 月に公開された WS-I Basic Profile 1.0 仕様に準拠しています。

さまざまな規格や技術により、各 Web サービスは、ネットワークを介してほかの Web サービスの機能にアクセスできるようになります。

アダプターが使用する規格は、以下のとおりです。

- SOAP バージョン 1.2 および 1.1
- WSDL 1.1 SOAP バインディング
- HTTP 1.0
- JMS 1.0.2

アダプターが使用する API は、以下のとおりです。

- Apache SOAP 2.3.1 API: このコネクターには、Apache Foundation が開発した SOAP API が組み込まれています。Apache SOAP API は、SOAP バージョン 1.1 をインプリメントしたオープン・ソース系の API です。Apache SOAP API には、以下の要件があります。
  - Java Activation Framework 1.0.2 (activation.jar)
  - JavaMail(TM) API 1.3.1 (mail.jar)
- JMS API バージョン 1.0.2
- WSDL4J 1.4 - Web Service Description Language for Java API (WSDL4J) は、WSDL 文書用のオブジェクト・モデルを規定しています。
- UDDI4J-WSDL 2.1.0 - UDDI4J-WSDL API は、UDDI4J API に存在するクラス、および WSDL4J API によって定義された一部のクラスも包含します。
- JNDI 1.2.1
- IBM JSSE 1.0.3
- XML4J 4.3.0

構成によっては、追加のソフトウェアをインストールする必要がある場合もあります。構成別の要件については、以下のセクションで説明します。

JMS プロトコル

JMS プロトコルを使用する場合は、JMS プロバイダーをインストールしてキューを作成する必要があります。キューの作成は、ユーザーの要件によってまったく異なります。JMS プロトコルは、コラボレーションを Web サービスとして公開すること、および外部 Web サービスを呼び出すことの両方に使用できます。詳しくは、92 ページの『コネクターおよび JMS』を参照してください。

JNDI: JNDI を構成してから、該当するパラメーターをコネクターの JNDI 構成プロパティーに入力する必要があります。また、接続ファクトリーおよび JMS 宛先 (キュー) オブジェクトが JNDI において使用可能になっていることを確認する必要もあります。JNDI を使用したいのに、JNDI がインプリメントされていない場合は、File System JNDI の参照インプリメンテーションを Sun Microsystems からダウンロードすることができます。詳しくは、92 ページの『コネクターおよび JMS』を参照してください。

SSL

SSL の使用を計画している場合は、鍵ストア、証明書、および鍵生成を管理するために、サード・パーティーのソフトウェアを使用する必要があります。鍵ストア、証明書のセットアップ、または鍵生成用のツールは提供されていません。Keytool (IBM JRE に同梱) の使用を選択して、自己署名証明書を作成し、鍵ストアを管理することもできます。詳しくは、94 ページの『SSL』を参照してください。

ロケール依存データ

コネクターは、2 バイト文字セットをサポートできるようにグローバル化対応されています。コネクターは、1 つの文字コードを使用する場所から別のコード・セットを使用する場所にデータを転送するとき、データの意味を保存するように文字変換を実行します。

このアダプターは、アラビア語、ヘブライ語、ウルドゥー語、ペルシア語、およびイディッシュ語のような言語に対して、双方向スクリプト・データの処理をサポートします。双方向の能力を使用するには、標準の双方向プロパティーを構成する必要があります。詳しくは、付録 A にある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。

Java 仮想マシン (JVM) 内での Java ランタイム環境は、Unicode 文字コード・セットでデータを表します。Unicode には、最もよく知られた文字コード・セット (単一バイトとマルチバイトの両方) の文字エンコードが含まれています。WebSphere Business Integration Server Express システムのほとんどのコンポーネントは Java で記述されています。したがって、ほとんどの統合コンポーネントの間でデータが転送されても、文字変換の必要はありません。

注: コネクターは、国際化対応されていません。このことは、トレースおよびログ・メッセージが変換されないということを意味します。

Web サービス・コネクター

このセクションでは、ローカリゼーションとコネクターについて説明します。

イベント通知: コネクターは、プラグ可能なプロトコル・リスナーを使用してイベント通知を行います。プロトコル・リスナーはトランスポートから SOAP メッセージを抽出し、SOAP データ・ハンドラーを呼び出します。このセクションでは、それぞれのリスナーがトランスポートを介してどのように SOAP メッセージをエンコードするのかを説明します。

- **SOAP/HTTP および SOAP/HTTPS リスナー** これらのリスナーは、HTTP 要求メッセージの本文をバイトとして読み取ります。本文のエンコード方式は、HTTP Content-Type ヘッダーの charset パラメーターで指定されます。charset パラメーターが指定されていない場合には、ISO-8859-1(ISO Latin 1) が使用されます。リスナーは、このエンコード方式を使用して要求メッセージの本文を Java ストリングに変換します。この Java ストリングは、SOAP データ・ハンドラーを呼び出すために使用されます。同期 (要求/応答) Web サービスでは、SOAP データ・ハンドラーは、コラボレーションによって戻されたビジネス・オブジェクトを使用して呼び出されます。SOAP データ・ハンドラーによって戻された Java ストリングは、HTTP 要求メッセージから得られたエンコード方式を使用してバイトに変換されます。
- **SOAP/JMS リスナー** このリスナーは、JMS テキスト・メッセージおよび JMS バイト・メッセージをサポートします。

要求処理: コネクターは、プラグ可能なプロトコル・ハンドラーを使用して要求処理を行います。プロトコル・ハンドラーは SOAP データ・ハンドラーを呼び出します。このセクションでは、それぞれのハンドラーがトランスポートを介してどのように SOAP メッセージをエンコードするのかを説明します。

- **SOAP/HTTP-HTTPS ハンドラー** これらのハンドラーは SOAP データ・ハンドラーを呼び出します。Web サービス要求を構成するために、データ・ハンドラーによって戻されたストリングが、UTF 8 エンコード方式を使用してバイトに変換されます。同期 (要求/応答) Web サービスの場合、プロトコル・ハンドラーは HTTP 応答メッセージの本文を読み取ります。本文のエンコード方式は、HTTP Content-Type ヘッダーの charset パラメーターで指定されます。charset パラメーターが指定されていない場合には、ISO-8859-1 が使用されます。ハンドラーは、このエンコード方式を使用して応答メッセージの本文を Java ストリングに変換します。SOAP データ・ハンドラーは、このストリングを使用して呼び出されます。
- **SOAP/JMS ハンドラー** このハンドラーは、JMS テキスト・メッセージおよび JMS バイト・メッセージをサポートします。

SOAP データ・ハンドラー

このセクションでは、ローカリゼーションと SOAP データ・ハンドラーについて説明します。

SOAP の文字制限: XML の要素名と属性名は、ビジネス・オブジェクト名、ビジネス・オブジェクト属性名、またはビジネス・オブジェクト・アプリケーション固有情報のいずれかで許容される、適格な ASCII 文字でなければなりません。

国際化対応した文字は、ビジネス・オブジェクト名またはビジネス・オブジェクト属性名ではサポートされません。国際化対応できるのは属性値のみです。

SOAP データ・ハンドラーの処理: SOAP メッセージをビジネス・オブジェクトに変換するときに、データ・ハンドラーはストリングのみを受け取ることができます。データ・ハンドラーは、ビジネス・オブジェクトにストリング値を取り込み、そのビジネス・オブジェクトを戻します。Java ストリングは UCS2 であるため、2 バイト対応文字は問題なく転送されます。非 ASCII 文字を使用することが許されるのは、XML の要素および属性値のみです (『文字制限』を参照)。ビジネス・オブジェクトを SOAP メッセージに変換する場合、データ・ハンドラーは XML4J パーサーを使用してビジネス・オブジェクトをストリングに変換します。Java ストリングは UCS2 であるため、2 バイト対応文字は問題なく転送されます。非 ASCII 文字を使用することが許されるのは、XML の要素および属性値のみです (『文字制限』を参照)。

WSDL ODA

このセクションでは、ローカリゼーションと WSDL ODA について説明します。

WSDL ファイルでは、WSDL ODA により、ファイル名および URL が任意の文字セットでサポートされます。ビジネス・オブジェクトの名前と属性における非 ASCII 文字セットの制限により、WSDL ファイルの内容は適格な ASCII 文字でなければなりません。

WSDL ODA の「エージェントの構成 (Configuring Agent)」テーブル内のプロパティーは、以下のようにグローバル化対応されました。

- **WSDL_URL** URL はネイティブ言語であってもかまわない
- **UDDI_InquiryAPI_URL** UDDI レジストリー・サポートを検査
- **WebServiceProvider** 適格な ASCII 文字のみ

- **WebService** 適格な ASCII 文字のみ
- **MimeType** 適格な ASCII 文字のみ
- **BOPrefix** 適格な ASCII 文字のみ
- **BOVerb** 適格な ASCII 文字のみ
- **Collaboration** 適格な ASCII 文字のみ
- **GenerateUniqueBOs** 適格な ASCII 文字のみ
- **SOAPVersion** 適格な ASCII 文字のみ
- **BiDi.ExtApplicationMetaData** 適格な ASCII 文字のみ

用語

本書では、以下の用語を使用します。

- **ASI (アプリケーション固有情報)** は、特定のアプリケーションまたはテクノロジーに合わせて調整されたコードです。ASI は、ビジネス・オブジェクト定義の属性レベルとビジネス・オブジェクト・レベルの両方のレベルで存在します。
- **ASBO (アプリケーション固有ビジネス・オブジェクト)** ASI を含めることのできるビジネス・オブジェクト。
- **双方向 (BiDi) 言語**は、主として中東で使用されます。双方向言語には、アラビア語、ウルドゥー語、ペルシア語、ヘブライ語、イディッシュ語があります。双方向言語では、一般的な文字の流れが横方向で右から左になりますが、数字は英語の場合と同じく、左から右の方向に書かれます。また、英語やそのほかの左から右に書かれる言語の文字が組み込まれる場合 (住所、頭字語、引用など) は、その文字も左から右方向に書かれます。
- **BO (ビジネス・オブジェクト)** ビジネス・エンティティー (Customer など) およびデータへの処置 (作成または更新操作など) を表す属性の集合。システムのコンポーネントは、情報を交換したりアクションを起動したりするためにビジネス・オブジェクトを使用します。
- **Content-Type** *タイプ/サブタイプ* およびオプショナル・パラメーターが含まれる HTTP プロトコル・ヘッダー。例えば、Content-Type 値 `text/xml;charset=ISO-8859-1` では、`text/xml` はタイプ/サブタイプであり、`charset=ISO-8859-1` はオプションの Charset パラメーターです。
- **ContentType** Content-Type ヘッダー値の*タイプ/サブタイプ* 部分のみを指します。例えば、Content-Type 値 `text/xml;charset=ISO-8859-1` の `text/xml` は、本書では、ContentType とみなされます。
- **MO_DataHandler_DefaultSOAPConfig** SOAP データ・ハンドラー固有の子データ・ハンドラー・メタオブジェクトです。
- **GBO (汎用ビジネス・オブジェクト)** ASI を含まず、どのアプリケーションとも関連していないビジネス・オブジェクト。
- **MO_DataHandler_Default** コネクター・エージェントが、どのデータ・ハンドラーのインスタンスを生成するかを決定するために使用するデータ・ハンドラー・メタオブジェクトです。これは、コネクターの DataHandlerMetaObjectName 構成プロパティーで指定されます
- **非トップレベルのビジネス・オブジェクト (非 TLO)** 非 TLO は、Web サービス TLO 構造に従わないビジネス・オブジェクトです。

- **プロトコル構成 MO** 要求処理中に、SOAP/JMS および SOAP/HTTP-HTTPS プロトコル・ハンドラーは、ターゲット Web サービスの宛先を判別するために、プロトコル構成 MO を使用します。イベント処理中に、コラボレーションを SOAP/JMS Web サービスとして公開する場合、コネクターは、プロトコル構成 MO を使用して、JMS メッセージ・ヘッダー情報を SOAP/JMS プロトコル・リスナーからコラボレーションへ転送します。
- **SOAP (Simple Object Access Protocol)** では、単純な要求メッセージと応答メッセージを使用するモデルが定義されます。これらのメッセージは、電子通信の基本プロトコルとして、XML で記述されます。SOAP メッセージングは、プラットフォームに無関係なリモート・プロシージャー呼び出し (RPC) 機能ですが、任意の種類の XML 情報の交換 (文書の交換) にも利用できます。
- **SOAP ビジネス・オブジェクト** SOAP ビジネス・オブジェクトは TLO の子であり、SOAP 要求、SOAP 応答、および SOAP 障害ビジネス・オブジェクトのいずれかになります。SOAP ビジネス・オブジェクトには、SOAP 構成 MO (SOAP ビジネス・オブジェクトの子) などの、SOAP データ・ハンドラーによる処理のために必要な情報が含まれ、また、SOAP ヘッダー・コンテナー・ビジネス・オブジェクトも含まれています。
- **SOAP 構成 MO (構成メタオブジェクト)** データ・ハンドラーは、(例えば SOAP メッセージから SOAP ビジネス・オブジェクトなどへの) 単一の変換に関する構成情報を含むオブジェクトを必要とします。この情報は、SOAP ビジネス・オブジェクトの子にメタデータとして保管されます。この子オブジェクトが SOAP 構成 MO です。
- **SOAP ヘッダー子ビジネス・オブジェクト** SOAP メッセージ内の単一のヘッダー要素を表すビジネス・オブジェクト。ヘッダー要素は、SOAP メッセージの SOAP-Env:Header 要素の直接の子です。ヘッダー・コンテナー・ビジネス・オブジェクトのすべての属性は、この型に属しています。これらのビジネス・オブジェクトには、actor および mustUnderstand 属性が割り当てられることがあります。これらの属性は、SOAP ヘッダー要素の actor および mustUnderstand 属性に対応しています。
- **SOAP ヘッダー・コンテナー・ビジネス・オブジェクト** SOAP メッセージのヘッダーに関する情報が格納されているビジネス・オブジェクト。このビジネス・オブジェクトには 1 つ以上の子ビジネス・オブジェクトが含まれます。それぞれの子ビジネス・オブジェクトは、SOAP メッセージのヘッダー項目を表します。SOAP データ・ハンドラー・ビジネス・オブジェクトには、SOAP ヘッダー・コンテナー・ビジネス・オブジェクトと同じタイプの属性が割り当てられることがあります。この属性は、SOAP ヘッダー属性とも呼ばれます。このような属性は、125 ページの『第 5 章 SOAP データ・ハンドラー』で述べるように、特別なアプリケーション固有情報を必要とします。この属性は、SOAP ビジネス・オブジェクトの直接の子でなければなりません。
- **トップレベル・ビジネス・オブジェクト (TLO)** Web サービスのトップレベル・ビジネス・オブジェクトには、1 つの SOAP 要求、1 つの SOAP 応答 (オプション)、および 1 つ以上の SOAP 障害 (オプション) ビジネス・オブジェクトが含まれます。TLO は、コネクターがイベント処理および要求処理の両方で使用します。
- **Web サービス**とは、内蔵タイプの動的な分散型モジュラー・アプリケーションのことです。このアプリケーションは、ネットワークを介して記述、パブリッシュ、配置、呼び出しを実行することにより、製品、プロセス、サプライ・チェー

ンを作成できます。製品、プロセス、サプライ・チェーンの種類は、ローカル、分散、Web ベースのいずれでも構いません。Web サービスは、TCP/IP、HTTP、Java、HTML、XML などのオープン・スタンダードを基盤として作成されます。Web サービスには、メッセージングについては SOAP (Simple Object Access Protocol)、パブリッシュについては UDDI (Universal Description, Discovery and Integration) や WSDL (Web Service Description Language) のように、新しい標準技術が採用されています。

- **UDDI (Universal Description, Discovery and Integration)** とは、Web サービスに関する情報の公開方法や検索方法を定義する仕様のことです。UDDI 仕様では、XML ベースのインターフェース (API) を提供することにより、UDDI 登録情報に対して、方針に基づいたアクセスが可能になります。SOAP は、これらの API の基盤となる RPC 機能です。
- **WSDL (Web サービス記述言語)** とは、Web サービス向けのソフトウェア・インターフェースを定義する XML 用語のことです。WSDL では、プログラミング・レベルでの自動的な統合に必要な Web サービス技術の詳細がすべて系統化されています。また、WSDL は、WebSphere コラボレーションを Web サービスとして公開するときにも使用します。WSDL と Web サービスとの関係は、IDL と CORBA オブジェクトとの関係に相当します。

  WSDL について詳しくは、次の URL にアクセスしてください。

  http://www.w3.org/TR/wsdl

Connector for Web Services のコンポーネント

図 1 は、Connector for Web Services を例示したもので、プロトコル・ハンドラー・フレームワーク、プロトコル・リスナー・フレームワーク、および SOAP データ・ハンドラーが含まれています。

注: Web Services Adapter には、XML データ・ハンドラーの限定使用ライセンスが付いています。ただし、このアダプターは、XML データ・ハンドラーがなくても機能します。

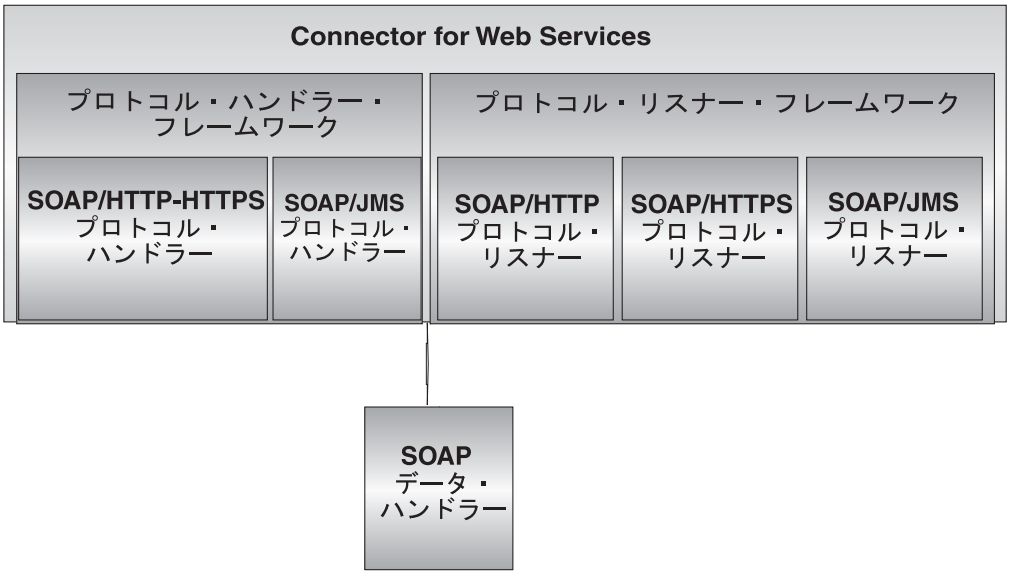


図 1. Connector for Web Services

以下のコンポーネントは、相互に作用することにより、インターネットを介したデータ交換を可能にします。

- SOAP データ・ハンドラーおよびプロトコルのリスナーとハンドラーを含む、Web サービス・コネクター
- Web サービス対応のコラボレーション
- ビジネス・オブジェクトおよび SOAP メッセージ
- InterChange Server Express

Web サービス・コネクター

要求処理中に、Web サービス・コネクターは、ビジネス・オブジェクトを SOAP 要求メッセージに変換し、それらを宛先 Web サービスに転送することによって、コラボレーション・サービス呼び出しに応答します。オプションで (同期要求処理の場合)、コネクターは、SOAP 応答メッセージを SOAP 応答ビジネス・オブジェクトに変換し、これらをコラボレーションに戻します。

イベント処理中に、コネクターは、SOAP 要求メッセージを SOAP 要求ビジネス・オブジェクトに変換し、それらを処理するためにコラボレーション (Web サービスとして公開されている) に引き渡すことによって、クライアント Web サービスからの SOAP 要求メッセージを処理します。コネクターは、オプションで SOAP 応答ビジネス・オブジェクトをコラボレーションから受け取ります。これらの SOAP 応答ビジネス・オブジェクトは、SOAP 応答メッセージに変換されてから、クライアント Web サービスに戻されます。

詳しくは、 65 ページの『第 4 章 Web サービス・コネクター』を参照してください。

注: この資料でコネクターと言う場合、特に指定のないかぎり、Web サービス・コネクターのことを指します。

プロトコル・リスナーおよびハンドラー

コネクターには、以下のプロトコル・リスナーおよびハンドラーが組み込まれています。

- SOAP/HTTP プロトコル・リスナー
- SOAP/HTTPS プロトコル・リスナー
- SOAP/JMS プロトコル・リスナー
- SOAP/HTTP-HTTPS プロトコル・ハンドラー
- SOAP/JMS プロトコル・ハンドラー

プロトコル・リスナーは、内部または外部 Web サービス・クライアントから、SOAP/HTTP、SOAP/HTTPS、または SOAP/JMS フォーマットのイベントを検出します。これらは、コネクターに対し、Web サービスとして公開されているコラボレーションによる処理が必要なイベントを通知します。その後、プロトコル・リスナーは、ビジネス・オブジェクト・レベルおよび属性レベル ASI、コネクター・プロパティー、およびプロトコル構成オブジェクトに組み込まれた変換規則を読み取って、コラボレーション、データ・ハンドラー、処理モード (同期/非同期)、および Web サービス・トランザクションのトランスポート固有の性質を判別します。プロトコル・リスナー処理の詳細な説明については、70 ページの『プロトコル・リスナー』を参照してください。

プロトコル・ハンドラーは、コラボレーションのために、Web サービスを SOAP/HTTP、SOAP/HTTPS、または SOAP/JMS フォーマットで呼び出します。プロトコル・ハンドラーは、TLO ASI およびプロトコル構成オブジェクトに組み込まれた変換規則を読み取って、データ・ハンドラーが SOAP メッセージから SOAP ビジネス・オブジェクトへの変換 (およびその逆の変換) に使用する要求の処理方法 (同期または非同期) を決定し、また、Web サービスのターゲット・アドレスを (SOAP 要求ビジネス・オブジェクトのプロトコル構成 MO の宛先属性から) 決定します。同期トランザクションの場合、プロトコル・ハンドラーは、SOAP 応答メッセージを処理して、SOAP 応答ビジネス・オブジェクトに変換し、コラボレーションに戻します。

プロトコル・ハンドラーについて詳しくは、84 ページの『プロトコル・ハンドラー』を参照してください。

SOAP データ・ハンドラー

SOAP データ・ハンドラーは、SOAP ビジネス・オブジェクトから SOAP への変換およびその逆の変換を行います。SOAP データ・ハンドラーについて詳しくは、125 ページの『第 5 章 SOAP データ・ハンドラー』を参照してください。

Web サービス構成ツール

Web サービスを呼び出したり、Web サービスとして公開されたりするコラボレーションを使用して、Web サービス・ソリューションを配置することができます。

要求処理のコラボレーションを使用可能にする場合は、WSDL Object Discovery Agent (ODA) を使用して Web サービス TLO を生成します。要求処理および WSDL ODA について詳しくは、165 ページの『第 6 章 要求処理のためのコラボレーションの有効化』を参照してください。

コラボレーションを Web サービスとして公開する場合は、WSDL 構成ウィザードを使用します。このウィザードは、後で、例えば、UDDI レジストリーを介して公開する、コラボレーション用の WSDL 文書の生成に役立ちます。コネクターは、この情報を公開するためのツールを備えていません。コラボレーションを Web サービスとして公開する場合の詳細は、167 ページの『第 7 章 Web サービスとしてのコラボレーションの公開』を参照してください。

コネクターの配置

Web サービス・コネクターを配置するには、以下の 2 つの方法があります。

- ファイアウォールの内側で、ビジネス・プロセス同士が SOAP/HTTP、SOAP/HTTPS、または SOAP/JMS Web サービス・フォーマットで通信しているエンタープライズ内のイントラネット・ベースのソリューション (図 2 を参照) として。

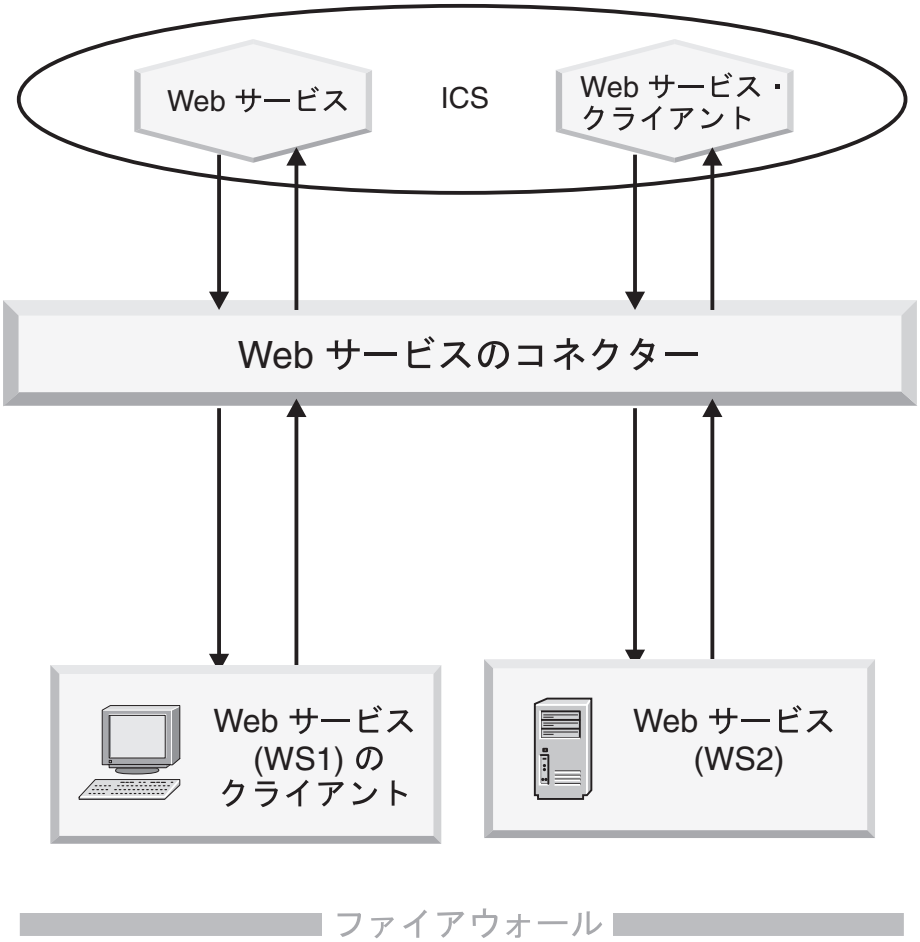


図 2. イントラネット・ソリューションとしての Web サービス・アダプター

- ファイアウォールの内側で、エンタープライズにとって外部の Web サービスとの通信を、処理、フィルター操作、または管理するフロントエンドまたはゲートウェイ・サーバーと連携して。

注: Web サービス・コネクターには、外部 Web サービスからの着信メッセージ、または外部 Web サービスへの発信メッセージを管理するためのゲートウェイまたはフロントエンドが組み込まれていません。ユーザー独自のゲートウェイを構成および配置する必要があります。***コネクターは、DMZ (demilitarized zone) 内やファイアウォールの外側ではなく、エンタープライズ内でのみ配置する必要があります。***

Connector for Web Services のアーキテクチャー

上位レベルのコンポーネントのアーキテクチャーを示すため、このセクションでは、2 つのデータ・フローについて説明します。図 3 に 2 つのシナリオを示します。これら 2 つのコンポーネントについて、以下で説明します。

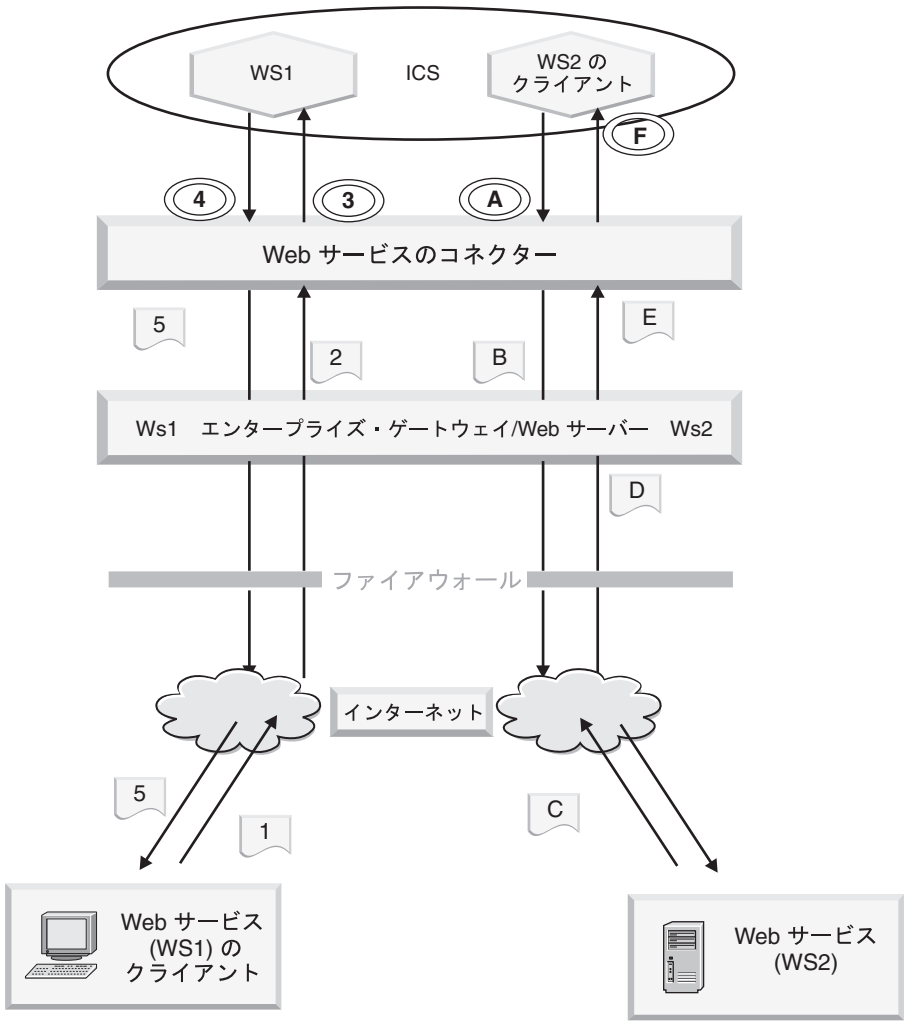


図 3. Web サービス・メッセージの流れ

要求処理は、コラボレーションが Web サービスを呼び出すために、コネクターに対してサービス呼び出し要求を実行したときに発生するイベントの流れを表しています。このシナリオでは、コラボレーションの果たす役割はクライアントであり、要求をサーバーに送信しています。

A　コラボレーションはサービス呼び出し要求をコネクターに送信し、コネクターは SOAP データ・ハンドラーを呼び出してビジネス・オブジェクトを SOAP 要求メッセージに変換します。

B　コネクターは、SOAP メッセージを送信して Web サービス WS2 を呼び出します。宛先が外部 Web サービスである場合、コネクターは SOAP メッセージをゲートウェイに送信します。ゲートウェイは、宛先の Web サービスに対応するエンドポイントに SOAP メッセージを送信します。これにより、Web サービスが呼び出されます。

C　呼び出された Web サービスは、SOAP 要求メッセージを受信して、要求された処理を実行します。

D　呼び出された Web サービスは、SOAP 応答 (または障害) メッセージを送信します。Web サービスがエンタープライズにとって外部の場合は、ゲートウェイは SOAP 応答メッセージを受信して発送します。

E　SOAP 応答 (または障害) メッセージは、コネクターに返送されます。コネクターは、データ・ハンドラーを呼び出して、このメッセージを応答または障害ビジネス・オブジェクトに変換します。

F　コネクターは、SOAP 応答または障害ビジネス・オブジェクトをコラボレーションに戻します。

イベント処理は、コラボレーションが Web サービスとして呼び出されたときに発生する、イベントの流れを表しています。このシナリオでは、コラボレーション (Web サービスとして公開されています) はサーバーの役割を果たし、(外部または内部) クライアントからの要求を受け入れ、必要に応じて応答を行います。

1　クライアント Web サービス (WS1) は、コラボレーションに対して生成された WSDL 文書に指定されている宛先に向けて SOAP 要求メッセージを送信します。

2　クライアント Web サービスが外部の場合、ゲートウェイはメッセージを受信してコネクターに発送します。

3　コネクターは、SOAP メッセージを SOAP データ・ハンドラーに送信して、SOAP メッセージをビジネス・オブジェクトに変換します。コネクターは、Web サービスとして公開されたコラボレーションを呼び出します。

4　コラボレーションは、SOAP 応答 (または障害) ビジネス・オブジェクトを戻します。

5　コネクターは、SOAP データ・ハンドラーを呼び出して、SOAP 応答 (または障害) ビジネス・オブジェクトを SOAP 応答メッセージに変換します。コネクターは、応答をゲートウェイに戻します。

6　クライアント Web サービスが外部の場合、ゲートウェイは SOAP 応答メッセージをクライアント Web サービス (WS1) に発送します。

インストール、構成、および設計のチェックリスト

このセクションでは、Web サービス・ソリューションをインストール、構成、および設計するときに行う必要のある作業を要約します。各セクションでは、作業を簡単に説明したうえで、その作業の実行方法や背景情報を説明する、本書中の該当セクションへのリンク (および関連資料への相互参照) を提供します。

アダプターのインストール

何をどこにインストールする必要があるのかについては、17 ページの『第 2 章 インストールおよび始動』を参照してください。

コネクター・プロパティーの構成

コネクターには、2 種類の構成プロパティーがあります。標準の構成プロパティーと、コネクター固有の構成プロパティーです。これらのプロパティーの一部には、デフォルト値が設定されていて、その値を変更する必要はありません。また、コネクターを実行する前に値を設定しなければならないプロパティーもあります。詳しくは、65 ページの『第 4 章 Web サービス・コネクター』を参照してください。

プロトコル・ハンドラーおよびプロトコル・リスナーの構成

プロトコル・ハンドラーおよびプロトコル・リスナーの構成は、これらのコンポーネントの振る舞いを制御するコネクター構成プロパティーに値を割り当てるときに行います。詳しくは、65 ページの『第 4 章 Web サービス・コネクター』を参照してください。

Web サービス向けコラボレーションの使用可能化

コラボレーションを Web サービスで使用可能にするには、Web サービスを呼び出すことのできるコラボレーション、あるいは Web サービスとして公開することのできるコラボレーションを作成してください。また、ビジネス・オブジェクトの作成あるいは改造も行ってください。関連する作業の概要については、10 ページの『Web サービス構成ツール』を参照してください。

Web サービスとしてのコラボレーションの公開

手順の説明については、167 ページの『第 7 章 Web サービスとしてのコラボレーションの公開』を参照してください。

Web サービスを呼び出すコラボレーションの使用可能化

手順の説明については、165 ページの『第 6 章 要求処理のためのコラボレーションの有効化』を参照してください。

SOAP データ・ハンドラーの構成

データ・ハンドラー・メタオブジェクト内の情報は、製品ファイルをインストールしてから始動するまでの間に構成してください。カスタム名ハンドラーを追加しないのであれば、デフォルトの SOAP データ・ハンドラー構成を使用して時間を節約することができます。ただし、それぞれのデータ・ハンドラーを変換するたびに特定のメタオブジェクト情報を構成しなければなりません。この情報は SOAP 構成 MO に収納されます。SOAP 構成 MO は、ビジネス・オブジェクトの作成時に指定します。この作業の大半は、Web サービス (要求処理) を呼び出すコラボレーションを開発するときに自動的に行われます。SOAP メッセージ用のビジネス・オブジェクトを生成するために WSDL ODA を使用すると、SOAP 構成 MO が自動的に生成されます。

データ・ハンドラーの構成に関する詳細については、125 ページの『第 5 章 SOAP データ・ハンドラー』を参照してください。

制約事項

- WSDL ODA は自動的にビジネス・オブジェクトを生成します。結果が要件に一致しない場合、Business Object Designer Express を使用してビジネス・オブジェクトを手動で更新または作成する必要があります。

  属性のスタイル、使用法、および型と要素を使用したパーツ定義のさまざまな組み合わせについては、WSDL ODA サポートの説明を参照してください。
- スタイル (rpc、document)、使用法 (リテラル、エンコード)、およびパーツの定義方法に関する XML の制約事項については、125 ページの『第 5 章 SOAP データ・ハンドラー』および 165 ページの『第 6 章 要求処理のためのコラボレーションの有効化』を参照してください。
- コネクターは、SOAP/HTTP および SOAP/JMS バインディングのみをサポートします。
- コネクターの SOAP/JMS プロトコル・リスナーは、キュー宛先のみをサポートします。トピックはサポートされません。JMS テキスト・メッセージおよびバイト・メッセージはサポートされます。
- HTTP POST 要求および応答はサポートされます。これ以外の HTTP メソッドはサポートされません。HTTP 1.1 の永続接続はサポートされません。

第 2 章 インストールおよび始動

- 『インストール作業の概要』
- 『コネクターおよび関連ファイルのインストール』
- 21 ページの『構成作業の概要』
- 22 ページの『アダプターの複数インスタンスの実行』
- 24 ページの『コネクターの始動』

この章では、Connector for Web Services をインプリメントするためのコンポーネントのインストール方法について説明します。WebSphere Business Integration Server Express のインストールの詳細については、ご使用のプラットフォームに合った「*WebSphere Business Integration Server Express インストール・ガイド*」(Windows 版、Linux 版、OS/400 および i5/OS 版) を参照してください。

インストール作業の概要

ブローカーの互換性、アダプター・フレームワーク、ソフトウェア前提条件、依存関係、および標準と API については、 2 ページの『Adapter for Web Services の環境』 を参照してください。

Connector for Web Services 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。

InterChange Server Express のインストール

この作業には、システムのインストールおよび InterChange Server Express の始動が含まれます。この作業の説明は、「WebSphere Business Integration Server Express インストール・ガイド」(Windows 版、Linux 版、OS/400 および i5/OS 版) にあります。 WebSphere Business Integration Server Express バージョン 4.4 をインストールする必要があります。

リポジトリーにファイルをロードするには、「 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。

コネクターおよび関連ファイルのインストール

この作業の内容は、ソフトウェア・パッケージからコネクター (および関連コンポーネント) のファイルを使用システムにインストールすることです。『コネクターおよび関連ファイルのインストール』を参照してください。

コネクターおよび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の詳細については、「*Websphere Business Integration Server Express インストール・ガイド*」*(Windows 版、Linux 版、OS/400 および i5/OS 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。

インストール済みファイルの構造

このセクションの表では、インストール済みファイルの構造を示します。

Windows コネクターのファイル構造

インストーラーにより、コネクターに関連付けられている標準ファイルがシステムにコピーされます。

ユーティリティーにより、コネクターがインストールされ、コネクター・エージェントのショートカットが「スタート」メニューに追加されます。

表 1 には、コネクターが使用する Windows ファイルの構造が説明されており、インストーラーによるコネクターのインストールを選択したときに自動的にインストールされるファイルが示されています。

表 1. アダプター用のインストール済み Windows ファイルの構造

***ProductDir* のサブディレクトリー**	**説明**
connectors¥WebServices¥CWWebServices.jar	Web サービス・コネクター
connectors¥WebServices¥start_WebServices.bat	コネクター・サービスの始動ファイル
connector¥WebServices¥start_WebServices_service.bat	コネクター・サービスの始動スクリプト
connectors¥WebServices¥README.htm	使用権
DataHandlers¥CwSOAPDataHandler.jar	SOAP データ・ハンドラー
repository¥DataHandlers¥MO_DataHandler_DefaultSOAPConfig.xsd	SOAP データ・ハンドラーの関連ファイル
bin¥Data¥App¥WebServicesConnectorTemplate	Web サービス・コネクターのテンプレート
ODA¥WSDL¥WSDLODA.jar	WSDL ODA
ODA¥WSDL¥start_WSDLODA.bat	WSDL ODA 始動ファイル
connectors¥WebServices¥dependencies¥soap.jar	SOAP コネクター、SOAP データ・ハンドラー、WSDL 構成ウィザード、および WSDL ODA に必要な Apache SOAP API
connectors¥WebServices¥dependencies¥LICENSE	Apache ライセンス・ファイル
connectors¥WebServices¥dependencies¥mail.jar	JavaMail API
connectors¥WebServices¥dependencies¥activation.jar	Java Activation Framework
connectors¥WebServices¥dependencies¥jms.jar	Java Messaging Service
connectors¥WebServices¥dependencies¥uddi4j-wsdl_2_1_0_040127.jar	WSDL ODA に必要
connectors¥WebServices¥dependencies¥uddi4jv2.jar	WSDL ODA に必要
connectors¥WebServices¥dependencies¥IPL10.txt	WSDL ODA に必要なライセンス・ファイル
connectors¥WebServices¥dependencies¥wsdl4j-1.4SR3.jar	WSDL ODA に必要
connectors¥WebServices¥dependencies¥CPL10.txt	WSDL ODA に必要なライセンス・ファイル
connectors¥WebServices¥dependencies¥qname.jar	WSDL ODA に必要
connectors¥WebServices¥dependencies¥j2ee.jar	WSDL ODA に必要
connectors¥WebServices¥dependencies¥wswb3.0¥common.jar	WSDL ODA に必要
connectors¥WebServices¥dependencies¥wswb3.0¥ecore.jar	WSDL ODA に必要
connectors¥WebServices¥dependencies¥wswb3.0¥xsd.jar	WSDL ODA に必要
connectors¥WebServices¥dependencies¥wswb3.0¥xsd.resources.jar	WSDL ODA に必要
connectors¥WebServices¥dependencies¥IBMReadme_de_DE.txt	ドイツ語 README ファイル
connectors¥WebServices¥dependencies¥IBMReadme_en_US.txt	英語 README ファイル
connectors¥WebServices¥dependencies¥IBMReadme_ex_ES.txt	スペイン語 README ファイル
connectors¥WebServices¥dependencies¥IBMReadme_fr_FR.txt	フランス語 README ファイル
connectors¥WebServices¥dependencies¥IBMReadme_it_IT.txt	イタリア語 README ファイル
connectors¥WebServices¥dependencies¥IBMReadme_ja_JA.txt	日本語 README ファイル
connectors¥WebServices¥dependencies¥IBMReadme_ko_KR.txt	韓国語 README ファイル
connectors¥WebServices¥dependencies¥IBMReadme_pt_BR.txt	ポルトガル (ブラジル) 語 README ファイル
connectors¥WebServices¥dependencies¥IBMReadme_zh_CN.txt	中国語 (簡体字) README ファイル
connectors¥WebServices¥dependencies¥IBMReadme_zh_TW.txt	中国語 (繁体字) README ファイル
connectors¥WebServices¥samples¥WebSphereICS¥CLIENT_SYNCH_TLO_OrderStatus.bo	Test Connector 用のサンプル (同期) ビジネス・オブジェクト
connectors¥messages¥WebServicesConnector.txt	コネクター・メッセージ・ファイル
ODA¥messages¥WSDLODAAgent.txt	WSDL ODA のメッセージ・ファイル

注: すべての製品パス名は、製品がシステムにインストールされているディレクトリーを基準にした相対パス名です。

i5/OS コネクターのファイル構造

インストーラーにより、コネクターに関連付けられている標準ファイルがシステムにコピーされます。

表 2 に、コネクターが使用する i5/OS ファイル構造が記載されており、インストーラーを介したコネクターのインストールを選択した際に自動的にインストールされるファイルを示します。

表 2. アダプター用のインストール済み i5/OS ファイルの構造

***ProductDir* のサブディレクトリー**	**説明**
/lib/WBIA.jar	WebSphere Business Integration Server Express Adapter JAR ファイル
/bin/CWConnEnv.sh	汎用コネクターの始動ファイル
/bin/CWODAEnv.sh	汎用 ODA 始動ファイル
/connectors/WebServices/README.htm	使用権
connectors/WebServices/CWWebServices.jar	Web サービス・コネクター
connectors/WebServices/start_WebServices.sh	コネクターの始動ファイル
DataHandlers/CwSOAPDataHandler.jar	SOAP データ・ハンドラー
DataHandlers/CwSOAPNameHandler.jar	SOAP 名前ハンドラー
bin/Data/App/WebServices	Web サービス・コネクターのテンプレート
ODA/WSDL/WSDLODA.jar	WSDL ODA
ODA/WSDL/start_WSDLODA.sh	WSDL ODA 始動ファイル
connectors/WebServices/dependencies/soap.jar	SOAP コネクター、SOAP データ・ハンドラー、WSDL 構成ウィザード、および WSDL ODA に必要な Apache SOAP API
connectors/WebServices/dependencies/LICENSE	Apache ライセンス・ファイル
connectors/WebServices/dependencies/mail.jar	JavaMail API
connectors/WebServices/dependencies/activation.jar	Java Activation Framework
connectors/WebServices/dependencies/ibmjsse.jar	IBM 製の JSSE (Java Secure Socket Extension) API
connectors/WebServices/dependencies/jms.jar	Java Messaging Service
connectors/WebServices/dependencies/uddi4j-wsdl_2_1_0_040127.jar	WSDL ODA に必要
connectors/WebServices/dependencies/uddi4jv2.jar	WSDL ODA に必要
connectors/WebServices/dependencies/IPL10.txt	WSDL ODA に必要なライセンス・ファイル
connectors/WebServices/dependencies/wsdi4j.jar	WSDL ODA に必要
connectors/WebServices/dependencies/CPL10.txt	WSDL ODA に必要なライセンス・ファイル
connectors/WebServices/dependencies/qname.jar	WSDL ODA に必要
connectors/WebServices/dependencies/j2ee.jar	WSDL ODA に必要
connectors/WebServices/dependencies/wswb3.0/common.jar	WSDL ODA に必要
connectors/WebServices/dependencies/wswb3.0/ecore.jar	WSDL ODA に必要
connectors/WebServices/dependencies/wswb3.0/xsd.jar	WSDL ODA に必要
connectors/WebServices/dependencies/wswb3.0/xsd.resources.jar	WSDL ODA に必要
connectors/WebServices/dependencies/IBMReadme_de_DE.txt	ドイツ語 README ファイル
connectors/WebServices/dependencies/IBMReadme_en_US.txt	英語 README ファイル。
connectors/WebServices/dependencies/IBMReadme_ex_ES.txt	スペイン語 README ファイル
connectors/WebServices/dependencies/IBMReadme_fr_FR.txt	フランス語 README ファイル
connectors/WebServices/dependencies/IBMReadme_it_IT.txt	イタリア語 README ファイル
connectors/WebServices/dependencies/IBMReadme_ja_JA.txt	日本語 README ファイル
connectors/WebServices/dependencies/IBMReadme_ko_KR.txt	韓国語 README ファイル
connectors/WebServices/dependencies/IBMReadme_pt_BR.txt	ポルトガル (ブラジル) 語 README ファイル
connectors/WebServices/dependencies/IBMReadme_zh_CN.txt	中国語 (簡体字) README ファイル
connectors/WebServices/dependencies/IBMReadme_zh_TW.txt	中国語 (繁体字) README ファイル
connectors/messages/WebServicesConnector.txt	コネクター・メッセージ・ファイル
ODA/messages/WSDLODAAgent.txt	WSDL ODA のメッセージ・ファイル

注: すべての製品パス名は、製品がシステムにインストールされているディレクトリーを基準にした相対パス名です。

コネクターをすばやく開始するには、WebSphere Business Integration Server Express Console を使用します。Console の詳細については、Console に付属するオンライン・ヘルプを参照してください。

インストールの詳細については、「*WebSphere Business Integration Server Express インストール・ガイド (OS/400 および i5/OS 版)*」を参照してください。

Linux コネクターのファイル構造

インストーラーにより、コネクターに関連付けられている標準ファイルがシステムにコピーされます。

表 3 には、コネクターが使用する Linux ファイルの構造が説明されており、インストーラーによるコネクターのインストールを選択したときに自動的にインストールされるファイルが示されています。

表 3. アダプター用のインストール済み Linux ファイルの構造

***ProductDir* のサブディレクトリー**	**説明**
connectors/WebServices/CWWebServices.jar	Web サービス・コネクター
connectors/WebServices/start_WebServices.sh	コネクターの始動ファイル
connectors/WebServices/README.htm	使用権
DataHandlers/CwSOAPDataHandler.jar	SOAP データ・ハンドラー
repository/DataHandlers/MO_DataHandler_DefaultSOAPConfig.xsd	SOAP データ・ハンドラーの関連ファイル
bin/Data/App/WebServicesConnectorTemplate	Web サービス・コネクターのテンプレート
ODA/WSDL/WSDLODA.jar	WSDL ODA
ODA/WSDL/start_WSDLODA.sh	WSDL ODA 始動ファイル
connectors/WebServices/dependencies/soap.jar	SOAP コネクター、SOAP データ・ハンドラー、WSDL 構成ウィザード、および WSDL ODA に必要な Apache SOAP API
connectors/WebServices/dependencies/LICENSE	Apache ライセンス・ファイル
connectors/WebServices/dependencies/mail.jar	JavaMail API
connectors/WebServices/dependencies/activation.jar	Java Activation Framework
connectors/WebServices/dependencies/jms.jar	Java Messaging Service
connectors/WebServices/dependencies/uddi4j-wsdl_2_1_0_040127.jar	WSDL ODA に必要
connectors/WebServices/dependencies/uddi4jv2.jar	WSDL ODA に必要
connectors/WebServices/dependencies/IPL10.txt	WSDL ODA に必要なライセンス・ファイル
connectors/WebServices/dependencies/wsdl4j-1.4SR3.jar	WSDL ODA に必要
connectors/WebServices/dependencies/CPL10.txt	WSDL ODA に必要なライセンス・ファイル
connectors/WebServices/README.htm	ライセンス
connectors/WebServices/dependencies/qname.jar	WSDL ODA に必要
connectors/WebServices/dependencies/j2ee.jar	WSDL ODA に必要
connectors/WebServices/dependencies/wswb3.0/common.jar	WSDL ODA に必要
connectors/WebServices/dependencies/wswb3.0/common.jar	WSDL ODA に必要
connectors/WebServices/dependencies/wswb3.0/ecore.jar	WSDL ODA に必要
connectors/WebServices/dependencies/wswb3.0/xsd.jar	WSDL ODA に必要
connectors/WebServices/dependencies/wswb3.0/xsd.resources.jar	WSDL ODA に必要
connectors/WebServices/dependencies/IBMReadme_de_DE.txt	ドイツ語 README ファイル
connectors/WebServices/dependencies/IBMReadme_en_US.txt	英語 README ファイル
connectors/WebServices/dependencies/IBMReadme_ex_ES.txt	スペイン語 README ファイル
connectors/WebServices/dependencies/IBMReadme_fr_FR.txt	フランス語 README ファイル
connectors/WebServices/dependencies/IBMReadme_it_IT.txt	イタリア語 README ファイル
connectors/WebServices/dependencies/IBMReadme_ja_JA.txt	日本語 README ファイル
connectors/WebServices/dependencies/IBMReadme_ko_KR.txt	韓国語 README ファイル
connectors/WebServices/dependencies/IBMReadme_pt_BR.txt	ポルトガル (ブラジル) 語 README ファイル
connectors/WebServices/dependencies/IBMReadme_zh_CN.txt	中国語 (簡体字) README ファイル
connectors/WebServices/dependencies/IBMReadme_zh_TW.txt	中国語 (繁体字) README ファイル

表 3. アダプター用のインストール済み Linux ファイルの構造 (続き)

ProductDir のサブディレクトリー	説明
connectors/WebServices/dependencies/wswb3.0/ecore.jar	WSDL ODA に必要
connectors/WebServices/dependencies/wswb3.0/xsd.jar	WSDL ODA に必要
connectors/WebServices/dependencies/wswb3.0/xsd.resources.jar	WSDL ODA に必要
connectors/messages/WebServicesConnector.txt	コネクター・メッセージ・ファイル
ODA/messages/WSDLODAAgent.txt	WSDL ODA のメッセージ・ファイル

注: すべての製品パス名は、製品がシステムにインストールされているディレクトリーを基準にした相対パス名です。

構成作業の概要

インストールしたら、始動する前に、コンポーネントを以下のように構成する必要があります。

コネクターの構成

この作業の内容は、コネクターのセットアップと構成です。 97 ページの『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・オブジェクトの構成手順は、製品スイートのインプリメント方法をどのように選択するかによって異なります。

- **要求処理** 以下の内容に対応するビジネス・オブジェクトを作成する必要があります。
  - 各 Web サービスへ送信される要求メッセージ
  - 要求に対して考えられる応答 (障害の場合も含む)

  詳しくは、 27 ページの『第 3 章 ビジネス・オブジェクトの要件』を検討してから、 165 ページの『第 6 章 要求処理のためのコラボレーションの有効化』を参照してください。
- **イベント処理** TLO または非 TLO ビジネス・オブジェクトを使用できます。詳しくは、 27 ページの『第 3 章 ビジネス・オブジェクトの要件』を検討してから、 167 ページの『第 7 章 Web サービスとしてのコラボレーションの公開』を参照してください。

データ・ハンドラーの構成

インストールが終了したら、SOAP データ・ハンドラーのメタオブジェクトを構成する必要があります。さらに、それぞれの SOAP ビジネス・オブジェクトごとに、SOAP 構成 MO を構成する必要があります。データ・ハンドラーを構成するには、 125 ページの『第 5 章 SOAP データ・ハンドラー』を参照してください。

コラボレーションの構成

- **要求処理** 処理の一部として Web サービスを呼び出すコラボレーションの場合には、WSDL ODA を使用してビジネス・オブジェクトを生成してから、コラボレーション・オブジェクトのポートをそのコネクターにバインドします。ステップ

ごとの手順などの、詳しい説明については、 165 ページの『第 6 章 要求処理のためのコラボレーションの有効化』を参照してください。

- **イベント処理** Web サービスとして公開されるコラボレーションの場合には、WSDL 構成ウィザードを使用して WSDL 文書を生成し、潜在的な顧客がその文書を利用できるようにしてから、顧客がコラボレーションを呼び出せるようにコラボレーション・オブジェクトのポートを構成する必要があります。ステップごとの手順などの、詳しい説明については、 167 ページの『第 7 章 Web サービスとしてのコラボレーションの公開』を参照してください。

アダプターの複数インスタンスの実行

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

- **Windows プラットフォームの場合**:

  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 サーバー名は、例えば start_WEBSERVICES.bat connName serverName のように、startup.bat のパラメーターとして指定できます。

- **i5/OS プラットフォームの場合:**

  `/QIBM/UserData/WBIServer44/WebShereICSName/connectors/connectorInstance`

  ここで、connectorInstance はコネクター・インスタンスを一意的に示し、WebSphereICSName はコネクターの実行に使用する Interchange Server Express インスタンスの名前です。

  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。

/QIBM/UserData/WBIServer44/WebSphereICSName/repository/connectorInstance。ここで WebSphereICSName はコネクターの実行に使用する Interchange Server Express インスタンスの名前です。

- **Linux プラットフォームの場合**:
  ProductDir/connectors/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。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、ディレクトリー ProductDir/repository/connectorInstance を作成し、ファイルをここに格納します。InterChange Server Express のサーバー名を connector_manager のパラメーターとして指定することができます。例: connector_manager -start connName WebSphereICSName [-cConfigFile]。

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、『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. (Windows の場合のみ) 始動スクリプトのショートカットを作成します。

4. (Windows の場合のみ) 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。
5. (i5/OS の場合のみ) 以下の情報を使用して、コネクターのジョブ記述を作成します。
   CRTDUPOBJ(QWBIWEBSERVICES)
   FROMLIB(QWBISVR44)OBJTYPE(*JOBD)TOLIB (QWBISVR44)
   NEWOBJ(newwebservicesname)。ここで、newwebservicesname は新規コネクターのジョブ記述で使用する 10 文字の名前です。
6. (i5/OS の場合のみ) 新規コネクターを WebSphere Business Integration Server Express Console に追加します。WebSphere Business Integration Server Express Console の詳細については、コンソールに付属のオンライン・ヘルプを参照してください。

コネクターの始動

重要: この章で前述したように、コネクター、ビジネス・オブジェクト、SOAP データ・ハンドラーのメタオブジェクト、およびコラボレーションは、適正な動作を保証するために、インストールしたらコネクターを始動する前に構成しておく必要があります。これらの作業の要約については、21 ページの『構成作業の概要』を参照してください。さらに、絶対にコネクターのポーリングを使用不可にしないでください (コネクター・ポーリングはデフォルトで使用可能になっています)。

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* に存在していなければなりません。ここで、*connName* はコネクターを示します。

Linux システムでは、始動スクリプトは *ProductDir*/bin ディレクトリーに存在していなければなりません。

i5/OS システムでは、始動スクリプトは、コネクターの実行に使用する /QIBM/UserData/WBIServer44/<instance>/connectors/<ConnInstance/ に存在しなければなりません。

始動スクリプトの名前は、表 4 に示すように、オペレーティング・システム・プラットフォームによって異なります。

表 4. 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
Linux	connector_manager
i5/OS	start_connName.sh
Windows	start_*connName*.bat

始動スクリプトが実行されると、デフォルトで構成ファイルは *Productdir* にあることが要求されます (下記のコマンドを参照)。ここに構成ファイルを格納します。

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。

- **Windows システムでのコネクターの開始**
  - 「**スタート**」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」となっています。ただし、これはカスタマイズすることができます。あるいは、ご使用のコネクタ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 コマンド行から、次を入力します: 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。
- **Linux システムでのコネクターの開始**:
  - コマンド行から、以下を入力します。

    ```
    connector_manager -start connName brokerName [-cconfigFile ]
    ```

    ここで、*connName* はコネクターの名前であり、*brokerName* はご使用の統合ブローカーを表します。
  - InterChange Server Express の場合は、*brokerName* に InterChange Server Express インスタンスの名前を指定します。
- **i5/OS システムでのコネクターの開始**
  - WebSphere Business Integrations Server Express Console がインストールされている Windows システムから、「**IBM WebSphere Business Integration Server Express**」>「**Toolset Express**」>「**管理**」>「**コンソール**」を選択します。次に、OS/400 システム名または i5/OS システム名、あるいは IP アドレスと、*JOBCTL の特殊権限を持つユーザー・プロファイルおよびパスワードを指定します。コネクターのリストからコネクターを選択して、「開始」をクリックします。
  - コンソールを使用してアダプターを自動的に開始するには、submit_adapter.sh スクリプトを使用します。これが、サーバーの自動開始ジョブ・エントリー内のサブシステムを使用してアダプターが開始する唯一の方法です。
  - バッチ・モードでは、i5/OS コマンド行から CL コマンド QSH を実行し、QSHELL 環境から /QIBM/ProdData/WBIServer44/bin/submit_adapter.sh *connName WebSphereICSName pathToConnNameStartScript jobDescriptionName* を実行する必要があります。ここで、*connName* はコネクター名、*WebSphereICSName* は Interchange Server Express サーバー名 (デフォルトは QWBIDFT44)、*pathToConnNameStartScript* はコネクター始動スクリプトの絶対パス、*jobDescriptionName* は QWBISVR44 ライブラリーで使用するジョブ記述の名前です。
  - 対話モードでは、CL コマンド QSH を実行し、QSHELL 環境から */QIBM/UserData/WBIServer44/WebSphereICSName/connectors/connName/start_connName.sh connNameWebsphereICSName* [-cConfigFile] を実行する必要

があります。ここで、*connName* はコネクターの名前であり、*WebSphereICSName* は InterChange Server Express インスタンスの名前です。

コマンド行の始動オプションなどのコネクターの始動方法の詳細については、「システム管理ガイド」を参照してください。

コネクターの停止

コネクターを停止する方法は、コネクターが始動された方法によって異なります。

- **Windows**:
  - コネクター用に別の「コンソール」ウィンドウを作成する始動スクリプトを呼び出すことができます。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。
  - コネクターを Windows のサービスとして始動するように構成できます。この場合、Windows システムがシャットダウンされるとコネクターは停止します。
- **i5/OS**:
  - Console、または QSHELL の「submit_adapter.sh」スクリプトを使用してコネクターを始動した場合、次のいずれかの方法でコネクターを停止することができます。
  - WebSphere Business Integration Server Express Console がインストールされている Windows システムから、「**IBM WebSphere Business Integration Express**」>「**Toolset Express**」>「**管理**」>「**コンソール**」を選択します。次に、OS/400 システム名または i5/OS システム名、あるいは IP アドレスと、*JOBCTL の特殊権限を持つユーザー・プロファイルおよびパスワードを指定します。リストから Web サービス・アダプターを選択し、「停止」ボタンを選択します。CL コマンド WRKACTJOB SBS (QWBISVR44) を使用して Server Express 製品に対するジョブを表示します。リストをスクロールして、コネクターのジョブ記述と一致するジョブ名を持つジョブを探します。例えば、Web サービス・コネクターの場合、ジョブ名は QWBIWEBSVC になります。このジョブに対してオプション 4 を選択し、F4 を押して ENDJOB コマンドのプロンプトを取得します。次に、オプション・パラメーターとして *IMMED を指定し、Enter を押します。

    注: QWBISVR44 サブシステムが終了すると、コネクターが終了します。
  - QSHELL から start_connName.sh スクリプトを使用してアダプターを始動した場合は、F3 を押してコネクターを終了します。/QIBM/ProdData/WBIServer44/bin ディレクトリーにある *stop_adapter.sh* というスクリプトを使用して、エージェントを停止することもできます。
- **Linux**:

  コネクターはバックグラウンドで稼働し、別のウィンドウは表示されません。その代わりに、以下のコマンドを実行してコネクターを停止します。

  ```
  connector_manager -stop connName
  ```

  ここで、*connName* はコネクター名です。

第 3 章 ビジネス・オブジェクトの要件

- 『ビジネス・オブジェクトのメタデータ』
- 『コネクター・ビジネス・オブジェクトの構造』
- 28 ページの『同期イベント処理 TLO』
- 43 ページの『非同期イベント処理 TLO』
- 47 ページの『イベント処理の非 TLO』
- 47 ページの『同期要求処理 TLO』
- 47 ページの『同期要求処理 TLO』
- 59 ページの『非同期要求処理 TLO』
- 64 ページの『ビジネス・オブジェクトの開発』

この章では、コネクター・ビジネス・オブジェクトの構造、要件、および属性について説明します。

ビジネス・オブジェクトのメタデータ

Connector for Web Services は、メタデータ主導型のコネクターです。ビジネス・オブジェクトでは、メタデータはアプリケーションに関するデータのことです。このデータはビジネス・オブジェクト定義に格納されており、コネクターとアプリケーションとの対話に役立ちます。メタデータ主導型のコネクターは、コネクター内にハードコーディングされている命令ではなく、ビジネス・オブジェクト定義内にエンコードされているメタデータに基づいて、コネクター自身がサポートしている各ビジネス・オブジェクトを処理します。

ビジネス・オブジェクトのメタデータには、ビジネス・オブジェクトの構造、その属性プロパティーの設定値、およびそのアプリケーション固有情報の内容が含まれています。コネクターは、メタデータ主導型なので、新規や変更後のビジネス・オブジェクトを処理する場合にコネクター・コードを変更する必要がありません。ただし、コネクターの構成済みデータ・ハンドラーでは、そのビジネス・オブジェクトの構造、オブジェクトの基数、アプリケーション固有のテキストのフォーマット、およびビジネス・オブジェクトのデータベース表現について前提事項が存在します。そのため、Web サービスのビジネス・オブジェクトを作成または変更する場合、その変更内容は、コネクターが従うべきルールに適合している必要があります。適合していないと、コネクターは新規または変更済みのビジネス・オブジェクトを正しく処理できません。

メタデータ、メタオブジェクト、ならびにこれらの構成およびビジネス・オブジェクトや SOAP メッセージとの相互作用については、 125 ページの『第 5 章 SOAP データ・ハンドラー』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、以下の 2 種類のビジネス・オブジェクトを処理します。

- **TLO** Web サービスのトップレベル・ビジネス・オブジェクト (TLO) には、要求ビジネス・オブジェクトと、オプションとして応答ビジネス・オブジェクトおよび障害ビジネス・オブジェクトが含まれます。これらの子オブジェクトには、内容データおよび SOAP 構成 MO、およびオプションでプロトコル構成 MO も含まれています。TLO、要求オブジェクト、応答オブジェクト、および障害オブジェクトのほかに、アプリケーション固有情報、属性、および要求とイベント処理に関する要件について、以下のセクションで説明および図解します。

  注: *TLO は、要求処理およびイベント処理のために使用されます。*

- **非 TLO** これらは、TLO 以外の汎用ビジネス・オブジェクト (GBO) およびアプリケーション固有ビジネス・オブジェクト (ASBO) ですが、WSDL 生成時の WSDL 構成ウィザードで使用されています。コネクターは、イベント処理中に、非 TLO を処理できます。これらのオブジェクトについては、以下の 47 ページの『イベント処理の非 TLO』で説明します。詳しくは、 170 ページの『WSDL 構成ウィザード』を参照してください。

  注: *非 TLO は、イベント処理のみに使用されます。*

注: SOAP ヘッダー・コンテナー、ならびに要求、応答、および障害の各ビジネス・オブジェクトに含まれるヘッダー・ビジネス・オブジェクトは、この章では説明しません。SOAP ヘッダー・コンテナーおよびヘッダー・ビジネス・オブジェクトについての詳細は、 125 ページの『第 5 章 SOAP データ・ハンドラー』を参照してください。

同期イベント処理 TLO

イベント処理のために、コネクターでは、同期および非同期の 2 種類の TLO を使用することが可能です。このセクションでは、同期イベント処理 TLO について説明します。

29 ページの図 4 は、同期イベント処理のためのビジネス・オブジェクト階層を示しています。要求オブジェクトおよび応答オブジェクトは必須であり、障害オブジェクトはオプションです。

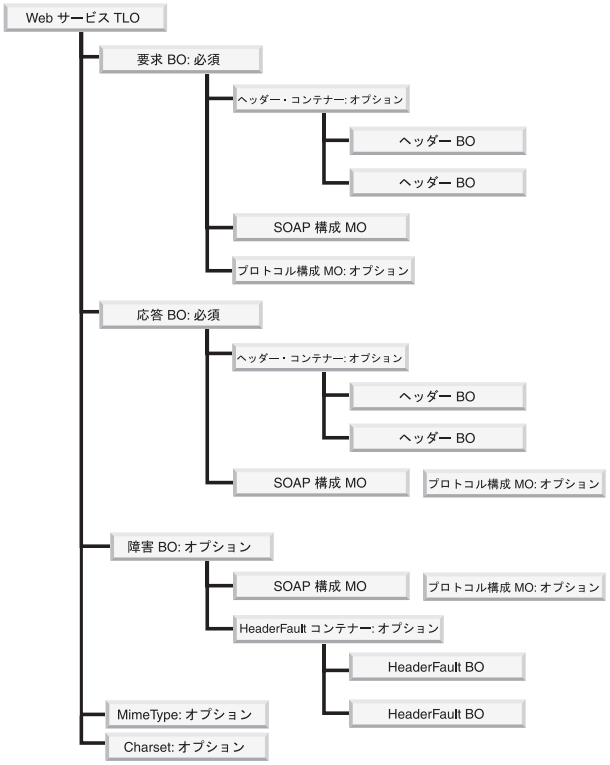


図 4. 同期イベント処理のためのビジネス・オブジェクト階層

TLO には、オブジェクト・レベルの ASI のほか、属性レベルの ASI を持った属性が含まれています。両方の種類の ASI について、以下で説明します。

同期イベント処理 TLO のオブジェクト・レベルの ASI

オブジェクト・レベルの ASI は、TLO の性質、および TLO に含まれるオブジェクトについての基本的情報を提供します。図 5 は、同期イベント処理のためのサンプル TLO である SERVICE_SYNCH_OrderStatus のオブジェクト・レベルの ASI を表しています。

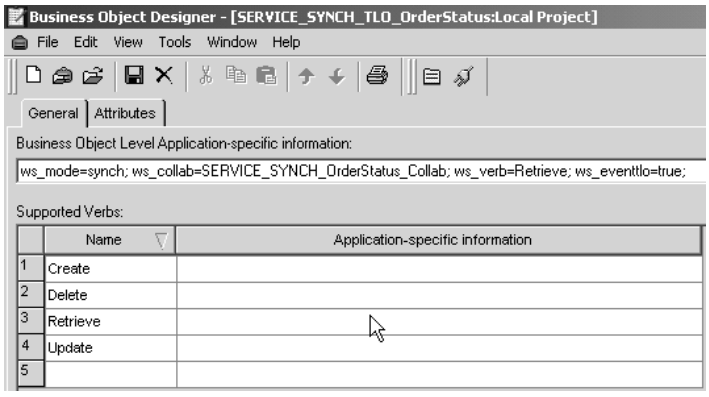


図 5. 同期イベント処理のためのトップレベル・ビジネス・オブジェクト

以下の表 5 では、同期イベント処理 TLO のためのオブジェクト・レベルの ASI について説明します。

表 5. 同期イベント処理 TLO のオブジェクト ASI

オブジェクト・レベルの ASI	説明
ws_eventtlo=true	この ASI プロパティーが true に設定されている場合には、コネクターはこのオブジェクトをイベント処理専用の TLO として扱います。 WSDL 構成ウィザードがこの ASI を使用して、ビジネス・オブジェクトが TLO であるかどうかを判別することに注意してください。これについての詳細は、 170 ページの『WSDL 構成ウィザード』を参照してください。
ws_collab=*collabname*	この ASI は、どのコラボレーションを呼び出すのかをコネクターに指示します。この値はコラボレーションの名前です。(この ASI は、WSDL の生成時にコラボレーション用の TLO を判別するためにも使用されます。詳しくは、 170 ページの『WSDL 構成ウィザード』を参照してください。) 図 5 に示されているサンプルでは、コラボレーション名は SERVICE_SYNCH_OrderStatus_Collab です。
ws_verb=*verb*	コネクターは、TLO をコラボレーションに引き渡す前に、TLO で動詞を設定するためにこの ASI を使用します。図 5 に示されているサンプルでは、動詞は Retrieve です。
ws_mode=synch	イベント通知の際に、コネクターは、この ASI プロパティーを使用して、コラボレーションを同期 (synch) で呼び出すのか非同期 (asynch) で呼び出すのかを決定します。同期処理の場合は、この ASI を synch に設定する必要があります。 デフォルトは asynch です。

同期イベント処理 TLO の属性レベルの ASI

それぞれの同期イベント処理 TLO は、属性および属性レベルの ASI を保有しています。図 6 は、サンプル TLO である SERVICE_SYNCH_OrderStatus の属性を表しています。また、この図には、属性レベルの ASI が「アプリケーション固有の情報 (App Spec Info)」列に表示されています。

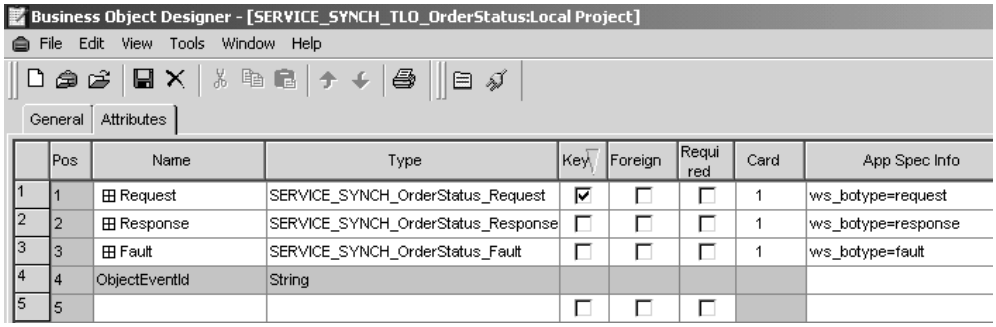


図 6. 同期イベント処理のための TLO 属性

表 6 は、同期イベント処理 TLO の Request、Response、Fault、MimeType、および Charset の各属性に対する属性レベルの ASI を要約したものです。

表 6. 同期イベント処理 TLO の属性 ASI

TLO 属性	**属性レベルの ASI**	**説明**
MimeType	なし	オプションの属性: 指定されていると、その値は、同期応答のために起動するデータ・ハンドラーの MIME タイプとして使用されます。String 型で、デフォルトは `xml/soap` です。
Charset	なし	このオプショナル・パラメーター (String 型) は、発信ビジネス・オブジェクトをメッセージに変換するときにデータ・ハンドラーに設定される charset を示します。注: この属性に指定された charset 値は、応答メッセージの Content-Type プロトコル・ヘッダーでは伝搬されません。
Request	`ws_botype=request`	この属性は、Web サービス要求に対応します。コネクターはこの ASI を使用して、この TLO 属性のタイプが SOAP 要求 BO であるかどうかを判別します。属性名ではなく、この ASI が属性タイプを判別します。複数の要求属性がある場合は、コネクターは最初の要求属性の ASI を使用します。 この属性は、同期イベント処理 TLO の場合には必須です。

表 6. 同期イベント処理 TLO の属性 ASI (続き)

TLO 属性	属性レベルの ASI	説明
Response	`ws_botype=response`	この属性は、Web サービスによって戻される応答に対応します。コネクターはこの ASI を使用して、この TLO 属性のタイプが SOAP 応答 BO であるかどうかを判別します。属性名ではなく、この ASI が属性タイプを判別します。複数の応答属性がある場合は、コネクターは最初の応答属性の ASI を使用します。 この属性は、同期イベント処理 TLO の場合には必須です。
Fault	`ws_botype=fault` `ws_botype=defaultfault`	この属性は、同期イベント処理の場合のオプションであり、応答を正常に取り込むことができないときにコラボレーションから戻される障害メッセージに対応します。コネクターは、属性名ではなくこの ASI を使用して、属性のタイプが SOAP 障害 BO であるかどうかを判別します。`ws_botype=defaultfault` である場合は、WSDL 構成ウィザードはこの障害ビジネス・オブジェクトをヘッダー処理に使用します。詳しくは、139 ページの『ヘッダー障害の処理』を参照してください。

同期イベント処理のための要求ビジネス・オブジェクト

要求ビジネス・オブジェクトは、TLO の子であり、同期イベント処理の場合には必須です。要求ビジネス・オブジェクトは、オブジェクト・レベルの ASI を保有しています。例えば、Business Object Designer Express で SERVICE_SYNCH_OrderStatus_Request をオープンし、「一般」タブをクリックすると、33 ページの図 7 に示すように、オブジェクト・レベルの ASI が表示されます。

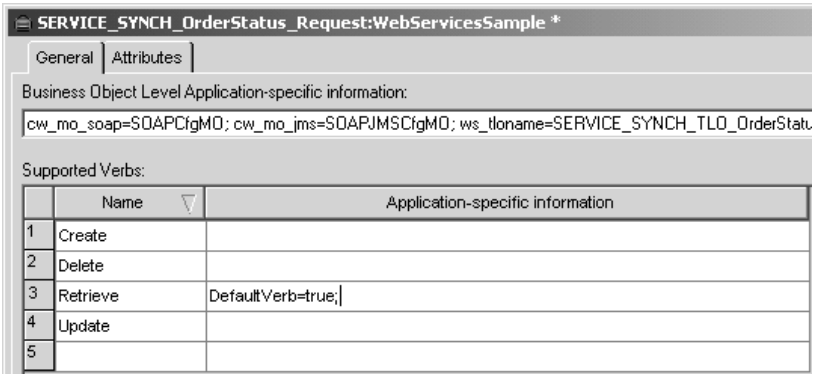


図 7. 同期イベント処理要求オブジェクトに関するオブジェクト・レベルの ASI

同期イベント処理のための要求ビジネス・オブジェクトに対するオブジェクト・レベルの ASI は、表 7 に説明されています。図 7 に示すように、要求ビジネス・オブジェクトに対して、デフォルトの動詞を指定できます。これを行うには、トップレベルの要求ビジネス・オブジェクトの「サポートされている動詞 (Supported Verbs)」リストにある動詞の ASI フィールドに、

```
DefaultVerb=true;
```

と指定します。DefaultVerb ASI が指定されず、データ・ハンドラーが動詞の設定されていないビジネス・オブジェクトを処理する場合、そのビジネス・オブジェクトは、動詞なしで戻されます。

表 7. 同期イベント処理: 要求ビジネス・オブジェクトのオブジェクト・レベルの ASI

オブジェクト・レベルの ASI	**説明**
cw_mo_soap=*SOAPCfgMO*	この ASI の値は、SOAP 構成 MO に対応する属性の名前と一致しなければなりません。これは、要求ビジネス・オブジェクトのためのデータ・ハンドラー変換を定義するメタオブジェクトです。詳しくは、35 ページの『SOAP 構成 MO』を参照してください。
cw_mo_jms=*SOAPJMSCfgMO* または cw_mo_http=*SOAPHTTPCfgMO*	この ASI の値は、プロトコル構成 MO に対応する属性の名前と一致しなければなりません。最初の ASI は SOAP/JMS プロトコル・リスナーを示し、2 番目の ASI は SOAP/HTTP または SOAP/HTTPS プロトコル・リスナーを示します。ASI およびプロトコル構成 MO は、両方ともオプションです。詳しくは、35 ページの『プロトコル構成 MO』を参照してください。

表 7. 同期イベント処理: 要求ビジネス・オブジェクトのオブジェクト・レベルの ASI (続き)

オブジェクト・レベルの ASI	説明
ws_tloname=*tloname*	この ASI は、このオブジェクトが属する Web サービス TLO の名前を指定します。イベント処理の際に、コネクターは、この ASI を使用して、データ・ハンドラーによって引き渡された要求ビジネス・オブジェクトが TLO の子であるかどうかを判別します。引き渡された要求ビジネス・オブジェクトが TLO の子である場合は、コネクターは、指定された TLO を作成し、要求ビジネス・オブジェクトをその TLO の子として設定し、TLO のオブジェクト・レベルの ASI を使用して、この要求ビジネス・オブジェクトを、サブスクライブしているコラボレーションに引き渡します。

同期イベント処理のための応答ビジネス・オブジェクト

応答ビジネス・オブジェクトは、TLO の子であり、同期イベント処理の場合には必須です。同期イベント処理のための応答ビジネス・オブジェクトに対するオブジェクト・レベルの ASI は、表 8 に説明されています。

表 8. 同期イベント処理: 応答ビジネス・オブジェクトのオブジェクト・レベルの ASI

オブジェクト・レベルの ASI	説明
cw_mo_soap=*SOAPCfgMO*	この ASI の値は、SOAP 構成 MO に対応する属性の名前と一致しなければなりません。これは、応答ビジネス・オブジェクトに関するデータ・ハンドラー変換を定義する SOAP 構成 MO です。詳しくは、 35 ページの『SOAP 構成 MO』を参照してください。

注: オプションで、要求 BO にプロトコル構成 MO のオブジェクト・レベルの ASI を組み込むことができます。

同期イベント処理のための障害ビジネス・オブジェクト

障害ビジネス・オブジェクトは、TLO の子であり、同期イベント処理の場合にはオプションです。同期イベント処理のための障害ビジネス・オブジェクトに対するオブジェクト・レベルの ASI は、表 9 に説明されています。

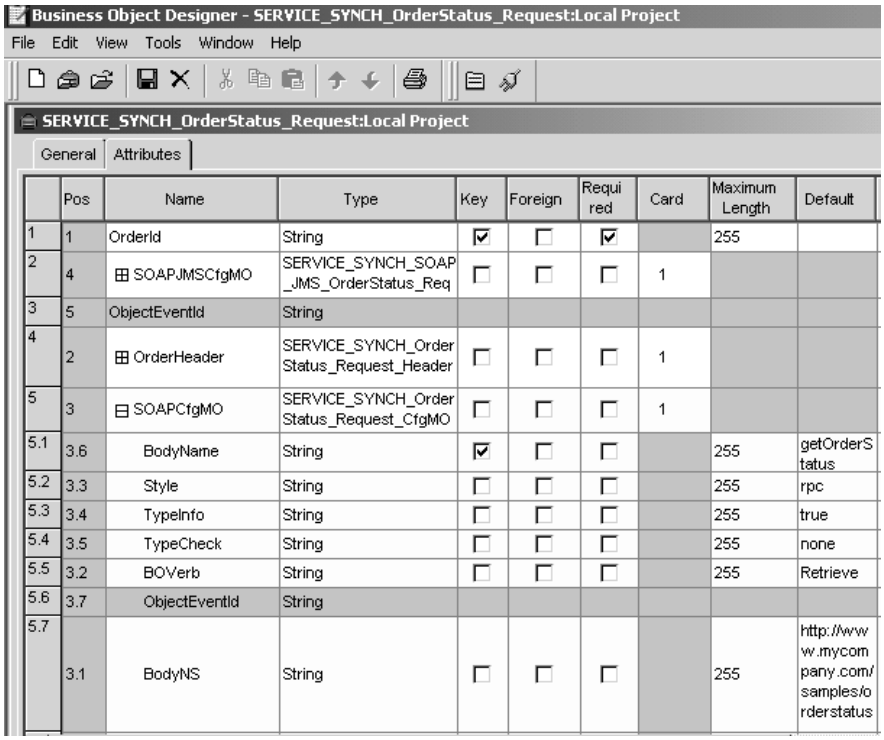
表 9. 同期イベント処理: 障害ビジネス・オブジェクトのオブジェクト・レベルの ASI

オブジェクト・レベルの ASI	説明
cw_mo_soap=*SOAPCfgMO*	この ASI の値は、SOAP 構成 MO に対応する属性の名前と一致しなければなりません。これは、障害ビジネス・オブジェクトに関するデータ・ハンドラー変換を定義する SOAP 構成 MO です。詳しくは、 35 ページの『SOAP 構成 MO』を参照してください。

注: オプションで、障害 BO にプロトコル構成 MO のオブジェクト・レベルの ASI を組み込むことができます。

SOAP 構成 MO

図 8 は、Business Object Designer Express で展開されたサンプル SOAP 構成 MO を表しています。



	Pos	Name	Type	Key	Foreign	Required	Card	Maximum Length	Default
1	1	OrderId	String	☑	☐	☑		255	
2	4	⊞ SOAPJMSCfgMO	SERVICE_SYNCH_SOAP_JMS_OrderStatus_Req	☐	☐	☐	1		
3	5	ObjectEventId	String						
4	2	⊞ OrderHeader	SERVICE_SYNCH_OrderStatus_Request_Header	☐	☐	☐	1		
5	3	⊟ SOAPCfgMO	SERVICE_SYNCH_OrderStatus_Request_CfgMO	☐	☐	☐	1		
5.1	3.6	BodyName	String	☑	☐	☐		255	getOrderStatus
5.2	3.3	Style	String	☐	☐	☐		255	rpc
5.3	3.4	TypeInfo	String	☐	☐	☐		255	true
5.4	3.5	TypeCheck	String	☐	☐	☐		255	none
5.5	3.2	BOVerb	String	☐	☐	☐		255	Retrieve
5.6	3.7	ObjectEventId	String						
5.7	3.1	BodyNS	String	☐	☐	☐		255	http://www.mycompany.com/samples/orderstatus

図 8. 同期イベント処理のための SOAP 構成 MO 属性

SOAP 構成 MO は、あるデータ・ハンドラー変換 (SOAP メッセージからビジネス・オブジェクトへの変換、またはビジネス・オブジェクトから SOAP メッセージへの変換) で行われるフォーマットの振る舞いを定義します。それぞれの要求、応答、または障害属性に SOAP 構成 MO が割り当てられます。その属性である BodyName、BodyNS、Style、Use、TypeInfo、TypeCheck、および BOVerb の型は、常に `String` になっています。これらの属性は、SOAP メッセージ要素に対応し、それらの値は、SOAP データ・ハンドラーがメッセージおよびオブジェクトをどのように読み取り、検証するのかを決定します。SOAP 構成 MO および属性についての詳細は、127 ページの『SOAP 構成メタオブジェクト: 各 SOAP ビジネス・オブジェクトの子』を参照してください。すべての SOAP 構成 MO は、要求、応答、または障害オブジェクトのいずれの場合にも、BodyName および BodyNS のデフォルト値に関する固有の項目を含んでいなければなりません。

プロトコル構成 MO

図 9 は、属性がインバウンド SOAP メッセージ内のヘッダーに対応している JMS プロトコル構成 MO を表しています。

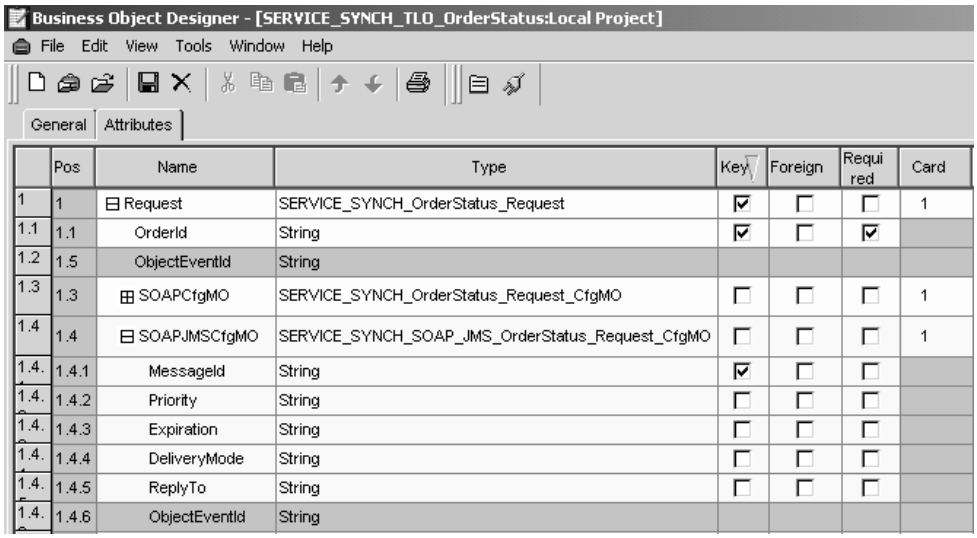


図 9. 同期イベント処理のための JMS プロトコル構成 MO 属性

この MO は、オプションで、イベント処理の要求、応答、または障害ビジネス・オブジェクトの子として組み込まれます。通常、プロトコル・ヘッダーとカスタム・プロパティーの (要求メッセージからの) 読み取りまたは (応答または障害メッセージへの) 伝搬を行う必要のある場合に指定します。前述のとおり、要求ビジネス・オブジェクトは、オプションでプロトコル構成 MO の名前をビジネス・オブジェクト・レベルの ASI として宣言します。

- `cw_mo_jms=`*`JMSProtocolListenerConfigMOAttribute`*
- `cw_mo_http=`*`HTTPProtocolListenerConfigMOAttribute`*

イベント処理中に、コネクターはプロトコル・リスナー (SOAP/HTTP、SOAP/HTTPS、または SOAP/JMS) を使用してトランスポートからイベントを検索します。これらのイベントは、Web サービスとして公開されているコラボレーションからのサービスを要求する、内部または外部 Web サービス・クライアントから送られるメッセージです。それぞれのトランスポートには、固有のヘッダー要件があります。コネクターは、プロトコル構成 MO を使用して、プロトコル固有のヘッダー情報をプロトコル・リスナーからコラボレーションへと転送します。プロトコル構成 MO の属性は、インバウンド SOAP/JMS メッセージ内のヘッダーに対応します。コネクターは、インバウンド SOAP メッセージの内容を使用して、これらの属性の値をビジネス・オブジェクト内に設定します。SOAP/JMS プロトコルの場合、イベント処理と要求処理用のプロトコル構成 MO の属性は以下のとおりです。

表 10. SOAP JMS プロトコル構成 MO の属性: イベント処理と要求処理

SOAP/JMS プロトコル構成 MO の属性	**JMSHeaderName**	**説明**
CorrelationID	JMSCorrelationID	インバウンド・メッセージ: この属性には JMSCorrelationID ヘッダーからの値が設定されます。アウトバウンド・メッセージ: この属性からの値が、発信メッセージの JMSCorrelationID ヘッダーとして設定されます。
MessageId	JMSMessageId	インバウンド・メッセージ: この属性には JMSMessageId ヘッダーからの値が設定されます。アウトバウンド・メッセージ: この属性は、アウトバウンド・メッセージには使用されません。
Priority	JMSPriority	インバウンド・メッセージ: この属性には JMSPriority ヘッダーからの値が設定されます。アウトバウンド・メッセージ: この属性からの値が、発信メッセージの JMSPriority ヘッダーに設定されます。
Expiration	JMSExpiration	インバウンド・メッセージ: この属性には JMSExpiration ヘッダーからの値が設定されます。アウトバウンド・メッセージ: この属性からの値が、発信メッセージの JMSExpiration ヘッダーに設定されます。
DeliveryMode	JMSDeliveryMode	インバウンド・メッセージ: この属性には JMSDeliveryMode ヘッダーからの値が設定されます。アウトバウンド・メッセージ: この属性からの値が、発信メッセージの JMSDeliveryMode ヘッダーに設定されます。

表 10. SOAP JMS プロトコル構成 MO の属性: イベント処理と要求処理 (続き)

SOAP/JMS プロトコル構成 MO の属性	JMSHeaderName	説明
Destination	JMSDestination	インバウンド・メッセージ: この属性には JMSDestination ヘッダーからの値が設定されます。 アウトバウンド・メッセージ: **要求処理** この属性からの値は、宛先キュー名として使用され、取得された宛先パスへの発信メッセージの JMSDestination ヘッダーに間接的に設定されます。 **イベント通知での同期応答**: この属性は使用されません。
Redelivered	JMSRedelivered	インバウンド・メッセージ: この属性には JMSRedelivered ヘッダーからの値が設定されます。アウトバウンド・メッセージ: この属性からの値が、発信メッセージの JMSRedelivered ヘッダーに設定されます。
ReplyTo	JMSReplyTo	インバウンド・メッセージ: この属性には JMSReplyTo ヘッダーからの値が設定されます。アウトバウンド・メッセージ: この属性からの値が、発信メッセージの JMSReplyTo ヘッダーに設定されます。
TimeStamp	JMSTimeStamp	インバウンド・メッセージ: この属性には JMSTimeStamp ヘッダーからの値が設定されます。アウトバウンド・メッセージ: この属性からの値が、発信メッセージの JMSTimeStamp ヘッダーに設定されます。
型	JMSType	インバウンド・メッセージ: この属性には JMSType ヘッダーからの値が設定されます。 アウトバウンド・メッセージ: この属性からの値が、発信メッセージの JMSType ヘッダーに設定されます。

表 10. SOAP JMS プロトコル構成 MO の属性: イベント処理と要求処理 (続き)

SOAP/JMS プロトコル構成 MO の属性	JMSHeaderName	説明
UserDefinedProperties	40 ページの『イベント処理のためのユーザー定義のプロパティー』を参照してください。	このオプションの読み取り/書き込み属性は、ユーザー定義のプロトコル・プロパティーのビジネス・オブジェクトを保持します。詳しくは、40 ページの『イベント処理のためのユーザー定義のプロパティー』を参照してください。

注: JMS プロトコル構成 MO で渡されたヘッダー値を、要求応答イベント環境で論理的に正しくすることは、コラボレーション側の責任となります。

SOAP/HTTP(S) プロトコルの場合、プロトコル構成 MO の属性は次のとおりです。

表 11. イベント処理用 HTTP/HTTPS プロトコル構成 MO 属性

属性	必須	型	説明
Content-Type	いいえ	String	この属性の値によって、発信メッセージの Content-Type ヘッダーが定義されます (ヘッダーには、発信メッセージのメッセージ ContentType および 0 個以上のパラメーター (charset) が組み込まれます)。構文は、HTTP プロトコルの Content-Type ヘッダーと同じです (例: `text/html; charset=ISO-8859-4`)。Content-Type 属性が定義されていない場合、コネクターは、応答/障害メッセージの ContentType として、要求の ContentType を使用します。
UserDefinedProperties	いいえ	ビジネス・オブジェクト	この属性は、ユーザー定義のプロトコル・プロパティーのビジネス・オブジェクトを保持します。
1 つ以上の HTTP ヘッダー	いいえ	String	この属性を使用すれば、ハンドラーは、指定された HTTP ヘッダーの値の受け渡しまたは検索ができます。
Authorization_UserID	いいえ	`String`	この属性は HTTP 基本認証の userID に対応します。
Authorization_Password	いいえ	`String`	この属性は HTTP 基本認証のパスワードに対応します。

これらの属性を以下に示します。

- 『イベント処理のためのユーザー定義のプロパティー』
- 41 ページの『イベント処理用の HTTP 証明書伝搬』

プロトコル・リスナーの詳細については、 70 ページの『プロトコル・リスナー』を参照してください (要求処理用のプロトコル構成 MO については、 47 ページの『同期要求処理 TLO』を参照してください)。

イベント処理のためのユーザー定義のプロパティー: オプションで、HTTP(S) プロトコル構成 MO にカスタム・プロパティーを指定できます。この指定を行うには、UserDefinedProperties 属性を組み込みます。この属性は、1 つ以上の子属性とプロパティー値を持つビジネス・オブジェクトに対応します。このビジネス・オブジェクトのすべての属性は、メッセージ・ヘッダーの変数部分で読み取られる (同期応答の場合は書き込まれる) 単一プロパティーを以下のように定義する必要があります。

- 属性の型は、プロトコル・プロパティー・タイプにかかわらず、常に String です。属性のアプリケーション固有情報には、属性がマップされるプロトコル・メッセージ・プロパティーの名前とフォーマットを定義した、2 組の名前と値のペアを入れることができます。

表 12 は、これらの属性のアプリケーション固有の情報を要約したものです。

表 12. ユーザー定義のプロトコル・プロパティー属性のアプリケーション固有情報: 名前 = 値のペアの内容

名前	値	説明
ws_prop_name (大/小文字を区別しない。指定されなければ、属性名がプロパティー名として使用される)	任意の有効なプロトコル・プロパティー名	これはプロトコル・プロパティーの名前です。ベンダーによっては、いくつかのプロパティーが拡張機能用に予約されています。一般に、このようなベンダー固有の機能にアクセスしようとする場合以外は、JMS で始まるカスタム・プロパティー (JMS プロトコルの場合) を定義しないでください。
ws_prop_type (大/小文字を区別しない。JMS 用オプション。指定されなければ、String とみなされる。HTTP(S) の場合、String 型しか意味を成さないため、関係しない)	String、Integer、Boolean、Float、Double、Long、Short	プロトコル・プロパティーのタイプ。JMS プロトコルの場合、JMS API には JMS メッセージにプロパティー値を設定するメソッドとして、setIntProperty、setLongProperty、setStringProperty などがあります。ここで指定される JMS プロパティーのタイプによって、これらのどのメソッドを使用してメッセージのプロパティー値を設定するかが決まります。

指定されたカスタム・プロパティー ASI (ws_prop_name または ws_prop_type) が無効で、このヘッダーを処理する論理的な方法 (HTTP 処理でプロパティー・タイプを無視するなど) がない場合、コネクターはログに警告を記録し、このプロパティーを無視します。ws_prop_name または ws_prop_type に対しての必要な検査は実行されたが、カスタム・プロパティーの値を設定することも検索することもできない場合、コネクターはログにエラーを記録し、イベントは失敗します。

UserDefinedProperties 属性が指定された場合、コネクターは、UserDefinedProperties ビジネス・オブジェクトのインスタンスを作成します。次に、コネクターはメッセージからプロパティー値を抽出し、ビジネス・オブジェクトに保管します。少なくとも 1 つのプロパティー値が正常に検索されれば、コネクターは、変更された UserDefinedProperties 属性をプロトコル構成 MO に設定します。

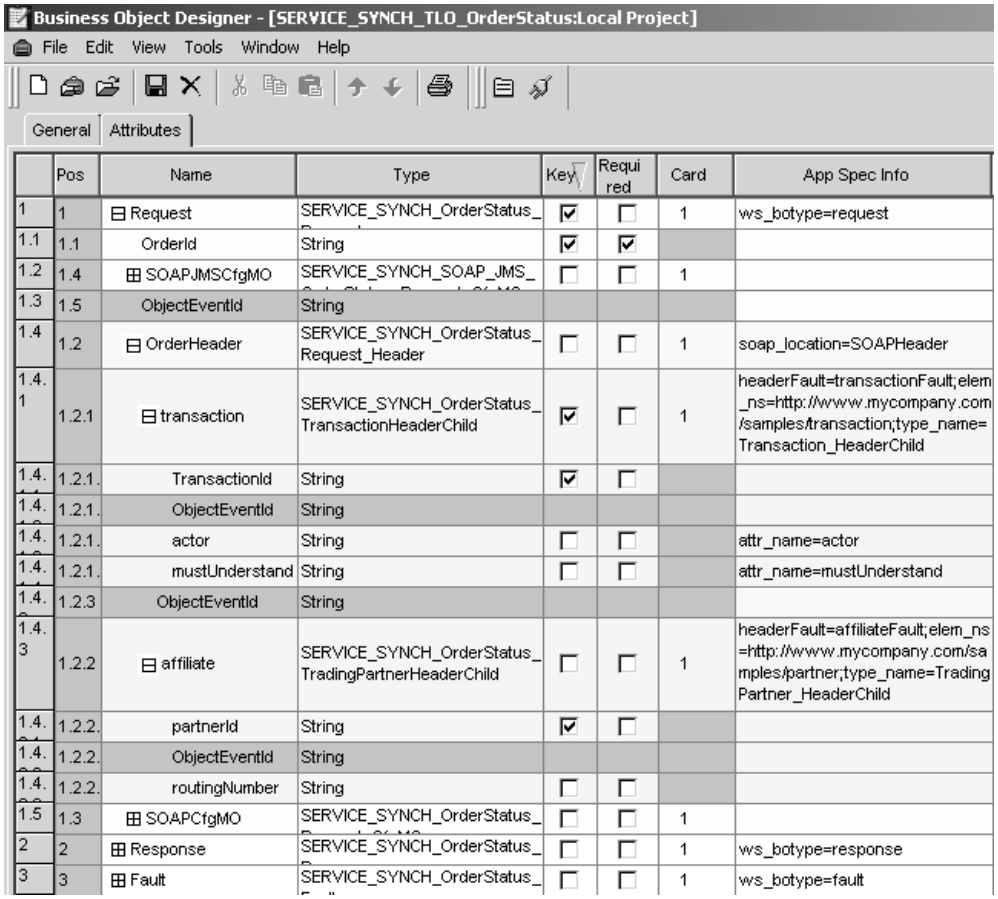
同期イベント処理の場合、UserDefinedProperties 属性が指定されて、そのビジネス・オブジェクトのインスタンスが作成されると、コネクターは、この子ビジネス・オブジェクトの各属性を処理し、それに対応するメッセージ・プロパティー値を設定します。

イベント処理用の HTTP 証明書伝搬: 証明書伝搬のために、コネクターは HTTP プロトコル構成 MO の Authorization_UserID 属性および Authorization_Password 属性をサポートします。このサポートは、HTTP 基本認証スキームの一部としてのこれらの証明書の伝搬に限られます。

SOAP/HTTP または SOAP/HTTPS プロトコル・リスナーが許可ヘッダーを含む SOAP/HTTP Web サービス要求を処理する場合は、リスナーはヘッダーを解析して HTTP 基本認証に従うかどうか判別します。従う場合は、リスナーはユーザー名およびパスワードを抽出し、Base64 を使用してデコードします。このデコードされたストリングは、コロンで区切られたユーザー名およびパスワードから構成されます。プロトコル・リスナーがプロトコル構成 MO で Authorization_UserID 属性および Authorization_Password 属性を検出した場合、リスナーは、イベント許可ヘッダーから抽出した値をこれらの属性に設定します。

ヘッダー・コンテナー・ビジネス・オブジェクト

図 10 は、展開されたヘッダー・コンテナー属性 OrderHeader を表しています。



	Pos	Name	Type	Key	Required	Card	App Spec Info
1	1	⊟ Request	SERVICE_SYNCH_OrderStatus_	☑	☐	1	ws_botype=request
1.1	1.1	OrderId	String	☑	☑		
1.2	1.4	⊞ SOAPJMSCfgMO	SERVICE_SYNCH_SOAP_JMS_	☐	☐	1	
1.3	1.5	ObjectEventId	String				
1.4	1.2	⊟ OrderHeader	SERVICE_SYNCH_OrderStatus_Request_Header	☐	☐	1	soap_location=SOAPHeader
1.4.1	1.2.1	⊟ transaction	SERVICE_SYNCH_OrderStatus_TransactionHeaderChild	☑	☐	1	headerFault=transactionFault;elem_ns=http://www.mycompany.com/samples/transaction;type_name=Transaction_HeaderChild
1.4.	1.2.1.	TransactionId	String	☑	☐		
1.4.	1.2.1.	ObjectEventId	String				
1.4.	1.2.1.	actor	String	☐	☐		attr_name=actor
1.4.	1.2.1.	mustUnderstand	String	☐	☐		attr_name=mustUnderstand
1.4.	1.2.3	ObjectEventId	String				
1.4.3	1.2.2	⊟ affiliate	SERVICE_SYNCH_OrderStatus_TradingPartnerHeaderChild	☐	☐	1	headerFault=affiliateFault;elem_ns=http://www.mycompany.com/samples/partner;type_name=TradingPartner_HeaderChild
1.4.	1.2.2.	partnerId	String	☑	☐		
1.4.	1.2.2.	ObjectEventId	String				
1.4.	1.2.2.	routingNumber	String	☐	☐		
1.5	1.3	⊞ SOAPCfgMO	SERVICE_SYNCH_OrderStatus_	☐	☐	1	
2	2	⊞ Response	SERVICE_SYNCH_OrderStatus_	☐	☐	1	ws_botype=response
3	3	⊞ Fault	SERVICE_SYNCH_OrderStatus_	☐	☐	1	ws_botype=fault

図 10. ヘッダー・コンテナーおよび子ビジネス・オブジェクト

ヘッダー・コンテナー属性は SOAP ヘッダー属性とも呼ばれ、子ビジネス・オブジェクトのみを含むビジネス・オブジェクトに対応します。それぞれの子は、SOAP メッセージのヘッダー項目を表します。図 10 に示す例では、要求ヘッダー・コンテナーは OrderHeader になっています。SOAP ヘッダー属性には、SOAP データ・ハンドラーが必要とするアプリケーション固有情報 (ASI) が含まれます。例えば、ヘッダー・コンテナー・ビジネス・オブジェクトは、その ASI によって、`soap_location=SOAPHeader` のように識別されます。ヘッダー処理についての詳細は、133 ページの『SOAP データ・ハンドラーの処理』を参照してください。

すべての SOAP ビジネス・オブジェクトは、要求オブジェクトであるのか、応答オブジェクトであるのか、あるいは障害オブジェクトであるのかに関係なく、ヘッダー・コンテナーを 1 つだけ含んでいます。

ヘッダー子ビジネス・オブジェクト

図 10 に示す例では、要求ヘッダー・コンテナー (OrderHeader) の 2 つの子属性は、1) SERVICE_SYNCH_OrderStatus_TransactionHeaderChild タイプのトランザクション、および 2) SERVICE_SYNCH_OrderStatus_TradingPartnerHeaderChild タイプの関連会社です。これらの属性は、ヘッダー子ビジネス・オブジェクトに対応します。それぞれの属性は、SOAP メッセージ内の単一のヘッダー要素を表します。ヘッダー要素は、SOAP メッセージの SOAP-Env:Header 要素の直接の子です。図 10 に示すように、ヘッダー子ビジネス・オブジェクトには actor および mustUnderstand 属性が割り当てられることがあります。これらの属性は、SOAP へ

ッダー要素の actor および mustUnderstand 属性に対応しています。ヘッダー処理についての詳細は、 133 ページの『SOAP データ・ハンドラーの処理』を参照してください。

SOAP ヘッダー・メッセージ要素を表すために必要な、任意の数のヘッダー子オブジェクトを含めることができます。

非同期イベント処理 TLO

図 11 は、非同期イベント処理のためのビジネス・オブジェクト階層を示しています。要求オブジェクトのみが必須です。

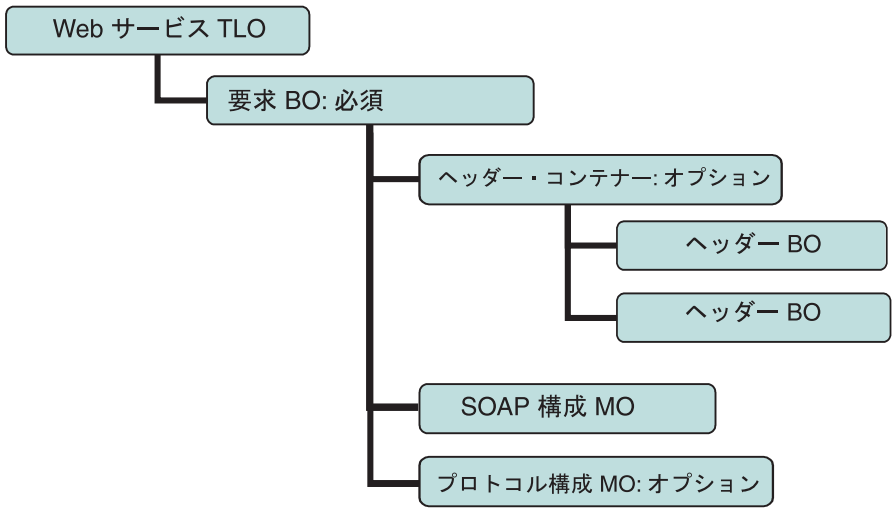


図 11. 非同期イベント処理のためのビジネス・オブジェクト階層

TLO には、オブジェクト・レベルの ASI のほか、属性レベルの ASI を持った属性が含まれています。両方の種類の ASI について、以下で説明します。ヘッダー・コンテナーおよびヘッダー子ビジネス・オブジェクトについての詳細は、 41 ページの『ヘッダー・コンテナー・ビジネス・オブジェクト』を参照してください。

非同期イベント処理 TLO のためのオブジェクト・レベルの ASI

オブジェクト・レベルの ASI は、TLO の性質、および TLO に含まれるオブジェクトについての基本的情報を提供します。図 12 は、非同期イベント処理のためのサンプル TLO である SERVICE_ASYNCH_TLO_Order のオブジェクト・レベルの ASI を表しています。

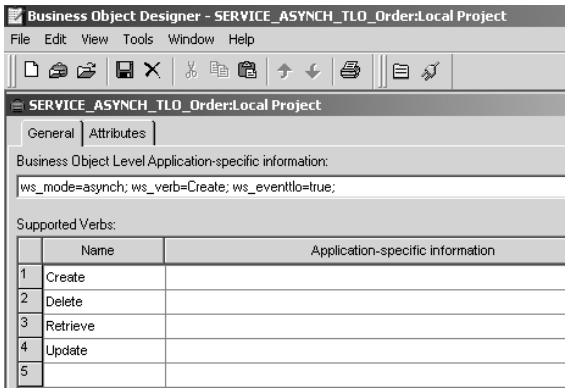


図 12. 非同期イベント処理のためのトップレベル・ビジネス・オブジェクト

以下の表 5 では、非同期イベント処理 TLO のためのオブジェクト・レベルの ASI について説明します。

表 13. 非同期イベント処理 TLO のオブジェクト ASI

オブジェクト・レベルの ASI	**説明**
`ws_eventtlo=true`	この ASI プロパティーが `true` に設定されている場合には、コネクターはこのオブジェクトをイベント処理用の TLO として扱います。 WSDL 構成ウィザードがこの ASI を使用して、ビジネス・オブジェクトが TLO であるかどうかを判別することに注意してください。これについての詳細は、170 ページの『WSDL 構成ウィザード』を参照してください。
`ws_verb=`*`verb`*	コネクターは、TLO をコラボレーションに引き渡す前に、TLO で動詞を設定するためにこの ASI を使用します。図 12 に示されているサンプルでは、動詞は `Create` です。
`ws_mode=asynch`	イベント通知の際に、コネクターは、この ASI プロパティーを使用して、コラボレーションを同期 (`synch`) で呼び出すのか非同期 (`asynch`) で呼び出すのかを決定します。非同期処理の場合は、この ASI を `asynch` に設定する必要があります。 デフォルトは `asynch` です。

注: 同期イベント処理の場合とは異なり、非同期イベント処理では、TLO レベルのコラボレーション名 ASI は必要ありません。その代わりに、統合ブローカーは、サブスクライブしているすべてのコラボレーションにアプリケーション・イベントが到達するものと想定します。

非同期イベント処理 TLO のための属性レベルの ASI

それぞれの非同期イベント処理 TLO には、要求ビジネス・オブジェクトに対応する単一の属性が含まれています。図 13 は、サンプル TLO の SERVICE_ASYNCH_TLO_Order の要求属性、および属性の ASI を表しています。

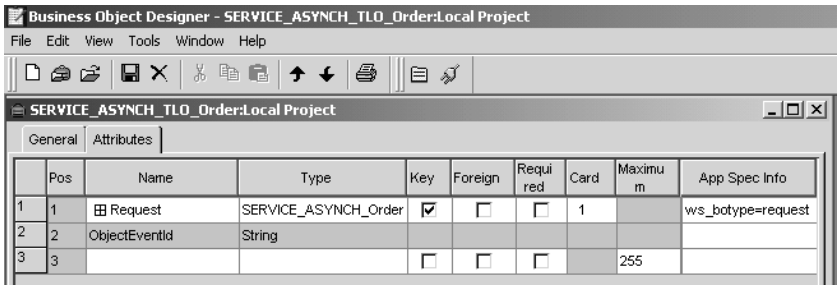


図 13. 非同期イベント処理のための TLO 属性

表 14 は、非同期イベント処理 TLO の要求属性に対する属性レベルの ASI を要約したものです。

表 14. 非同期イベント処理 TLO の属性 ASI

TLO 属性	属性レベルの ASI	説明
Request	`ws_botype=request`	この属性は、Web サービス要求に対応します。コネクターはこの ASI を使用して、この TLO 属性のタイプが SOAP 要求 BO であるかどうかを判別します。属性名ではなく、この ASI が属性タイプを判別します。複数の要求属性がある場合は、コネクターは最初の要求属性の ASI を使用します。 この属性は、同期イベント処理 TLO の場合には必須です。

非同期イベント処理のための要求ビジネス・オブジェクト

要求ビジネス・オブジェクトは、TLO の子であり、非同期イベント処理の場合には必須です。要求ビジネス・オブジェクトに対して、デフォルトの動詞を指定できます。これを行うには、トップレベルの要求ビジネス・オブジェクトの「サポートされている動詞 (Supported Verbs)」リストにある動詞の ASI フィールドに、

```
DefaultVerb=true;
```

と指定します。`DefaultVerb` ASI が指定されず、データ・ハンドラーが動詞の設定されていないビジネス・オブジェクトを処理する場合、そのビジネス・オブジェクトは、動詞なしで戻されます。非同期イベント処理のための要求ビジネス・オブジェクトに対するオブジェクト・レベルの ASI は、表 15 に説明されています。

表 15. 非同期イベント処理: 要求ビジネス・オブジェクトのためのオブジェクト・レベルの ASI

オブジェクト・レベルの ASI	説明
cw_mo_soap=*SOAPCfgMO*	この ASI の値は、SOAP 構成 MO に対応する属性の名前と一致しなければなりません。これは、要求ビジネス・オブジェクトに対するデータ・ハンドラー変換を定義する SOAP 構成 MO です。詳しくは、35 ページの『SOAP 構成 MO』を参照してください。
cw_mo_jms=SOAPJMSCfgMO または cw_mo_http=*SOAPHTTPCfgMO*	この ASI の値は、プロトコル構成 MO に対応する属性の名前と一致しなければなりません。最初の ASI は SOAP/JMS プロトコル・リスナーを示し、2 番目の ASI は SOAP/HTTP または SOAP/HTTPS プロトコル・リスナーを示します。ASI およびプロトコル構成 MO は、両方ともオプションです。詳しくは、35 ページの『プロトコル構成 MO』を参照してください。
ws_tloname=*tloname*	この ASI は、このオブジェクトが属する Web サービス TLO の名前を指定します。イベント処理の際に、コネクターは、この ASI を使用して、データ・ハンドラーによって引き渡された要求ビジネス・オブジェクトが TLO の子であるかどうかを判別します。引き渡された要求ビジネス・オブジェクトが TLO の子である場合は、コネクターは、指定された TLO を作成し、要求ビジネス・オブジェクトをその TLO の子として設定し、TLO のオブジェクト・レベルの ASI を使用して、この要求ビジネス・オブジェクトを、サブスクライブしているコラボレーションに引き渡します。

図 14 に示すサンプルでは、要求属性には、SOAP 構成 MO、ヘッダー・コンテナー (OrderHeader) のほか、内容に関連した属性 (OrderLineItems) が含まれています。非同期イベント処理の場合の SOAP 構成 MO、プロトコル構成 MO、SOAP ヘッダー・コンテナー、およびヘッダー子ビジネス・オブジェクトの要件および特性は、同期イベント処理の場合と同じです。詳しくは、28 ページの『同期イベント処理 TLO』における該当のトピックを参照してください。

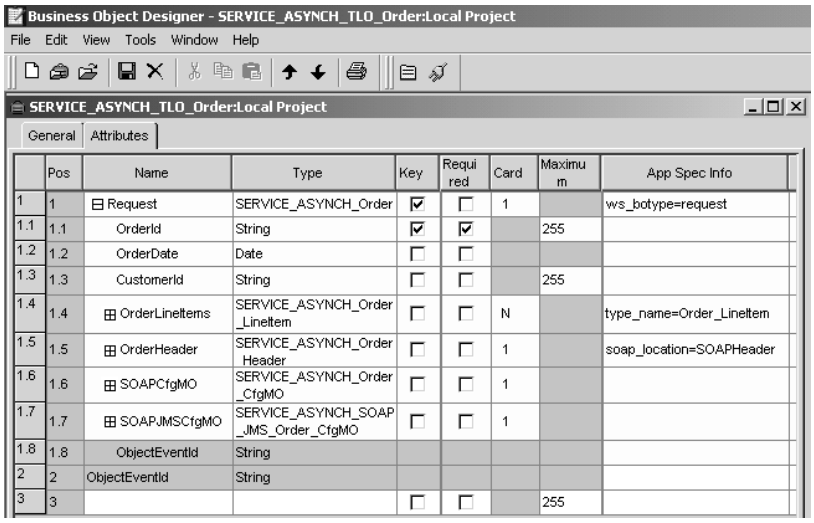


図 14. 非同期イベント処理のための要求属性

イベント処理の非 TLO

オブジェクト・レベルの ASI である `ws_eventtlo=true` がビジネス・オブジェクト内に存在しない場合、コネクターはそのオブジェクトが TLO ではないと判断します。イベント処理の際に、コネクターは、非 TLO、すなわち、汎用ビジネス・オブジェクトおよびアプリケーション固有ビジネス・オブジェクトを処理できます。非 TLO の場合、同じビジネス・オブジェクトが、要求ビジネス・オブジェクトと応答ビジネス・オブジェクトを表します。

非 TLO には、SOAP 構成 MO が含まれていません。コラボレーションを Web サービスとして公開すると、WSDL 構成ウィザードはコネクターの WSCollaborations プロパティーを構成します。コネクターは、WSCollaborations プロパティーを使用して、要求メッセージの BodyName および BodyNS を判別します。非 TLO の場合、WSCollaborations プロパティーがビジネス・オブジェクトの解決のために使用されるということに注意してください。

非 TLO を使用する利点は、Web サービス・ソリューションで使用する TLO 構造のビジネス・オブジェクトを新規に開発する必要がないということです。しかし、TLO を使用すると、データ、すなわち、顧客や会社などをより正確で経済的に公開することができます。また、TLO ビジネス・オブジェクトは、非 TLO よりもカスタマイズに適しています。

非 TLO を WSDL 構成ウィザードへの入力データとして使用する場合の要件については、168 ページの『ビジネス・オブジェクトの識別または開発』を参照してください。

同期要求処理 TLO

要求処理のために、コネクターでは、同期および非同期の 2 種類の TLO を使用することができます。このセクションでは、同期要求処理 TLO について説明します。

図 15 は、同期要求処理のための TLO ビジネス・オブジェクト階層を示しています。要求オブジェクトおよび応答オブジェクトは必須であり、障害オブジェクトはオプションです。イベント処理とは異なり、プロトコル構成 MO は、要求オブジェクトの場合には必須であり、応答および障害オブジェクトの場合には、オプションです。ヘッダー・コンテナーおよびヘッダー子ビジネス・オブジェクトについての詳細は、41 ページの『ヘッダー・コンテナー・ビジネス・オブジェクト』を参照してください。

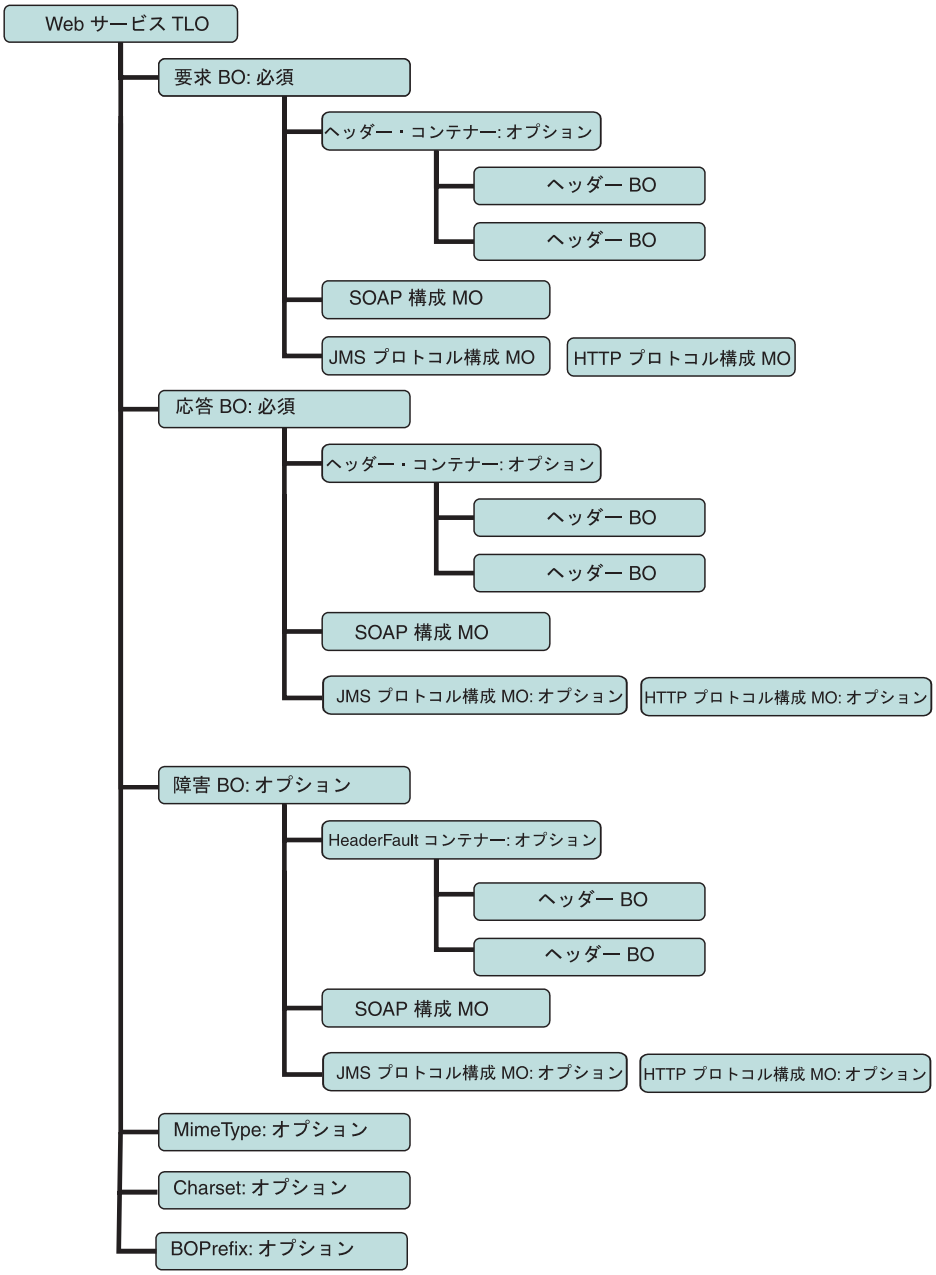


図 15. 同期要求処理のためのビジネス・オブジェクト階層

同期要求処理 TLO のオブジェクト・レベルの ASI

オブジェクト・レベルの ASI は、TLO の性質、および TLO に含まれるオブジェクトについての重要情報を提供します。図 16 は、同期要求処理のためのサンプル TLO である CLIENT_SYNCH_TLO_OrderStatus を表しています。

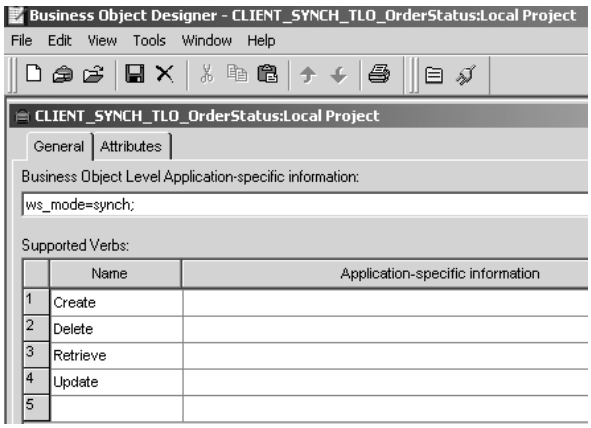


図 16. 同期要求処理のためのトップレベル・ビジネス・オブジェクト

表 16 では、同期要求処理 TLO のためのオブジェクト・レベルの ASI について説明します。同期イベント処理 TLO のための ASI とは異なり、ws_collab、ws_verb、または ws_eventtlo ASI は、このレベルでの要求処理には必須ではありません。

表 16. 同期要求処理 TLO のオブジェクト ASI

オブジェクト・レベルの ASI	説明
ws_mode=synch	要求処理の際に、コネクターは、この ASI プロパティーを使用して、Web サービスを同期 (synch) で呼び出すのか非同期 (asynch) で呼び出すのかを決定します。synch が指定された場合、コネクターは応答を予期します。したがって、TLO には、要求および応答ビジネス・オブジェクト、およびオプションとして 1 つまたは複数の障害オブジェクトを組み込む必要があります。 デフォルトは asynch です。

同期要求処理 TLO のための属性レベルの ASI

図 17 は、CLIENT_SYNCH_TLO_OrderStatus TLO の属性、および属性レベルの ASI を示しています。

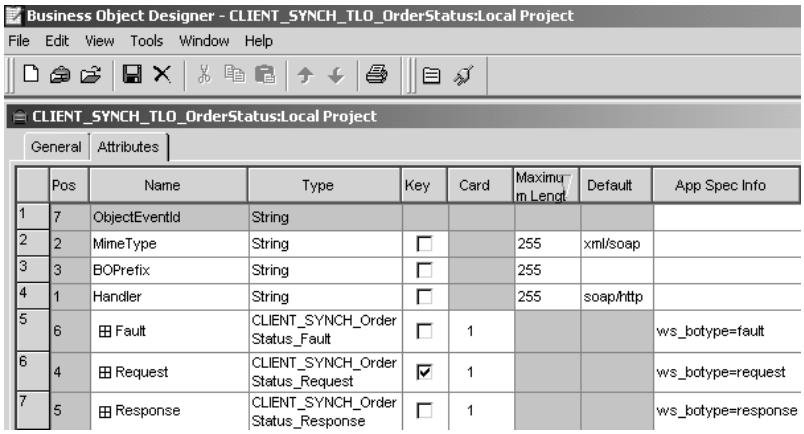


図 17. 同期要求処理のための TLO 属性

図 17 に表示された属性および ASI について、表 17 で説明します。

表 17. 要求処理の TLO 属性

TLO 属性	属性レベルの ASI	説明
MimeType	なし	この属性は、要求ビジネス・オブジェクトを要求メッセージに変換するためにコネクターが起動するデータ・ハンドラーの MIME タイプを指定します。この値は、メッセージ変換規則構成に応じて、同期応答/障害メッセージをビジネス・オブジェクトに変換するために使用される場合があります。
BOPrefix	なし	型 String のこの属性は、データ・ハンドラーに渡されます。
Handler	なし	この属性は、Web サービス要求の処理に使用するプロトコル・ハンドラーを指定もので、要求処理専用です。この値は、次のいずれかになります。 • soap/jms: コネクターは、SOAP/JMS プロトコル・ハンドラーを使用して要求を処理します。 • soap/http: コネクターは、SOAP/HTTP、SOAP/HTTPS プロトコル・ハンドラーを使用して、この Web サービス要求を処理します。 デフォルトは soap/http です。

表 17. 要求処理の TLO 属性 (続き)

TLO 属性	属性レベルの ASI	説明
Charset	なし	型 String のこのオプショナル・パラメーターは、要求ビジネス・オブジェクトをメッセージに変換するときにデータ・ハンドラーに設定される charset を指定します。注: この属性に指定された charset 値は、要求メッセージの Content-Type プロトコル・ヘッダーでは伝搬されません。
Request	`ws_botype=request`	この属性は、Web サービス要求ビジネス・オブジェクトに対応します。コネクターはこの属性 ASI を使用して、この TLO 属性のタイプが SOAP 要求 BO であるかどうかを判別します。属性名ではなく、この ASI が属性タイプを判別します。複数の要求属性がある場合は、コネクターは最初に取り込まれた属性の ASI を使用します。
Response	`ws_botype=response`	この属性は、コラボレーションによって戻される応答に対応し、同期要求処理の場合には必須です。コネクターはこの属性 ASI を使用して、この TLO 属性のタイプが SOAP 応答 BO であるかどうかを判別します。属性名ではなく、この ASI が属性タイプを判別します。

Fault	`ws_botype=fault`
	または
	`ws_botype=defaultfault`
		この属性は同期要求処理の場合のオプションであり、応答を正常に取り込むことができないときに Web サービスから戻される障害メッセージに対応するものです。
		
		コネクターはこの ASI を使用して、TLO の属性のタイプが SOAP 障害 BO であるかどうかを判別します。属性名ではなく、この ASI が属性タイプを判別します。障害メッセージが詳細要素である場合は、`defaultfault` ビジネス・オブジェクトが戻されます。`defaultfault` は、デフォルトのビジネス・オブジェクトの解決に使用されます。詳しくは、125 ページの『第 5 章 SOAP データ・ハンドラー』を参照してください。

同期要求処理のための要求ビジネス・オブジェクト

要求ビジネス・オブジェクトは、TLO の子であり、同期要求処理の場合には必須です。要求ビジネス・オブジェクトは、オブジェクト・レベルの ASI を保有しています。

例えば、CLIENT_SYNCH_OrderStatus_Request をオープンし、「一般」タブをクリックすると、図 18 に示すように、オブジェクト・レベルの ASI が表示されます。

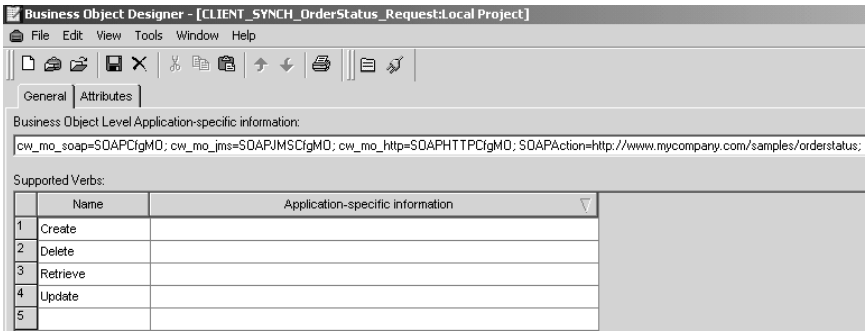


図 18. 同期要求処理のための要求オブジェクト ASI

表 18 では、同期要求処理のための要求ビジネス・オブジェクトのオブジェクト・レベルの ASI について説明します。

表 18. 同期要求処理: 要求ビジネス・オブジェクトのオブジェクト・レベルの ASI

オブジェクト・レベルの ASI	説明
cw_mo_soap=*SOAPCfgMO*	この ASI の値は、SOAP 構成 MO に対応する属性の名前と一致しなければなりません。これは、要求ビジネス・オブジェクトに対するデータ・ハンドラー変換を定義する SOAP 構成 MO です。詳しくは、 35 ページの『SOAP 構成 MO』を参照してください。
cw_mo_jms=*SOAPJMSCfgMO*	この ASI の値は、プロトコル構成 MO に対応する属性の名前と一致しなければなりません。これは、JMS プロトコル・ハンドラーの宛先 Web サービスを指定するプロトコル構成 MO です。詳しくは、 54 ページの『要求処理のための要求ビジネス・オブジェクトの JMS プロトコル構成 MO』を参照してください。
cw_mo_http=*SOAPHTTPCfgMO*	このオプショナル ASI の値は、プロトコル構成 MO に対応する属性の名前と一致しなければなりません。これは、SOAP/HTTP-HTTPS プロトコル・ハンドラーの宛先を指定する、別のプロトコル構成 MO です。この ASI は、SOAP/HTTP および SOAP/HTTPS プロトコル・ハンドラーによって使用されます。TLO 要求属性には、要求処理のための JMS または HTTP プロトコル構成 MO (使用する Web サービス・プロトコルによって異なる) がなければなりません。詳しくは、 55 ページの『要求処理のための HTTP プロトコル構成 MO』を参照してください。

表 18. 同期要求処理: 要求ビジネス・オブジェクトのオブジェクト・レベルの ASI (続き)

オブジェクト・レベルの ASI	説明
SOAPAction=*SOAPActionURI*	コネクターはこの ASI を使用して、要求メッセージに SOAPAction ヘッダーを設定するかどうかを判別します。この ASI は、宛先 Web サービスが SOAPAction ヘッダーを必要とする場合にのみ指定してください。この ASI は要求処理に使用されるものであり、イベント通知には使用されません。

同期要求処理のための応答ビジネス・オブジェクト

応答ビジネス・オブジェクトは、TLO の子であり、同期要求処理の場合には必須です。同期要求処理のための応答ビジネス・オブジェクトに対するオブジェクト・レベルの ASI は、表 19 に説明されています。

表 19. 同期要求処理: 応答ビジネス・オブジェクトのオブジェクト・レベルの ASI

オブジェクト・レベルの ASI	説明
cw_mo_soap=*SOAPCfgMO*	この ASI の値は、SOAP プロトコル構成 MO に対応する属性の名前と一致しなければなりません。これは、応答ビジネス・オブジェクトに関するデータ・ハンドラー変換を定義する SOAP 構成 MO です。詳しくは、 35 ページの『SOAP 構成 MO』を参照してください。
cw_mo_jms=*SOAPJMSCfg MO* または cw_mo_http=*SOAPHTTPCfgMO*	この ASI の値は、プロトコル構成 MO に対応する属性の名前と一致しなければなりません。これは、JMS または HTTP プロトコル・ハンドラーの応答 SOAP メッセージ内のヘッダーを指定するプロトコル構成 MO であり、応答ビジネス・オブジェクトの場合にはオプションです。詳しくは、 35 ページの『プロトコル構成 MO』を参照してください。

応答ビジネス・オブジェクトに対して、デフォルトの動詞を指定できます。これを行うには、トップレベルの要求ビジネス・オブジェクトの「サポートされている動詞 (Supported Verbs)」リストにある動詞の ASI フィールドに、

```
DefaultVerb=true;
```

と指定します。DefaultVerb ASI が指定されず、データ・ハンドラーが動詞の設定されていないビジネス・オブジェクトを処理する場合、応答ビジネス・オブジェクトが、動詞なしで戻されます。

同期要求処理のための障害ビジネス・オブジェクト

障害ビジネス・オブジェクトは、TLO の子であり、同期要求処理の場合にはオプションです。同期要求処理のための障害ビジネス・オブジェクトに対するオブジェクト・レベルの ASI は、表 9 に説明されています。

表 20. 同期要求処理: 障害ビジネス・オブジェクトのためのオブジェクト・レベルの ASI

オブジェクト・レベルの ASI	説明
cw_mo_soap=SOAPCfgMO	この ASI の値は、SOAP プロトコル構成 MO に対応する属性の名前と一致しなければなりません。これは、障害ビジネス・オブジェクトに関するデータ・ハンドラー変換を定義する SOAP 構成 MO です。詳しくは、 35 ページの『SOAP 構成 MO』を参照してください。
cw_mo_jms=*SOAPJMSCfg MO* または cw_mo_http=*SOAPHTTPCfgMO*	この ASI の値は、プロトコル構成 MO に対応する属性の名前と一致しなければなりません。これは、JMS プロトコル・ハンドラーの応答 SOAP メッセージ内のヘッダーを指定するプロトコル構成 MO であり、障害ビジネス・オブジェクトの場合にはオプションです。詳しくは、 35 ページの『プロトコル構成 MO』を参照してください。

SOAP 構成 MO

SOAP 構成 MO (SOAPCfgMO) の属性は、イベント処理用の SOAP 構成 MO の場合と同じです。詳しくは、 35 ページの『SOAP 構成 MO』および 127 ページの『SOAP 構成メタオブジェクト: 各 SOAP ビジネス・オブジェクトの子』を参照してください。

要求処理のための要求ビジネス・オブジェクトの JMS プロトコル構成 MO

JMS Web サービスを使用するときは、JMS プロトコル構成 MO は要求ビジネス・オブジェクトの場合には必須であり、応答オブジェクトと障害オブジェクトの場合はオプションです。表 21 では、要求処理の JMS プロトコル構成 MO について説明します。ここで示す Destination は、最も重要な属性であり、唯一の必須属性です。JMS プロトコル・ハンドラーはこの属性を使用して、要求された Web サービスを探し出します。このほかに、 35 ページの『プロトコル構成 MO』で示した JMS 構成 MO のすべての属性は、オプションとして指定できます。

表 21. 要求処理のための JMS プロトコル構成 MO 属性

属性	必須	型	説明
Destination	はい	String	ターゲット Web サービスの宛先キュー名。JMS プロトコル・ハンドラーはこの属性を使用して、Web サービスの宛先を判別します。コネクター固有の JNDI プロパティー LookupQueuesUsingJNDI が true に設定されている場合、JMS プロトコル・ハンドラーは JNDI を使用してこのキューを検索します。この属性が宛先キューの JNDI 名を提供することを確認してください。

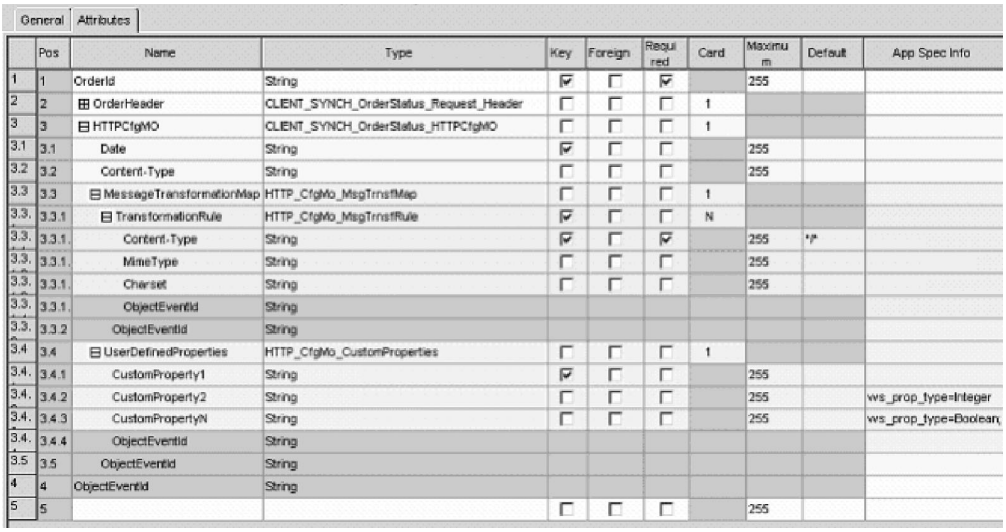
要求処理のための HTTP プロトコル構成 MO

要求処理の際に、SOAP/HTTP-HTTPS プロトコル・ハンドラーは、ターゲット Web サービスの宛先を判別するために、HTTP プロトコル構成 MO を使用します。このプロトコル構成 MO は、要求ビジネス・オブジェクトの場合には必須です。SOAP/HTTP-HTTPS プロトコル・ハンドラーは、HTTP 1.0 POST 要求のみをサポートします。 表 22 に示すように、唯一の必須属性 (Destination) はターゲット Web サービスの完全 URL です。オプションの許可属性については、以下のセクションで説明します。

表 22. 要求処理のための HTTP プロトコル構成 MO 属性

属性	必須	型	説明
Destination	はい	String	ターゲット Web サービスの宛先 URL。SOAP/HTTP-HTTPS プロトコル・ハンドラーは、この属性を使用して、Web サービスの宛先を判別します。
Content-Type	要求ビジネス・オブジェクトでは必須。それ以外ではオプション。	String	この属性の値は、発信メッセージの Content-Type ヘッダー (発信メッセージのメッセージ ContentType、およびオプションで charset が含まれる) を定義します。構文は、HTTP プロトコルの Content-Type ヘッダーと同じです (例: text/html; charset=ISO-8859-4)。Content-Type 属性が定義されていない場合、コネクターは、メッセージの ContentType として text/xml を使用します。
Authorization_UserID	いいえ	String	この属性は HTTP 基本認証の userID に対応します。詳しくは、 59 ページの『要求処理用の HTTP 証明書伝搬』を参照してください。
Authorization_Password	いいえ	String	この属性は HTTP 基本認証のパスワードに対応します。詳しくは、 59 ページの『要求処理用の HTTP 証明書伝搬』を参照してください。
1 つ以上の HTTP ヘッダー	いいえ	String	この属性を使用すれば、ハンドラーは、指定された HTTP ヘッダーの値の受け渡しまたは検索ができます。
UserDefinedProperties	いいえ	ビジネス・オブジェクト	この属性は、ユーザー定義のプロトコル・プロパティーのビジネス・オブジェクトを保持します。詳しくは、 56 ページの『要求処理のためのユーザー定義のプロパティー』を参照してください。
MessageTransformationMap	いいえ	単一カーディナリティー・ビジネス・オブジェクト	この属性は、0 個またはそれ以上のメッセージ変換規則を保持するビジネス・オブジェクトを指します。この変換規則には、規則に指定された着信メッセージに適用する MIME タイプおよび charset に関する情報が入っています。詳しくは、 58 ページの『メッセージ変換マップ』を参照してください。

図 19 に、Business Object Designer Express における HTTP プロトコル構成 MO 属性の一部を示します。



General | Attributes

	Pos	Name	Type	Key	Foreign	Required	Card	Maximum	Default	App Spec Info
1	1	OrderId	String	☑	☐	☑		255		
2	2	⊞ OrderHeader	CLIENT_SYNCH_OrderStatus_Request_Header	☐	☐	☐	1			
3	3	⊟ HTTPCfgMO	CLIENT_SYNCH_OrderStatus_HTTPCfgMO	☐	☐	☐	1			
3.1	3.1	Date	String	☑	☐	☐		255		
3.2	3.2	Content-Type	String	☐	☐	☐		255		
3.3	3.3	⊟ MessageTransformationMap	HTTP_CfgMo_MsgTrnsfMap	☐	☐	☐	1			
3.3.	3.3.1	⊟ TransformationRule	HTTP_CfgMo_MsgTrnsfRule	☑	☐	☐	N			
3.3.	3.3.1.	Content-Type	String	☑	☐	☑		255	*/*	
3.3.	3.3.1.	MimeType	String	☐	☐	☐		255		
3.3.	3.3.1.	Charset	String	☐	☐	☐		255		
3.3.	3.3.1.	ObjectEventId	String							
3.3.	3.3.2	ObjectEventId	String							
3.4	3.4	⊟ UserDefinedProperties	HTTP_CfgMo_CustomProperties	☐	☐	☐	1			
3.4.	3.4.1	CustomProperty1	String	☑	☐	☐		255		
3.4.	3.4.2	CustomProperty2	String	☐	☐	☐		255		ws_prop_type=Integer
3.4.	3.4.3	CustomPropertyN	String	☐	☐	☐		255		ws_prop_type=Boolean
3.4.	3.4.4	ObjectEventId	String							
3.5	3.5	ObjectEventId	String							
4	4	ObjectEventId	String							
5	5			☐	☐	☐		255		

図 19. 要求処理のための HTTP プロトコル構成 MO 属性

HTTP プロトコル構成 MO の属性を以下に示します。

- 『要求処理のためのユーザー定義のプロパティー』
- 58 ページの『メッセージ変換マップ』
- 59 ページの『要求処理用の HTTP 証明書伝搬』

要求処理のためのユーザー定義のプロパティー: オプションで、HTTP プロトコル構成 MO にカスタム・プロパティーを指定できます。この指定を行うには、UserDefinedProperties 属性を組み込みます。この属性は、1 つ以上の子属性とプロパティー値を持つビジネス・オブジェクトに対応します。このビジネス・オブジェクトのすべての属性は、メッセージ・ヘッダーの変数部分で読み取られる (同期応答の場合は書き込まれる) 単一プロパティーを以下のように定義する必要があります。

- 属性の型は、プロトコル・プロパティー・タイプにかかわらず、常に `String` です。属性のアプリケーション固有情報には、属性がマップされるプロトコル・メッセージ・プロパティーの名前とフォーマットを定義した、2 組の名前と値のペアを入れることができます。

表 23 は、これらの属性のアプリケーション固有の情報を要約したものです。

表 23. ユーザー定義のプロトコル・プロパティー属性のアプリケーション固有情報: 名前 = 値のペアの内容

名前	値	説明
ws_prop_name (大/小文字を区別しない。指定されなければ、属性名がプロパティー名として使用される)	任意の有効なプロトコル・プロパティー名	これはプロトコル・プロパティーの名前です。ベンダーによっては、いくつかのプロパティーが拡張機能用に予約されています。一般に、このようなベンダー固有の機能にアクセスしようとする場合以外は、JMS で始まるカスタム・プロパティー (JMS プロトコルの場合) を定義しないでください。
ws_prop_type (大/小文字を区別しない。JMS 用オプション。指定されなければ、String とみなされる。HTTP(S) の場合、String 型しか意味を成さないため、関係しない)	String、Integer、Boolean、Float、Double、Long、Short	プロトコル・プロパティーのタイプ。JMS プロトコルの場合、JMS API には JMS メッセージにプロパティー値を設定するメソッドとして、setIntProperty、setLongProperty、setStringProperty などがあります。ここで指定される JMS プロパティーのタイプによって、これらのどのメソッドを使用してメッセージのプロパティー値を設定するかが決まります。

指定されたカスタム・プロパティー ASI (ws_prop_name または ws_prop_type) が無効で、このヘッダーを処理する論理的な方法 (HTTP 処理でプロパティー・タイプを無視するなど) がない場合、コネクターはログに警告を記録し、このプロパティーを無視します。ws_prop_name または ws_prop_type に対しての必要な検査は実行されたが、カスタム・プロパティーの値を設定することも検索することもできない場合、コネクターはログにエラーを記録し、イベントは失敗します。

UserDefinedProperties 属性が指定され、そのビジネス・オブジェクトのインスタンスが作成されると、コネクターは、この子ビジネス・オブジェクトの各属性を処理し、それに対応するメッセージ・プロパティー値を設定します。

同期要求処理の場合は、Web サービス/URL から応答メッセージを受け取ったときに、UserDefinedProperties 属性が指定されていると、コネクターは UserDefinedProperties ビジネス・オブジェクトのインスタンスを作成します。次に、メッセージからプロパティー値を抽出して、それを新規ビジネス・オブジェクトに保管します。少なくとも 1 つのプロパティー値が正常に検索されれば、コネクターは、変更された UserDefinedProperties ビジネス・オブジェクトをプロトコル構成 MO に設定します。

メッセージ変換マップ: メッセージ変換マップ (MTM) 機能は、要求処理 HTTP(S) プロトコル・ハンドラーでのみサポートされます。MessageTransformationMap はプロトコル構成 MO のオプションの属性で、ビジネス・オブジェクトを指します。このビジネス・オブジェクトには、規則に指定された MIME タイプおよび charset を持つメッセージを変換する規則が格納されています。属性名が MessageTransformationMap (大/小文字を区別する) で、この属性がビジネス・オブジェクト・オブジェクト・タイプの場合 (図 19 を参照)、コネクターは、そのオブジェクト内の規則を使用して、メッセージを変換します。

図 19 に示すように、MTM 属性には、カーディナリティー N の子ビジネス・オブジェクト属性が 1 つあり、その属性名は TransformationRule です。メッセージの TransformationRule を検索するときに、SOAP/HTTP(S) プロトコル・ハンドラーはまず、そのメッセージが、すべての TransFormationRule に指定された ContentType と正確に一致するかを確認します。一致しない場合、コネクターは、複数のメッセージ・タイプに適用される規則を検索します。プロトコル・ハンドラーの処理について詳しくは、85 ページの『SOAP/HTTP-HTTPS プロトコル・ハンドラー処理』を参照してください。

TransformationRule ビジネス・オブジェクトの各インスタンスでは、表 24 に示すとおりに属性を指定する必要があります。

表 24. HTTP プロトコル構成 MO における MessageTransformationMap の TransformationRule 属性

属性名	必須	型	デフォルト値	説明
TransformationRule	いいえ	ビジネス・オブジェクト、カーディナリティー N	なし	この属性は、メッセージ変換に関する規則を 1 つ保持します。MessageTranformationMap 属性の下にあるこの属性のインスタンスは 0 個の場合も、それ以上の場合もあります。
+ContentType	はい	String	*/*	このプロパティーの値は、この変換規則が適用されるメッセージの HTTP ContentType を示します。この属性にデフォルト値 */* が指定された場合、コネクターは、この規則を任意の ContentType に適用できます。プロトコル・ハンドラーの処理の詳細については、85 ページの『SOAP/HTTP-HTTPS プロトコル・ハンドラー処理』を参照してください。他の規則と同じ ContentType を持つ規則を複数検出した場合、プロトコル・ハンドラーはログに警告を記録し、重複する規則をすべて無視しますが、固有の規則は使用します。
+MimeType	いいえ	String	なし	このビジネス・オブジェクトに指定された ContentType のメッセージの処理でデータ・ハンドラーを呼び出すときに使用する MIME タイプ。

表 24. *HTTP プロトコル構成 MO における MessageTransformationMap の TransformationRule 属性 (続き)*

属性名	必須	型	デフォルト値	説明
+Charset	いいえ	String	なし	このビジネス・オブジェクトに指定された ContentType の要求を変換するときに使用する charset。

要求処理用の HTTP 証明書伝搬: 証明書伝搬のために、コネクターは HTTP プロトコル構成 MO の Authorization_UserID 属性および Authorization_Password 属性をサポートします。このサポートは、HTTP 基本認証スキームの一部としてのこれらの証明書の伝搬に限られます。

要求処理中に証明書を伝搬する必要がある場合は、WSDL ODA によって生成されたプロトコル構成 MO に Authorization_UserID 属性および Authorization_Password 属性を手動で追加する必要があります。この操作は、ビジネス・オブジェクト定義およびメタ・オブジェクト定義を生成した後に Business Object Designer Express で行います。(WSDL ODA の詳細については、165 ページの『第 6 章 要求処理のためのコラボレーションの有効化』を参照してください。)

コラボレーションは、プロトコル構成 MO の Authorization_UserID 属性および Authorization_Password 属性の値を設定します。これらの属性が null でも空でもない場合は、コネクターは、ターゲット Web サービスに送信する要求に許可ヘッダーを作成します。SOAP HTTP/HTTPS プロトコル・ハンドラーは、許可ヘッダーの作成時に *HTTP Authentication: Basic and Digest Access Authentication (RFC 2617)* に従います。

注: RFC2617 で規定されている HTTP 認証のダイジェスト認証方式も、オプションのチャレンジ・レスポンス機構もサポートされていません。信任状を要求するサーバーを HTTP(S) プロトコル・ハンドラーが呼び出した場合、コネクターはサーバーからのチャレンジ・レスポンスを待ちません。代わりに、信任状を連続して送信します。

非同期要求処理 TLO

図 20 は、非同期要求処理のためのビジネス・オブジェクト階層を示しています。必須オブジェクトは要求オブジェクトのみで、このオブジェクトには、SOAP データ・ハンドラーのための SOAP 構成 MO、および SOAP/JMS プロトコル・ハンドラーと SOAP/HTTP/HTTPS プロトコル・ハンドラー用の 2 つのプロトコル構成 MO が含まれます。これらについて、以下のセクションで説明します。

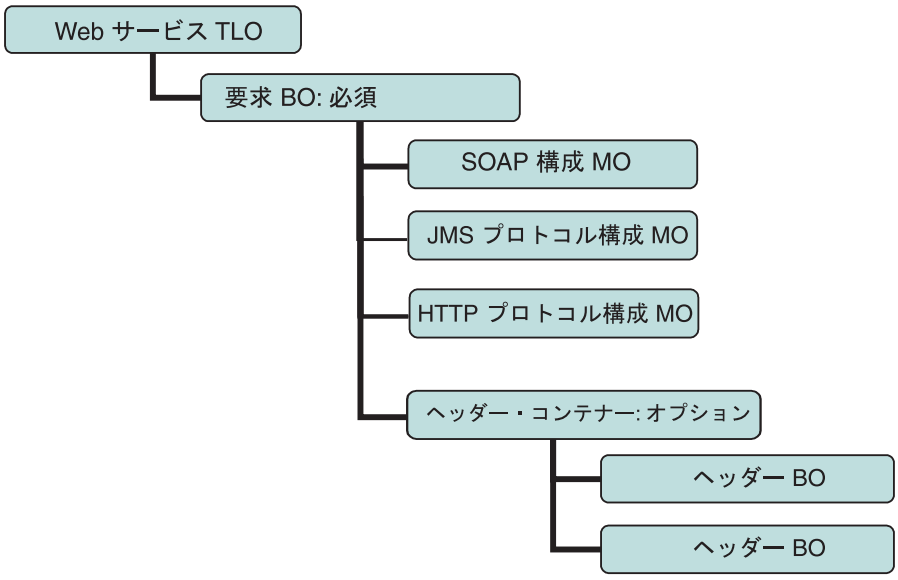


図 20. 非同期要求処理のためのビジネス・オブジェクト階層

TLO には、オブジェクト・レベルの ASI のほか、属性レベルの ASI を持った属性が含まれています。両方の種類の ASI について、以下で説明します。ヘッダー・コンテナーおよびヘッダー子ビジネス・オブジェクトについての詳細は、41 ページの『ヘッダー・コンテナー・ビジネス・オブジェクト』を参照してください。

非同期イベント処理 TLO のためのオブジェクト・レベルの ASI

図 21 は、非同期要求処理のためのサンプル TLO である CLIENT_ASYNCH_Order_TLO を表しています。

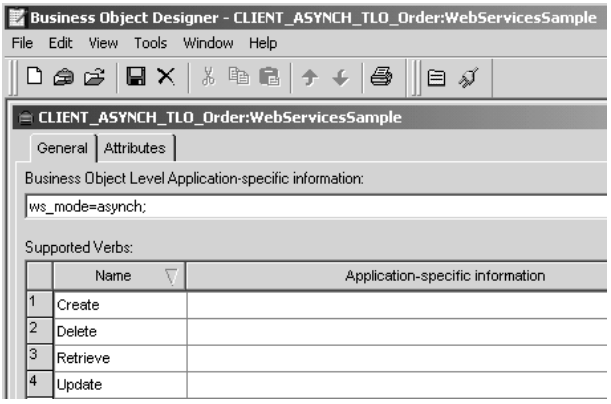


図 21. 非同期要求処理のためのトップレベル・ビジネス・オブジェクト

以下の表 25 では、非同期要求処理 TLO のオブジェクト・レベルの ASI について説明します。

表 25. 非同期要求処理 TLO のオブジェクト ASI

オブジェクト・レベルの ASI	説明
ws_mode=asynch	要求処理の際に、コネクターは、この ASI プロパティーを使用して、コラボレーションを同期 (synch) で呼び出すのか非同期 (asynch) で呼び出すのかを決定します。非同期要求処理の場合は、この ASI を asynch に設定する必要があります。 デフォルトは asynch です。

非同期要求処理 TLO のための属性レベルの ASI

図 22 は、要求処理 TLO のサンプルである CLIENT_ASYNCH_TLO_Order の属性を表しています。

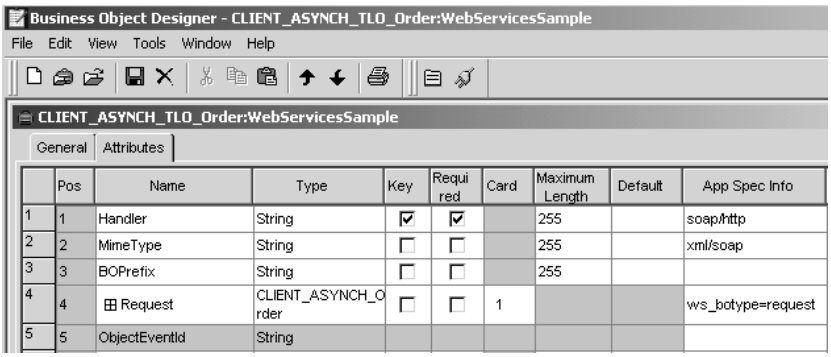


図 22. 非同期要求処理のための TLO 属性

表 26 は、非同期要求処理 TLO の要求属性に対する属性レベルの ASI を要約したものです。

表 26. 非同期要求処理 TLO の属性

TLO 属性	属性レベルの ASI	説明
MimeType	なし	この属性は、コネクターが呼び出すデータ・ハンドラーの MIME タイプを指定します。この属性は要求処理のみに使用されます。(イベント処理の場合、プロトコル・リスナーは SOAPDHMimeType コネクター固有構成プロパティーを使用します。) デフォルトは xml/soap です。
BOPrefix	なし	String タイプのこの属性は、今後の開発のために予約されているもので、必須ではありません。

表 26. 非同期要求処理 TLO の属性 (続き)

TLO 属性	属性レベルの ASI	説明
Handler	なし	この属性は、Web サービス要求の処理に使用するプロトコル・ハンドラーを指定もので、要求処理専用です。この値は、次のいずれかになります。 • `soap/jms`: コネクターは、SOAP/JMS プロトコル・ハンドラーを使用して要求を処理します。 • `soap/http`: コネクターは、SOAP/HTTP-HTTPS プロトコル・ハンドラーを使用して、この Web サービス要求を処理します。 デフォルトは `soap/http` です。
Request	`ws_botype=request`	この属性は、Web サービス要求ビジネス・オブジェクトに対応します。コネクターはこの属性 ASI を使用して、この TLO 属性のタイプが SOAP 要求 BO であるかどうかを判別します。属性名ではなく、この ASI が属性タイプを判別します。複数の要求属性がある場合は、コネクターは最初の要求属性の ASI を使用します。

非同期要求処理のための要求ビジネス・オブジェクト

要求ビジネス・オブジェクトは、TLO の子であり、非同期要求処理の場合には必須です。非同期要求処理のための要求ビジネス・オブジェクトに対するオブジェクト・レベルの ASI は、表 27 で説明します。

表 27. 非同期要求処理: 要求ビジネス・オブジェクトのオブジェクト・レベルの ASI

オブジェクト・レベルの ASI	説明
`cw_mo_soap=`*`SOAPCfgMO`*	この ASI の値は、SOAP 構成 MO に対応する属性の名前と一致しなければなりません。これは、要求ビジネス・オブジェクトに対するデータ・ハンドラー変換を定義する SOAP 構成 MO です。詳しくは、 35 ページの『SOAP 構成 MO』を参照してください。
`cw_mo_jms=`*`SOAPJMSCfgMO`*	この ASI の値は、プロトコル構成 MO に対応する属性の名前と一致しなければなりません。これは、JMS プロトコル・ハンドラーの宛先 Web サービスを指定するプロトコル構成 MO です。詳しくは、 54 ページの『要求処理のための要求ビジネス・オブジェクトの JMS プロトコル構成 MO』を参照してください。

表 27. 非同期要求処理: 要求ビジネス・オブジェクトのオブジェクト・レベルの ASI (続き)

オブジェクト・レベルの ASI	説明
cw_mo_http=*SOAPHTTPCfgMO*	この ASI の値は、プロトコル構成 MO に対応する属性の名前と一致しなければなりません。これは、SOAP/HTTP-HTTPS プロトコル・ハンドラーの宛先を指定する、別のプロトコル構成 MO です。この ASI は、SOAP/HTTP-HTTPS プロトコル・ハンドラーによって使用されます。要求処理のためには、TLO 要求属性が JMS と HTTP の両方のプロトコル構成 MO を備えている必要があります。詳しくは、55 ページの『要求処理のための HTTP プロトコル構成 MO』を参照してください。
SOAPAction=*SOAPActionURI*	コネクターはこの ASI を使用して、要求メッセージに SOAPAction ヘッダーを設定するかどうかを判別します。この ASI は、宛先 Web サービスが SOAPAction ヘッダーを必要とする場合にのみ指定してください。この ASI は要求処理に使用されるものであり、イベント通知には使用されません。

図 14 に示すサンプルでは、要求属性には、SOAP 構成 MO、ヘッダー・コンテナー (OrderHeader) のほか、内容に関連した属性 (OrderLineItems) が含まれています。非同期要求処理の場合の SOAP 構成 MO、プロトコル構成 MO、SOAP ヘッダー・コンテナー、およびヘッダー子ビジネス・オブジェクトの要件および特性は、同期要求処理の場合と同じです。詳しくは、47 ページの『同期要求処理 TLO』における該当のトピックを参照してください。

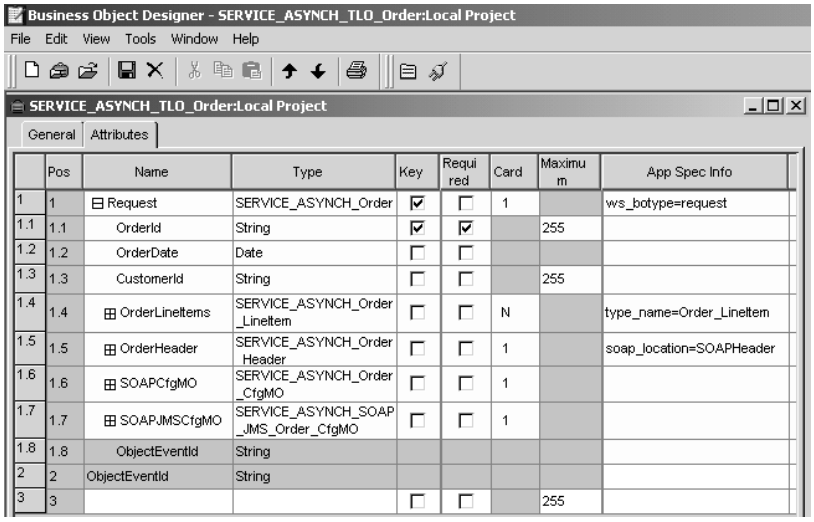


図 23. 非同期イベント処理のための要求属性

非同期要求処理のための構成 MO

SOAP 構成 MO (SOAPCfgMO) の属性は、イベント処理用の SOAP 構成 MO の場合と同じです。詳しくは、 35 ページの『SOAP 構成 MO』を参照してください。また、 127 ページの『SOAP 構成メタオブジェクト: 各 SOAP ビジネス・オブジェクトの子』も参照してください。

JMS Web サービスを使用するときは、JMS プロトコル構成 MO は要求ビジネス・オブジェクトの場合には必須です。詳しくは、 54 ページの『要求処理のための要求ビジネス・オブジェクトの JMS プロトコル構成 MO』を参照してください。

要求処理の際に、SOAP/HTTP-HTTPS プロトコル・ハンドラーは、ターゲット Web サービスの宛先を判別するために、HTTP プロトコル構成 MO を使用します。このプロトコル構成 MO は、要求ビジネス・オブジェクトの場合には必須です。詳しくは、 55 ページの『要求処理のための HTTP プロトコル構成 MO』を参照してください。

ビジネス・オブジェクトの開発

ビジネス・オブジェクトを作成するには、Business Object Designer Express を使用し、ビジネス・オブジェクトをサポートするようにコネクターを構成するには、Connector Configurator Express を使用します。Business Object Designer Express ツールの詳細については、「*ビジネス・オブジェクト開発ガイド*」および 167 ページの『第 7 章 Web サービスとしてのコラボレーションの公開』を参照してください。Connector Configurator Express の詳細については、 217 ページの『付録 B. Connector Configurator Express』を参照してください。

第 4 章 Web サービス・コネクター

この章では、Web サービス・コネクターとその構成方法について説明します。

すべての WebSphere Business Integration コネクターは、統合ブローカーと連動して動作します。Web サービス・コネクターは、InterChange Server Express 統合ブローカーと連動します。これについては、「*システム・インプリメンテーション・ガイド*」で説明されています。

コネクターは、アダプターのランタイム・コンポーネントです。コネクターは、アプリケーション固有のコンポーネントとコネクター・フレームワークからなります。アプリケーション固有のコンポーネントには、特定のアプリケーションに合わせて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 (このコードは、すべてのコネクターで共通です) 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を果たします。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サービスを提供します。

- ビジネス・オブジェクトの送受信
- 開始メッセージおよび管理メッセージの交換の管理

本書には、アプリケーション固有のコンポーネントおよびコネクター・フレームワークに関する情報が記載されています。本書では、これら両方のコンポーネントをコネクターと呼んでいます。

統合ブローカーとコネクターの関係に関する詳細については、「*システム管理ガイド*」を参照してください。

コネクター処理

コネクターには、イベント処理を行うプロトコル・リスナー・フレームワークと要求処理を行うプロトコル・ハンドラー・フレームワークがあります。コネクター・フレームワークでは、この双方向の機能を利用して以下のことが可能になります。

- コラボレーションを Web サービスとして公開して、Web サービス・クライアントからの呼び出しを処理する
- Web サービスを呼び出すコラボレーションの要求を処理する

SOAP データ・ハンドラーについて詳しくは、125 ページの『第 5 章 SOAP データ・ハンドラー』を参照してください。

注: コネクターは、SOAP/HTTP および SOAP/JMS バインディングのみをサポートします。

イベント処理の概要

コネクター・イベント処理 (またはイベント通知) は、Web サービスからの要求を処理するために使用します。このイベント処理機能は、以下のコンポーネントを含む、プロトコル・リスナー・フレームワークを採用しています。これらのコンポーネントについては、この章の後半でさらに詳しく説明します。

- SOAP/HTTP プロトコル・リスナー
- SOAP/HTTPS プロトコル・リスナー
- SOAP/JMS プロトコル・リスナー

コネクターはリスナーを使用して、コラボレーションを Web サービスとして公開し、Web サービス・クライアントから公開済みコラボレーションへの呼び出しのためのトランスポートを listen します。

SOAP/HTTP および SOAP/HTTPS プロトコル・リスナーは、コラボレーションを SOAP/HTTP Web サービスとして公開します。SOAP/JMS プロトコル・リスナーは、コラボレーションを SOAP/JMS Web サービスとして公開します。

Web サービス・クライアントからの要求が到着すると、リスナーは、SOAP 要求メッセージをビジネス・オブジェクトに変換し、コラボレーションを呼び出します。これが同期要求である場合、コネクターは同一タイプの応答ビジネス・オブジェクトを、要求ビジネス・オブジェクトとして受信します。リスナーは、応答ビジネス・オブジェクトを SOAP 応答メッセージに変換します。リスナーは、次に SOAP 応答メッセージを Web サービス・クライアントにトランスポートします。イベントの順序付けは、このコネクターの要件ではないので注意してください。つまり、コネクターはイベントをどのような順序でもデリバリーできます。

Web サービス・コネクターは、SOAP データ・ハンドラーを使用して、着信した SOAP 要求メッセージをビジネス・オブジェクトに変換します。着信した SOAP 要求メッセージをどのビジネス・オブジェクトで解決すべきか、データ・ハンドラーが判別できるように、コネクターは、サポートされているビジネス・オブジェクトに関するメタ情報をデータ・ハンドラーに提供します。コネクターはまず最初に、サポートされているビジネス・オブジェクトの中から変換の候補となるすべてのビジネス・オブジェクトのリストを作成します。このリストは、TLO と非 TLO の両方から構成される可能性があります。サポートされている TLO ビジネス・オブジェクトは、オブジェクト・レベル ASI が `ws_eventtlo=true` のビジネス・オブジェクトです。

TLO を使用する場合、プロトコル・リスナーは、以下のようにして TLO のオブジェクト・レベル ASI を読み取ります。

- ws_collab= これは、どのコラボレーションを呼び出すのかを決定します。
- ws_mode= これは、コラボレーションを呼び出す方法 (同期 (synch) または非同期 (asynch)) を決定します。

非 TLO を使用する場合、プロトコル・リスナーは、WSDL 構成ウィザードにより生成された WSCollaborations 構成プロパティー値から、コラボレーションおよび処理モードを読み取ります。

コネクターは SOAP 要求の BodyName と BodyNamespace を変換候補のビジネス・オブジェクトの名前と比較し、突き合わせます。TLO の場合、この BodyName/BodyNamespace のペアは、SOAP 要求ビジネス・オブジェクトの SOAP 構成 MO プロパティーを使用して検出されます。非 TLO の場合、BodyName/BodyNamespace のペアは、WSCollaborations コネクター構成プロパティーを使用して検出されます。(コネクターは、WSCollaborations プロパティーにエントリーのある 非 TLO の場合のみを検討しますので注意してください。) データ・ハンドラーは、この BodyName/BodyNamespace ペアを用いて、SOAP 要求をビジネス・オブジェクトに変換する際に使用するビジネス・オブジェクトを判別します。

コネクターは SOAP データ・ハンドラーから戻された要求ビジネス・オブジェクトを検査します。このビジネス・オブジェクトに ws_tloname ASI が存在する場合は、コネクターはこの TLO に要求ビジネス・オブジェクトを設定します。この TLO はコラボレーションを呼び出すために使用されます。ただし、この ASI が設定されていなければ、コネクターは、SOAP データ・ハンドラーから戻された要求ビジネス・オブジェクトを使用して、コラボレーションを呼び出します。

コラボレーションを同期実行する場合、コネクターは SOAP データ・ハンドラーを使用して、クライアントに戻す SOAP 応答または障害メッセージを作成します。この場合、コネクターは、SOAP ビジネス・オブジェクト (TLO の子) または非 TLO のいずれかをデータ・ハンドラーに渡すだけです。SOAP データ・ハンドラーは、渡されたビジネス・オブジェクトに基づいて SOAP メッセージを戻します。

要求処理の概要

コネクターは、コラボレーションのために、SOAP/HTTP(S) および SOAP/JMS を介して Web サービスを呼び出すことができます。この要求処理機能は、WSDL Object Discovery Agent (ODA) およびプロトコル・ハンドラー・フレームワークによってサポートされています。WSDL ODA は、ターゲット Web サービスに関する情報を持つ SOAP ビジネス・オブジェクトを生成する際に使用する設計時ツールです。詳しくは、165 ページの『第 6 章 要求処理のためのコラボレーションの有効化』を参照してください。プロトコル・ハンドラー・フレームワークは、以下のコンポーネントから成る、構成可能なランタイム・モジュールです。これらのコンポーネントについては、この章の後半でさらに詳しく説明します。

- SOAP/HTTP-HTTPS プロトコル・ハンドラー
- SOAP/JMS プロトコル・ハンドラー

プロトコル・ハンドラー・フレームワークは、コラボレーションの要求ビジネス・オブジェクト (WSDL ODA を介して常に TLO に設定されている) を受け取ると、該当するプロトコル・ハンドラーをロードします。プロトコル・ハンドラーは、Web サービスの呼び出しおよび (オプションで) 応答の保護に必要なトランスポー

ト・レベルの詳細を管理し、コラボレーション要求ビジネス・オブジェクトの SOAP 要求メッセージへの変換、要求メッセージによるエンドポイント Web サービスの呼び出し、および SOAP 応答メッセージのビジネス・オブジェクトへの変換ならびにそのオブジェクトのコラボレーションへのリターン (要求/応答 (同期) モードの場合)、という 3 つの主要なタスクを実行します。コネクターは、SOAP/HTTP-HTTPS プロトコル・ハンドラーを使用して SOAP/HTTP(S) Web サービスを呼び出し、SOAP/JMS プロトコル・ハンドラーを使用して SOAP/JMS Web サービスを呼び出します。

Web サービス・コネクターは常に TLO を使用してコラボレーションから呼び出されます。コネクターは、TLO から SOAP 要求ビジネス・オブジェクトを判別して、このビジネス・オブジェクトにより SOAP データ・ハンドラーを呼び出します。データ・ハンドラーは、コネクターにより送信された要求メッセージを Web サービスに戻します。

Web サービスを同期実行する場合、コネクターは SOAP データ・ハンドラーを使用して、SOAP 応答および障害メッセージをそれぞれ SOAP 応答ビジネス・オブジェクトおよび障害ビジネス・オブジェクトに変換します。データ・ハンドラーが、これらの SOAP 応答/障害をビジネス・オブジェクトに変換する際に、どのビジネス・オブジェクトにより解決すべきかを判別できるようにするために、コネクターは、特定のメタ情報をデータ・ハンドラーに提供します。具体的には、コネクターは、呼び出し側の TLO の子であるすべての応答および障害ビジネス・オブジェクトのリストを作成します。応答ビジネス・オブジェクトは 1 つだけ存在している必要があり、オプションで多数の障害ビジネス・オブジェクトが存在します。defaultfault ビジネス・オブジェクトが 1 つだけ存在している場合もあります。コネクターは SOAP の BodyName と BodyNamespace をすべての応答ビジネス・オブジェクトのリスト内にあるビジネス・オブジェクト名と突き合わせ、次にマップします。SOAP 応答ビジネス・オブジェクトの場合、このペアは、SOAP 応答ビジネス・オブジェクトの SOAP 構成 MO プロパティーを使用して検出されます。SOAP 障害ビジネス・オブジェクトの場合、このペアは、詳細エレメントの最初の子の elem_name および elem_ns 属性レベル ASI プロパティーを使用して検出されます。defaultfault ビジネス・オブジェクトの場合、コネクターは、データ・ハンドラーに defaultfault ビジネス・オブジェクトの名前を通知するだけです。この変換で解決する障害ビジネス・オブジェクトが他にない場合、最後の手段として、データ・ハンドラーによって defaultfault ビジネス・オブジェクトを解決する必要があります。

SOAP/HTTP(S) Web サービス

Web サービスは HTTP トランスポート・プロトコルをサポートします。HTTP は、HTTP クライアントが接続を開き、要求メッセージを HTTP サーバーに送信するというクライアント/サーバー・モデルを実現します。クライアント要求メッセージは Web サービスを呼び出します。HTTP サーバーは、呼び出しを含むメッセージをディスパッチして、接続を閉じます。

コネクターの SOAP/HTTP および SOAP/HTTPS プロトコル・リスナーは、Web サービスとして公開されているコラボレーションに対するクライアント要求を処理する際に HTTP クライアント/サーバーおよび要求/応答モデルを利用します。ただし、SOAP/HTTP リスナーは、HTTP サーバー (プロキシー、仲介、またはそれ以

外) として機能するようになっていません。SOAP/HTTP リスナーは、どちらかといえば、企業内およびファイアウォールの内側で使用するエンドポイントとして機能します。したがって、クライアント要求をリスナーに送るには、ファイアウォール内に別の Web サーバーまたはゲートウェイを配置する必要があります。詳しくは、1 ページの『第 1 章 コネクターの概要』を参照してください。

SOAP/HTTP および SOAP/HTTPS プロトコル・リスナーは、コラボレーションを SOAP/HTTP(S) Web サービスとして公開します。コネクターは、SOAP/HTTP-HTTPS プロトコル・ハンドラーを使用して SOAP/HTTP(S) Web サービスを呼び出します。

同期 SOAP/HTTP(S) Web サービス

コネクター処理の観点からは、同期 HTTP Web サービスは、要求/応答の流れをたどるサービスです。SOAP/HTTP または SOAP/HTTPS プロトコル・リスナーにより HTTP 要求メッセージが正常に処理された場合、メッセージ本文には Web サービス応答および HTTP 状況コード `200 OK` が記載されます。障害が戻された場合、本文には障害メッセージおよび状況コード `500` が記載されます。

非同期 SOAP/HTTP(S) Web サービス

コネクター処理の観点からは、非同期 HTTP Web サービスは、要求専用の流れをたどるサービスです。SOAP/HTTP または SOAP/HTTPS プロトコル・リスナーにより、要求専用 Web サービス操作が正常に受信および処理された場合、HTTP 状況コード `202 Accepted` が生成されます。HTTP 状況コード `200 OK` が生成されるようにコネクターを構成することもできます。詳しくは、表 41 の HTTPAsyncResponseCode プロパティーを参照してください。障害が発生すると、HTTP 状況コード `500` が生成されます。応答はありませんが、障害の本文が戻されることがあります。

SOAP/JMS Web サービス

JMS は、企業が、メッセージング、データ永続性、および Java ベースのアプリケーションへのアクセスを実行するために Web サービス・ソリューションと結合できる、トランスポート・レベルの API です。SOAP/JMS Web サービスは、JMS キュー・ベースのトランスポートをインプリメントする Web サービスです。

Web サービス・ソリューションは、キューまたはトピック用の JMS 宛先をインプリメントする場合があります。コネクターの SOAP/JMS プロトコル・リスナーは、キュー宛先のみをサポートします。トピックはサポートされません。JMS テキスト・メッセージだけをサポートしています。

イベント処理時に SOAP/JMS Web サービス・クライアントは、要求メッセージを JMS メッセージでラップし、コネクターを JMS 宛先として持つキューにそれをパブリッシュします。JMS 宛先は Web サービス要求が含まれる JMS メッセージを検索し、そこから SOAP 要求メッセージを抽出します。次に SOAP 要求メッセージを処理します。

同期 SOAP/JMS Web サービス

同期コネクター処理 (要求/応答) の場合、応答メッセージは JMS メッセージでラップされます (要求メッセージの場合と同様)。Web サービス応答を含んだ JMS メッセージは、次に、着信要求から JMSReplyTo キューに送信されます。応答メッセージ内の JMS ヘッダーは、以下のようにして JMS 要求メッセージ内のヘッダーの値に設定されます。

- 応答メッセージの JMSCorrelationID は、JMS 要求メッセージの JMSMessageID の値に設定する必要がある。
- 応答メッセージの JMS DeliveryMode は、要求の JMSDeliveryMode に設定する。
- 応答メッセージの JMSPriority は、要求の JMSPriority に設定する。
- 要求メッセージの JMSExpiration は、要求の JMSExpiration に設定する。

この処理の詳細については、76 ページの『SOAP/JMS プロトコル・リスナー処理』で説明しています。

非同期 SOAP/JMS Web サービス

コネクター処理の観点からは、非同期 SOAP/JMS Web サービスは、要求専用の流れをたどるサービスです。SOAP/JMS プロトコル・リスナーが要求専用の Web サービス・メッセージを正常に受信および処理しても、応答を含む JMS メッセージがクライアントに戻されることはありません。ReplyToQueue が構成されている場合に JMS メッセージの受信時に障害が発生すると、障害メッセージは Web サービス・クライアントに戻ります。また、SOAP/JMS リスナーに ErrorQueue が指定されている場合、障害メッセージはリスナーにアーカイブされます。

イベント処理

イベント処理機能をインプリメントする際の最初のステップとして、ビジネス・プロセス (コラボレーション) を Web サービスとして公開します。次に、この Web サービスを、例えば UDDI レジストリーにパブリッシュし、コラボレーションを呼び出す Web サービス・クライアントに応答するようにコネクターを構成します。

イベント処理中に、コネクターはプロトコル・リスナーと SOAP データ・ハンドラーを使用して、Web サービス・クライアントの SOAP 要求メッセージを、Web サービスとして公開されているコラボレーションで操作できるビジネス・オブジェクトに変換します。プロトコル・リスナーは、イベント処理において極めて重要な役割を果たしています。

プロトコル・リスナー

Web サービス要求は、HTTP、HTTPS、および JMS などをはじめとする、さまざまなトランスポートを介して送られてきます。Web サービス・プロトコル・リスナーは、トランスポート・チャネルにこのような要求が到着するのをモニターします。プロトコル・リスナーには次の 3 種類があり、それぞれに対応するチャネルがあります。

- SOAP/HTTP プロトコル・リスナー
- SOAP/HTTPS プロトコル・リスナー

- SOAP/JMS プロトコル・リスナー

これらの各リスナーは、トランスポートで listen するスレッドで構成されます。クライアントから SOAP 要求メッセージを受け取ると、リスナーは、プロトコル・リスナー・フレームワークにそのイベントを登録します。

プロトコル・リスナー・フレームワークは、プロトコル・リスナーを管理し、リソースが使用可能になったときに要求を処理するようにスケジューリングします。コネクター固有のプロパティーに値を設定するときに、プロトコル・リスナー・フレームワークのリスナーおよび性質を構成してください。構成可能なプロトコル・リスナー・フレームワークのプロパティーには、以下のものが含まれます。

- **WorkerThreadCount** プロトコル・リスナー・フレームワークが使用することのできるスレッドの合計数。これは、プロトコル・リスナー・フレームワークが同時に処理できる要求の数です。
- **RequestPoolSize** プロトコル・リスナー・フレームワークに登録できる要求の最大数。この最大数を超える要求を受け取ると、新規要求は登録されなくなります。

これら 2 種類のコネクター固有のプロパティーは、プロトコル・リスナーが際限なく Web サービス・イベントを発生させてコネクターをふさいでしまわないように、メモリー割り振りを制御します。この割り振りアルゴリズムは、「コネクターは、WorkerThreadCount + RequestPoolSize に等しいイベントの合計数を常に受信できる」というものです。ここでは、*WorkerThreadCount* 数の要求を並行して処理できます。

追加のプロトコル・リスナーをプロトコル・リスナー・フレームワークに付け加えることができます。詳しくは、119 ページの『複数のプロトコル・リスナーの作成』および 98 ページの『コネクター固有の構成プロパティー』を参照してください。

SOAP/HTTP および SOAP/HTTPS プロトコル・リスナー処理

SOAP/HTTP(S) プロトコル・リスナーは、Web サービス・クライアントの HTTP(S) 要求について継続的に listen するスレッドから構成されています。リスナー・スレッドは、コネクター固有の構成 (リスナー) プロパティー Host および Port で指定されているホストとポートをバインドします。別の構成プロパティー (RequestWaitTimeout) は、コネクターがシャットダウンされたかどうか確認するまでリスナーが要求を待つインターバルを定義します。

図 24 は、同期操作のための SOAP/HTTP プロトコル・リスナー処理を表しています。

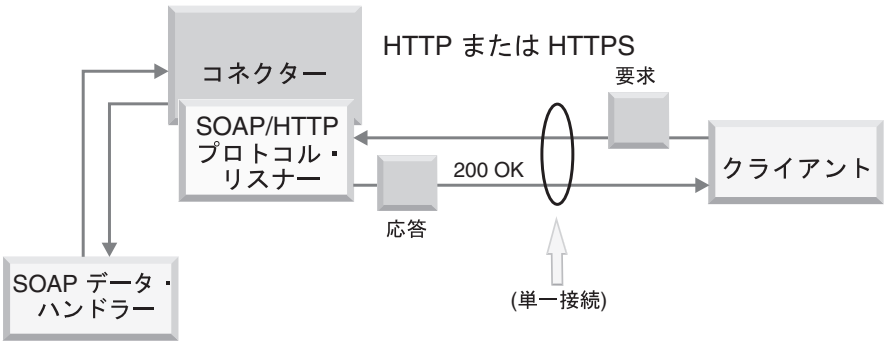


図 24. SOAP/HTTP プロトコル・リスナー: 同期イベント処理

図 25 は、非同期操作のための SOAP/HTTP プロトコル・リスナー処理を表しています。

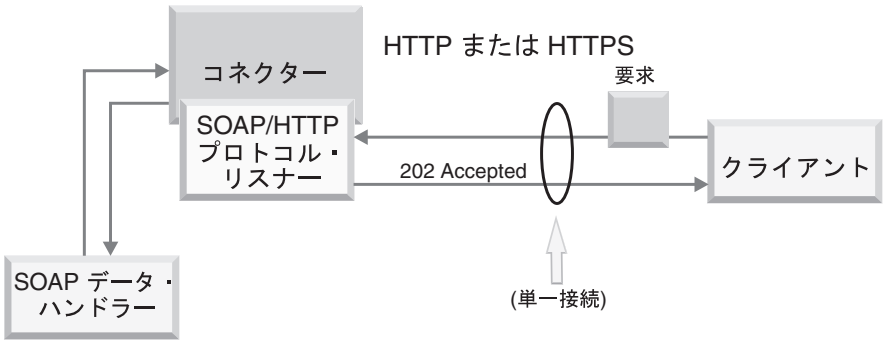


図 25. SOAP/HTTP プロトコル・リスナー: 非同期イベント処理

Web サービス・クライアントは、SOAP/HTTP または SOAP/HTTPS 要求を開始するときに、SOAP/HTTP または SOAP/HTTPS リスナーの URL に SOAP 要求メッセージを通知します。プロトコル・リスナー URL を呼び出すためには、クライアントは HTTP POST メソッドを使用する必要があります。

HTTP(S) 要求が到着すると、リスナーは、要求をプロトコル・リスナー・フレームワークに登録し、リソースが使用可能になったときにイベントを処理するようにスケジュールします。リスナーは、次に、要求からプロトコル・ヘッダーとペイロードを抽出します。

73 ページの表 28 は、インバウンド・メッセージの Charset、MimeType、ContentType、および Content-Type ヘッダーを決定する場合にリスナーが使用する規則の優先順位を要約したものです。

表 28. インバウンド・メッセージに対する SOAP/HTTP(S) プロトコル・リスナーの処理規則

優先順位	Charset	MimeType	ContentType	Content-Type ヘッダー
1	着信 HTTP メッセージの Content-Type ヘッダー値から得られる Charset パラメーター値	このリスナーの URLsConfiguration コネクター・プロパティー値	Content-Type ヘッダー値から得られる着信 HTTP メッセージ・タイプ/サブタイプ値	着信 HTTP メッセージの Content-Type ヘッダー
2	このリスナーの URLsConfiguration プロパティー値	SOAPDHMimeType コネクター・プロパティー値		
3	要求メッセージ ContentType のタイプが、任意のサブタイプ (例えば、`text/xml`、`text/plain` など) を持つ `text` の場合は、ISO-8859-1 にデフォルト設定されます。それ以外の場合は、charset を使用しません。	ContentType (デフォルト)		

表 28 に示したように、以下を決定します。

- プロトコル・リスナーは、次の規則に従って、インバウンド・メッセージの Charset を決定します。
  1. プロトコル・リスナーは、HTTP メッセージ Content-Type ヘッダー値の charset パラメーターから Charset の抽出を試みます。
  2. Charset 値が、Content-Type ヘッダーから取得されない場合、プロトコル・リスナーは、そのリスナー用に URLsConfiguration プロパティー値を読み取ろうとします。
  3. Charset 値が、上記のステップで説明されている方法を使用しても取得されず、また、メッセージ ContentType のタイプが、任意のサブタイプ (例えば、`text/xml`、`text/plain` など) を持つ `text` の場合、リスナーは、デフォルト Charset 値の ISO-8859-1 を使用します。それ以外の場合は、Charset 値を使用しません。
- リスナーは、次の規則に従って、応答メッセージの MimeType を決定します。
  1. ユーザーが、着信要求メッセージの使用する URL に対する TransformationRules の構成を終了し、要求 ContentType が TransformationRule の ContentType に一致する場合、リスナーは、TransformationRule を使用して、要求メッセージを SOAP 要求ビジネス・オブジェクトに変換するために MimeType 抽出します。リスナーは、要求された URL に対して、URLsConfiguration プロパティーの中で、ContentType 値 (例えば、`text/xml`) に基づき、正確に一致する TransformationRule を検索しようとします。
  2. 検索が失敗すると、リスナーは、要求 URL (例えば、*/*) のもとで複数の ContentType に適用される TransformationRule を検索しようとします。

3. MimeType に一致する TransformationRule が存在しない場合、リスナーは、SOAPDHMimeType コネクター構成プロパティーを、MimeType 値として使用します。
4. 上記のステップがすべて失敗し、MimeType が決定できない場合は、SOAP データ・ハンドラーを起動し、要求メッセージを、SOAP 要求ビジネス・オブジェクトに変換するために、ContentType の値が MimeType として使用されます。

- リスナーは、着信 HTTP メッセージ Content-Type ヘッダーからタイプ/サブタイプを抽出して、ContentType を決定します。
- リスナーは、着信 HTTP メッセージ Content-Type ヘッダーのそれから、Content-Type ヘッダーを決定します。

コラボレーションを非同期に呼び出す場合、リスナーは、要求ビジネス・オブジェクトを統合ブローカーにデリバリーし、HTTP 状況コード `202 Accepted` で Web サービス・クライアントに応答します。これでリスナー処理が完結します。

同期呼び出しの場合は、リスナーは同期させてコラボレーションを呼び出します。コラボレーションは SOAP 応答ビジネス・オブジェクトを使用して応答します。

表 29 は、応答メッセージの Charset、MimeType、ContentType、Content-Type ヘッダーを決定する際にリスナーが使用する規則の優先順位を要約したものです。

表 29. アウトバウンド同期応答メッセージに対する SOAP/HTTP(S) プロトコル・リスナーの処理規則

優先順位	Charset	MimeType	ContentType	Content-Type ヘッダー
1	プロトコル構成 MO の Content-Type ヘッダー	TLO の MimeType プロパティー	プロトコル構成 MO の Content-Type ヘッダー	プロトコル構成 MO の Content-Type ヘッダー
2	TLO の Charset プロパティー値	要求メッセージの MimeType。ただし要求と応答の ContentType が一致する場合のみ。	要求メッセージの ContentType	ContentType および Charset を使用して、Content-Type ヘッダーを構成します。
3	要求メッセージの Charset。ただし要求と応答の ContentType が一致する場合のみ。	SOAPDHMimeType コネクター・プロパティー値		
4	ContentType が `text/*` の場合、デフォルトは ISO-8859-1 です。それ以外の場合は、charset を使用しません。	MimeType として ContentType 値を使用します。		

表 29 に示したように、以下を決定します。

- リスナーは、次の規則に従って、応答メッセージの Charset を決定します。
  1. 応答ビジネス・オブジェクトのプロトコル構成 MO に Charset が指定されている場合は、その値を使用します。
  2. 応答ビジネス・オブジェクトのプロトコル構成 MO ヘッダーに Charset 値が指定されていない場合は、要求と応答のビジネス・オブジェクトが TLO の子であれば、リスナーは、TLO に Charset が指定されているかを検査します。

3. TLO に Charset が指定されていない場合、または応答ビジネス・オブジェクトが TLO でない場合に、応答と要求の ContentType が同じであれば、要求の Charset が応答に使用されます。
4. 上記のステップが失敗し、応答 Charset 値が決定できない場合に、メッセージ ContentType のタイプ部分が、任意のサブタイプ (例えば、`text/xml`、`text/plain` など) を持つ `text` であれば、リスナーは、デフォルト Charset 値の ISO-8859-1 を使用します。それ以外の場合は、Charset 値を使用しません。

- リスナーは、次の規則に従って、応答メッセージの MimeType を決定します。
  1. TLO の MimeType 属性を使用します。
  2. TLO MimeType 属性が欠落し、要求と応答の ContentType が一致する場合、リスナーは、応答メッセージに要求の MimeType を使用します。
  3. 上記のステップが失敗する場合、リスナーは、SOAPDHMimeType コネクター・プロパティーの値を使用します。
  4. 上記以外の場合、リスナーは、MimeType として、ContentType 値を使用します。
- リスナーは、次の規則に従って、応答メッセージの ContentType を決定します。
  1. Content-Type ヘッダーが、応答ビジネス・オブジェクトのプロトコル構成 MO に指定されている場合は、Content-Type ヘッダーのタイプ/サブタイプ部分が、ContentType として使用されます。
  2. Content-Type ヘッダーが、応答ビジネス・オブジェクトのプロトコル構成 MO に指定されていない場合、リスナーは、(Charset が応答メッセージに対して決定されている場合) 決定されている ContentType と Charset を使用して、Content-Type ヘッダーを構成します。

リスナーは HTTP プロトコル構成 MO を処理します。HTTP プロトコル構成 MO で渡されたヘッダー値を、この要求応答イベント環境に合った正しい値にすることは、コラボレーション側の責任となります。リスナーは、次の規則に従って、標準ヘッダーおよびカスタム・プロパティーを取り込みます。

1. リスナーは、特殊な属性 (ObjectEventId など) を無視するため、HTTP プロトコル構成 MO の各項目を調べます。
2. 空でない各ヘッダーが、発信メッセージに書き込まれ、追加処理 (例えば、Content-Type ヘッダーに対する処理) が行われる場合があります。
3. 上記の方法では、メッセージに標準でないヘッダーが設定される可能性があるにもかかわらず、リスナーは、メッセージが論理的または意味的に正しいかをチェックしないことに注意してください。
4. HTTP プロトコル構成 MO UserDefinedProperties 属性に 1 つ以上のカスタム・プロパティーがある場合、リスナーは、それらをエンティティー・ヘッダー・セクション (最後のヘッダー・セクション) に追加します。カスタム・プロパティーの詳細については、40 ページの『イベント処理のためのユーザー定義のプロパティー』を参照してください。

注: HTTP プロトコル構成 MO のヘッダー Connection、Trailer、Transfer-Encoding、 Content-Encoding、Content-Length、 Content-MD5、Content-Range の中からいずれを指定しても、ほとんどの場合、正しくない HTTP メッセージが作成されます。

リスナーは、次に、SOAP データ・ハンドラーを呼び出して、コラボレーションによって戻された応答ビジネス・オブジェクトを SOAP 応答メッセージに変換します。

リスナーは、応答メッセージを Web サービス・クライアントにデリバリーし、200 OK HTTP 状況コードを組み込みます。コラボレーションにより SOAP 障害ビジネス・オブジェクトが戻された場合は、障害メッセージに変換されます。この障害メッセージは、500 Internal Server Error HTTP コードと共に Web サービス・クライアントにデリバリーされます。

リスナーは、次に、接続を閉じ、イベントを処理したスレッドは使用可能になります。

SOAP/HTTP プロトコル・リスナーでサポートされない処理機能

SOAP/HTTP プロトコル・リスナーでは、以下のサポートをしていません。

- キャッシュ: プロトコル・リスナーは、HTTP 仕様 (RFC2616) で規定されているキャッシュ機能を実行しません。
- プロキシー: プロトコル・リスナーは、HTTP 仕様 (RFC2616) で規定されているプロキシー機能を実行しません。
- 持続接続: プロトコル・リスナーは、HTTP 仕様 (RFC2616) で規定されている持続接続をサポートしていません。その代わり、プロトコル・リスナーでは、各 HTTP 接続のスコープは単一のクライアント要求であると想定されています。したがって、サービス要求が完了すると、接続は閉じられます。プロトコル・リスナーは、複数のサービス呼び出しの間で接続を再利用することはありません。
- リダイレクト: プロトコル・リスナーはリダイレクトをサポートしていません。
- 大容量ファイル転送: プロトコル・リスナーは、大容量ファイルの転送には使用できません。その代わり、参照による大容量ファイルの受け渡しは可能です。
- 状態管理: プロトコル・リスナーは、RFC2965 で示されている HTTP 状態管理機構をサポートしていません。
- Cookies: プロトコル・リスナーは Cookies をサポートしていません。

セキュア・ソケットを使用した SOAP/HTTPS リスナー処理

SOAP/HTTPS プロトコル・リスナー処理は SOAP/HTTP プロトコル・リスナー処理のセクションで説明されているとおりですが、HTTPS ではセキュア・ソケットを使用するという点をが異なります。詳しくは、94 ページの『SSL』を参照してください。

SOAP/JMS プロトコル・リスナー処理

SOAP/JMS プロトコル・リスナーは、Web サービス・クライアントからの要求の JMS 宛先となる InputQueue で継続的に listen するスレッドから構成されています。RequestWaitTimeout コネクター構成プロパティーは、コネクターがシャットダウンされたかどうか確認するまでリスナーが要求を待つ時間を定義します。

図 26 は、同期操作のための SOAP/JMS プロトコル・リスナー処理を表しています。この図は JMS プロバイダー情報を示していません。

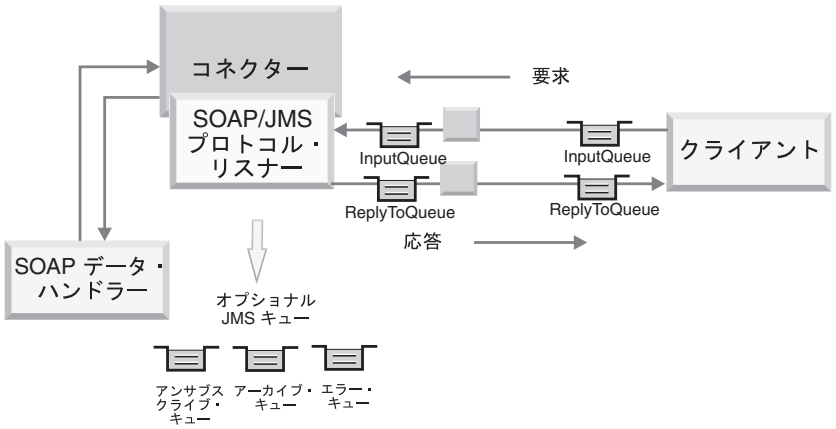


図 26. SOAP/JMS プロトコル・リスナー: 同期イベント処理

図 27 は、非同期操作のための SOAP/JMS プロトコル・リスナー処理を表しています。

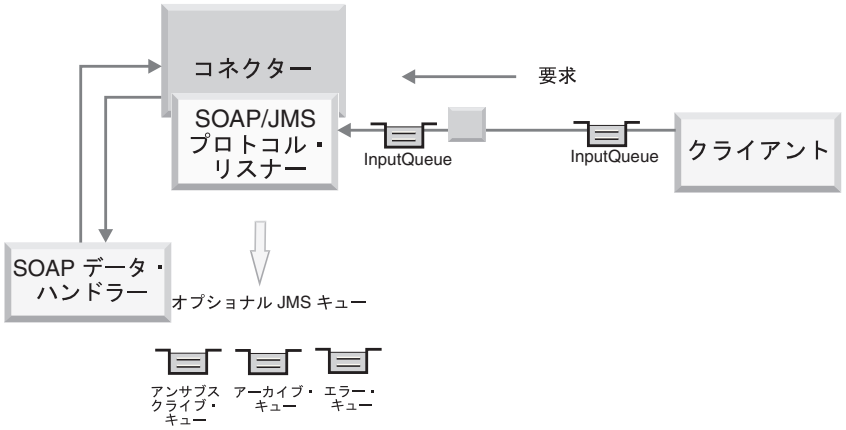


図 27. SOAP/JMS プロトコル・リスナー: 非同期イベント処理

注: LookupQueueUsingJNDI 構成プロパティーが `true` に設定されている場合、SOAP/JMS プロトコル・リスナーは、JNDI を使用してキューを検索します。JNDI プロパティーは、コネクター・プロパティーに指定されます。詳しくは、92 ページの『コネクターおよび JMS』および 98 ページの『コネクター固有の構成プロパティー』の JNDI 関連のプロパティーを参照してください。

Web サービス・クライアントは、SOAP/JMS 要求を開始するときに、SOAP/JMS リスナーが listen している InputQueue に SOAP 要求メッセージを送信します。InputQueue から SOAP 要求メッセージを受け取ると、SOAP/JMS プロトコル・リスナーは、プロトコル・リスナー・フレームワークにその要求を登録します。リソースが使用可能である限り、プロトコル・リスナー・フレームワークはその要求をスケジュールします。

注: コネクター構成プロパティー InDoubtEvents が `Reprocess` に設定されている場合、プロトコル・リスナー・フレームワークは、InProgressQueue の JMS メッセージをスケジュールしてから InputQueue のメッセージをスケジュールします。

次のリスナーは、このメッセージ (本文と必要な JMS ヘッダー (JMSCorrelationID、 JMSMessageID、 JMSPriority、 JMSExpiration、 JMSDeliveryMode、 JMSReplyTo、 JMSTimeStamp、 JMSType)) を InProgressQueue にディスパッチします。次に、プロトコル・リスナー・フレームワークはイベントを登録します。

リスナーは、次に、InProgressQueue から JMS メッセージを読み取り、メッセージの本文および以下のヘッダーを抽出します。

- JMSDestination
- JMSRedelivered
- JMSCorrelationID
- JMSMessageID
- JMSPriority
- JMSExpiration
- JMSDeliveryMode
- JMSReplyTo
- JMSTimeStamp
- JMSType
- JMS メッセージ・ペイロード・タイプ (ヘッダーではなく、メッセージの情報)

JMS メッセージ・ペイロード・タイプ: リスナーは、着信メッセージのペイロード・タイプを判別し、この情報を JMS プロトコル構成 MO の MessageType 属性に格納します。ペイロードは、TextMessage または BytesMessage です。TextMessage フォーマットの場合、リスナーは、ストリングとして抽出された Web サービス要求メッセージを使用して、ストリング API 経由でデータ・ハンドラーを呼び出します。BytesMessage の場合、リスナーは、バイト配列として抽出された Web サービス要求メッセージを使用して、バイト・データ・ハンドラー API 経由でデータ・ハンドラーを呼び出します。

リスナーは、SOAPDHMimeType コネクター構成プロパティーを使用して、SOAP データ・ハンドラーを呼び出し、要求メッセージを SOAP 要求ビジネス・オブジェクトに変換します。変換中にエラーが発生し、JMSReplyTo JMS ヘッダーが指定されていると、リスナーは、SOAP 障害メッセージで応答し、faultcode は `Client` に設定され、faultstring は `Cannot Parse` に設定されます。障害メッセージには、これ以外の詳細はありません。

リスナーは、データ・ハンドラーによって戻される SOAP 要求ビジネス・オブジェクトのオブジェクト・レベル `cw_mo_jms` ASI を使用して、プロトコル構成 MO を判別します。イベント処理の場合、ASI およびプロトコル構成 MO はいずれもオプションとなるので注意してください。リスナーは、プロトコル構成 MO を検出すると、事前に抽出済みの JMS メッセージ・ヘッダーをそれに取り込みます。表 43 は、プロトコル構成 MO 属性と JMS メッセージ・ヘッダー間のマッピングを示しています。

表 30. JMS ヘッダーとプロトコル構成 MO 属性のマッピング

プロトコル構成 MO 属性	JMS ヘッダー名	説明
CorrelationID	JMSCorrelationID	要求メッセージから得られる JMSCorrelationID ヘッダー
MessageId	JMSMessageId	要求メッセージから得られる JMSMessageID ヘッダー
Priority	JMSPriority	要求メッセージから得られる JMSPriority ヘッダー
Expiration	JMSExpiration	要求メッセージから得られる JMSExpiration ヘッダー
DeliveryMode	JMSDeliveryMode	要求メッセージから得られる JMSDeliveryMode ヘッダー
ReplyTo	JMSReplyTo	要求メッセージから得られる JMSReplyTo ヘッダー。JMS API はこのヘッダーを JMSDestination として戻しますが、SOAP/JMS プロトコル・リスナーはキュー名を戻します。
Timestamp	JMSTimestamp	要求メッセージから得られる JMSTimestamp ヘッダー
Redelivered	JMSRedelivered	要求メッセージから得られる JMSRedelivered ヘッダー
Type	JMSType	要求メッセージから得られる JMSType ヘッダー
Destination	JMSDestination	要求メッセージから得られる JMSDestination ヘッダー
MessageType	na	要求メッセージ・ペイロードのタイプ。この属性の値は、TextMessage ペイロードの場合は Text であり、BytesMessage ペイロードの場合は Bytes です。

SOAP/JMS プロトコル構成 MO UserDefinedProperties 属性に 1 つ以上のカスタム・プロパティーがある場合、リスナーは、メッセージからカスタム・プロパティーの値を抽出し、UserDefinedProperties ビジネス・オブジェクトにデータを設定しようとします。 カスタム・プロパティーの詳細については、 40 ページの『イベント処理のためのユーザー定義のプロパティー』を参照してください。

TLO (非 TLO SOAP 要求ビジネス・オブジェクトの場合) が統合ブローカーによってサブスクライブされなければ、リスナーはエラーをログに記録します。要求メッセージに JMSReplyTo ヘッダーが指定されている場合は、リスナーにより SOAP 障害メッセージが作成され、JMSReplyTo キューに配置されます。faultcode は `Client` に設定され、faultString は `Not subscribed to this message` に設定されます。障害メッセージにこれ以外の詳細はありません。また、そのように構成されている場合、リスナーは、JMS ヘッダーを含めて元の JMS 要求メッセージを UnsubscribedQueue に保存します。

コラボレーションを非同期に呼び出す場合、リスナーは、要求ビジネス・オブジェクトを統合ブローカーにデリバリーします。次に、リスナーはそのメッセージを InProgressQueue から除去します。また、そのように構成されている場合、リスナーは、JMS ヘッダーを含めて元の JMS 要求メッセージを ArchiveQueue に保存します。

非同期処理中にエラーが発生し、JMSReplyTo が指定されていると、リスナーは、障害メッセージで応答します。メッセージの faultcode は `Server` に設定され、faultstring は `Internal Error` に設定されます。また、そのように構成されている場合、リスナーは、JMS ヘッダーを含めて元の JMS 要求メッセージを ErrorQueue に保存します。

同期呼び出しの場合は、リスナーは同期させてコラボレーションを呼び出します。コラボレーションは SOAP 応答ビジネス・オブジェクトを使用して応答します。リスナーは、SOAP データ・ハンドラーを呼び出して、コラボレーションによって戻された応答ビジネス・オブジェクトを SOAP/JMS 応答メッセージに変換します。応答ペイロードのタイプは、SOAP 応答ビジネス・オブジェクトの JMS プロトコル構成 MO の MessageType 属性の値に依存します。MessageType が Text の場合、リスナーは、ストリング・データ・ハンドラー API 経由で SOAP 応答ビジネス・オブジェクトをストリングに変換します。MessageType が Bytes の場合、リスナーは、バイト・データ・ハンドラー API 経由で SOAP 応答ビジネス・オブジェクトをバイト配列に変換します。(デフォルトのメッセージ・ペイロード・タイプは、対応する同期要求のタイプです。) リスナーは、応答メッセージを ReplyTo キュー (これは、元の要求メッセージの JMSReplyTo ヘッダーで提供されます) にデリバリーします。その後、リスナーは、データ・ハンドラーによって戻された応答メッセージを (あらかじめ判別してある MessageType に応じて) TextMessageまたは BytesMessage に設定し、表 31 に示すヘッダーを設定します。

表 31. SOAP/JMS プロトコル・リスナーによって応答メッセージに設定されるヘッダー値

JMS ヘッダー名	**値**
JMSCorrelationId	要求メッセージの JMSMessageId
JMSDeliveryMode	要求メッセージの JMSDeliveryMode
JMSPriority	要求メッセージの JMSPriority
JMSExpiration	要求メッセージの JMSExpiration
JMSRedelivered	要求メッセージの JMSRedelivered
JMSReplyTo	要求メッセージの JMSReplyTo
JMSTimestamp	要求メッセージの JMSTimestamp
JMSType	要求メッセージの JMSType

リスナーは、JMS カスタム・プロパティーが応答または障害ビジネス・オブジェクトの JMS プロトコル構成 MO UserDefinedProperties 属性に存在する場合に、その JMS カスタム・プロパティーを応答メッセージに設定します。

そのように構成されている場合、リスナーは、元の JMS メッセージ (Web サービス・クライアントからの要求) をそのヘッダーも含めて、InProgressQueue から ArchiveQueue に移動します。

エラーが発生し、JMSReplyTo が指定されていると、リスナーは、障害メッセージで応答します。また、そのように構成されている場合は、元の JMS 要求メッセージを ErrorQueue に保存します。

イベントの永続性とデリバリー

イベントの永続性は、プロトコルによって決まります。

- **SOAP/HTTP プロトコル・リスナー** 永続性がなく、デリバリーは保証されません
- **SOAP/HTTPS プロトコル・リスナー** 永続性がなく、デリバリーは保証されません
- **SOAP/JMS プロトコル・リスナー** JMS キュー・イベントには永続性があり、少なくとも 1 度のデリバリーは保証されます。JMS キューに関する詳細については、98 ページの『コネクター固有の構成プロパティー』を参照してください。

イベントの順序付け

コネクターは、任意の順序でイベントをデリバリーすることができます。

イベントのトリガー

イベント・トリガーのメカニズムは、プロトコル・リスナーの構成方法によって異なります。

- **SOAP/HTTP プロトコル・リスナー** HTTP 接続要求の場合、listen は ServerSocket を介して行われます。
- **SOAP/HTTPS プロトコル・リスナー** HTTPS 接続要求の場合、listen は ServerSocket 層を介して行われます。
- **SOAP/JMS プロトコル・リスナー** Web サービス要求を伝達する着信 JMS メッセージの場合、listen は入力キューを介して行われます。JMS に関する詳細については、98 ページの『コネクター固有の構成プロパティー』を参照してください。

注: コネクターは、Create (作成)、Update (更新)、Retrieve (検索) または Delete (削除) の区別を行いません。これらのイベントは、すべて同じ方法で扱われます。

イベントの検出

イベントの検出は、それぞれのプロトコル・リスナーによって行われます。イベント検出のメカニズムは、トランスポート、および各リスナーごとのコネクター固有プロパティーの構成方法に、完全に依存しています。これらのプロパティーの詳細については、98 ページの『コネクター固有の構成プロパティー』を参照してください。

イベント状況

イベント状況はプロトコル・リスナーによって管理され、トランスポート、およびリスナーの構成方法によって異なります。

- **SOAP/HTTP プロトコル・リスナー** HTTP は本来、非永続的で同期的なものです。したがって、イベント状況は維持されません。
- **SOAP/HTTPS プロトコル・リスナー** HTTPS は本来、非永続的で同期的なものです。したがって、イベント状況は維持されません。

- **SOAP/JMS プロトコル・リスナー** JMS は永続的なトランスポートです。イベント状況はキューを使用して維持されます。JMS キューに関する詳細については、98 ページの『コネクター固有の構成プロパティー』を参照してください。

イベントの検索

イベントの検索はプロトコル・リスナーによって管理され、トランスポート、およびリスナーの構成方法によって異なります。

- **SOAP/HTTP プロトコル・リスナー・**イベントの検索は、ソケットから HTTP 要求を取り出すことによって行われます。
- **SOAP/HTTPS プロトコル・リスナー・**イベントの検索は、ソケットから HTTP 要求を取り出すことによって行われます。
- **SOAP/JMS プロトコル・リスナー・**イベントの検索は、JMS API を使用して行われます。JMS プロトコル・リスナーは JMS 入力キューからイベントを検索し、それらのイベントを進行中のキューに移動します。JMS キューに関する詳細については、 98 ページの『コネクター固有の構成プロパティー』を参照してください。

イベントのアーカイブ

イベントのアーカイブはプロトコル・リスナーによって管理され、トランスポート、およびリスナーの構成方法によって異なります。

- **SOAP/HTTP プロトコル・リスナー・**トランスポートが非永続的であり、同期的であるため、アーカイブは行われません。
- **SOAP/HTTPS プロトコル・リスナー・**トランスポートが非永続的であり、同期的であるため、アーカイブは行われません。
- **SOAP/JMS プロトコル・リスナー・**アンサブスクライブされたイベント、正常なイベント、および失敗したイベントのキューを含む JMS キューにイベントをアーカイブするように、コネクターを構成することができます。JMS キューに関する詳細については、 98 ページの『コネクター固有の構成プロパティー』を参照してください。

イベントのリカバリー

イベントのリカバリーはプロトコル・リスナーによって管理され、トランスポート、およびリスナーの構成方法によって異なります。

- **SOAP/HTTP プロトコル・リスナー・**トランスポートが非永続的であるため、イベント・リカバリーは行われません。
- **SOAP/HTTPS プロトコル・リスナー・**トランスポートが非永続的であるため、イベント・リカバリーは行われません。
- **SOAP/JMS プロトコル・リスナー** JMS は永続的なトランスポートです。イベントの処理中にコネクターがシャットダウンされた場合でも、イベントは、引き続き InProgressQueue で使用可能です。コネクターは、これらのイベントを始動時に処理するように構成することができるため、イベント・リカバリーが可能になります。InDoubtEvents コネクター構成プロパティーは、イベント・リカバリーのメカニズムを決定します。

注: SOAP/JMS リスナーは、統合ブローカーへ少なくとも 1 度はデリバリーすることを保証します。ただし、リスナーは、デリバリーが必ず 1 度だけであるということは保証できません。また、リスナーが受信したイベントは、順序を問わず統合ブローカーにデリバリーされる可能性があります。

JMS プロトコル・リスナーは、始動時にまず、InProgressQueue からイベントを検索しようとします。InDoubtEvents 構成プロパティーに割り当てる値によって次の動作が決まります。リカバリーのシナリオを、以下の表に示します。JMS キューに関する詳細については、 98 ページの『コネクター固有の構成プロパティー』を参照してください。

表 32. SOAP/JMS プロトコル・リスナーによって応答メッセージに設定されるヘッダー値

InDoubtEvents 値	**イベント・リカバリー処理**
FailOnStartup	イベントを InProgressQueue で検出すると、リスナーは、致命的エラーをログに記録し、直ちにシャットダウンします。
Reprocess	イベントを InProgressQueue で検出すると、リスナーはまず最初にこれらのイベントを処理します。リスナーは、InProgressQueue で検出されたメッセージ数をトレースできます。
Ignore	InProgressQueue 内のイベントは無視されます。リスナーは、InProgressQueue で検出されたイベントをトレース可能で、これらのイベントを無視します。
LogError	イベントを InProgressQueue で検出すると、リスナーは、エラーをログに記録し、処理を続行します。

要求処理

コネクターの要求処理機能を使用して、コラボレーションから Web サービスを呼び出すことができます。開発タスクでは、WSDL ODA を使用して Web サービス・トップレベル・オブジェクト (TLO) を生成し、それを配置するようにコラボレーションを構成します。詳しくは、 165 ページの『第 6 章 要求処理のためのコラボレーションの有効化』を参照してください。コネクターおよびその要求処理コンポーネント (プロトコル・ハンドラー・フレームワークおよびプロトコル・ハンドラー) も構成する必要があります。

コネクターは、実行時にビジネス・オブジェクトの形でコラボレーションから要求を受け取ります。ビジネス・オブジェクト (SOAP 要求、およびオプションで SOAP 応答および SOAP 障害ビジネス・オブジェクト) は、WSDL ODA によって生成される TLO に格納され、Web サービスを使用するように構成されているコラボレーションによって発行されます。TLO およびその子ビジネス・オブジェクトには、処理モード (同期または非同期) を指定する属性および ASI、データ・ハンドラー MIME タイプ、使用するプロトコル・ハンドラーの種類、およびターゲット Web サービスのアドレスが入っています。プロトコル・ハンドラーは、この情報を使用して、SOAP データ・ハンドラーのインスタンスを呼び出し、要求ビジネス・オブジェクトを SOAP 要求メッセージに変換し、ターゲット Web サービスを呼び出します。同期モードの場合、プロトコル・ハンドラーは、データ・ハンドラーを再度呼び出し、応答メッセージを SOAP 応答ビジネス・オブジェクトに変換して、これをコラボレーションに返送します。

コネクターは、SOAP 要求メッセージに対する応答として、リモート側の取引先から次のいずれかを受け取ることができます。

- データを含んでいる SOAP 応答メッセージ
- 障害情報を含んでいる SOAP 応答メッセージ

プロトコル・ハンドラーは、要求処理において重要な役割を果たしています。

プロトコル・ハンドラー

コラボレーションは、HTTP、HTTPS、または JMS トランスポートを介して Web サービスを呼び出すことができます。このコネクターには次の 2 つのプロトコル・ハンドラーがあり、それぞれに対応するチャネルがあります。

- SOAP/HTTP および SOAP/HTTPS ウェブ・サービスを呼び出すための SOAP/HTTP-HTTPS プロトコル・ハンドラー
- SOAP/JMS ウェブ・サービスを呼び出すための SOAP/JMS プロトコル・ハンドラー

プロトコル・ハンドラー・フレームワークはプロトコル・ハンドラーを管理し、起動時にプロトコル・ハンドラーをロードします。コネクターが要求ビジネス・オブジェクトを受け取ると、要求スレッド (それぞれのコラボレーション要求は、独自のスレッドで送られてきます) は、プロトコル・ハンドラー・フレームワークを呼び出して、要求を処理します。

プロトコル・ハンドラー・フレームワークは、TLOs Handler 属性 ASI を読み取り、使用するプロトコル・ハンドラーを決定します。一連の規則 (85 ページの『SOAP/HTTP-HTTPS プロトコル・ハンドラー処理』および 89 ページの『SOAP/JMS プロトコル・ハンドラー処理』を参照) を適用すると、プロトコル・ハンドラーはデータ・ハンドラーを呼び出して、要求ビジネス・オブジェクトを SOAP 要求メッセージに変換します。プロトコル・ハンドラーは、要求メッセージをトランスポート (HTTP(S) または JMS) メッセージにパッケージします。要求ビジネス・オブジェクトで SOAPAction ASI を検出すると、プロトコル・ハンドラーは、これを要求メッセージのヘッダーに追加します。

次に、プロトコル・ハンドラーは、要求ビジネス・オブジェクトのプロトコル構成 MO の Destination 属性を読み取り、ターゲット・アドレスを判別します。プロトコル・ハンドラーは、次に、要求メッセージを使用してターゲット Web サービスを呼び出します。

プロトコル・ハンドラーは、`ws_mode` TLO ASI を読み取り、処理モードが同期または非同期のいずれであるかを判別します。この ASI が `asynch` に設定されていると、プロトコル・ハンドラー処理は完了します。このように設定されていない場合、プロトコル・ハンドラーは応答メッセージを待ちます。応答メッセージが到着すると、プロトコル・ハンドラーは、プロトコル・ヘッダーとペイロードを抽出します。次にデータ・ハンドラー (MimeType TLO 属性によって指示される) を呼び出して、メッセージを応答または障害ビジネス・オブジェクトに変換します。プロトコル・ハンドラーは、プロトコル構成 MO を再度使用して、プロトコル・ヘッダーをビジネス・オブジェクトに設定します。プロトコル・ハンドラーは、次に、応答または障害ビジネス・オブジェクトをコラボレーションに戻します。

コネクターの構成によっては、1 つまたは複数のプロトコル・ハンドラーがコネクターにプラグされている場合があります。コネクター固有のプロパティーを指定することにより、プロトコル・ハンドラーを構成することができます。

SOAP/HTTP-HTTPS プロトコル・ハンドラー処理

SOAP/HTTP(S) プロトコル・ハンドラーは、このセクションに記載している点を除き、84 ページの『プロトコル・ハンドラー』で説明しているように動作します。図 28 は、同期操作のための SOAP/HTTP-HTTPS プロトコル・ハンドラーを表しています。

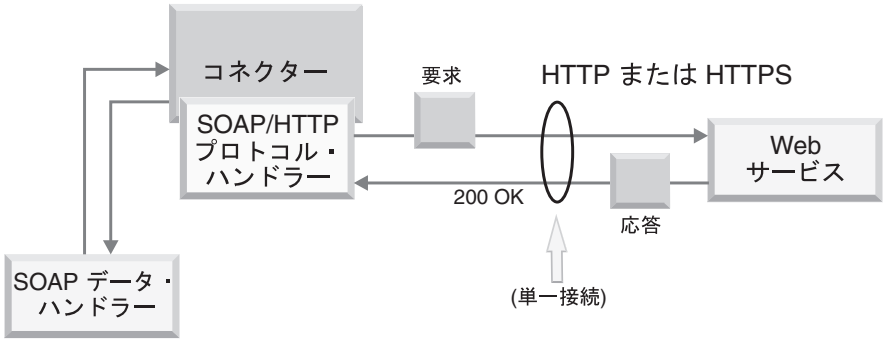


図 28. SOAP/HTTP-HTTPS プロトコル・ハンドラー: 同期要求処理

図 29 は、非同期要求処理のための SOAP/HTTP-HTTPS プロトコル・ハンドラーを表しています。

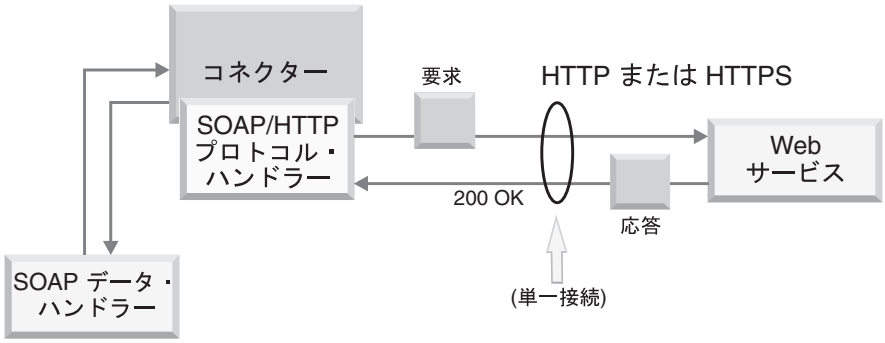


図 29. SOAP/HTTP-HTTPS プロトコル・ハンドラー: 非同期要求処理

注: このセクションでは、SOAP/HTTP プロトコル処理についてのみ説明します。

SOAP/HTTP-HTTPS プロトコル・ハンドラーは、SOAP 要求ビジネス・オブジェクトのオブジェクト・レベル ASI (`cw_mo_http`) を使用して、プロトコル構成 MO を決定します。SOAP/HTTP-HTTPS プロトコル・ハンドラーは、HTTP プロトコル構成 MO の Destination 属性を読み取り、ターゲット Web サービスの URL を決定します。URL が欠落している、あるいは不完全であれば、プロトコル・ハンドラーはサービス呼び出しで失敗します。HTTP プロトコル構成 MO およびその属性の詳細については、55 ページの『要求処理のための HTTP プロトコル構成 MO』を参照してください。

SOAP/HTTP-HTTPS プロトコル・ハンドラーは、SOAP データ・ハンドラーによって戻される SOAP 要求メッセージを使用して、Web サービスを呼び出します。HTTP プロキシー・コネクターの構成プロパティーが指定されている場合、SOAP/HTTP(S) プロトコル・ハンドラーは、それに応じた振る舞いをします。応答が戻されると、SOAP/HTTP(S) プロトコル・ハンドラーはそれを読み取ります。

表 33 は、発信要求メッセージの Charset、MimeType、ContentType、および ContentType ヘッダーを決定する際に SOAP/HTTP-HTTPS プロトコル・ハンドラーが使用する規則の優先順位を要約したものです。

表 33. アウトバウンド・メッセージに対する SOAP/HTTP-HTTPS プロトコル・ハンドラーの処理規則

優先順位	Charset	MimeType	ContentType	ContentType ヘッダー
1	プロトコル構成 MO の Content-Type ヘッダー	TLO 属性の MimeType プロパティー	プロトコル構成 MO の Content-Type ヘッダー	プロトコル構成 MO の Content-Type ヘッダー
2	TLO 属性の Charset プロパティー	ContentType (デフォルト)	`text/xml` (デフォルト)	ContentType および Charset を使用して、Content-Type ヘッダーを構成します。
3	ContentType が `text/*` の場合、デフォルトは ISO-8859-1 です。それ以外の場合は、charset を使用しません。			

表 33 に示したように、以下を決定します。

- SOAP/HTTP-HTTPS プロトコル・ハンドラーは、次の規則に従って、応答メッセージの Charset を決定します。
  1. 要求ビジネス・オブジェクトのプロトコル構成 MO ヘッダーに Charset 値が指定されている場合は、その値を使用します。
  2. 上記のステップで Charset が決定しなかった場合、プロトコル・ハンドラーは TLO 属性から Charset を抽出しようとします。
  3. 上記のステップに示した操作が失敗した場合は、次の表を使用して Charset を決定します。

表 34. Charset のデフォルト要求処理

ContentType	デフォルトの Charset
`text/*`	ISO-8859-1 詳しくは、RFC2616 を参照してください。
`application/*`	デフォルトなし
その他	デフォルトなし

  4. 上記のステップで Charset が決定した場合は、その Charset がデータ・ハンドラーに設定されます。
  5. データ・ハンドラーは、ストリームまたはバイト配列 API (要求の作成に必要なデータ構造に依存する) を使って呼び出されます。
- SOAP/HTTP-HTTPS プロトコル・ハンドラーは、次の規則に従って、要求の MimeType を決定します。

1. TLO MimeType 属性を使用します。
2. TLO の MimeType 属性がない場合、プロトコル・ハンドラーは ContentType を使用して MimeType を決定します。

- SOAP/HTTP-HTTPS プロトコル・ハンドラーは、次の規則に従って、要求メッセージの ContentType を決定します。
  1. Content-Type ヘッダーが、要求ビジネス・オブジェクトのプロトコル構成 MO に指定されている場合は、Content-Type ヘッダーのタイプ/サブタイプが、ContentType として使用されます。
  2. 上記以外の場合、ハンドラーはデフォルトの ContentType である `text/xml` を使用します。
- SOAP/HTTP-HTTPS プロトコル・ハンドラーは、次の規則に従って、要求メッセージの Content-Type ヘッダーを決定します。
  1. 要求ビジネス・オブジェクトのプロトコル構成 MO に Content-Type ヘッダーが指定されている場合は、その値を発信メッセージに設定します。
  2. Content-Type ヘッダーが、要求ビジネス・オブジェクトのプロトコル構成 MO に指定されていない場合、リスナーは、(Charset が要求メッセージに対して決定されている場合)、ContentType と Charset の両パラメーターを使用して、Content-Type ヘッダーを構成します。

表 35 は、応答メッセージの Charset、MimeType、ContentType、および ContentType ヘッダーを決定する際にハンドラーが使用する規則の優先順位を要約したものです。

表 35. インバウンド同期応答メッセージに対する SOAP/HTTP(S) プロトコル・ハンドラーの処理規則

優先順位	Charset	MimeType	ContentType	ContentType ヘッダー
1	着信 HTTP メッセージの Content-Type ヘッダー値から得られる Charset パラメーター値	要求ビジネス・オブジェクトのプロトコル構成 MO の MessageTransformationMap 子ビジネス・オブジェクト	Content-Type ヘッダー値から得られる着信 HTTP メッセージ・タイプ/サブタイプ値	着信 HTTP メッセージの Content-Type ヘッダー
2	要求ビジネス・オブジェクトのプロトコル構成 MO の MessageTransformationMap 子ビジネス・オブジェクト	要求メッセージの MimeType。ただし要求と応答の ContentType が一致する場合のみ。		
3	要求メッセージの Charset。ただし要求と応答の ContentType が一致する場合のみ。	TLO の MimeType プロパティー		
4	TLO の Charset プロパティー	ContentType (デフォルト)		

表 35. インバウンド同期応答メッセージに対する SOAP/HTTP(S) プロトコル・ハンドラーの処理規則 (続き)

5	Content-Type が `text/*` の場合、デフォルトは ISO-8859-1 です。それ以外の場合は、Charset を使用しません。			

表 35 に示したように、以下を決定します。

- プロトコル・ハンドラーは、次の規則に従って、同期応答メッセージの Charset を決定します。
  1. Charset パラメーターが着信応答メッセージの Content-Type ヘッダーに設定されている場合、プロトコル・ハンドラーは、データ・ハンドラーに設定されている Charset 値を使用します。
  2. 応答メッセージ・ヘッダーに Charset 値がない場合、プロトコル・ハンドラーは、TLO 要求プロトコル構成 MO の MessageTranformationMap から、コラボレーションが定義した Charset を読み取ろうとします。
  3. TLO に Charset 値が指定されていない場合、または TLO がない場合は、応答と要求の ContentType が同じであれば、要求の Charset が応答に使用されます。
  4. 上記のステップで Charset 値を決定できない場合、プロトコル・ハンドラーは TLO の Charset 属性を読み取ろうとします。
  5. Charset 値が、上記のステップで説明されている方法を使用しても取得されず、また、メッセージ ContentType のタイプが、任意のサブタイプ (例えば、`text/xml` 、`text/plain` など) を持つ `text` の場合は、ISO-8859-1 にデフォルト設定されます。それ以外の場合は、Charset 値を使用しません。
- プロトコル・ハンドラーは、次の規則に従って、同期応答メッセージの MimeType を決定します。
  1. プロトコル・ハンドラーはまず、TLO 要求プロトコル構成 MO の MessageTransformationMap から MimeType を抽出しようとします。特に、プロトコル・ハンドラーは、MTM の中で正確に一致する ContentType を検索して、MessageTransformationRule を抽出した後、その中の MimeType プロパティー値を使用しようとします。検索できない場合、プロトコル・ハンドラーは、複数の ContentType に適用される MessageTransformationRule (ContentType が */*) を検索します。
  2. MessageTransformationMap を使用しても MimeType が決定しない場合は、プロトコル・ハンドラーは、要求と応答の ContentType が一致する場合に限り、要求の MimeType を応答に使用します。
  3. 上記のステップを使用しても MimeType を抽出できない場合、プロトコル・ハンドラーは、プロトコル・ハンドラーで使用可能な場合に、TLO の MimeType 属性またはデフォルトの MimeType を使用します。
  4. 上記のすべてのステップが失敗した場合、プロトコル・ハンドラーは、ContentType を使用して MimeType を設定します。
- ハンドラーは、着信 HTTP メッセージ Content-Type ヘッダーからタイプ/サブタイプを抽出して、ContentType を決定します。

ハンドラーは HTTP プロトコル構成 MO を処理します。HTTP プロトコル構成 MO で渡されたヘッダー値を、この要求応答イベント環境に合った正しい値にすることは、コラボレーション側の責任となります。ハンドラーは、次の規則に従って、標準ヘッダーおよびカスタム・プロパティーを取り込みます。

1. ハンドラーは、特殊な属性 (ObjectEventId など) を無視するため、HTTP プロトコル構成 MO の各項目を調べます。
2. 空でない各ヘッダーが、発信メッセージに書き込まれ、追加処理 (例えば、Content-Type ヘッダーに対する処理) が行われる場合があります。
3. 上記の方法では、メッセージに標準でないヘッダーが設定される可能性があるにもかかわらず、ハンドラーは、メッセージの論理的または意味的な正しさを保証しないことに注意してください。
4. HTTP プロトコル構成 MO UserDefinedProperties 属性に 1 つ以上のカスタム・プロパティーがある場合、ハンドラーは、それらをエンティティー・ヘッダー・セクション (最後のヘッダー・セクション) に追加します。カスタム・プロパティーの詳細については、56 ページの『要求処理のためのユーザー定義のプロパティー』を参照してください。

注: HTTP プロトコル構成 MO のヘッダー Connection、Trailer、Transfer-Encoding、 Content-Encoding、Content-Length、Content-MD5、Content-Range の中からいずれを指定しても、ほとんどの場合、正しくない HTTP メッセージが作成されます。

SOAP/JMS プロトコル・ハンドラー処理

SOAP/JMS プロトコル・ハンドラーは、このセクションに記載している点を除き、84 ページの『プロトコル・ハンドラー』で説明しているように動作します。

注: LookupQueueUsingJNDI 構成プロパティーが `true` に設定されている場合、SOAP/JMS プロトコル・ハンドラーは、JNDI を使用して宛先キューを検索します。JNDI プロパティーは、コネクター・プロパティーに指定されます。詳しくは、92 ページの『コネクターおよび JMS』および 98 ページの『コネクター固有の構成プロパティー』の JNDI 関連のプロパティーを参照してください。

SOAP/JMS プロトコル・ハンドラーは、SOAP データ・ハンドラーによって戻された Web サービス要求メッセージの本文と表 36 に示すとおりに設定されている JMS ヘッダーを使用して、JMS トランスポート・メッセージを作成します。応答ペイロードのタイプは、SOAP 要求ビジネス・オブジェクトの JMS プロトコル構成 MO の MessageType 属性の値に依存します。MessageType が Text の場合、ハンドラーは、ストリング・データ・ハンドラー API 経由で SOAP 要求ビジネス・オブジェクトをストリングに変換します。MessageType が Bytes の場合、ハンドラーは、バイト・データ・ハンドラー API 経由で SOAP 要求ビジネス・オブジェクトをバイト配列に変換します。(デフォルトのメッセージ・ペイロード・タイプは TextMessage です。)

表 36. SOAP/JMS プロトコル・ハンドラーによって要求メッセージに設定されるヘッダー値

JMS ヘッダー名	**SOAP/JMS プロトコル構成 MO に設定されていない場合のデフォルト値**
JMSPriority	4
JMSExpiration	0

表 36. SOAP/JMS プロトコル・ハンドラーによって要求メッセージに設定されるヘッダー値 (続き)

JMSDeliveryMode	PERSISTENT
JMSReply	
JMSCorrelationId	
JMSRedelivered	
JMSTimestamp	
JMSType	

ターゲット Web サービスが非同期に呼び出される場合、JMSReplyTo ヘッダーは設定されません。そうでない場合 (同期処理の場合)、SOAP/JMS プロトコル・ハンドラーにより JMSReplyTo ヘッダーが設定されます。SOAP/JMS プロトコル・ハンドラーは、ReplyToQueue 構成プロパティーを使用して、JMSDestination (ターゲット Web サービスからの応答または障害が戻される宛先) を取得し、それを JMS トランスポート・メッセージの JMSReplyTo ヘッダーに設定します。

図 30 は、同期要求操作のための SOAP/JMS プロトコル・ハンドラー処理を表しています。

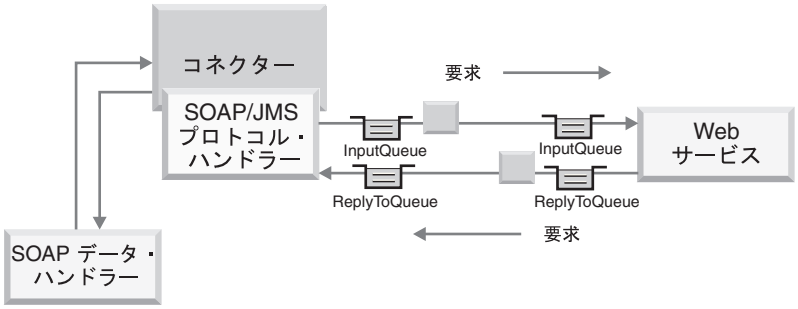


図 30. SOAP/JMS プロトコル・ハンドラー: 同期要求処理

図 31 は、非同期要求操作のための SOAP/JMS プロトコル・ハンドラー処理を表しています。

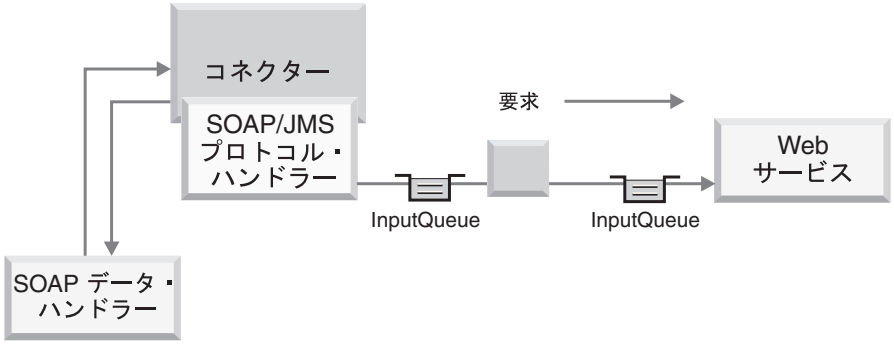


図 31. SOAP/JMS プロトコル・ハンドラー: 非同期要求処理

SOAP/JMS プロトコル・ハンドラーは、SOAP 要求ビジネス・オブジェクトのオブジェクト・レベル ASI (cw_mo_jms) を使用して、プロトコル構成 MO を決定します。プロトコル構成 MO の Destination 属性には、ターゲット Web サービスのキュー名を指定されます。JNDI が使用可能である場合、SOAP/JMS プロトコル・ハンドラーは、JNDI オブジェクトを検索して、SOAP 要求メッセージの JMSDestination を取得します。そうでない場合は、SOAP プロトコル構成 MO の Destination 属性を使用します。

ResponseWaitTimeout プロパティーに指定されているインターバル中に応答が到着しない場合、SOAP/JMS プロトコル・ハンドラーはコラボレーション要求の処理に失敗します。SOAP 応答 (または障害) メッセージが到着すると、SOAP/JMS プロトコル・ハンドラーは、SOAP データ・ハンドラーにより変換するために JMS ヘッダーおよびペイロードを抽出します。ハンドラーは、着信メッセージのペイロード・タイプを判別し、この情報を JMS プロトコル構成 MO の MessageType 属性に格納します。ペイロードは、TextMessage または BytesMessage です。TextMessage フォーマットの場合、ハンドラーは、ストリングとして抽出された Web サービス応答メッセージを使用して、ストリング API 経由でデータ・ハンドラーを呼び出します。BytesMessage の場合、ハンドラーは、バイト配列として抽出された Web サービス応答メッセージを使用して、バイト・データ・ハンドラー API 経由でデータ・ハンドラーを呼び出します。次に、SOAP/JMS プロトコル・ハンドラーは、応答 (または障害) ビジネス・オブジェクトでプロトコル構成 MO を使用して JMS ヘッダーをマップし、SOAP 応答 (または障害) ビジネス・オブジェクトを TLO に設定します。表 37 にこのマッピングを示しています。

表 37. 同期要求処理における応答のプロトコル構成 MO と JMS ヘッダー属性のマッピング

プロトコル構成 MO 属性	**JMS ヘッダー名**	**説明**
Destination	JMSDestination	応答メッセージから得られる JMSDestination ヘッダー
MessageId	JMSMessageId	応答メッセージから得られる JMSMessageId ヘッダー
Priority	JMSPriority	応答メッセージから得られる JMSPriority ヘッダー
Expiration	JMSExpiration	応答メッセージから得られる JMSExpiration ヘッダー
DeliveryMode	JMSDeliveryMode	応答メッセージから得られる JMSDeliveryMode ヘッダー
ReplyTo	JMSReplyTo	応答メッセージから得られる JMSReplyTo ヘッダー。JMS API はこのヘッダーを JMSDestination として戻しますが、SOAP/JMS プロトコル・リスナーはキュー名を戻します。
CorrelationId	JMSCorrelationId	応答メッセージから得られる JMSCorrelationId ヘッダー
Redelivered	JMSRedelivered	応答メッセージから得られる JMSRedelivered ヘッダー
TimeStamp	JMSTimeStamp	応答メッセージから得られる JMSTimeStamp ヘッダー

表 37. 同期要求処理における応答のプロトコル構成 MO と JMS ヘッダー属性のマッピング (続き)

Type	JMSType	応答メッセージから得られる JMSType ヘッダー
MessageType	n/a	応答メッセージ・ペイロードのタイプ。この属性の値は、TextMessage ペイロードの場合は Text であり、BytesMessage ペイロードの場合は Bytes です。

SOAP/JMS プロトコル・ハンドラーは、次に、TLO をコラボレーションに戻します。

コネクターおよび JMS

注: このセクションでは、JMS および JNDI、特に JMS の動作について十分に理解していることを前提としています。詳しくは JMS および JNDI ソースの資料を参照してください。

コネクターは、コラボレーションを SOAP/JMS Web サービスとして公開し、コラボレーションから SOAP/JMS Web サービスを呼び出すことができるようにします。Web サービス・コネクターで SOAP/JMS を使用するための要件は以下のとおりです。

1. JMS サービス・プロバイダーをインストールおよび構成済みである。
2. JNDI をインストールおよび構成済みである。
3. JMS プロバイダーが JMS API バージョン 1.0.2 をサポートしている。
4. 必要なすべての jar ファイルがコネクターのクラスパスにある。(必要なすべての jar ファイルを判別するには JMS プロバイダーの資料を参照してください。)
5. 必要なすべてのライブラリーがコネクターのパスにある。(必要なすべてのライブラリーを判別するには JMS プロバイダーの資料を参照してください。)

JNDI

SOAP/JMS の場合、コネクターは、JNDI を使用して JNDI コンテキストにより接続ファクトリーを検索します。コネクターは、初期化中にコネクター固有の JNDI プロパティーを読み取り JNDI に接続します。JNDI プロパティーを構成しなければ、SOAP/JMS は使用できません。コネクター固有の JNDI プロパティーとして、以下のプロパティーを指定することができます。

- JNDIProviderURL
- InitialContextFactory
- JNDIConnectionFactoryName
- CTX_ObjectFactories
- CTX_StateFactories
- CTX_URLPackagePrefixes
- CTX_DNS_URL

- CTX_Authoritative
- CTX_Batchsize
- CTX_Referral
- CTX_SecurityProtocol
- CTX_SecurityAuthentication
- CTX_SecurityPrincipal
- CTX_SecurityCredentials
- CTX_Language
- LookupQueuesUsingJNDI

これらのプロパティーを指定する方法については JNDI の資料を参照してください。コネクターで SOAP/JMS を使用するには、以下のコネクター固有の JNDI プロパティーが必要になります。

- **JNDIProviderURL** このプロパティーは、JNDI サービス・プロバイダーの URL に設定します。このプロパティーの値については、JNDI プロバイダーの資料を参照してください。
- **InitialContextFactory** このプロパティーは、JNDI 初期コンテキストを作成するファクトリー・クラスの完全修飾クラス名に設定します。このプロパティーの値については、JNDI プロバイダーの資料を参照してください。このクラスとその従属クラスがコネクターのクラスパスにあることを確認してください。
- **JNDIConnectionFactoryName** このプロパティーは、(JNDI コンテキストを使用して) 検索する、接続ファクトリーの JNDI 名に設定します。この名前は、JNDI を使用して検索できることを確認してください。

LookupQueuesUsingJNDI を `true` に設定した場合、コネクターで使用されるキューはすべて JNDI を使用して検索できることを確認してください。

SOAP/JMS Web サービスとしてのコラボレーションの公開

コラボレーションを SOAP/JMS Web サービスとして公開するには、SOAP/JMS プロトコル・リスナーを使用しなければなりません。SOAP/JMS プロトコル・リスナーを使用するには、JNDI コネクター・プロパティーを指定する必要があります。

JMS プロバイダー構成では、SOAP/JMS プロトコル・リスナーの要求を反映する必要があります。SOAP/JMS プロトコル・リスナーで必要なキューはすべて、JMS サービス・プロバイダーで定義されることを確認してください。キュー定義の作業はプロバイダーによって異なるので、JMS プロバイダーの資料で確認してください。SOAP/JMS プロトコル・リスナーには 6 つのキューを定義する必要があります。SOAP/JMS リスナー構成プロパティーにキュー名を設定する必要があり、JNDI ” LookupQueuesUsingJNDI を `true` に設定している場合は、SOAP/JMS リスナー構成プロパティーにキューの JNDI 名も指定する必要があります。

以下の SOAP/JMS リスナー構成プロパティーの値としてキュー名を指定する必要があります。

- InputQueue
- InProgressQueue
- ArchiveQueue

- UnsubscribedQueue
- ErrorQueue
- ReplyToQueue

InputQueue および InProgressQueue は必要なプロパティーです。これらのキューが正しく構成されていることを確認してください。

ArchiveQueue、UnsubscribedQueue および ErrorQueue はオプションのプロパティーです。これらのキューは、Web サービス要求を保存する場合に使用します。これらのプロパティーのいずれかを使用する場合は、対応する JMS キューが正しく構成されていることを確認してください。JMS プロバイダーでこれらのキューを定義する場合は、これらのキューの容量の指定は慎重に行う必要があります。

SOAP/JMS Web サービスを呼び出すコラボレーション

コラボレーションで SOAP/JMS Web サービスを呼び出せるようにするには、SOAP/JMS プロトコル・ハンドラーを使用します。SOAP/JMS プロトコル・ハンドラーを使用するには、JNDI コネクター・プロパティーを指定する必要があります。Web サービス・プロバイダーと連動させて、JMS および JNDI 要件を決定します。

SOAP/JMS Web サービスを呼び出すには、コネクターで、ターゲット Web サービスの入力キューに SOAP/JMS プロトコル構成 MO の Destination 属性の値を設定する必要があります。JNDI ” LookupQueuesUsingJNDI を `true` に設定している場合は、入力キューの JNDI 名を指定する必要があります。

要求/応答 Web サービスを呼び出す場合は、Web サービス・プロバイダーと連動させて、ReplyTo キューの要件を決定する必要があります。ReplyTo キューが定義されていることを確認してください。SOAP/JMS プロトコル・ハンドラーの ReplyToQueue 構成プロパティーに ReplyTo キューの名前を指定していることも確認してください。JNDI ” LookupQueuesUsingJNDI が `true` に設定されている場合、ReplyToQueue 構成プロパティーの値には、このキューの JNDI 名を指定する必要があります。

プロトコル・ハンドラーは、プロトコル・リスナーとは異なり、Web サービス・コネクターにプラグ可能ではないので注意しておくことが大切です。そのためコネクターは、それが呼び出すすべての要求/応答 Web サービスに同じ ReplyTo キューを使用することになります。

SSL

このセクションでは、コネクターの SSL 機能のインプリメント方法について説明します。背景情報については SSL の資料を参照してください。このセクションでは、SSL テクノロジーについて十分に理解していることを前提としています。

JSSE

コネクターは、コラボレーションを SOAP/HTTPS Web サービスとして公開し、コラボレーションから SOAP/HTTPS Web サービスを呼び出すことができるようにします。コネクターは、JSSE を使用して HTTPS および SSL をサポートします。

IBM JSSE はコネクターと共に出荷されます。この機能を使用可能にするには、java.security ファイルに次の項目があることを確認してください。このファイルは、コネクターと一緒にインストールされるファイルの 1 つです。

```
security.provider.5=com.ibm.jsse.IBMJSSEProvider
```

java.security は、コネクターのインストール先システムの $ProductDir¥lib¥security ディレクトリーにあります。コネクターは、JavaProtocolHandlerPackages コネクター・プロパティーの値を使用して、システム・プロパティー java.protocol.handler.pkgs を設定します。コネクターと共に出荷された IBM JSSE の場合、このプロパティーの値は com.ibm.net.ssl.internal.www.protocol に設定する必要があることに注意してください。JavaProtocolHandlerPackages 構成プロパティーでは、この値がデフォルトになります。ただし、ご使用のシステムで java.protocol.handler.pkgs システム・プロパティーが非空値の場合は、JavaProtocolHandlerPackages コネクター・プロパティーの値も設定されている場合にのみ、値が上書きされます。

コネクターは、初期化中に、JSSE でサポートされる無名の暗号スイートをすべて使用不可にします。

KeyStore および TrustStore

コネクターで SSL を使用するには、鍵ストアとトラストストアをセットアップする必要があります。鍵ストア、証明書、および鍵生成のセットアップ用ツールは提供されていません。これらの作業を完了するには、サード・パーティーのソフトウェア・ツールを使用する必要があります。

SSL プロパティー

コネクター固有の SSL プロパティーとして、以下のプロパティーを指定することができます。

- SSLVersion
- SSLDebug
- KeyStore
- KeyStoreAlias
- KeyStorePassword
- TrustStore
- TrustStorePassword

これらのプロパティーは、コネクター・インスタンスに適用されることに注目してください。コネクターにプラグインされるすべての SOAP/HTTPS プロトコル・リスナーと、コネクター・インスタンスごとの SOAP/HTTP-HTTPS プロトコル・ハンドラーにより、同じ SSL プロパティー値のセットが使用されます。HTTPS/SSL セットアップの詳細については、257 ページの『付録 E. HTTPS/SSL の構成』を参照してください。

コラボレーションを SOAP/HTTPS Web サービスとして公開

コラボレーションを SOAP/HTTPS Web サービスとして公開する際には、SOAP/HTTPS プロトコル・リスナーを使用します。SOAP/HTTPS プロトコル・リス

ナーを使用するには、コネクター固有の SSL プロパティーを指定する必要があります。これらのプロパティーに割り当てる値は、以下の SSL 要件を満たしている必要があります。

- **SSLVersion** 使用する SSLVersion が JSSE によってサポートされていることを確認してください。
- **KeyStore** SOAP/HTTPS プロトコル・リスナーは、SSL 通信のサーバーとして動作するので、鍵ストアを指定する必要があります。リスナーは、SSL ” KeyStore 構成プロパティーに指定されている鍵ストアを使用します。このプロパティーの値は、鍵ストア・ファイルの完全パスでなければなりません。鍵ストアにはコネクター用の鍵ペア (秘密鍵と公開鍵) があることを確認してください。秘密鍵の別名を、SSL ” KeyStoreAlias プロパティーで指定する必要があります。鍵ストアにアクセスする際に必要なパスワードは、SSL ” KeyStorePassword プロパティーで指定しなければなりません。鍵ストアにアクセスする際に必要なパスワードと秘密鍵 (鍵ストアにある) が同じであることも確認してください。最後に、コネクターのディジタル証明書を Web サービス・クライアントに配布して、コネクターの認証ができるようにする必要があります。
- **TrustStore** SOAP/HTTPS プロトコル・リスナーで Web サービス・クライアントの認証を行うようにする場合は、クライアント認証をアクティブしておく必要があります。このためには、SSL ” UseClientAuth プロパティーを `true` に設定します。以下も指定する必要があります。
  - トラストストアのロケーション (SSL ” TrustStore 構成プロパティーの値として)
  - トラストストアにアクセスする際に必要なパスワード (SSL ” TrustStorePassword プロパティーの値として)

  トラストストアには、Web サービス・クライアントのディジタル証明書が含まれていることを確認してください。Web サービス・クライアントで使用されるディジタル証明書は、自己署名するか、または CA から発行されます。トラストストアが CA のルート証明書を信頼すると、JSSE は、その CA によって発行されたすべてのディジタル証明書を認証することになるので注意してください。

HTTPS/SSL セットアップの詳細については、 257 ページの『付録 E. HTTPS/SSL の構成』を参照してください。

SOAP/HTTPS Web サービスを呼び出すコラボレーション

コラボレーションで SOAP/HTTPS Web サービスを呼び出せるようにするには、SOAP/HTTP-HTTPS プロトコル・ハンドラーを使用します。SOAP/HTTP-HTTPS プロトコル・ハンドラーで SSL を使用する場合は、コネクター固有の SSL プロパティーを指定する必要があります。これらのプロパティーに割り当てる値は、Web サービス・プロバイダーの以下の HTTPS/SSL 要件を満たしている必要があります。

- **SSLVersion** 使用する SSLVersion が Web サービス・プロバイダーおよび JSSE によってサポートされていることを確認してください。
- **TrustStore** SOAP/HTTP-HTTPS プロトコル・ハンドラーは SSL 通信ではクライアントとして動作するため、トラストストアをセットアップする必要があります。ハンドラーは、SSL -> Truststore 構成プロパティーに指定されているトラストストアを使用します。このプロパティーの値は、トラストストア・ファイルの完全パスでなければなりません。トラストストアにアクセスする際に必要なパス

ワードは、SSL -> TrustStorePassword プロパティーに指定しなければなりません。トラストストアには、Web サービス・プロバイダーのディジタル証明書が含まれていることを確認してください。Web サービス・プロバイダーで使用されるディジタル証明書は、自己署名するか、または CA から発行されます。トラストストアが CA のルート証明書を信頼すると、JSSE は、その CA によって発行されたすべてのディジタル証明書を認証することになるので注意してください。

- **KeyStore** Web サービス・プロバイダーでクライアント認証が必要な場合は、鍵ストアをセットアップする必要があります。SOAP/HTTP-HTTPS プロトコル・ハンドラーは、SSL ” KeyStore 構成プロパティーに指定されている鍵ストアを使用します。この値は、鍵ストア・ファイルの完全パスでなければなりません。鍵ストアにはコネクター用に構成された鍵ペア (秘密鍵と公開鍵) があることを確認してください。秘密鍵の別名を、SSL ” KeyStoreAlias プロパティーに指定する必要があります。鍵ストアにアクセスする際に必要なパスワードは、SSL ” KeyStorePassword プロパティーに指定しなければなりません。最後に、鍵ストアにアクセスする際に必要なパスワードと秘密鍵 (鍵ストアにある) が同じであることを確認してください。認証のためにコネクターのディジタル証明書を Web サービス・プロバイダーに配布する必要があります。

HTTPS/SSL セットアップの詳細については、 257 ページの『付録 E. HTTPS/SSL の構成』を参照してください。

コネクターの構成

Installer を使用してコネクター・ファイルをシステムにインストールした後で、標準およびアプリケーション固有のコネクター構成プロパティーを設定する必要があります。

構成プロパティーの設定

コネクターには、2 種類の構成プロパティーがあります。標準の構成プロパティーと、コネクター固有の構成プロパティーです。コネクターを稼働させる前に、System Manager (SM) 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。

標準構成プロパティー

標準構成プロパティーは、すべてのコネクターによって使用される情報を提供します。これらのプロパティーの説明については、 193 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。下の表には、この付録で説明する構成プロパティーに関する、このコネクターに固有の情報が示されています。

プロパティー	説明
CharacterEncoding	この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に対応していないため、このプロパティーの値は変更できません。現在サポートされているロケールについては、コネクターのリリース情報を参照してください。

このコネクターが統合ブローカーとしてサポートするのは InterChange Server Express のみであるため、関連する構成プロパティーは InterChange Server Express 用のプロパティーのみです。

少なくとも、以下の標準コネクター構成プロパティーを設定する必要があります。

- AgentTraceLevel
- ApplicationName
- ControllerTraceLevel
- DeliveryTransport

コネクター固有の構成プロパティー

コネクター固有の構成プロパティーは、実行時にコネクター・エージェントが必要とする情報を提供します。また、コネクター固有のプロパティーを使用することにより、エージェントの再コーディングおよび再ビルドを行わずに、コネクター・エージェント内の静的な情報または論理を変更することができます。

表 38 には、コネクター固有の構成プロパティーがリストされています。プロパティーの説明については、以下の各セクションを参照してください。プロパティーの中には、別のプロパティーを含んでいるものもあります。+ 文字は、プロパティー階層内の項目の位置を示しています。

注: コネクターと連動させずに、SOAP/JMS プロトコル・リスナーまたは SOAP/JMS プロトコル・ハンドラーをコネクターを使用したい場合は、SOAP/JMS 関連のコネクター固有のプロパティーを必ず削除するか、ブランクのままにしておいてください。

表 38. 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	必須
ConnectorType	*任意の有効なコネクター・タイプ*	`WebService`	はい
DataHandlerMetaObjectName	データ・ハンドラーのメタオブジェクト名	`MO_DataHandler_ Default`	はい
JavaProtocolHandlerPackages	*有効な Java プロトコル・ハンドラー・パッケージ*	`com.ibm.net.ssl. internal.www.protocol`	いいえ
ProtocolHandlerFramework	*これは階層プロパティーであり、値はありません。*	なし	いいえ
+ProtocolHandlers	*これは階層プロパティーであり、値はありません。*		いいえ
++SOAPHTTPHTTPSHandler	*これは階層プロパティーです。このプロパティーのサブプロパティーについては、* 101 ページの *『SOAPHTTPHTTPSHandler』* を参照してください。		はい
++SOAPJMSHandler	*これは階層プロパティーです。このプロパティーのサブプロパティーについては、* 102 ページの *『SOAPJMSHandler』* を参照してください。		
ProtocolListenerFramework	*これは階層プロパティーであり、値はありません。*		いいえ

表 38. 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	必須
+WorkerThreadCount	*使用可能なリスナー・スレッド数を指定する 1 以上の整数。*	10	いいえ
+RequestPoolSize	*リソース・プール・サイズを指定する WorkerThreadCount より大きい整数*	20	いいえ
+ProtocolListeners	*これは階層プロパティーであり、値はありません。*		
++Listener1	*固有の名前が指定されたプロトコル・リスナー*		はい
+++Protocol	soap/http、soap/https、soap/jms		はい
+++SOAPDHMimeType	*SOAP データ・ハンドラーの任意の有効な MIME タイプ*	xml/soap	
+++ListenerSpecific	*リスナーで固有または必要なプロパティー。* 104 ページの『ListenerSpecific』を参照してください。		
ProxyServer	*これは階層プロパティーであり、値はありません。*		いいえ
+HttpProxyHost	*HTTP プロキシー・サーバーのホスト名*		いいえ
+HttpProxyPort	*HTTP プロキシー・サーバーのポート番号*	80	いいえ
+HttpNonProxyHosts	*直接接続が必要な HTTP ホスト*		いいえ
+HttpsProxyHost	*HTTPS プロキシー・サーバーのホスト名*		いいえ
+HttpsProxyPort	*HTTPS プロキシー・サーバーのポート番号*	443	いいえ
+HttpsNonProxyHosts	*直接接続が必要な HTTPS ホスト*		いいえ
+SocksProxyHost	*Socks プロキシー・サーバーの名前*		いいえ
+SocksProxyPort	*Socks プロキシー・サーバーのポート*		いいえ
+HttpProxyUsername	*HTTP プロキシー・サーバー・ユーザー名*		いいえ
+HttpProxyPassword	*HTTP プロキシー・サーバー・パスワード*		いいえ
+HttpsProxyUsername	*HTTPS プロキシー・サーバー・ユーザー名*		いいえ
+HttpsProxyPassword	*HTTPS プロキシー・サーバー・パスワード*		いいえ
SSL	*これは階層プロパティーであり、値はありません。*		いいえ
+SSLVersion	SSL、SSLv2、SSLv3、TLS、TLSv1	SSL	いいえ
+SSLDebug	true、false	false	いいえ
+KeyStoreType	*任意の有効な鍵ストア・タイプ*	JKS	いいえ
+KeyStore	*KeyStore ファイルへのパス*		いいえ
+KeyStorePassword	*KeyStore における秘密鍵のパスワード*		いいえ
+KeyStoreAlias	*KeyStore における鍵ペアの別名*		いいえ
+TrustStore	*TrustStore ファイルへのパス*		いいえ
+TrustStorePassword	*TrustStore のパスワード*		いいえ
+UseClientAuth	true、false	false	いいえ

表 38. 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	必須
WSCollaborations	*これは、WSDL 構成ウィザードで作成する階層プロパティーであり、値はありません。* 116 ページの『WSCollaborations』を参照してください。		
+Collaboration1	*これは階層プロパティーであり、値はありません。*		
++CollaborationPort1	*コラボレーション・ポートの名前*		はい
+++WebServiceOperation1	*これは階層プロパティーであり、値はありません。*		はい
++++BodyName	*Web サービス・メソッドの名前であり、有効な XML エレメント名でなければなりません。*		はい
++++BodyNS	*Web サービス・メソッドのネーム・スペースであり、有効な XML ネーム・スペースでなければなりません。*		はい
++++BOName	*操作のための要求ビジネス・オブジェクトの名前*		はい
++++Mode	synch、asynch	asynch	いいえ
JNDI	*これは、JMS 関連の階層プロパティーであり、値はありません。*		いいえ
+LookupQueuesUsingJNDI	true、false	false	いいえ
+JNDIProviderURL	*有効な JNDI URL*		いいえ
+InitialContextFactory	*初期コンテキストに関するファクトリー・クラスの名前*		いいえ
+JNDIConnectionFactoryName	*JNDI コンテキストを使用して検索する接続ファクトリーの名前*		いいえ
+CTX Properties +CTX_*プロパティー*	*JNDI コンテキストにおけるセキュリティーおよびオブジェクト検索に関する追加情報を指定するプロパティー*		いいえ

ConnectorType: このプロパティーが WebService に設定されている場合、コラボレーション・ポートをバインドすると、System Manager は、そのコネクターを Web サービス・コネクターとして表示します。それ以外の場合には、そのコネクターは通常のコネクターとして表示されます。

デフォルト値は WebService です。

DataHandlerMetaObjectName: これは、データ・ハンドラーが構成プロパティーを設定するために使用する、メタオブジェクトの名前です。

デフォルト値は MO_DataHandler_Default です。

JavaProtocolHandlerPackages: このプロパティーの値は、Java プロトコル・ハンドラー・パッケージを指定します。コネクターは、このプロパティーの値を使用して、システム・プロパティー java.protocol.handler.pkgs を設定します。

デフォルトは com.ibm.net.ssl.internal.www.protocol です。

ProtocolHandlerFramework: プロトコル・ハンドラー・フレームワークは、そのプロトコル・ハンドラーをロードおよび構成するために、このプロパティーを使用します。これは階層プロパティーであり、値はありません。

デフォルト値はありません。

ProtocolHandlers: この階層プロパティーには、値がありません。この第 1 レベルの子は、個別のプロトコル・ハンドラーを表します。

デフォルト値はありません。

SOAPHTTPHTTPSHandler: SOAP/HTTP-HTTPS プロトコル・ハンドラーの名前。これは階層プロパティーです。リスナーとは異なり、プロトコル・ハンドラーは重複しない場合もあるので、それぞれのプロトコルごとにハンドラーは 1 つだけしかないことがあります。次の表 39 は SOAP/HTTP-HTTPS プロトコル・ハンドラーのサブプロパティーを表しています。+ 文字は、プロパティー階層内の項目の位置を示しています。

表 39. SOAP/HTTP-HTTPS プロトコル・ハンドラー構成プロパティー

名前	指定可能な値	デフォルト値	必須
++SOAPHTTPHTTPSHandler	これは階層プロパティーであり、値はありません。		はい
+++Protocol	*ハンドラーがインプリメントしようとしているプロトコルの種類。SOAP/HTTP および SOAP/HTTPS の場合の値は* `soap/http` です。 **注:** このプロパティーの値を指定しなければ、コネクターはプロトコル・ハンドラーを初期化しません。		はい
+++HTTPReadTimeout	*リモート・ホスト (Web サービス) から読み取りを行う際のタイムアウト・インターバル (ミリ秒単位) を指定する、SOAP/HTTP 固有のプロパティー。このプロパティーが指定されていない場合、または 0 に設定されている場合、プロトコル・ハンドラーは、リモート・ホストからの読み取りを行っている間、いつまでもブロックします。*	`0`	いいえ

図 32 は、Connector Configurator Express で表示されたプロパティーを表しています。

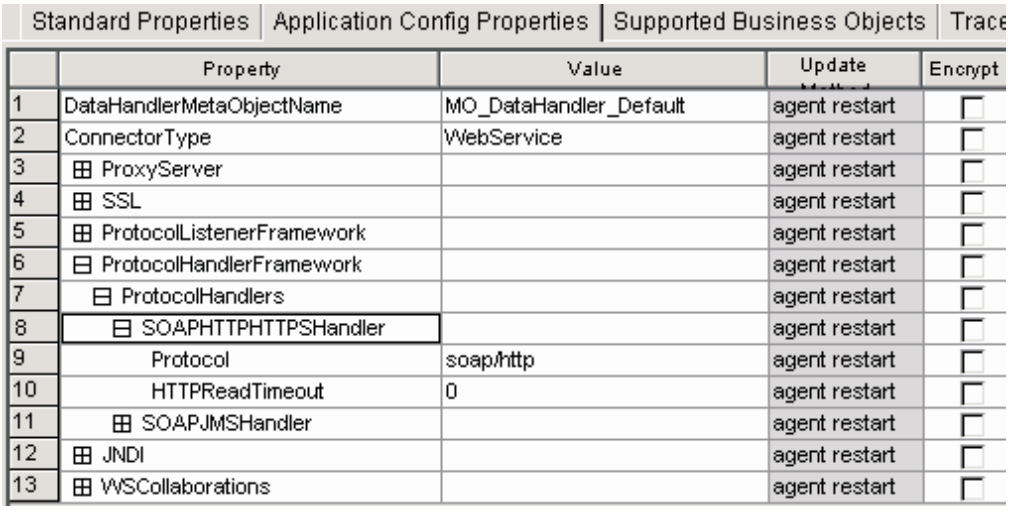


図 32. SOAP/HTTP-HTTPS プロトコル・ハンドラー・プロパティー

SOAPJMSHandler: SOAP/JMS プロトコル・ハンドラーの名前。これは階層プロパティーです。リスナーとは異なり、プロトコル・ハンドラーは重複しない場合もあるので、それぞれのプロトコルごとにハンドラーは 1 つだけしかないことがあります。次の、表 40 は SOAP/JMS プロトコル・ハンドラーのサブプロパティーを表しています。 + 文字は、プロパティー階層内の項目の位置を示しています。

表 40. SOAP/JMS プロトコル・ハンドラー構成プロパティー

名前	指定可能な値	デフォルト値	必須
++SOAPJMSHandler	これは階層プロパティーであり、値はありません。		はい
+++Protocol	ハンドラーがインプリメントしようとしているプロトコルの種類。*SOAP/JMS* の場合の値は `soap/jms` です。 **注:** このプロパティーの値を指定しなければ、コネクターはプロトコル・ハンドラーを初期化しません。		はい
+++ResponseWaitTimeout	*これは、JMS プロトコル・ハンドラー固有のプロパティーで、プロトコル・ハンドラーが ReplyToQueue で同期要求処理を待つ際のタイムアウト・インターバル (ミリ秒単位) を指定します。このインターバル中に応答が到着しない場合、ハンドラーはコラボレーション要求に失敗します。このプロパティーが指定されていない場合、または 0 に設定されている場合、プロトコル・ハンドラーは ReplyToQueue でいつまでも待機します。*	`0`	いいえ

表 40. SOAP/JMS プロトコル・ハンドラー構成プロパティー (続き)

名前	指定可能な値	デフォルト値	必須
+++ReplyToQueue	これは、*ReplyTo* キューの名前を指定する *JMS* プロトコル・ハンドラー固有の必要なプロパティーです。同期要求処理の場合、ハンドラーは *JMSReplyTo* フィールドの値としてこの *JMS* 宛先を指定します。*LookupQueuesUsingJNDI* が `true` であれば、*SOAP/JMS* プロトコル・ハンドラーは *JNDI* を使用してこのキューを検索します。	なし	はい

図 33 は、Connector Configurator Express で表示されたプロパティーを表しています。

Standard Properties | Application Config Properties | Supported Business Objects | Trace/Log Files | Da

	Property	Value	Update	Encrypt	Description
1	DataHandlerMetaObjectName	MO_DataHandler_Default	agent restart	☐	
2	ConnectorType	WebService	agent restart	☐	
3	⊞ ProxyServer		agent restart	☐	
4	⊞ SSL		agent restart	☐	
5	⊞ ProtocolListenerFramework		agent restart	☐	
6	⊟ ProtocolHandlerFramework		agent restart	☐	
7	⊟ ProtocolHandlers		agent restart	☐	
8	⊞ SOAPHTTPHTTPSHandler		agent restart	☐	
9	⊟ SOAPJMSHandler		agent restart	☐	
10	Protocol	soap/jms	agent restart	☐	
11	ResponseWaitTimeout	0	agent restart	☐	
12	ReplyToQueue		agent restart	☐	
13	⊞ JNDI		agent restart	☐	
14	⊞ WSCollaborations		agent restart	☐	

図 33. SOAP/JMS プロトコル・ハンドラー・プロパティー

ProtocolListenerFramework: プロトコル・リスナー・フレームワークは、このプロパティーを使用してプロトコル・リスナーをロードします。これは階層プロパティーであり、値はありません。

WorkerThreadCount: このプロパティー (これは 1 以上の整数でなければなりません) は、プロトコル・リスナー・フレームワークで使用できるプロトコル・リスナー・ワーカー・スレッドの数を設定します。詳しくは、70 ページの『プロトコル・リスナー』を参照してください。デフォルト値は `10` です。

RequestPoolSize: このプロパティー (これは、WorkerThreadCount より大きい整数でなければなりません) は、プロトコル・リスナー・フレームワークのリソース・プール・サイズを設定します。このフレームワークは、最大で、WorkerThreadCount 要求と RequestPoolSize 要求の合計数を並行して処理することができます。

デフォルト値は `20` です。

ProtocolListeners: これは階層プロパティーであり、値はありません。このプロパティーの第 1 レベルの子は、それぞれ、個別のプロトコル・リスナーを表します。

Listener1: プロトコル・リスナーの名前。複数のプロトコル・リスナーが存在する場合があります。これは階層プロパティーです。このプロパティーのインスタンスを複数作成して、固有の名前を持つ、追加のリスナーを作成することができます。このようにする場合、リスナー固有のプロパティーは変更できますが、プロトコル・プロパティーは変更できません。複数のリスナーの名前は固有でなければなりません。指定可能な名前 (値ではありません) は、`SOAPHTTPListener1`、`SOAPHTTPSListener1`、`SOAPJMSListener1` のいずれかです。

Protocol: このプロパティーは、このリスナーがインプリメントしようとしているプロトコルを指定します。指定可能な値: `soap/http`、`soap/https`、`soap/jms`

注: このプロパティーの値を指定しなければ、コネクターはプロトコル・リスナーを初期化しません。

SOAPDHMimeType: このリスナーによって受け取られた要求のために使用される、SOAP データ・ハンドラーの MIME タイプ。

デフォルト値は `xml/soap` です。

ListenerSpecific: リスナー固有のプロパティーは、指定されたプロトコル・リスナーに固有の、あるいは指定されたプロトコル・リスナーにとって必須のプロパティーです。例えば、HTTP リスナーにはリスナー固有のプロパティー Port が設定されますが、これは、リスナーが要求をモニターするポート番号を表します。表 41 は、HTTP/HTTPS リスナーに固有のプロパティーを要約したものです。+ 文字は、プロパティー階層内の項目の位置を示しています。

表 41. SOAP/HTTP および SOAP/HTTPS プロトコル・リスナーに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	必須
+++SOAPHTTPListener1	*HTTP プロトコル・リスナーの固有の名前。これは、ProtocolListenerFramework -> ProtocolListeners 階層プロパティーの子です。リスナーは複数存在する場合があります。このプロパティーおよびその階層の別のインスタンスを作成することにより、追加の HTTP リスナーを付け加えることができます。*		はい
++++Protocol	`soap/http` (SOAP/HTTP プロトコル・リスナーの場合) `soap/https` (SOAP/HTTPS プロトコル・リスナーの場合) **注:** このプロパティーの値を指定しなければ、コネクターはプロトコル・リスナーを初期化しません。		はい
++++SOAPDHMimeType	`xml/soap`	`xml/soap`	いいえ
++++BOPrefix	*このプロパティーの値は、データ・ハンドラーに渡されます。*		いいえ

表 41. SOAP/HTTP および SOAP/HTTPS プロトコル・リスナーに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	必須
++++Host	リスナーは、このプロパティーの値によって指定された IP アドレスで要求を listen します。Host が指定されない場合、デフォルト値として localhost が使用されます。リスナーを実行しているマシンのホスト名 (DNS 名) または IP アドレスのどちらを指定してもかまいません。マシンには、複数の IP アドレスまたは複数の名前がある場合があります。	localhost	いいえ
++++Port	リスナーが要求を listen するポート。ポートが指定されていない場合、ポートのデフォルトとして、SOAP/HTTP の場合は 80、SOAP/HTTPS の場合は 443 が使用されます。コネクター内でリスナーを複製した場合、Host プロパティーと Port プロパティーの組み合わせは固有にします。そうしないと、リスナーは、要求を受け入れるためにポートにバインドすることができなくなる可能性があります。	SOAP/HTTP リスナーの場合は 80、SOAP/HTTPS リスナーの場合は 443	いいえ
++++SocketQueueLength	着信接続要求のためのキュー (ソケット・キュー) の長さ。ホストに接続を拒否されずに、一度に保管できる着信接続の数を指定します。キューの最大長は、オペレーティング・システムに依存します。	5	いいえ
++++RequestWaitTimeout	Web サービス要求の到着を待つ間、リスナー・スレッドがホストおよびポートでブロックするミリ秒単位の時間間隔。この間隔以内に要求を受け取った場合、リスナーはその要求を処理します。それ以外の場合、リスナー・スレッドは、コネクター・シャットダウン・フラグが設定されているかどうかを調べます。このフラグが設定されている場合、コネクターは終了します。このフラグが設定されていない場合、RequestWaitTimeout 間隔の間、ブロックが継続します。このプロパティーを 0 に設定した場合、ブロックはいつまでも継続します。このプロパティーが指定されていないと、デフォルトとして 60000 ミリ秒が使用されます。	60000 (ms)	いいえ
++++HTTPReadTimeout	クライアントからの Web サービス要求を読み取っているときにリスナーがブロックされる時間間隔を、ミリ秒単位で指定します。このパラメーターを 0 に設定すると、リスナーは、要求メッセージ全体を受け取るまで、いつまでもブロックされます。	0	いいえ
++++HttpAsyncResponseCode	リスナーへの非同期要求に対する HTTP 対応コード: 200 (OK) 202 (ACCEPTED)	202 (ACCEPTED)	いいえ

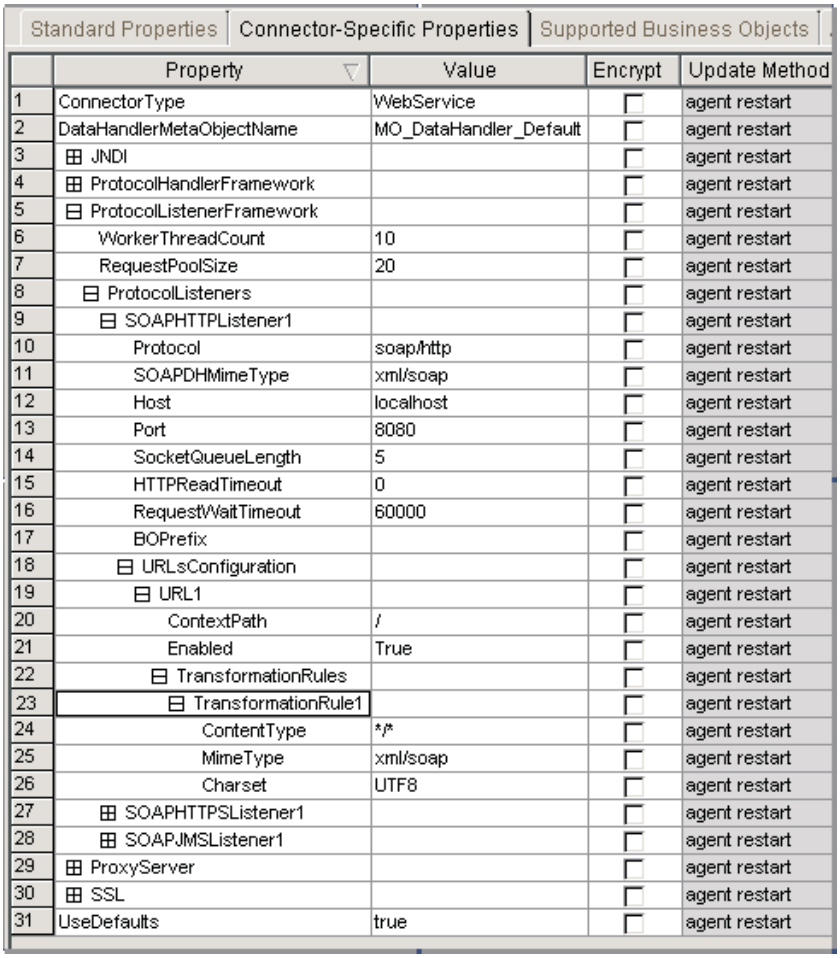
表 41. SOAP/HTTP および SOAP/HTTPS プロトコル・リスナーに固有の構成プロパティー (続き)

名前	指定可能な値	デフォルト値	必須
++++URLsConfiguration	*これは階層プロパティーであり、値はありません。ここには、このリスナーがサポートする URL の構成が 1 つ以上格納され、さらにオプションで、MIME タイプと charset の値が格納されます。これは、*ProtocolListenerFramework-> ProtocolListeners-> SOAPHTTPListener1 *階層プロパティーの子プロパティーです。このプロパティーが指定されない場合、リスナーはデフォルト値を想定します。*	ContextPath: / 使用可能: true データ・ハンドラー MimeType: 要求の ContentType に等しい Charset: なし。詳しくは、「SOAP/HTTP および SOAP/HTTPS プロトコル・リスナー」を参照してください。	いいえ
+++++URL1	これは階層プロパティーであり、値はありません。この子には、このリスナーがサポートする *URL* の名前があります。サポートされる *URL* は、*複数存在する場合があります。このプロパティーおよびその階層を複製すれば、追加の URL を付け加えることができます。*		いいえ
++++++ContextPath	*リスナーが受け取る HTTP 要求の URI。この値は、URLsConfiguration プロパティーの下の ContextPath 値の中で固有でなければなりません。そうでないと、コネクターはログにエラーを記録し、始動に失敗します。ContextPath の値については、大/小文字の区別が行われます。ただし、大/小文字を区別しないプロトコル、ホスト名、およびポートを含めることができます。プロトコルを ContextPath に指定する場合は、http にする必要があります。ホストを指定する場合には、Host リスナー・プロパティーの値と同じにする必要があります。ポートを指定する場合には、Port リスナー・プロパティーの値と同じにする必要があります。このプロパティーは、双方向言語の変換を行う場合に使用可能にします。*		いいえ
++++++Enabled	*このプロパティーの値によって、親 URL 階層プロパティーがコネクターで使用可能かどうかが決まります。*	True	いいえ
++++++TransformationRules	これは階層プロパティーであり、値はありません。*1 つ以上の変換規則が保持されます。*		
+++++++TransformationRule1	これは階層プロパティーであり、値はありません。*ここには変換規則が保持されます。*		いいえ

表 41. SOAP/HTTP および SOAP/HTTPS プロトコル・リスナーに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	必須
++++++++ContentType	このプロパティーの値は、特殊処理 (データ・ハンドラー MIME タイプまたは charset) が適用される着信要求の ContentType を示します。TransformationRuleN 階層プロパティーによって ContentType が指定されていない場合、コネクターは、ログに警告メッセージを記録して、その TransformationRuleN プロパティーを無視します。このプロパティーに特殊値 */* を指定すると、プロトコル・リスナーは、このルールを任意の ContentType に適用できます。同じコンテキスト・パスで、ContentType を共有する規則が複数検出された場合、リスナーはログにエラーを記録し、初期化に失敗します。		いいえ
++++++++MimeType	指定された ContentType の要求の処理中に、データ・ハンドラーの呼び出しで使用する MIME タイプ。		いいえ
++++++++Charset	指定された ContentType の要求をビジネス・オブジェクトに変換するときに使用する文字セット。		いいえ

図 34 は、Connector Configurator Express で表示されたプロパティーを表しています。



	Property	Value	Encrypt	Update Method
1	ConnectorType	WebService	☐	agent restart
2	DataHandlerMetaObjectName	MO_DataHandler_Default	☐	agent restart
3	⊞ JNDI		☐	agent restart
4	⊞ ProtocolHandlerFramework		☐	agent restart
5	⊟ ProtocolListenerFramework		☐	agent restart
6	WorkerThreadCount	10	☐	agent restart
7	RequestPoolSize	20	☐	agent restart
8	⊟ ProtocolListeners		☐	agent restart
9	⊟ SOAPHTTPListener1		☐	agent restart
10	Protocol	soap/http	☐	agent restart
11	SOAPDHMimeType	xml/soap	☐	agent restart
12	Host	localhost	☐	agent restart
13	Port	8080	☐	agent restart
14	SocketQueueLength	5	☐	agent restart
15	HTTPReadTimeout	0	☐	agent restart
16	RequestWaitTimeout	60000	☐	agent restart
17	BOPrefix		☐	agent restart
18	⊟ URLsConfiguration		☐	agent restart
19	⊟ URL1		☐	agent restart
20	ContextPath	/	☐	agent restart
21	Enabled	True	☐	agent restart
22	⊟ TransformationRules		☐	agent restart
23	⊟ TransformationRule1		☐	agent restart
24	ContentType	*/*	☐	agent restart
25	MimeType	xml/soap	☐	agent restart
26	Charset	UTF8	☐	agent restart
27	⊞ SOAPHTTPSListener1		☐	agent restart
28	⊞ SOAPJMSListener1		☐	agent restart
29	⊞ ProxyServer		☐	agent restart
30	⊞ SSL		☐	agent restart
31	UseDefaults	true	☐	agent restart

図 34. SOAP/HTTP プロトコル・リスナー・プロパティー

表 42 は、SOAP/JMS プロトコル・リスナーに固有のプロパティーを要約したものです。+ 文字は、プロパティー階層内の項目の位置を示しています。

表 42. SOAP/JMS プロトコル・リスナーに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	必須
+++SOAPJMSListener1	*JMS プロトコル・リスナーの固有の名前。これは、ProtocolListenerFramework -> ProtocolListeners 階層プロパティーの子です。リスナーは複数存在する場合があります。このプロパティーおよびその階層の別のインスタンスを作成することにより、追加の JMS リスナーを付け加えることができます。*		はい
++++Protocol	`soap/jms`		はい
++++SOAPDHMimeType	`xml/soap`	`xml/soap`	いいえ
++++BOPrefix	*このプロパティーの値は、SOAPDHMimeType プロパティーによって指定されたデータ・ハンドラーに渡されます。*		いいえ

表 42. SOAP/JMS プロトコル・リスナーに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	必須
++++RequestWaitTimeout	このプロパティーは、*Web* サービス要求を待つ際に *SOAP/JMS* リスナー・スレッドが *InputQueue* をブロックする時間間隔を設定します。この間隔以内に *Web* サービス要求を受け取った場合、リスナーはその要求を処理します。この間隔以内に要求を受け取らなかった場合、リスナー・スレッドは最初に、コネクター・シャットダウン・フラグが設定されているかどうかを調べます。コネクター・シャットダウン・フラグが設定されている場合、コネクターは終了します。このフラグが設定されていない場合、*RequestWaitTimeout* 間隔の間、ブロックが継続します。このプロパティーを *0* に設定した場合、ブロックはいつまでも継続します。	60000 ミリ秒	いいえ
++++SessionPoolSize	任意のリスナーとそのワーカー・スレッドに割り振り可能なセッションの最大数。セッションの最小数は *2* です *(*これがデフォルトです*)*。セッション・プール・サイズを大きくすると、コネクターが必要とするメモリーも多くなります。	2	いいえ
++++InputQueue	このプロパティーでは、リスナーが *Web* サービスからのインバウンド・メッセージをポーリングする入力キューの名前を指定します。*LookupQueuesUsingJNDI* が `true` である場合、リスナーが *JNDI* を使用してこのキューを検索し、*InputQueue* プロパティーの値が *SOAP/JMS* バインディングの *jms:address* 要素の *jndiDestinationName* 属性に設定されます。*jms:address* 要素は、*WSDL* 文書の *wsdl:port* セクションで指定されています。*WSDL* の生成中に *SOAP/JMS* リスナーを選択した場合、*System Manager* は、このプロパティーの値を使用して自動的に *jndiDestinationName* 属性を作成します。*LookupQueueUsingJNDI* が `false` である場合、*System Manager* は *jmsProviderDestinationName* 属性を作成します。		はい
++++InProgressQueue	このプロパティーでは、進行中のキューの名前を指定します。リスナーは、*InputQueue* から *InProgressQueue* にインバウンド・メッセージのコピーを送信します。*LookupQueuesUsingJNDI* = `true` である場合、リスナーは *JNDI* を使用してこのキューを検索します。		はい
++++ArchiveQueue	このプロパティーでは、アーカイブ・キューの名前を指定します。リスナーは、*InProgressQueue* から *ArchiveQueue* に、正常に処理されたメッセージのコピーを送信します。*LookupQueuesUsingJNDI* = `true` である場合、リスナーは *JNDI* を使用してこのキューを検索します。		いいえ

表 42. SOAP/JMS プロトコル・リスナーに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	必須
++++UnsubscribedQueue	このプロパティーでは、アンサブスクライブされたキューの名前を指定します。リスナーは、*InProgressQueue* から *UnsubscribedQueue* に、アンサブスクライブされたメッセージのコピーを送信します。*LookupQueueUsingJNDI* = true である場合、リスナーは *JNDI* を使用してこのキューを検索します。		いいえ
++++ErrorQueue	このプロパティーでは、エラー・キューの名前を指定します。リスナーは、失敗したメッセージのコピーを *ErrorQueue* に送信します。*LookupQueueUsingJNDI* = true である場合、リスナーは *JNDI* を使用してこのキューを検索します。		いいえ
++++InDoubtEvents	このプロパティーでは、予期しないコネクター終了のために完全に処理されなかった、*InProgressQueue* 内のメッセージの取り扱い方法を指定します。このプロパティーには、次のいずれかの値を指定することができます。 • FailOnStartup: エラーをログに記録し、ただちにシャットダウンします。 • Reprocess: *InProgressQueue* 内の残りのメッセージを処理します • Ignore: 進行中キュー内のメッセージをすべて無視します • LogError: エラーをログに記録しますが、シャットダウンしません	Ignore	いいえ
++++ReplyToQueue	このプロパティーでは、*ReplyTo* キューの名前を指定します。*WSDL* 構成ウィザードはこのプロパティーを読み取り、それを *WSDL* 文書に書き込みます。このプロパティーが指定されない場合、このユーティリティーは、*WSDL* 文書内の *SOAP/JMS* バインディングに *ReplyTo JMS* ヘッダーを作成しません。(リスナーはこのプロパティーを使用しません。) *JNDI* プロパティーが指定されていて、*LookupQueueUsingJNDI* が false である場合にも、*WSDL Generation Utility* は *WSDL* 文書内に *JNDI* 固有の属性を作成します。これらの *JNDI* 固有属性が必要であるのは、*JNDI* がなければ *SOAP/JMS* バインディングが *ReplyTo* 属性を指定できないためです。*JNDI* による *InputQueue* の検索は必要ではありませんが、*ReplyTo* キューを使用するためには、*JNDI* 固有のプロパティーが必要です。*JNDI* 固有のプロパティーが見付からなかった場合、*WSDL* ユーティリティーは、*SOAP/JMS* バインディング内で *ReplyTo* 属性を作成することができません。		

表 42. SOAP/JMS プロトコル・リスナーに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	必須
++++ JMSVendorURI	*JMS インプリメンテーションを一意的に識別し、SOAP/JMS バインディングの jms:address 要素の jmsVendorURI 属性に対応するストリング。jms:address 要素は、WSDL 文書の wsdl:port セクションで指定されています。リスナーはこのプロパティーを使用しません。このプロパティーは、双方向言語の変換を行う場合に使用可能にします。*		いいえ

図 35 は、Connector Configurator Express で表示されたプロパティーを表しています。

Standard Properties | Application Config Properties | Supported Business Objects | Trace/Log Files | D

	Property	Value	Update	Encrypt	Description
1	DataHandlerMetaObjectName	MO_DataHandler_Default	agent restart	☐	
2	ConnectorType	WebService	agent restart	☐	
3	⊞ ProxyServer		agent restart	☐	
4	⊞ SSL		agent restart	☐	
5	⊟ ProtocolListenerFramework		agent restart	☐	
6	WorkerThreadCount	10	agent restart	☐	
7	RequestPoolSize	20	agent restart	☐	
8	⊟ ProtocolListeners		agent restart	☐	
9	⊞ SOAPHTTPListener1		agent restart	☐	
10	⊟ SOAPJMSListener1		agent restart	☐	
11	Protocol	soap/jms	agent restart	☐	
12	SOAPDHMimeType	xml/soap	agent restart	☐	
13	InputQueue		agent restart	☐	
14	InProgressQueue		agent restart	☐	
15	ArchiveQueue		agent restart	☐	
16	UnsubscribedQueue		agent restart	☐	
17	ErrorQueue		agent restart	☐	
18	InDoubtEvents		agent restart	☐	
19	ReplyToQueue		agent restart	☐	
20	JMSVendorURI		agent restart	☐	
21	RequestWaitTimeout	60000	agent restart	☐	
22	BOPrefix		connector rest	☐	Property Nam
23	⊞ SOAPHTTPSListener1		agent restart	☐	
24	⊞ ProtocolHandlerFramework		agent restart	☐	
25	⊞ JNDI		agent restart	☐	
26	⊞ WSCollaborations		agent restart	☐	

図 35. SOAP/JMS プロトコル・リスナー・プロパティー

注: 以下のプロパティーに指定されたキュー名が固有であることを確認してください。

- InputQueue

- InProgressQueue
- ArchiveQueue
- UnsubscribedQueue
- ErrorQueue

ProxyServer: ネットワークがプロキシー・サーバーを使用する場合には、このプロパティー以下の値を構成してください。これは階層プロパティーであり、値はありません。このプロパティー以下で指定された値は、SOAP/HTTP/HTTPS プロトコル・ハンドラーによって使用されます。

図 36 は、Connector Configurator Express で表示されたプロキシー・サーバーのプロパティーを表しており、以下にその説明をしています。

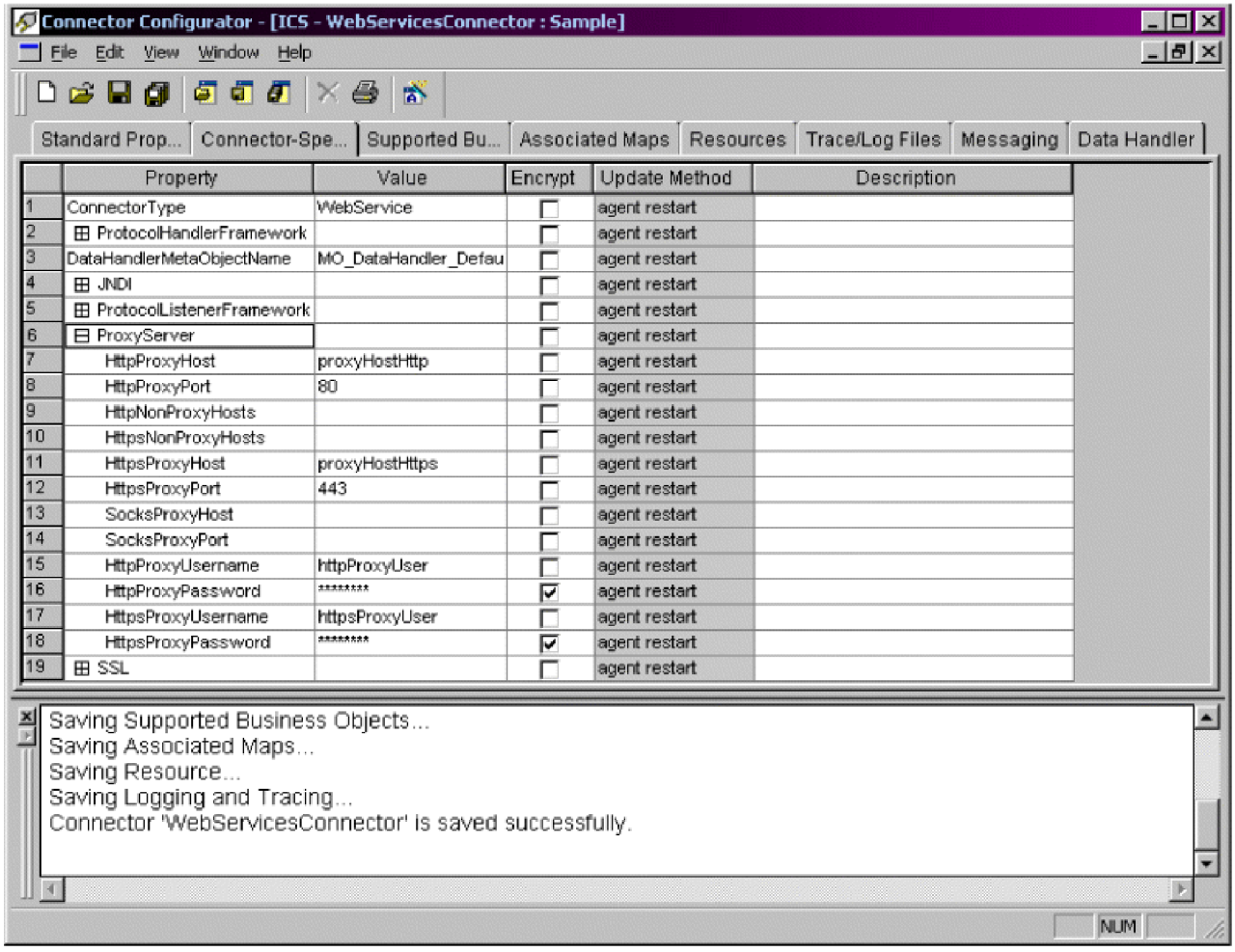


図 36. ProxyServer プロパティー

HttpProxyHost: HTTP プロキシー・サーバーのホスト名。ネットワークが HTTP プロトコルでプロキシー・サーバーを使用する場合には、このプロパティーを指定してください。

デフォルト値はありません。

HttpProxyPort: HTTP プロキシー・サーバーに接続するためにコネクターが使用するポート番号。

デフォルト値は 80 です。

HttpNonProxyHosts: このプロパティーの値は、プロキシー・サーバー経由ではなく直接接続する必要のある、1 つまたは複数のホスト (HTTP の場合) を表します。この値は、各ホストを ″|″ で区切った、ホストのリストの形で指定することができます。

デフォルト値はありません。

HttpsProxyHost: HTTPS プロキシー・サーバーの場合のホスト名。

デフォルト値はありません。

HttpsProxyPort: HTTPS プロキシー・サーバーに接続するためにコネクターが使用するポート番号。

デフォルト値は 443 です。

HttpsNonProxyHosts: このプロパティーの値は、プロキシー・サーバー経由ではなく直接接続する必要のある、1 つまたは複数のホスト (HTTPS の場合) を表します。この値は、各ホストを ″|″ で区切った、ホストのリストの形で指定することができます。

デフォルト値はありません。

SocksProxyHost: Socks Proxy サーバーの場合のホスト名。ネットワークが Socks プロキシーを使用する場合には、このプロパティーを指定してください。

注: 基礎となる JDK は Socks をサポートしていなければなりません。

デフォルト値はありません。

SocksProxyPort: Socks Proxy サーバーに接続するためのポート番号。ネットワークが Socks プロキシーを使用する場合には、このプロパティーを指定してください。

デフォルト値はありません。

HttpProxyUsername: HTTP プロキシー・サーバーのユーザー名。Web サービス要求の宛先が HTTP URL であり、ProxyServer ->HttpProxyUsername が指定された場合、SOAP HTTP/HTTPS プロトコル・ハンドラーは、このプロキシーでの認証のときに Proxy-Authorization ヘッダーを作成します。ハンドラーは、認証に CONNECT メソッドを使用します。

このプロキシー認証ヘッダーは Base64 方式でエンコードされており、以下の構造になっています。
Proxy-Authorization: Basic
Base64EncodedString

ハンドラーは、ユーザー名とパスワードのプロパティー値を連結し、その間をコロン (:) で区切って、Base64 エンコードされたストリングを作成します。

デフォルト値はありません。

HttpProxyPassword: HTTP プロキシー・サーバーのパスワード。これらの値の使用に関する詳細については、113 ページの『HttpProxyUsername』を参照してください。

デフォルト値はありません。

HttpsProxyUsername: HTTPS プロキシー・サーバーのユーザー名。Web サービス要求の宛先が HTTPS URL であり、ProxyServer ->HttpsProxyUsername が指定された場合、SOAP HTTP/HTTPS プロトコル・ハンドラーは、このプロキシーでの認証に対して Proxy-Authorization ヘッダーを作成します。ハンドラーは、HttpsProxyUsername と HttpsProxyPassword 構成プロパティー値を連結し、その間をコロン (:) で区切って、Base64 エンコードされたストリングを作成します。

デフォルト値はありません。

HttpsProxyPassword: HTTPS プロキシー・サーバーのパスワード。これらの値の使用に関する詳細については、『HttpsProxyUsername』を参照してください。

デフォルト値はありません。

SSL: コネクターのための SSL を構成するためには、このプロパティー以下の値を指定してください。これは階層プロパティーであり、値はありません。

図 37 は、Connector Configurator Express で表示された SSL のプロパティーを表しており、以下にその説明をしています。

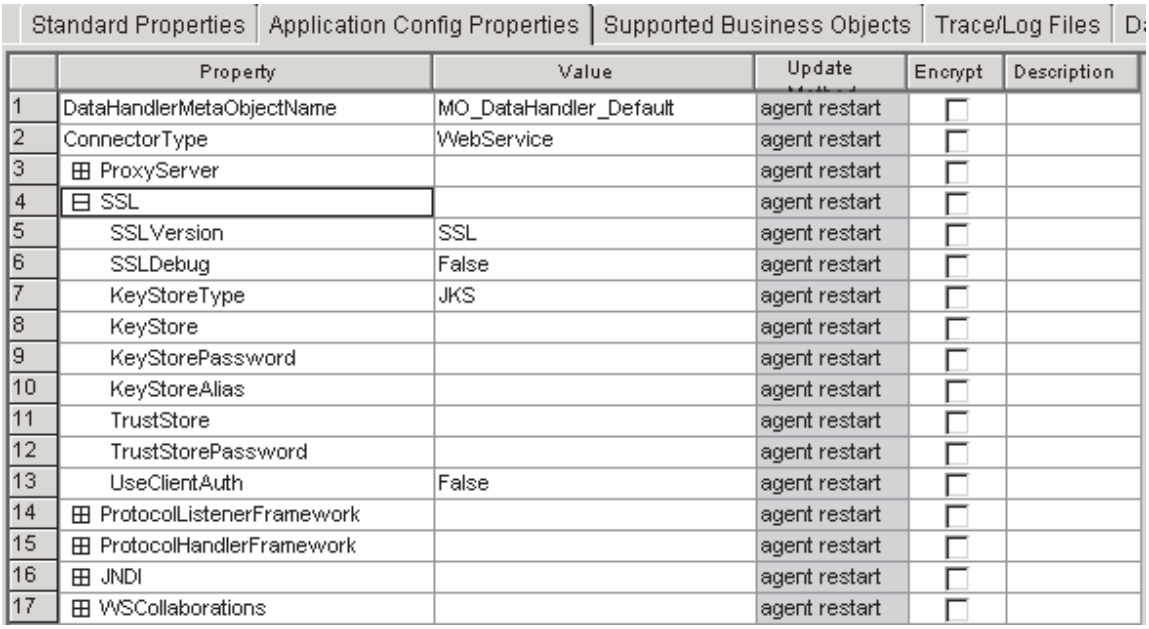


図 37. SSL プロパティー

SSLVersion: コネクターによって使用される SSL のバージョン。詳しくは、IBM JSSE 資料に記載された、サポートされる SSL バージョンに関する説明を参照してください。

デフォルト値は `SSL` です。

SSLDebug: このプロパティーの値が `true` に設定されている場合、コネクターは `javax.net.debug` システム・プロパティーの値を `true` に設定します。IBM JSSE はこのプロパティーを使用して、トレース機能をオンにします。詳しくは IBM JSSE の資料を参照してください。

デフォルト値は `false` です。

KeyStoreType: このプロパティーの値は、KeyStore および TrustStore のタイプを表します。詳しくは、IBM JSSE 資料に記載された、有効な鍵ストア・タイプに関する説明を参照してください。

デフォルト値は `JKS` です。

KeyStore: このプロパティーでは、鍵ストア・ファイルへの完全パスを指定します。KeyStore または KeyStoreAlias プロパティー、あるいはその両方が指定されていない場合には、KeyStorePassword、KeyStoreAlias、TrustStore、および TrustStorePassword の各プロパティーは無視されます。コネクターは、このプロパティーで指定されたパスを使用して鍵ストアをロードすることができない場合、始動に失敗します。このパスは、鍵ストア・ファイルの完全パスでなければなりません。

デフォルト値はありません。

KeyStorePassword: このプロパティーでは、Keystore 内の秘密鍵のパスワードを指定します。

デフォルト値はありません。

KeyStoreAlias: このプロパティーでは、KeyStore 内の鍵ペアの別名を指定します。SOAP/HTTPS リスナーは、KeyStore からこの秘密鍵を入手して使用します。また、SOAP/HTTP-HTTPS プロトコル・ハンドラーは、クライアント認証を必要とする Web サービスを呼び出すときに、KeyStore からこの別名を入手して使用します。このプロパティーは、有効な JSSE 別名に設定しなければなりません。

デフォルト値はありません。

TrustStore: このプロパティーでは、TrustStore への完全パスを指定します。TrustStore は、コネクターに信頼される証明書を保管するために使用されます。TrustStore のタイプは KeyStore と同じでなければなりません。TrustStore ファイルの完全パスを指定しなければなりません。

デフォルト値はありません。

TrustStorePassword: このプロパティーは、Truststore のパスワードを指定します。

デフォルト値はありません。

UseClientAuth: このプロパティーでは、SSL クライアント認証を使用するかどうかを指定します。このプロパティーが `true` に設定されている場合、SOAP/HTTPS リスナーはクライアント認証を使用します。

デフォルト値は `false` です。

WSCollaborations: このプロパティーは、ユーザーがコラボレーション・オブジェクトを Web サービスとして公開するときに自動的に作成され、非 TLO で使用されます。これは階層プロパティーであり、値はありません。このプロパティーの第 1 レベルのそれぞれの子は、Web サービスとして公開されたコラボレーションを表します。これらのプロパティーを自動的に作成するのに使用するためのツールについては、167 ページの『第 7 章 Web サービスとしてのコラボレーションの公開』を参照してください。

注: System Manager でコラボレーションまたはそのポートを削除しても、コネクターはコラボレーションを表すプロパティーを自動的には削除しません。これらのプロパティーは、Connector Configurator Express を使用して削除しなければなりません。

図 38 は、Connector Configurator Express で表示された WSCollaborations のプロパティーを表しており、以下にその説明をしています。

図 38. WSCollaborations プロパティー

Collaboration1: このプロパティーでは、このコネクターを介して Web サービスとして公開されるコラボレーション・オブジェクトを指定します。これは階層プロパティーであり、値はありません。Web サービスとして公開されるそれぞれのコラボレーション・オブジェクトごとに 1 つずつ、このようなプロパティーを複数指定することができます。このプロパティーの第 1 レベルのそれぞれの子は、このコラボレーション・オブジェクトのポートを表します。

CollaborationPort1: このプロパティーでは、コラボレーション・ポートを指定します。これは階層プロパティーであり、値はありません。コネクターにバインドされるこのコラボレーションのそれぞれのポートごとに 1 つずつ、このようなプロパティーを複数指定することができます。このプロパティーの第 1 レベルのそれぞれの子は、Web サービスの操作を表します。

WebServiceOperation1: このプロパティーでは、コラボレーション・オブジェクトに関する Web サービス操作を表します。これは階層プロパティーであり、値はありません。 WSDL 文書生成時にユーザーが定義したそれぞれの Web サービス操作ごとに 1 つずつ、このようなプロパティーを複数指定することができます。

BodyName: このプロパティーでは、Web サービス方式の名前を指定します。この名前は有効な XML 要素名でなければなりません。

デフォルト値はありません。

BodyNS: このプロパティーでは、Web サービス方式のネーム・スペースを指定します。この名前は有効な XML 要素ネーム・スペースでなければなりません。

デフォルト値はありません。

BOName: このプロパティーでは、この操作に関する要求ビジネス・オブジェクトの名前を指定します。

デフォルト値はありません。

Mode: このプロパティーでは、操作の処理モードを指定します。このプロパティーを synch に設定すると、コネクターはコラボレーションを同期的に呼び出すようになります。それ以外の値を設定した場合、およびデフォルト値の場合、コネクターは、コラボレーションを要求専用操作として非同期的に呼び出すようになります。

デフォルト値は asynch です。

JNDI: コネクターは、JNDI (Java Naming and Directory Interface) への接続時に SOAP/JMS プロトコル・ハンドラーおよび JMS プロトコル・リスナーによって使用される JNDI プロバイダー・プロパティーの 1 組のセットを維持します。これは階層プロパティーであり、値はありません。コネクターは、JNDI を使用して JMS 接続ファクトリー・オブジェクトを検索します。このプロパティーは、SOAP/JMS バインディングの生成時に WSDL 構成ウィザードによって使用されることに注意してください。

図 39 は、Connector Configurator Express で表示された JNDI のプロパティーを表しており、以下にその説明をしています。

Standard Properties | Application Config Properties | Supported Business Objects | Trace/Log Files | D

	Property	Value	Update	Encrypt	Description
1	DataHandlerMetaObjectName	MO_DataHandler_Default	agent restart	☐	
2	ConnectorType	WebService	agent restart	☐	
3	⊞ ProxyServer		agent restart	☐	
4	⊞ SSL		agent restart	☐	
5	⊞ ProtocolListenerFramework		agent restart	☐	
6	⊞ ProtocolHandlerFramework		agent restart	☐	
7	⊟ JNDI		agent restart	☐	
8	LookupQueuesUsingJNDI	False	agent restart	☐	
9	InitialContextFactory		agent restart	☐	
10	JNDIConnectionFactoryName		agent restart	☐	
11	CTX_ObjectFactories		agent restart	☐	
12	CTX_StateFactories		agent restart	☐	
13	CTX_URLPackagePrefixes		agent restart	☐	
14	CTX_DNS_URL		agent restart	☐	
15	CTX_Authoritative		agent restart	☐	
16	CTX_Batchsize		agent restart	☐	
17	CTX_Referral		agent restart	☐	
18	CTX_SecurityProtocol		agent restart	☐	
19	CTX_SecurityAuthentication		agent restart	☐	
20	CTX_SecurityPrincipal		agent restart	☐	
21	CTX_SecurityCredentials		agent restart	☐	
22	CTX_Language		agent restart	☐	
23	⊞ WSCollaborations		agent restart	☐	

図 39. JNDI プロパティー

LookupQueuesUsingJNDI: このプロパティーの値が true に設定されている場合、コネクターの SOAP/JMS リスナーおよび SOAP/JMS プロトコル・ハンドラーは、JNDI を使用してキューを検索します。

デフォルト値は false です。

JNDIProviderURL: このプロパティーでは、JNDI サービス・プロバイダーの URL を指定します。これは、SOAP/JMS バインディングの jms:address 要素の jndiProviderURL 属性に対応します。jms:address 要素は、wsdl:port セクションで指定されています。これは、デフォルトの JNDI プロバイダーとして使用されるものであり、有効な JNDI URL でなければなりません。詳しくは JNDI 仕様を参照してください。

このプロパティーは、双方向言語の変換を行う場合に使用可能にします。

デフォルト値はありません。

InitialContextFactory: このプロパティーでは、初期コンテキストを作成するファクトリー・クラスの完全修飾クラス名 (例えば、com.ibm.NamingFactory) を指定します。これは、SOAP/JMS バインディングの jms:address 要素の initialContextFactory 属性に対応します。jms:address 要素は、wsdl:port セクションで指定されています。

デフォルト値はありません。

JNDIConnectionFactoryName: このプロパティーでは、JNDI コンテキストを使用して検索を行う接続ファクトリーの名前を指定します。これは、SOAP/JMS バインディングの jms:address 要素の jndiConnectionFactoryName 属性に対応します。jms:address 要素は、wsdl:port セクションで指定されています。

デフォルト値はありません。

CTX Properties: JNDI コンテキストにおけるセキュリティーおよびオブジェクト検索に関する追加情報を指定するプロパティー。表 43 は、これらのプロパティーを要約したものです。 + 文字は、プロパティー階層内の項目の位置を示しています。

+CTX_DNS_URL プロパティーは、双方向言語の変換を行う場合に使用可能にします。

表 43. Java Naming and Directory Interface (JNDI) プロバイダーのプロパティー

プロパティー名	説明
+CTX_ObjectFactories +CTX_StateFactories +CTX_URLPackagePrefixes +CTX_DNS_URL +CTX_Authoritative +CTX_Batchsize +CTX_Referral +CTX_SecurityProtocol +CTX_SecutiryAuthentication +CTX_SecurityPrincipal +CTX_SecurityCredentials +CTX_Language	JNDI コンテキストにおけるセキュリティーおよびオブジェクト検索に関する追加情報を指定するプロパティー。詳しくは J2EE 資料を参照してください。これらのプロパティーは、Adapter for JMS で使用されるものを反映しています。

複数のプロトコル・リスナーの作成

プロトコル・リスナーのインスタンスを複数作成することができます。プロトコル・リスナーは、ProtocolListenerFramework> ProtocolListeners コネクター・プロパ

ティーの子プロパティーとして構成されます。(ProtocolListenerFramework -> ProtocolListeners の) それぞれの子プロパティーは、コネクターのプロトコル・リスナーを明確に識別します。したがって、ProtocolListeners プロパティーの下で、新しい子プロパティーを構成することにより、追加のプロトコル・リスナーを作成することができます。新規に作成したリスナー・プロパティーの子プロパティーをすべて指定していることを、確認してください。各リスナーの名前は固有でなければなりません。ただし、リスナーのプロトコル・プロパティー (soap/http、soap/https、soap/jms) は変更しません。これらは、リスナーの複数インスタンス用に同じ名前で残ります。

注: このプロトコル・プロパティーはスイッチの役目をしているため非常に重要です。リスナーまたはハンドラーを使用したくない場合は、このプロパティーを空のままにします。

SOAP/HTTP リスナーまたは SOAP/HTTPS リスナーの複数インスタンスを作成している場合は、それぞれのインスタンスに別々のポートおよびホスト・プロパティーを必ず指定してください。複数の SOAP/JMS リスナーを指定している場合は、それぞれのインスタンスに対して必ず別々のキューのセットを使用します。

1 つのハンドラーの複数のインスタンスは作成できません。プロトコルごとにハンドラーは 1 つしか存在できません。

始動時のコネクター

コネクターを始動すると、init() メソッドは、System Manager の Connector Configurator Express を使用して設定された構成プロパティーを読み取ります。正常に機能させるために、絶対にコネクターのポーリングを使用不可にしないでください (コネクターのポーリングはデフォルトで使用可能になっています)。この際にどのようなことが行われるのかについて、以下のセクションで説明します。

プロキシーのセットアップ

コネクター固有の ProxyServer プロパティーを指定すると、コネクターはプロキシー・システム・プロパティーをセットアップします。プロキシー・サーバーは、要求処理の場合のみ、SOAP/HTTP-HTTPS プロトコル・ハンドラーとともに使用されます。コネクターはまた、セットアップしたそれぞれのシステム・プロパティーをトレースします。ProxyServer プロパティーの詳細については、98 ページの『コネクター固有の構成プロパティー』を参照してください。

JNDI の初期化

コネクター固有の JNDI プロパティーでは、コネクターが使用する JNDI を指定します。コネクターは、JNDI を使用して JMS 接続ファクトリー・オブジェクトを検索します。JNDI ” LookupQueuesUsingJNDI が true に設定されている場合、コネクターは JNDI を使用して JMS キュー・オブジェクトを検索します。

SOAP/JMS (SOAP/JMS プロトコル・リスナーおよび SOAP/JMS プロトコル・ハンドラー) を使用しない場合は、JNDI プロパティーを指定する必要はありません。JNDI プロパティーを指定した場合に JNDI を初期化できないときには、コネクターは終了します。コネクターは、以下のコネクター固有の JNDI プロパティーがすべて指定されていないかぎり、JNDI を初期化しません。

- JNDIProviderURL
- InitialContextFactory
- JNDIConnectionFactoryName

注: JNDI インプリメンテーションは、コネクターでは提供されません。

プロトコル・リスナー・フレームワークの初期化

コネクターは、始動時にプロトコル・リスナー・フレームワークのインスタンスを作成し、それを初期化します。このフレームワークがコネクター固有の ProtocolListenerFramework プロパティーを読み取り、さらにコネクターがコネクター・プロパティー WorkerThreads および RequestPoolSize の値を読み取ります。ProtocolListenerFramework プロパティーが指定されていないかあるいは欠落している場合、コネクターは Web サービス・クライアントから要求を受け取ることができず、警告をログに記録します。

コネクターは、次に ProtocolListenerFramework -> ProtocolListeners プロパティーを読み取ります。ProtocolListeners プロパティーのすべての第 1 レベル・プロパティーは、プロトコル・リスナーを表します。プロトコル・リスナー・フレームワークは、それぞれのリスナーのロードと初期化を試み、それらをトレースします。永続的イベントに対応している場合には、リスナーはイベント・リカバリーを試みます。

プロトコル・ハンドラー・フレームワークの初期化

コネクターは、コネクター固有の ProtocolHandlerFramework プロパティーを読み取り、プロトコル・ハンドラー・フレームワークのインスタンスを作成してそれを初期化します。このプロパティーが欠落しているか、あるいは正しく設定されていない場合、コネクターは要求を処理することができず、警告をログに記録します。次に、コネクターはすべての ProtocolHandlerFramework ” ProtocolHandlers プロパティー (これは、プロトコル・ハンドラーに対応します) を読み取り、それらのプロパティーのロード、初期化、およびトレースを試みます。プロトコル・ハンドラーはコネクターの初期化中にロードされ、コラボレーションがサービス要求を行うときにはインスタンス化されません。プロトコル・ハンドラーはマルチスレッドに対応しています。

ロギング

コネクターは、以下の場合にログに警告を記録します。

- ProtocolListenerFramework プロパティーが指定されていない場合。コネクターは、イベント通知を行えないことを警告します。(Web サービスとして公開されたコラボレーションは、コネクターによって呼び出すことはできません。)
- ProtocolHandlerFramework プロパティーが指定されていない場合。コネクターは、(コラボレーション) 要求処理を行えないことを警告します。

トレース

トレースは、コネクターの振る舞いを綿密にたどるために使用することのできる、オプションのデバッグ機能です。デフォルトでは、トレース・メッセージは STDOUT に書き込まれるようになっています。トレース・メッセージの構成に関する詳細については、コネクター構成プロパティーを参照してください。

コネクターのトレース・レベルは次のとおりです。

レベル 0　このレベルは、コネクターのバージョンを識別するトレース・メッセージに使用されます。

レベル 1　pollForEvents メソッドが呼び出されるたびにトレースを行います。InterChange Server Express へのデリバリーのためにリスナーによって作成された TLO 名をトレースします。要求ビジネス・オブジェクト名、およびそれに対応する TLO 内の属性名をトレースします。

レベル 2　このレベルは、`gotApplEvent()` または `executeCollaboration()` から InterChange Server Express にビジネス・オブジェクトがポストされるたびにトレース・メッセージをログに記録する場合に使用してください。また、どのプロトコル・ハンドラーが要求を処理しているのかもトレースします。

レベル 3　処理中のビジネス・オブジェクトの ASI をトレースします。処理中のビジネス・オブジェクトの属性をトレースします。イベント通知中に SOAP 要求ビジネス・オブジェクトの TLO をトレースします。データ・ハンドラーによって戻されたビジネス・オブジェクトをトレースします。

レベル 4　以下のものと関連したトランスポート・ヘッダーをトレースします。

- プロトコル・リスナーによってトランスポートから検索された SOAP 要求メッセージ
- プロトコル・リスナーによってクライアントに送信された応答メッセージ

スレッドの作成、処理されるすべての ASI、および重要な機能を持つすべての入り口および出口をトレースします。

レベル 5　以下のものをトレースします。

- それぞれの重要なメソッドごとの入り口および出口
- 構成に固有のすべてのプロパティー
- それぞれのプロトコル・リスナーのロード
- プロトコル・リスナーによってトランスポートから検索された要求メッセージ
- プロトコル・リスナーによってトランスポートでクライアントに送信された応答メッセージ
- それぞれのプロトコル・ハンドラーのロード
- SOAP データ・ハンドラーによって戻されたメッセージ

- コラボレーションに送られた TLO のビジネス・オブジェクト・ダンプ
- データ・ハンドラーによって戻されたビジネス・オブジェクトのダンプ

第 5 章 SOAP データ・ハンドラー

SOAP データ・ハンドラーは、ビジネス・オブジェクトから SOAP メッセージへ、および SOAP メッセージからビジネス・オブジェクトへの変換を主な役割とするデータ変換モジュールです。 SOAP データ・ハンドラーは以下の機能を実行します。

- 要求処理
  - SOAP 要求ビジネス・オブジェクトから SOAP 要求メッセージへ
  - SOAP 応答メッセージから SOAP 応答ビジネス・オブジェクトへ
  - SOAP 障害メッセージから SOAP 障害ビジネス・オブジェクトへ
- イベント処理
  - SOAP 要求メッセージから SOAP 要求ビジネス・オブジェクトへ
  - SOAP 応答ビジネス・オブジェクトから SOAP 応答メッセージへ
  - SOAP 障害ビジネス・オブジェクトから SOAP 障害メッセージへ

この章では、SOAP データ・ハンドラーの構成方法、SOAP データ・ハンドラーによるメッセージおよびオブジェクトの処理方法、およびデータ・ハンドラーのカスタマイズ方法について説明します。

SOAP データ・ハンドラーの構成

SOAP データ・ハンドラーは、Connector for Web Services の中核コンポーネントです。コネクターは、ビジネス・オブジェクトを Web サービスに対応する SOAP メッセージに変換するため、SOAP データ・ハンドラーを呼び出します。

コラボレーションが Web サービスとして公開されている場合には、コネクターも SOAP データ・ハンドラーを呼び出します。次にデータ・ハンドラーは、リモート取引先 (または内部クライアント) から送信された SOAP メッセージをビジネス・オブジェクトに変換します。コネクターは、変換されたビジネス・オブジェクトを、Web サービス用に構成されたコラボレーションに渡します。

上記の変換の中で、データ・ハンドラー・メタオブジェクトに格納された情報が重要な役割を果たします。この情報の構成は、製品ファイルのインストール後、ただし始動前に実行してください。カスタム名ハンドラーを追加しないのであれば、デフォルトの SOAP データ・ハンドラー構成を使用して時間を節約することができます。ただし、それぞれのデータ・ハンドラーを変換するたびに特定のメタオブジェクト情報を構成しなければなりません。データ・ハンドラーのメタオブジェクトについて、以下のセクションで説明します。

メタオブジェクトの要件

メタオブジェクトは、構成情報を格納しているビジネス・オブジェクトです。コネクターは、実行時にメタオブジェクトを使用してデータ・ハンドラーを構成し、そのインスタンスを作成します。SOAP データ・ハンドラーもメタオブジェクトを使用することにより、SOAP メッセージの本文を見つけ、本文に対応するビジネス・オブジェクトおよび動詞を決定し、SOAP メッセージ内のビジネス・オブジェクトをエンコードし、またこの章で説明する他のいくつかのタスクを実行します。ここでは、これらのメタオブジェクトの要件について説明します。

メタオブジェクトの階層および用語

図 28 は、Web サービス製品用アダプターに対応するメタオブジェクト構造を示しています。メタオブジェクトは図の中で太字で示され、図の後で説明されています。

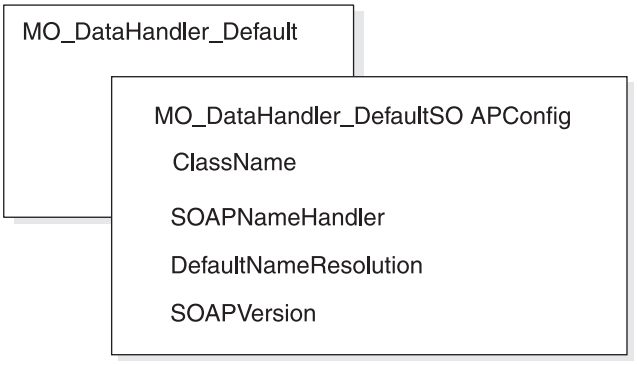


図 40. メタオブジェクトの構造

本書では、メタオブジェクトについて述べるとき次の用語を使用します。

- `MO_DataHandler_Default` コネクター・エージェントが、どのデータ・ハンドラーのインスタンスを生成するかを決定するために使用するデータ・ハンドラー・メタオブジェクトです。これは、コネクターの DataHandlerMetaObjectName プロパティーで指定されます。
- `MO_DataHandler_DefaultSOAPConfig` SOAP データ・ハンドラー固有の子データ・ハンドラー・メタオブジェクトです。
- SOAP 構成メタオブジェクト (SOAP 構成 MO) それぞれの SOAP ビジネス・オブジェクトの子として指定されるメタオブジェクトであり、ビジネス・オブジェクトから SOAP メッセージへの、またその逆方向への単一の変換を行うための構成情報を含みます。

MO_DataHandler_Default

MO_DataHandler_Default は、コネクターから呼び出されるすべてのデータ・ハンドラーに関する、トップレベル・メタオブジェクトです。これらのメタオブジェクトに設定されている MIME 型により、使用されるデータ・ハンドラーが決まります。コネクター・エージェントは、このメタオブジェクトを使用して、SOAP データ・ハンドラーのインスタンスを作成します。したがって、MO_DataHandler_Default オブジェクトには、`xml_soap` という名前で `MO_DataHandler_DefaultSOAPConfig` という型の属性が設定されている必要があります。

MO_DataHandler_Default オブジェクトは、インストール後に構成できます。MO_DataHandler_DefaultSOAPConfig タイプの xml_soap を追加する必要があります。

MO_DataHandler_DefaultSOAPConfig

コネクター・エージェントは、このメタオブジェクトを使用して、実行時に SOAP データ・ハンドラーを作成および構成します。MO_DataHandler_DefaultSOAPConfig には、string 型の属性が 2 つあり、以下を指定します。

- SOAP データ・ハンドラーのクラス名
- SOAP 名ハンドラー
- カスタム名前ハンドラーが失敗した場合のデフォルトのネーム解決
- SOAP バージョン (1.1 または 1.2)

これらの属性を、表 44 に示します。

カスタム名ハンドラー (これについては、この章の後半で説明します) をインプリメントしたい場合を除き、MO_DataHandler_DefaultSOAPConfig は、配布およびインストールされたままの状態で使用することができます。構成は不要です。

表 44. MO_DataHandler_DefaultSOAPConfig のメタオブジェクト属性

名前	型	デフォルト値	説明
ClassName	String	com.ibm.adapters .dataHandlers.xml. soap	createHandler メソッドに渡された MIME 型に基づいて、データ・ハンドラー基本クラスがクラス名を見つけるために使用する標準属性。
SOAPName Handler	String		使用する SOAP 名ハンドラーの名前。
DefaultName Resolution	String	false	カスタム名前ハンドラーが失敗した場合にデフォルトのネーム解決を使用するかどうかを指定します。
SOAPVersion	String	1.1	SOAP メッセージの読み取りおよび書き込みでデータ・ハンドラーが使用する SOAP 標準 (1.1 または 1.2) を指定します。

SOAP 構成メタオブジェクト: 各 SOAP ビジネス・オブジェクトの子

SOAP 構成 MO は、あるデータ・ハンドラー変換 (SOAP メッセージからビジネス・オブジェクトへの変換、またはビジネス・オブジェクトから SOAP メッセージへの変換) で行われるデータ・フォーマットの振る舞いを定義します。SOAP 構成 MO は SOAP ビジネス・オブジェクトの子です。これらの子 SOAP 構成 MO は、デフォルトのビジネス・オブジェクトの解決には不可欠なものです。デフォルトのビジネス・オブジェクト解決を使用する場合、すべての子 SOAP 構成 MO は、要求、応答、または障害オブジェクトのいずれの場合にも、BodyName および BodyNS のデフォルト値に関する固有の項目を含んでいなければなりません。表 45 には、これらの属性および SOAP 構成 MO のその他の属性が示されています。

表 45. SOAP 構成 MO の属性

名前	必須	説明
BodyNS	はい	SOAP 本文に使用されるネーム・スペース。
BodyName	はい	SOAP メッセージの本文の名前。SOAP 障害に対して、デフォルト値を `soap:fault` に設定します。
BOVerb	はい	SOAP 構成 MO を含むビジネス・オブジェクトの動詞。
TypeInfo	いいえ	SOAP 要素で型情報 (xsi:type) の書き込みおよび読み取りが行われるかどうかを示す、true または false 属性。デフォルト値は `false` です。
TypeCheck	いいえ	このプロパティーは、TypeInfo が `true` に設定されている場合にのみ読み取られます。指定可能な値は `none` および `strict` です。`none` を指定した場合、このビジネス・オブジェクトに SOAP メッセージを読み込むときに型の検証が省略されます。`strict` を指定した場合、データ・ハンドラーは、すべての SOAP 型名およびネーム・スペースを、ビジネス・オブジェクトのアプリケーション固有情報と対比して、厳密に検証します。デフォルト値は `none` です。
Style	いいえ	このプロパティーは SOAP メッセージ・スタイルを指定するものであり、BodyName や BodyNS などの他の属性と密接に関連しています。この属性として指定可能な値は、`rpc` および `document` です。デフォルト値は `rpc` です。
Use	いいえ	このプロパティーは SOAP メッセージの使用を指定するものであり、ビジネス・オブジェクトから SOAP 本文を構成する方法に影響を及ぼします。指定可能な値は `literal` および `encoded` です。デフォルト値は `literal` です。

図 41 は、SOAP ビジネス・オブジェクトと SOAP 構成 MO の関係を示しています。

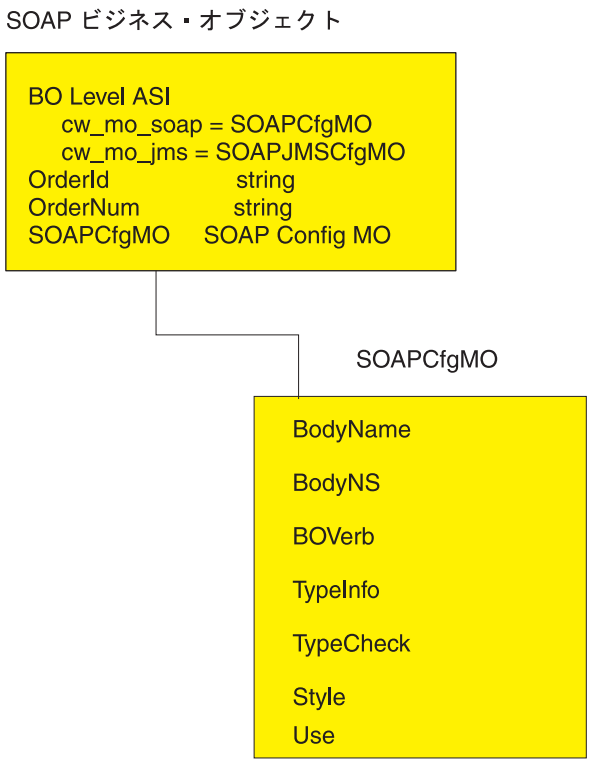


図 41. SOAP 構成のメタオブジェクト

図 41 は、SOAP 応答ビジネス・オブジェクトとその子ビジネス・オブジェクトを示したものです。子ビジネス・オブジェクト SOAPCfgMO は、ビジネス・オブジェクト応答から SOAP 応答メッセージへの変換を行う際の SOAP データ・ハンドラーの振る舞いを指定する、SOAP 構成 MO です。子 SOAP 構成 MO を指定する属性では、`cw_mo_soap` で始まる名前と値のペアを使用しなければなりません。

規約により、データ・ハンドラーは、`cw_mo_` で始まるビジネス・オブジェクト・レベルのアプリケーション固有情報を読み取るときには、名前と値のペアで指定された子オブジェクトには変換メタオブジェクト情報が含まれているものと認識するため、変換するメッセージの本文の内容としてこの子を含みません。この例で、名前と値のペア `cw_mo_jms` と `cw_mo_soap` で指定されている子オブジェクトは、メタオブジェクトとして認識され、SOAP 応答メッセージには書き込まれません。また、SOAP データ・ハンドラーは、`cw_mo_soap` を除き、`cw_mo_` で始まるすべてのビジネス・オブジェクト・レベルのアプリケーション固有の情報を無視します。したがって、SOAP データ・ハンドラーは、`cw_mo_tpi` のようなアプリケーション固有の情報は無視します。ただし、SOAP データ・ハンドラーは、ビジネス・オブジェクトから SOAP メッセージへの SOAP 応答変換を実行するために、`cw_mo_soap` で指定された SOAP 構成 MO については、読み取って使用します。

すべての SOAP ビジネス・オブジェクトには子 SOAP 構成 MO が必要であり、これらの SOAP 構成 MO は、ビジネス・オブジェクト・レベルでアプリケーション固有情報として指定されていなければなりません。この作業の大半は、自動的に行われます。SOAP メッセージ用のビジネス・オブジェクトを生成するために WSDL ODA を使用すると、SOAP 構成 MO が自動的に生成されます。

SOAP メッセージに対する Style と Use の影響

SOAP 構成 MO オプション・プロパティー Style および Use は、SOAP メッセージの作成方法に影響を及ぼします。Style に指定可能な値は rpc および document です。Use に指定可能な値は literal および encoded です。以下のセクションでは、Style および Use の組み合わせが SOAP メッセージの作成に及ぼす影響について説明します。

rpc/literal: Style プロパティーを rpc に設定し、Use プロパティーを literal に設定した場合、SOAP メッセージの Body Name および Body Namespace は、それぞれ SOAP 構成 MO の BodyName プロパティーおよび BodyNS プロパティーから読み取られます。

Body Name および Body Namespace がそれぞれ getOrderStatus および OrderStatusNS に解決された rpc/literal スタイル・メッセージの例を以下に示します。

```
<?xml version='1.0' encoding='UTF-8'?>
<SOAP-ENV:Envelope xmlns:SOAP-ENV=îhttp://schemas.xmlsoap.org/soap/envelope/ì
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
   <SOAP-ENV:Body>
      <ns1:getOrderStatus xmlns:ns1="http://www.ibm.com/">
         <Part1>
            <ns2:Elem1 xmlns:ns2="http://www.ibm.com/elem1">
               <Child1>1</Child1>
               <Child2>2</Child2>
            </ns2:Elem1>
            <ns3:Elem1 xmlns:ns3="http://www.ibm.com/elem1">
               <Child1>3</Child1>
               <Child2>4</Child2>
            </ns2:Elem1>
            <Elem2>10</Elem2>
         </Part1>
      </ns1:getOrderStatus>
   </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

図 42 は、この rpc/literal メッセージに対応するビジネス・オブジェクトを示しています。

Name	Type	Key	Card	Default	App Spec Info
⊟ Part1	SOAP_Part1Type	☑	1		
⊟ Elem1	SOAP_MaxType	☑	N		maxoccurs=5;elem_ns=http://www.ibm.com/elem1
Child1	String	☑			
Child2	String	☐			
ObjectEventId	String				
Elem2	String	☐			
ObjectEventId	String				
⊟ SOAPConfigMO	SOAP_Req_Cfg_MO	☐	1		
BodyName	String	☑		getOrderStatus	
BodyNS	String	☐		http://www.ibm.com	
BOVerb	String	☐		Retrieve	
TypeInfo	String	☐		false	
TypeCheck	String	☐		none	
Style	String	☐		rpc	
Use	String	☐		literal	

図 42. rpc/literal SOAP 構成 MO

注: 対応する SOAP メッセージが作成されるように、これらのプロパティーおよびビジネス・オブジェクト属性を適切に構成してください。

rpc/encoded: Style プロパティーを rpc に設定し、Use プロパティーを encoded に設定した場合、SOAP メッセージの Body Name および Body Namespace は、それぞれ子構成 MO の BodyName プロパティーおよび BodyNS プロパティーから読み取られます。また、
SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" 属性も Body タグに追加されます。

Body Name および Body Namespace がそれぞれ getOrderStatus および OrderStatusNS に解決された rpc/encoded メッセージの例を以下に示します。

```
<?xml version='1.0' encoding='UTF-8'?>
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
   <SOAP-ENV:Body SOAP-ENV:encodingStyle=
    "http://schemas.xmlsoap.org/soap/encoding/">
      <ns1:getOrderStatus xmlns:ns1="http://www.ibm.com/">
         <Part1 xsi:type="ns1:SOAP_Part1Type">
            <ns2:Elem1 SOAP-ENC:arrayType="ns2:SOAP_MaxType[2]"
             xsi:type="SOAP-ENC:Array" xmlns:ns2="http://www.ibm.com/elem1">
               <item>
                  <Child1 xsi:type="xsd:string">1</Child1>
            <Child2 xsi:type="xsd:string">2</Child2>
               </item>
               <item>
                  <Child1 xsi:type="xsd:string">3</Child1>
            <Child2 xsi:type="xsd:string">4</Child2>
               </item>
            </ns2:Elem1>
            <Elem2 xsi:type="xsd:string">10</Elem2>
         </Part1>
      </ns1:getOrderStatus>
   </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

図 43 に、この rpc/encoded メッセージに対応するビジネス・オブジェクトを示します。

Name	Type	Key	Card	Default	App Spec Info
⊟ Part1	SOAP_Part1Type	☑	1		
⊟ Elem1	SOAP_MaxType	☑	N		elem_ns=http://www.ibm.com/elem1
Child1	String	☑			
Child2	String	☐			
ObjectEventId	String				
Elem2	String	☐			
ObjectEventId	String				
⊟ SOAPConfigMO	SOAP_Req_Cfg_MO	☐	1		
BodyName	String	☑		getOrderStatus	
BodyNS	String	☐		http://www.ibm.com	
BOVerb	String	☐		Retrieve	
TypeInfo	String	☐		true	
TypeCheck	String	☐		none	
Style	String	☐		rpc	
Use	String	☐		encoded	

図 43. rpc/encoded SOAP 構成 MO

document/literal: Style プロパティーを document に設定し、Use プロパティーを literal に設定した場合、Body Name タグを囲むものは一切存在しません。次に示すのは、上のビジネス・オブジェクトに基づいた、document スタイルの SOAP メッセージの例です。

```
<?xml version='1.0' encoding='UTF-8'?>
<SOAP-ENV:Envelope xmlns:SOAP-ENV=
"http://schemas.xmlsoap.org/soap/envelope/"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
   <SOAP-ENV:Body>
      <ns1:Elem1 xmlns:ns1="http://www.ibm.com/elem1">
         <Child1>1</Child1>
         <Child2>2</Child2>
      </ns1:Elem1>
      <ns2:Elem1 xmlns:ns2="http://www.ibm.com/elem1">
         <Child1>3</Child1>
         <Child2>4</Child2>
      </ns2:Elem1>
   </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

図 44 に、この document/literal メッセージに対応するビジネス・オブジェクトを示します。

Name	Type	Key	Card	Default	App Spec Info
⊟ Elem1	SOAP_Elem1	☑	1		maxoccurs=3;elem_ns=http://www.ibm.com/elem1
Child1	String	☑			
Child2	String	☐			
ObjectEventId	String				
⊟ SOAPConfigMO	SOAP_Req_Cfg_MO	☐	1		
BodyName	String	☑		getOrderStatus	
BodyNS	String	☐		http://www.ibm.com	
BOVerb	String	☐		Retrieve	
TypeInfo	String	☐		false	
TypeCheck	String	☐		none	
Style	String	☐		document	
Use	String	☐		literal	

図 44. document/literal SOAP 構成 MO

この XML コードでは encodingStyle 属性が設定されていません。

document/encoded: この Style/Use の組み合わせはサポートされません。データ・ハンドラーは、Style を `document` に設定し、Use を `encoded` に設定した SOAP 構成 MO を検出すると失敗します。

SOAP データ・ハンドラーの処理

SOAP データ・ハンドラーは、SOAP メッセージとビジネス・オブジェクトの間の変換を次の方法で実行します。

- **SOAP メッセージからビジネス・オブジェクトへの処理**
  - 要求メッセージから SOAP 要求ビジネス・オブジェクトへのデータ処理は、イベント処理の際に、Web サービスとして公開されているコラボレーションを Web サービス・クライアントが呼び出すときに行われます。
  - 応答メッセージから SOAP 応答ビジネス・オブジェクトへのデータ処理は、要求処理の際に、Web サービスを呼び出したコラボレーションに、Web サービスが SOAP 応答メッセージを戻すときに行われます。あるいは、この段階で障害メッセージから SOAP ビジネス・オブジェクトへのデータ処理が行われる可能性もあります。

  この処理について詳しくは、このセクションで後述される 134 ページの『SOAP 本文メッセージからビジネス・オブジェクトへの処理』を参照してください。
- **ビジネス・オブジェクトから SOAP メッセージへの処理**
  - ビジネス・オブジェクトから SOAP 応答メッセージへのデータ処理は、イベント処理の際に、Web サービスとして公開されているコラボレーションによって応答ビジネス・オブジェクトが戻されたときに行われます。あるいは、この段階で障害ビジネス・オブジェクトから SOAP 障害メッセージへのデータ処理が行われる可能性もあります。
  - ビジネス・オブジェクトから SOAP 要求メッセージへのデータ処理は、要求処理の際に、ビジネス・オブジェクトを SOAP 要求メッセージに変換するために、コラボレーションがコネクターに対してサービス呼び出しを行ったときに行われます。

この処理について詳しくは、このセクションで後述される 136 ページの『ビジネス・オブジェクトから SOAP メッセージ本文への処理』を参照してください。

SOAP 本文メッセージからビジネス・オブジェクトへの処理

このセクションでは、SOAP 本文メッセージからビジネス・オブジェクトへの変換について、段階を追って説明します。

1. SOAP データ・ハンドラーが SOAP メッセージを受け取ります。
2. データ・ハンドラーは、Apache SOAP API を使用して、SOAP メッセージを解析します。
3. データ・ハンドラーは、SOAP メッセージのコンポーネントであるエンベロープ、ヘッダー、および本文を抽出します。
4. **ヘッダーの処理** 詳しくは、135 ページの『SOAP ヘッダー・メッセージからビジネス・オブジェクトへの処理』を参照してください。
5. **本文の処理** データ・ハンドラーは、SOAP 本文が障害に関するものか、データの通知かを確認するため、SOAP 本文の最初の要素を読み取ります。本文の内容が障害でない場合には、データ・ハンドラーは次の処理を実行します。
   a. どのビジネス・オブジェクトが変換に使用されるのかを判別するために、ビジネス・オブジェクト解決を実行します。カスタム名前ハンドラーが構成されている場合には、下記のデフォルトのビジネス・オブジェクト解決は適用されません。プラグ可能な名前ハンドラーの指定については、161 ページの『プラグ可能な名前ハンドラーの指定』を参照してください。
   b. データ・ハンドラーは、変換に使用される SOAP 構成 MO (データ・ハンドラーが作成している SOAP ビジネス・オブジェクトの子) も解決します。SOAP 構成 MO のインスタンスが存在しない場合、データ・ハンドラーはインスタンスを作成し、そのデフォルト値を読み取ります。データ・ハンドラーは、ConfigMO の属性値からビジネス・オブジェクトの動詞を読み取ります。データ・ハンドラーは SOAP ビジネス・オブジェクトのインスタンスを生成し、該当の動詞を設定します。このビジネス・オブジェクトは、データ・ハンドラーが SOAP メッセージの書き込みを試みる対象となります。
   c. データ・ハンドラーは、一度に 1 要素ずつ SOAP メッセージの構文解析を続行します。rpc の場合、データ・ハンドラーは、先頭の要素を親と判断します。
   d. データ・ハンドラーは、ビジネス・オブジェクト (またはそのアプリケーション固有情報、詳しくは 142 ページの『ビジネス・オブジェクトから SOAP メッセージへの変換における ASI』を参照) の属性の名前が、子要素と同じであるものと想定します。その属性がビジネス・オブジェクトから見つからない場合、データ・ハンドラーは例外をスローします。子要素は、単純タイプの場合と複合タイプの場合があります。複合要素とは、子要素を持つ要素のことです。
   e. **単純要素** 子要素が単純要素の場合、デフォルトでは、データ・ハンドラーは、ビジネス・オブジェクト属性が単純要素のビジネス・オブジェクト属性と同じ名前 (または ASI) であることを想定します。データ・ハンドラーは単純要素の値を読み取り、ビジネス・オブジェクトにその値を設定します。
   f. **複合要素** 子要素が複合タイプの要素である場合、データ・ハンドラーは、ビジネス・オブジェクトの属性の名前 (または ASI) および型が子ビジネス・オ

ブジェクトのものと同じであるものと想定します。この属性は、複合 SOAP 要素が存在するか、それとも SOAP 配列が存在するかによって、単一カーディナリティーの場合と複数カーディナリティーの場合があります。次に、データ・ハンドラーは子ビジネス・オブジェクトのインスタンスを作成し (デフォルトでは、属性の型に応じて子ビジネス・オブジェクトの名前が決まります)、この複合要素のすべての子要素を読み取って、子ビジネス・オブジェクトのそれらの値を設定します。データ・ハンドラーは、親ビジネス・オブジェクト属性のカーディナリティーを検査した後で、この子ビジネス・オブジェクトを親ビジネス・オブジェクト属性に組み込みます。属性がカーディナリティー n の場合、データ・ハンドラーは、このビジネス・オブジェクトをコンテナーに追加します。複合要素の子要素は、単純要素の場合と複合要素の場合とがあります。これらも、同じように扱われます。単純要素である場合には、データ・ハンドラーは値を子ビジネス・オブジェクトで設定し、複合要素である場合には、データ・ハンドラーは子ビジネス・オブジェクトのインスタンスを作成します。

6. **障害の処理** データ・ハンドラーは、SOAP 本文の最初の要素の名前を読み取り、SOAP 本文が障害に関するものかどうかを判別します。最初の要素の名前が `Fault` の場合、データ・ハンドラーはこれが障害メッセージであると認識します。この障害メッセージをどのビジネス・オブジェクトに変換するのかを決定するために、障害ビジネス・オブジェクトの解決が行われます。その後データ・ハンドラーは、本文の処理の場合と同じ処理を行います。データ・ハンドラーは、子ビジネス・オブジェクトで指定されたビジネス・オブジェクトが以下の属性を備えているものと想定します。
   a. faultcode: 必須。String 属性
   b. faultstring: 必須。String 属性
   c. faultactor: 任意。String 属性
   d. detail: 任意。子ビジネス・オブジェクト
7. なんらかの理由で障害処理が失敗した場合、スローされる例外には、SOAP 障害メッセージの faultcode、faultstring、および faultactor 要素から得られたテキストが含まれます。

注: 障害メッセージに関する SOAP 仕様によると、faultcode、faultstring、および faultactor は単純要素であり、detail は複合要素 (子要素を含む要素) です。また、faultcode、faultstring、faultactor、および detail は SOAP エンベロープ・ネーム・スペースに属しますが、detail 子要素はユーザー定義のネーム・スペースに属することもあります。

SOAP ヘッダー・メッセージからビジネス・オブジェクトへの処理

ここでは、データ・ハンドラーによる SOAP メッセージのヘッダーからビジネス・オブジェクトへの変換の方法について説明します。

1. SOAP データ・ハンドラーは SOAP メッセージの本文を処理します。本文の処理により SOAP ビジネス・オブジェクトが作成されます。
2. SOAP メッセージに SOAP ヘッダー要素があると、SOAP データ・ハンドラーは、本文の処理の結果生成されたビジネス・オブジェクト内に SOAP ヘッダー属性が存在すると想定します。`SOAPHeader` 属性は、ビジネス・オブジェクトの

子属性であり、soap_location=SOAPHeader をそのアプリケーション固有の情報として持っています。このような属性が存在しない場合、SOAP データ・ハンドラーはエラーをスローします。SOAPHeader 属性は、SOAP ヘッダー・コンテナー・ビジネス・オブジェクトの型であることが必要です。SOAP データ・ハンドラーは、ステップ 1 で取得した SOAP ビジネス・オブジェクト内に、この属性のインスタンスを作成します。

3. SOAP-Env:Header 要素の直接の子それぞれに対して、次の処理が実行されます。
   a. データ・ハンドラーは、SOAP ヘッダー・コンテナー・ビジネス・オブジェクト内に子属性が存在すると想定します。この属性の名前は、ヘッダー要素の名前と同じであること、および SOAP ヘッダー子ビジネス・オブジェクトに準拠していることが必要です。このような属性を見つけることができない場合、データ・ハンドラーはエラーをスローします。さらに、ネーム・スペースこの要素のネーム・スペースは、この属性の elem_ns アプリケーション固有情報に指定されたネーム・スペースと同じです。同じでない場合、データ・ハンドラーはエラーをスローします。
   b. データ・ハンドラーは SOAP ヘッダー子ビジネス・オブジェクトのインスタンスを作成し、ステップ 2 で作成された SOAP ヘッダー・コンテナー・ビジネス・オブジェクトのインスタンスにこれを格納します。
   c. このヘッダー要素に actor 属性がある場合、データ・ハンドラーは、actor 属性が上記で作成された子ビジネス・オブジェクト内に存在すると想定します。actor 属性が見つからない場合、データ・ハンドラーはエラーをスローします。

      注: actor 属性を追加する必要がある場合は、146 ページの『SOAP 属性の指定』を参照してください。
   d. このヘッダー要素に mustUnderstand 属性がある場合、データ・ハンドラーは、上記で作成された子ビジネス・オブジェクト内に mustUnderstand 属性が存在すると想定します。mustUnderstand 属性が見つからないと、データ・ハンドラーはエラーをスローします。

      注: mustUnderstand 属性を追加する必要がある場合は、146 ページの『SOAP 属性の指定』を参照してください。
   e. このヘッダー要素の各子要素について、データ・ハンドラーは、子ビジネス・オブジェクト内に同じ名前の属性が存在すると想定します。これらの要素は、SOAP-Env:Body 要素の子要素と同じ方法で処理されます。

ビジネス・オブジェクトから SOAP メッセージ本文への処理

ビジネス・オブジェクトから SOAP メッセージ本文への変換は、次のような手順を経て実行されます。アプリケーション固有情報が使用される特殊なケースについては、142 ページの『ビジネス・オブジェクトから SOAP メッセージへの変換における ASI』を参照してください。

1. SOAP データ・ハンドラーは、変換対象の SOAP ビジネス・オブジェクトに対応する SOAP 構成 MO を探します。
2. データ・ハンドラーは、SOAP メッセージのエンベロープとヘッダーを作成します。

3. データ・ハンドラーは SOAP 構成 MO を解決します。SOAP 構成 MO のインスタンスが存在しない場合、データ・ハンドラーはインスタンスを作成し、デフォルト値を読み取ります。デフォルトでは、データ・ハンドラーは、SOAP 構成 MO の BodyName 属性の値を読み取って、処理しているビジネス・オブジェクトが障害ビジネス・オブジェクトであるかどうかを判別します。この値が `soap:fault` に設定されている場合には、そのビジネス・オブジェクトは SOAP 障害ビジネス・オブジェクトであるとみなされます。それが障害ビジネス・オブジェクトでない場合、データ・ハンドラーは、下記の『本文の作成』で述べられた処理を行い、それ以外の場合には『障害メッセージの作成』で述べられた処理を行います。
4. **本文の作成** データ・ハンドラーがビジネス・オブジェクトから SOAP メッセージの本文を作成するために行う処理について、以下のステップで詳しく説明します。
   - データ・ハンドラーは、SOAP 構成 MO 属性から BodyName および BodyNS を入手してから、SOAP メッセージ本文の最初の (親) 要素を作成します。デフォルトでは、最初の要素の名前は BodyName の値になります。本書では、これを本文要素とも呼びます。本文要素のネーム・スペースは、デフォルトでは BodyNS について決められた値となります。SOAP 構成 MO の Style 属性が `document` に設定されている場合、このステップ (最初の本文要素の作成) はスキップされます。
   - データ・ハンドラーはビジネス・オブジェクトの属性を読み取り、それら属性を型ごとに処理します。各型の属性の処理について以下で説明します。
     - **単純属性** 属性が単純タイプの場合、データ・ハンドラーが本文要素から子要素を作成し、名前は属性と同じになります (特別なアプリケーション固有情報で別の値が指定されている場合を除きます)。データ・ハンドラーは、この要素の値を、ビジネス・オブジェクト属性の値に設定します。
     - **カーディナリティー 1 の子ビジネス・オブジェクト属性**

       属性が単一カーディナリティーの子ビジネス・オブジェクトの場合、データ・ハンドラーは本文要素の子要素を作成します。これは、子ビジネス・オブジェクト要素と呼ばれます。作成された子要素の名前は、属性の名前と同じになります (ただし、特別な ASI プロパティーで別の名前が指定されている場合を除きます)。次にデータ・ハンドラーは、子ビジネス・オブジェクトの属性を全探索し、着信ビジネス・オブジェクトの属性を処理する場合と同じ方法で、それらの属性の子要素を作成します。ただし、それらの子要素は、本文要素の子ではなく、子ビジネス・オブジェクト要素の子になります。
     - **カーディナリティー n の子ビジネス・オブジェクト属性** 属性がカーディナリティー n の子ビジネス・オブジェクトである場合、データ・ハンドラーは SOAP 配列を作成します。それぞれの属性は、単一カーディナリティーの子ビジネス・オブジェクトと同じ方法で処理されます。
5. **障害メッセージの作成** 以下のセクションでは、データ・ハンドラーが障害メッセージを作成するプロセスについて、順に説明します。
   - データ・ハンドラーは、ビジネス・オブジェクト内に以下の属性が存在するものと想定します。
     - faultcode: 必須。String 属性

– faultstring: 必須。String 属性
– faultactor: 任意。String 属性
– detail: 任意。子ビジネス・オブジェクト属性

必須属性が欠落している場合には、データ・ハンドラーはエラー終了します。

- データ・ハンドラーは faultcode の要素を作成します。そして、ビジネス・オブジェクトの faultcode 属性により指定された値を設定します。
- データ・ハンドラーは faultstring の要素を作成します。そして、ビジネス・オブジェクトの faultstring 属性により指定された値を設定します。
- データ・ハンドラーは faultactor を作成します。そして、ビジネス・オブジェクトの faultactor 属性により指定された値を設定します。
- ビジネス・オブジェクトに detail 属性が存在する場合、この属性は、子ビジネス・オブジェクトと同じ型になっている必要があります。そのようになっていない場合、データ・ハンドラーはエラー終了します。データ・ハンドラーは各 detail ビジネス・オブジェクトの属性を、上記の『**本文の作成**』で詳しく説明したように処理します。

6. **CxIgnore の処理** データ・ハンドラーは、属性の値が CxIgnore に設定されていることを検出した場合には、この属性に対応する要素を作成しません。
7. **CxBlank の処理** データ・ハンドラーは、属性の値が CxBlank に設定されていることを判別した場合には、この属性に対応する要素を作成しますが、その値を設定しません。

ビジネス・オブジェクトから SOAP メッセージ・ヘッダーへの処理

このセクションでは SOAP ヘッダー属性の処理についてのみ説明します。他のすべての属性は、136 ページの『ビジネス・オブジェクトから SOAP メッセージ本文への処理』で説明されているとおりに処理されます。

1. SOAP データ・ハンドラーは、ビジネス・オブジェクトから SOAPHeader 属性を取得します。この属性には、そのアプリケーション固有情報として、soap_location=SOAPHeader があります。SOAP データ・ハンドラーは、この属性の値がヌル以外のときに限って、SOAP-Env:Header 要素を作成します。ビジネス・オブジェクトに複数の SOAPHeader 属性が格納されている場合、最初の属性のみが処理され、残りは本文の一部として扱われます。
2. SOAP データ・ハンドラーは、SOAPHeader 属性が、SOAP Header Container ビジネス・オブジェクトを表す単一カーディナリティーの子であると想定します。データ・ハンドラーは、SOAP Header Container ビジネス・オブジェクトの属性の中で、SOAP Header Child ビジネス・オブジェクトの型の属性を処理します。
3. SOAP Header Container ビジネス・オブジェクトの各属性に対して、データ・ハンドラーは次の処理を実行します。
   a. カーディナリティーをチェックし、この属性がカーディナリティー 1 の子オブジェクトでもカーディナリティー n の子オブジェクトでもなければ、その属性を無視します。
   b. 値をチェックし、この属性の値が NULL であれば、その属性を無視します。
   c. 属性がカーディナリティー 1 または n の子オブジェクトの場合、SOAP データ・ハンドラーは、ステップ 1 で作成された SOAP-Env:Header 要素の直

接の子であるヘッダー要素を作成します。このヘッダー要素の名前は属性の名前と同じです。この要素のネーム・スペースは、この属性の elem_ns アプリケーション固有情報により指定されます。

d. 属性が SOAP Header Child ビジネス・オブジェクトであれば、このビジネス・オブジェクトのすべての属性が処理されます。この属性は actor および mustUnderstand 属性を持つ場合があります。

   注: mustUnderstand 属性または actor 属性を追加する必要がある場合は、146 ページの『SOAP 属性の指定』を参照してください。

e. SOAP Header Child ビジネス・オブジェクトにヌル以外の actor 属性が存在する場合、データ・ハンドラーは、ステップ c で作成されたヘッダー要素内に actor 属性を作成します。

f. SOAP Header Child ビジネス・オブジェクトにヌル以外の mustUnderstand 属性が存在する場合、データ・ハンドラーは、ステップ c で作成されたヘッダー要素内に mustUnderstand 属性を作成します。

g. SOAP Header Child ビジネス・オブジェクトのヌル以外のその他すべての属性は、このヘッダー要素の子要素となります。これらの要素は、SOAP-Env:Body 要素の子要素と同じ方法で構成されます。

ヘッダー障害の処理

SOAP の仕様では、ヘッダーに関するエラーはヘッダーで戻さなければならないと規定されています。このようなヘッダーは、SOAP 障害メッセージで戻します。メッセージ・ヘッダーが要求および応答のビジネス・オブジェクトの SOAPHeader 属性で指定されるように、障害ヘッダーは、障害ビジネス・オブジェクトの SOAPHeader 属性で指定されます。

要求ビジネス・オブジェクトまたは応答ビジネス・オブジェクトに付くどのヘッダーも、エラーの発生を招くことがあります。このようなエラーは、障害メッセージのヘッダーに報告されます。

WSDL 文書には SOAP バインディング・ヘッダー障害要素があり、これを使用することにより、障害ヘッダーを指定することができます。詳しくは、第 1 章に表で示した SOAP 仕様および WSDL 仕様を参照してください。

headerfault のアプリケーション固有情報により、各ヘッダーの障害を指定することができます。SOAP Header Container ビジネス・オブジェクトの各属性について、headerfault アプリケーション固有情報を指定できます。障害ビジネス・オブジェクトに対する SOAP Header Container ビジネス・オブジェクト内の属性リストは次のとおりです。

```
headerfault=attr1, attr2, attr3...
```

WSDL 構成ウィザードにより、要求または応答オブジェクトの SOAP Header Child ビジネス・オブジェクト内に headerfault アプリケーション固有情報が検出されると、このユーティリティーは、これらのヘッダーのために生成された WSDL の中に headerfault 要素を作成します。WSDL では、要求 (入力) ヘッダーおよび応答 (出力) ヘッダーのそれぞれに対して、複数のヘッダー障害を指定することができます。したがって、アプリケーション固有情報の値は、コンマ区切りの属性リストです。

アプリケーション固有情報機能の使用

オブジェクト・レベルおよび属性レベルのアプリケーション固有情報 (ASI) を指定して、SOAP データ・ハンドラー機能を拡張および強化することができます。表 46 に、これらの属性を示します。各属性については、以下のセクションで説明します。表の項目はすべて、特に明記されていない限り、属性レベルの ASI です。

表 46. SOAP オブジェクト ASI の要約

ASI	指定可能な値	説明
soap_location	SOAPHeader	このビジネス・オブジェクト属性をヘッダー属性として指定します。
headerfault	String	障害ビジネス・オブジェクト内の対応する SOAP ヘッダーのビジネス・オブジェクト属性名を指定します。
elem_name	String	このビジネス・オブジェクト属性に対応する SOAP 要素の名前を指定します。
elem_ns	String	このビジネス・オブジェクト属性に対応する SOAP 要素のネーム・スペースを指定します。
type_name	String	このビジネス・オブジェクト属性に対応する SOAP 要素の型を指定します。
type_ns	String	このビジネス・オブジェクト属性に対応する要素の型ネーム・スペースを指定します。
xsdtype	true	このビジネス・オブジェクト属性に対応する要素の型ネーム・スペースとして xsd を指定し、古いバージョンの xsd (1999、2000 など) を最新バージョンの xsd (2001 など) でオーバーライドします。
attr_name	String	このビジネス・オブジェクト属性に対応する SOAP 属性の名前を指定します。
attr_ns	String	このビジネス・オブジェクト属性に対応する SOAP 属性のネーム・スペースを指定します。

表 46. SOAP オブジェクト ASI の要約 (続き)

ASI	指定可能な値	説明
arrayof	String	単純タイプ配列項目のプレースホルダーとして使用する必要のある、n カーディナリティーの子ビジネス・オブジェクト属性の名前を指定します。
dh_mimetype	String	この複合タイプの属性を変換するために使用される、データ・ハンドラーの mimeType を指定します。
cw_mo_*	String	このビジネス・オブジェクト・レベルの ASI では、SOAP データ・ハンドラーによって内容ではなくメタデータとして解釈される、子構成 MO の名前を指定します。メタデータとして処理される子構成 MO を指定する値は、cw_mo_soap のみです。それ以外のすべての cw_mo_* は、別のコンポーネントを表すため、SOAP データ・ハンドラーの処理対象から除外されます。他の cw_mo* はすべて無視されます。
cw_mo_soap	String	このビジネス・オブジェクト・レベルの ASI では、このビジネス・オブジェクトを変換するときに使用する必要のある、子構成 MO 属性の名前を指定します。
cw_mo_jms	String	このビジネス・オブジェクト・レベルの ASI では、使用する JMS プロトコル構成 MO の名前を指定します。
cw_mo_http	String	このビジネス・オブジェクト・レベルの ASI では、使用する HTTP プロトコル構成 MO の名前を指定します。
wrapper	true	このビジネス・オブジェクト内のラッパー・オブジェクトの属性名を指定します。ラッパー・オブジェクトは特定のスキーマ標識で使用され、直列化してはなりません。

表 46. SOAP オブジェクト ASI の要約 (続き)

ASI	指定可能な値	説明
maxoccurs	Integer	このビジネス・オブジェクト属性について出現可能な、最大オカレンス件数を指定します。maxoccurs の値によって、ビジネス・オブジェクトにラッパーを含められる場合と、含められない場合があります。
minoccurs	Integer	このビジネス・オブジェクト属性について可能な、最小オカレンス件数を指定します。minoccurs の値によって、ビジネス・オブジェクトにラッパーを含められる場合と、含められない場合があります。
all	String	スキーマの all 標識を表す子属性を指定します。
choice	String	スキーマの choice 標識を表す子属性を指定します。

ビジネス・オブジェクトから SOAP メッセージへの変換における ASI

SOAP データ・ハンドラーは、SOAP メッセージの構成方法を決定するために、ビジネス・オブジェクトの ASI を使用します。別の指示が示されていないかぎり、以下のセクションで述べるすべての ASI は属性レベルの ASI を表し、ストリング・ベースでの比較はすべて、大/小文字を区別しないで行われます。

elem_name および elem_ns の処理

このセクションで示す例では、属性名が OrderId であって、SOAP 要素ネーム・スペースの接頭部が ns0 であると想定されています。

1. elem_name も elem_ns も指定されていない場合、elem_name のデフォルト値としては属性名が使用され、elem_ns のデフォルト値としては要素の親のネーム・スペースが使用されます。ASI は指定されません。

   ```
   <OrderId>1</OrderId>
   ```

2. elem_name が指定され、elem_ns が指定されていない場合、elem_name は ASI の elem_name 値に設定され、elem_ns についてはデフォルト値として SOAP 本文のネーム・スペースが使用されます。ASI は以下のようになります。

   ```
   elem_name=CustOrderId
   <CustOrderId>2</CustOrderId>
   ```

3. elem_ns が指定され、elem_name が指定されていない場合、elem_name についてはデフォルト値として属性名が使用され、elem_ns は ASI の elem_ns 値に設定されます。xmlns 属性が明示的に書き込まれるのは、この要素のスコープ内における他のどの個所にも要素ネーム・スペースが見付からない場合のみです。要素ネーム・スペースが検出された場合には、すでに定義されているネーム・スペー

スの接頭部が使用されます。それ以外の場合 (要素ネーム・スペースが見付からない場合)、elem_ns のための固有の接頭部が生成されます。次の例では、スコープ内で接頭部がすでに定義されているものと想定しています (ns1 は、この要素のスコープ内ですでに定義されているネーム・スペースに対応する接頭部を表します)。ASI は以下のようになります。

```
elem_ns= http://www.w3.org/2001/XMLSchema
<ns1:OrderId>3</ns1:OrderId>
```

次の例では、接頭部が見付からない場合が想定されています (ns2 は固有の接頭部を表します)。ASI は以下のようになります。

```
elem_ns=CustOrderIdNamespace
<ns2:OrderId xmlns:ns2="CustOrderIdNamespace">3</ns2:OrderId>
```

4. elem_name と elem_ns の両方が指定されている場合、elem_name および elem_ns は、ASI の値に設定されます。すでに定義されているネーム・スペースに関しては、上記 3 の場合と同じ検査が行われます。3 の場合と同様に、ネーム・スペースがまだ定義されていない場合には、elem_ns のために固有の接頭部が生成されます。ASI は以下のようになります。

```
elem_name=CustOrderId;elem_ns=CustOrderIdNamespace
<ns2:CustOrderId xmlns:ns2="CustOrderIdNamespace">1</ns2:OrderId>
```

単純属性の場合の type_name および type_ns の処理

このセクションの例では、属性名は `OrderId`、SOAP 要素ネーム・スペースの接頭部は `ns0`、そして属性タイプは `String` になっています。

注: type_name および type_ns の処理は、構成 MO の属性 TypeInfo が `true` になっている場合にのみ行われます。

1. type_name も type_ns も指定されていない場合、type_name のデフォルト値として単純タイプが使用され、type_ns のデフォルト値として、xml スキーマで定義されたネーム・スペース (xsd) が使用されます。ASI は指定されません。

```
<OrderId xsi:type="xsd:string">1</OrderId>
```

2. type_name が指定され、type_ns が指定されていない場合、type_name は ASI の type_name 値に設定され、type_ns についてはデフォルト値として要素のネーム・スペースが使用されます。ASI は以下のようになります。

```
type_name=CustString
<OrderId xsi:type="ns0:CustString">2</OrderId>
```

3. type_ns が指定され、type_name が指定されていない場合、type_ns のデフォルト値として単純タイプ名が使用され、type_name は ASI の type_ns 値に設定されます。この接頭部は、elem_ns の作成に準じた方法で処理されます。要素スコープ内にその型ネーム・スペースがすでに存在している場合を除き、型ネーム・スペースのために固有の接頭部が生成されます。ASI は以下のようになります。

```
type_ns=CustStringNamespace
<OrderId xmlns:ns2="CustStringNamespace" xsi:type=
"ns2:String">3</OrderId>
```

4. type_name と type_ns の両方が指定されている場合、これらは、割り当てられた ASI 値に設定されます。型ネーム・スペースのために固有の接頭部が生成されます。ASI は以下のようになります。

```
type_name=CustString;type_ns=CustStringNamespace
<OrderId xmlns:ns2="CustStringNamespace" xsi:type=
"ns2:CustString">1</OrderId>
```

単一カーディナリティー属性の場合の type_name および type_ns の処理

このセクションの例では、属性名は OrderStaus、SOAP 要素ネーム・スペースの接頭部は ns0、そして属性タイプは OrderStatus になっています。

注: type_name および type_ns の処理は、構成 MO の属性 TypeInfo が true になっている場合にのみ行われます。

1. type_name も type_ns も指定されていない場合、type_name のデフォルト値としてビジネス・オブジェクト名が使用され、型ネーム・スペースのデフォルト値として要素のネーム・スペースが使用されます。ASI は指定されません。

   ```
   <OrderStatus  xsi:type="ns0:OrderStatus">1</OrderStatus>
   ```

2. type_name が指定され、type_ns が指定されていない場合、type_name は割り当てられた ASI の値に設定され、type_ns については、デフォルト値として要素のネーム・スペースが使用されます。ASI は以下のようになります。

   ```
   type_name=CustOrderStatus
   <OrderStatus  xsi:type="ns0:CustOrderStatus">1</OrderStatus>
   ```

3. type_ns が指定され、type_name が指定されていない場合、type_name のデフォルト値としてビジネス・オブジェクト名が使用され、type_ns は割り当てられた type_ns 値に設定されます。型ネーム・スペースのために固有の接頭部が生成されます。ASI は以下のようになります。

   ```
   type_ns=CustTypeNS
   <OrderStatus  xsi:type="ns2:SOAP_OrderStatusLine
   " xmlns:ns2="CustTypeNS">1</OrderStatus>
   ```

4. type_name と type_ns の両方が指定されている場合、これらは、割り当てられた ASI 値に設定されます。型ネーム・スペースのために固有の接頭部が生成されます。ASI は以下のようになります。

   ```
   type_name=CustOrderStatus;type_ns=CustTypeNS
   <OrderStatus
   xsi:type="ns2:CustOrderStatus" xmlns:ns2="CustTypeNS">1</OrderStatus>
   ```

複数カーディナリティー属性の場合の type_name および type_ns の処理

このセクションで示すすべての例では、属性名が MultiLines であって、SOAP 要素ネーム・スペースの接頭部が ns0 であると想定されています。属性タイプは OrderStatus であるものとします。

注: type_name および type_ns の処理は、構成 MO の属性 TypeInfo が true になっている場合にのみ行われます。

1. type_name も type_ns も指定されていない場合、type_name のデフォルト値としてビジネス・オブジェクト名が使用され、type_ns のデフォルト値として要素のネーム・スペースが使用されます。ASI は以下のようになります。

   ```
   <MultiLines SOAP-ENC:arrayType="ns0:OrderStatus[2]"
   xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/"
   xsi:type="SOAP-ENC:Array">
   ```

2. type_name が指定され、type_ns が指定されていない場合、type_name は、割り当てられた ASI の type_name 値に設定され、type_ns については、デフォルト値として要素のネーム・スペースが使用されます。ASI は以下のようになります。

```
type_name=CustOrderStatus
<MultiLines SOAP-ENC:arrayType="ns0:CustOrderStatus[2]"
xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/"
xsi:type="SOAP-ENC:Array">
```

3. type_ns が指定され、type_name が指定されていない場合、type_name のデフォルト値としてビジネス・オブジェクト名が使用され、type_ns は、割り当てられた ASI の type_ns 値に設定されます。型ネーム・スペースのために固有の接頭部が生成されます。ASI は以下のようになります。

```
type_ns=CustTypeNS
<MultiLines SOAP-ENC:arrayType="ns2:OrderStatus[2]"
xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/
" xmlns:ns2="CustTypeNS" xsi:type="SOAP-ENC:Array">
```

4. type_name と type_ns の両方が指定されている場合、これらは、割り当てられた ASI 値に設定されます。型ネーム・スペースのために固有の接頭部が生成されます。ASI は以下のようになります。

```
type_name=CustOrderStatus;type_ns=CustTypeNS
<MultiLines SOAP-ENC:arrayType="ns2:CustOrderStatus[2
]" xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:ns2="CustTypeNS" xsi:type="SOAP-ENC:Array">
```

注: 各 Array 要素の親を表す項目要素の型およびネーム・スペースは、arrayType と同じです。

単純型、単一カーディナリティー型、および複数カーディナリティー型の場合の xsdtype

単純型、単一カーディナリティー型、および複数カーディナリティー型の場合、型名が SOAP メッセージの現行 XSD に従うときには、xsdtype ASI 属性を true に設定してください。xsdtype プロパティーが読み取られるのは、type_name と type_ns の両方のプロパティーが設定されている場合のみです。type_name および type_ns が指定された場合、SOAP データ・ハンドラーは、最初に、SOAP API を使用してそれらのペアを Java 型にマップすることを試みます。データ・ハンドラーは、その後で、SOAP メッセージのための現行 XSD を使用して、Java 型を SOAP 要素型に変換することを試みます。例えば、現行 XSD が以下のとおりであって

```
http://www.w3.org/2001/XMLSchema
```

ASI が以下のとおりである場合、

```
type_name=timeInstant;type_ns=http://www.w3.org/1999/XMLSchema;xsdtype=true
```

SOAP メッセージ型名は次のように書かれます。

```
<OrderDate xsi:type="xsd:dateTime">
```

これは、2001 XSD における dateTime が、1999 XSD における timeInstant と同等であるためです。

xsdtype および単純型の配列

複数カーディナリティー・オブジェクトの場合、次のような単純型の配列を作成することができます。

```
<MultiLines SOAP-ENC:arrayType="xsd:string[4]"
xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/"
xsi:type="SOAP-ENC:Array">
```

そのためには、type_name プロパティーを、希望する単純型 (例えば、string) に設定し、type_ns プロパティーを適切な XSD 仕様に設定してください。その後で、xsdtype プロパティーを true に設定して、この型が現行の XSD 型に変換されるようにしてください。最後に、arrayOf プロパティーを、単純型値を保持すべきコンテナー内の属性の名前に設定する必要があります。ストリング配列の場合の ASI の例を次に示します。

```
arrayof=size;type_name=string;type_ns=http://www.w3.org/2001/XMLSchema;xsdtype=true
```

障害処理における ASI の効果

faultcode、faultactor、faultstring、および detail の各要素は、以下の規則に従います。

1. これらの属性における elem_name、elem_ns、type_name、および type_ns ASI は無視されます。
2. detail 要素のすべての子は、本文処理の説明で述べたとおりに書かれます。

ヘッダー処理における ASI の効果

ヘッダーの子オブジェクトのレベルおよびそれ以下では、すべての ASI プロパティー (表 46 を参照) を使用することができます。

SOAP 属性の指定

単純タイプの場合の attr_name の処理

simpleContent 拡張または制限を持つ complexTypes が値と属性の両方を持っている XML スキーマの事例があります。例えば、次の SOAP タグがあるとします。

```
<size system="us">10</size>
```

これは、次のスキーマに基づいています。

```
<complexType name="SizeType">
   <simpleContent>
       <extension base="int">
          <attribute name="system" type="string"/>
       </extension>
   </simpleContent>
</complexType>
<element name="size" type="ns:SizeType"/>
```

複合タイプに対応する、単純コンテント拡張または制限を持つビジネス・オブジェクトは、複合タイプ属性に対応するその他の属性に加えて、1 つの追加属性を持っていなければなりません。追加属性は、単純コンテント値 (前述の例の場合は要素サイズの値 10) を持っていなければなりません。このような複合タイプに対応するビジネス・オブジェクトをタイプとして持っているビジネス・オブジェクト属性は、属性レベル ASI として elem_value=simpleContentValue を持ちます。

図 45 は、対応するビジネス・オブジェクトを示しています。

Name	Type	Key	Card	Application Specific Informatio
⊟ Request	SOAP_getQuote_N09218329332_Request	☐	1	ws_botype=request
⊟ size	SOAP_getQuote_C09218329332_SizeType	☐	1	elem_value=simpleContentValue; type_name=SizeType
simpleContentValue	String	☑		
system	String	☐		attr_name=system

図 45. 単純タイプの attr_name ビジネス・オブジェクト

単一カーディナリティー型および複数カーディナリティー型の場合の attr_name の処理

ビジネス・オブジェクト属性を SOAP 要素ではなく SOAP 属性に変換する ASI を指定することができます。データ・ハンドラーは、単一および n カーディナリティーの複合タイプに対してのみ SOAP 属性の追加をサポートしています。次の例について考えてみましょう。

```
<CustInfo City="4" State="5" Street="2" Zip="6">
   <Name xsi:type="xsd:string">1</Name>
   <Street2 xsi:type="xsd:string">3</Street2>
</CustInfo>
```

このビジネス・オブジェクト定義構造 (図 46 の各属性の右側に属性レベル ASI が指定されています) の場合、データ・ハンドラーは以下の処理ステップに従います。

Name	Type	App Spec Info
⊟ CustInfo	CustomerInfo	
Name	String	
Street1	String	attr_name=Street
Street2	String	
City	String	attr_name=City
State	String	attr_name=State
Zip	String	attr_name=Zip

図 46. attr_name ビジネス・オブジェクト

1. データ・ハンドラーは、複合属性全体を探索する際に、まず、この複合ビジネス・オブジェクト属性に対応するタグを生成します。この例では、CustInfo が複合ビジネス・オブジェクト属性を表します。
2. データ・ハンドラーは、複合ビジネス・オブジェクトの複数の子について繰り返し処理を行います。属性を作成する際に考慮されるのは、単純型属性のみです。ある単純型の ASI プロパティーの名前が attr_name である場合、データ・ハンドラーはこの単純型を SOAP 要素の属性として記述します。この例では、要素 (CustInfo) には Street、City、State、および Zip の 4 つの属性があります。
3. ビジネス・オブジェクトのそれ以外の属性は、標準的な BODY 処理を使用して記述されます。これはつまり、関係のあるすべての ASI が、attr_name ASI を持たないビジネス・オブジェクト属性に対しても評価されるということです。

複数カーディナリティー型を処理する際のロジックは、単一カーディナリティー型を処理するロジックと同じです。具体的には、それぞれの <item> タグは複数カーディナリティー・オブジェクトにおける各ビジネス・オブジェクト・インスタンスに対応し、ASI を使用して処理されます。例えば、この複数カーディナリティー・ビジネス・オブジェクト定義構造に、対応する ASI があるとします。

Name	Type	Card	App Spec Info
⊟ CustInfo	CustomerInfo	N	
Name	String		
Street1	String		attr_name=Street
Street2	String		
City	String		attr_name=City
State	String		attr_name=State
Zip	String		attr_name=Zip

図 47. attr_name 複数カーディナリティー・ビジネス・オブジェクト

データ・ハンドラーに送られたイベントに、この複数カーディナリティー・オブジェクトのインスタンスが 2 つ含まれている場合、次のような SOAP メッセージが作成されます。

```
<CustInfo>
   <item City="Armonk" Street="Main Street">
      <Name>IBM</Name>
      <Street2>None</Street2>
   </item>
   <item City="Burlingame" State="Ca" Street="577 Airport Blvd" Zip="94010">
      <Name>Burlingame Labs</Name>
      <Street2>Suite 600</Street2>
   </item>
</CustInfo>
```

item タグが複合要素型として扱われることに注意してください。ビジネス・オブジェクト定義に含まれている属性は、対応する item タグの SOAP 属性になります。

単純タイプ配列の arrayof 処理

arrayof ASI プロパティーは、SOAP でエンコードされた単純タイプ配列の場合にのみ使用してください。例えば、次のような直列化があるとします。

```
<CustomerNames SOAP-ENC:arrayType="xsd:string[4]" xmlns:SOAP-ENC=
"http://schemas.xmlsoap.org/soap/encoding/" xsi:type="SOAP-ENC:Array">
<item xsi:type="xsd:string">value1</item>
<item xsi:type="xsd:string">value2</item>
<item xsi:type="xsd:string">value3</item>
<item xsi:type="xsd:string">value4</item>
</CustomerNames>
```

これは、図 48 に示すようなビジネス・オブジェクト定義を必要とします。

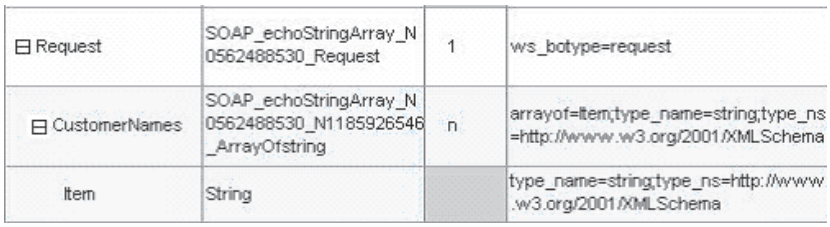


図 48. arrayof ビジネス・オブジェクト

(わかりやすいように、ビジネス・オブジェクトは要求レベルから示します。)

注: ここには示されていませんが、ビジネス・オブジェクト構造から前述の SOAP 直列化を派生させるために、この例の SOAP 構成 MO の TypeInfo プロパティーを `true` に設定する必要があります。

また、`arrayof` プロパティーを使用して、`item` 以外の名前の配列項目を作成できます。前述の例を使用する際、ビジネス・オブジェクト属性名および「arrayof」 ASI プロパティー値の両方が name である場合は、<item> タグを <name> タグに置き換えることができます。この場合、直列化は次のようになります。

```
<CustomerNames SOAP-ENC:arrayType="xsd:string[4]"
xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/"
xsi:type="SOAP-ENC:Array">
<name xsi:type="xsd:string">value1</name>
<name xsi:type="xsd:string">value2</name>
<name xsi:type="xsd:string">value3</name>
<name xsi:type="xsd:string">value4</name>
</CustomerNames>
```

attr_name および attr_ns の処理

作成される SOAP 属性に対応したネーム・スペースを指定することをお勧めします。そのためには、単純型の場合には ASI プロパティー attr_ns を指定します。データ・ハンドラーが attr_ns プロパティーを処理するのは、同じ属性の ASI に attr_name が存在する場合のみです。attr_name および attr_ns には、以下の規則が適用されます。

1. attr_name も attr_ns も設定されていない場合には、ビジネス・オブジェクト属性は SOAP 要素に変換されます。
2. attr_name のみが設定されている場合には、SOAP 属性のネーム・スペースのデフォルト値として、その要素のネーム・スペースが使用されます。

   ```
   <CustInfo Street="577 Airport"></CustomerInfo>
   ```

3. attr_ns のみが設定されている場合には、このプロパティーは無視され、ビジネス・オブジェクト属性は SOAP 要素に変換されます。
4. attr_name と attr_ns の両方が設定されている場合には、次のような SOAP 属性が作成されます。

   ```
   <CustInfo ns2:Street="577 Airport" xmlns:ns2=
   "AttrNS"></CustomerInfo>
   ```

dh_mimetype: データ・ハンドラーの呼び出し

SOAP データ・ハンドラーを使用して別のデータ・ハンドラーを呼び出し、ビジネス・オブジェクトを、対応するデータ・ハンドラーが存在する任意の形式に書き出

すことができます。これは、SOAP 子ビジネス・オブジェクトを SOAP ストリングに変換するときに、エンコードされたテキストを SOAP メッセージに追加することによって行います。

SOAP 要素の値をエンコードすることのできるフォーマットの 1 つに、RNIF 文書があります。この機能を使用するためには、任意のレベルの SOAP 子ビジネス・オブジェクトに RNIF BO を追加してください。この RNIF ビジネス・オブジェクトをストリングに変換するときに別のデータ・ハンドラーを呼び出すように SOAP データ・ハンドラーに通知するためには、その属性の ASI に dh_mimetype プロパティーを追加します。ASI プロパティー dh_mimetype の値は、MO_DataHandler_Default メタオブジェクトで指定された適格な mimeType でなければなりません。この mimeType は、ビジネス・オブジェクトを処理するために呼び出すデータ・ハンドラーを判別するために使用されます。

図 49 に示す SOAP 子ビジネス・オブジェクトでは、`CustomerInfo` は複合子であり、`RNET_Pip3A2PriceAndAvailabilityQuery` は RNIF ビジネス・オブジェクトになっています。

Name	Type	App Spec Info
⊟ CustomerInfo	CustomerInfo	
Name	String	
CustID	String	
⊞ RNIFMsg	RNET_Pip3A2PriceAndAvailabilityQuery	elem_name=RNIFexample;dh_mimetype=application/x_rosettanet_agent;type_name=base64Binary;type_ns=http://www.w3.org/2001/XMLSchema;xsdtype=true

図 49. dh_mimetype を有する RNIF ビジネス・オブジェクト

このビジネス・オブジェクトから作成される SOAP メッセージは、次のようになります。

```
<CustomerInfo>
<Name>IBM Corporation</Name>
<CustID>95626</CustID>
<RNIFexample
xsi:type="xsd:base64Binary">1AWERYER238W98EYR923872837487189278 7ASRJK23423
JKAWERJ234AWERIJHI423488R4HASF1AWERYER238W98EYR923872837487189278 7ASRJK234
34JKAWERJ234AWERIJHI423488R4HASF1AWERYER238W98EYR923872837487189278 7ASRJK2
4234JKAWERJ234AWERIJHI423488R4HASF1AWERYER238W98EYR923872837487189278 7ASRJ
234234JKAWERJ234AWERIJHI423488R4HASFWR234
</RNIFexample>
</CustomerInfo>
```

RNIF サンプル要素には RNIF エンコード・ストリングが含まれていますが、これは、その要素値としてエンコードされた base64 バイナリーだったものです。また、ASI プロパティー elem_name、elem_ns、type_name、type_ns、および xsdtype は、このビジネス・オブジェクト属性との関係を維持します。この例では、メッセージが作成されると、指定された elem_name が SOAP 要素の名前になります。

注: 呼び出し先のデータ・ハンドラーによって戻される要素値がエンコード・テキストである場合には、type_name プロパティーは `base64Binary` に設定されていなければならず、type_ns は xsd ネーム・スペースに対応していなければならず、また xsdtype は `true` に設定されていなければなりません。

xsd:base64Binary: type_name および type_ns を xsd:base64Binary に解決されるように設定すると、SOAP データ・ハンドラーは、対応する要素の値を設定する前に、ビジネス・オブジェクトから得られた値をエンコードします。データ・ハンドラーは Apache API を使用してレジストリーに対して base64Binary シリアライザーの照会を行い、呼び出し先データ・ハンドラーから戻されたストリングをシリアライズし、要素の値を設定します。

スキーマ complexType の標識

以下のセクションでは、スキーマ complexType の標識がビジネス・オブジェクトに与える影響について説明します。標識には以下のものがあります。

- maxOccurs
- minOccurs
- all
- sequence
- choice

単純タイプの場合の maxOccurs および minOccurs 標識: maxOccurs 標識は、1 つの複合タイプの中である要素が出現することのできる、最大回数を表します。minOccurs 標識は、1 つの複合タイプの中である要素が出現する必要のある、最小回数を表します。

次のスキーマを検討してください。

```
<xs:element name="Address" type="Address">
<xs:complexType name="Address">
     <xs:sequence>
        <xs:element name="AddressLine" type="xsd:string" maxOccurs="10"/>
        <xs:element name="SuiteNumber" type="xsd:string" minOccurs="3"
                              maxoccurs="unbounded"/>
     <xs:element name="City" type="xsd:string"/>
  </xs:sequence>
  </xs:complexType>
</xs:element>
```

上の例では、Address 要素内で AddressLine 要素が最大 10 回出現することができ、SuiteNumber 要素が少なくとも 3 回出現することを示しています。このスキーマに対応するビジネス・オブジェクトでは、次の ASI を備えたそれぞれの maxoccurs/minoccurs 標識ごとに、N カーディナリティーのラッパー・オブジェクトが 1 つ存在しなければなりません。

```
maxOccurs=N;wrapper=true
```

または

```
minOccurs=3;wrapper=true;
```

wrapper=true ASI は、このオブジェクトがラッパーであること、およびその理由から、SOAP メッセージに明示的に書き込まれないことを示しています。明示的な書き込みが行われないため、このラッパー・オブジェクトには、単純型の子が 1 つ存在しなければなりません。実行時に SOAP からビジネス・オブジェクトへの変換を行うために、データ・ハンドラーは、ラッパーの子オブジェクトを N 個読み取り、それぞれの子オブジェクトごとに対応する要素を作成します。ビジネス・オブジェ

クトから SOAP メッセージへの変換を行う際には、データ・ハンドラーは、検出されたそれぞれの要素ごとに、N カーディナリティー・ラッパー内に子オブジェクトを作成します。

これに対応する SOAP ビジネス・オブジェクトは、図 50 のようなものとなります。

Pos	Name	Type	Key	Card	App Spec Info
1	⊟ Address	Address	☑	1	
1.1	⊟ AddressLine	AddressLine_wrap	☑	N	maxoccurs=10;wrapper=true
1.1.1	AddressLine	String	☑		
1.1.2	ObjectEventId	String			
1.2	⊟ SuiteNumber	SuiteNumber_wrap	☑	N	minoccurs=3;wrapper=true
1.2.1	SuiteNumber	String	☑		
1.2.2	ObjectEventId	String			
1.3	City	String	☐		
1.4	ObjectEventId	String			
2	ObjectEventId	String			

図 50. SOAP ビジネス・オブジェクトにおける単純タイプ ASI の minOccurs および maxOccurs

図 50 に示したビジネス・オブジェクトに対応する SOAP メッセージは、次のようになります。

```
<Address xsi:type="ns0:Address">
    <AddressLine xsi:type="xsd:string">Line1</AddressLine>
    <AddressLine xsi:type="xsd:string">Line2</AddressLine>
    <SuiteNumber xsi:type="xsd:string">600</SuiteNumber>
    <SuiteNumber xsi:type="xsd:string">650</SuiteNumber>
    <SuiteNumber xsi:type="xsd:string">700</SuiteNumber>
  <City xsi:type="xsd:string">San Francisco</City>
</Address>
```

注: SOAP データ・ハンドラーは、maxOccurs 標識を処理する場合にも minOccurs 標識を処理する場合にも、要素の最大および最小出現回数を検証しません。データ・ハンドラーは、maxOccurs および minOccurs 標識を備えた特定の要素の複数のインスタンスを保持するためのコンテナー構造を提供するだけです。この方法は、単純タイプおよび複合タイプに適用されます。

複合タイプの場合の maxOccurs および minOccurs 標識: <maxOccurs> 標識は、1 つの複合タイプの中である要素が出現することのできる、最大回数を表します。<minOccurs> 標識は、1 つの複合タイプの中である要素が出現する必要のある、最小回数を表します。次のスキーマにおける maxOccurs 標識について検討してください。

```
<xs:element name="Address" type="Address">
<xs:complexType name="Address">
     <xs:sequence>
     <xs:element name="AddressInfo" type="AddressInfo" maxOccurs="3"/>
     <xs:element name="City" type="xsd:string"/>
  </xs:sequence>
  </xs:complexType>
</xs:element>

<xs:complexType name="AddressInfo">
```

```
     <xs:sequence>
     <xs:element name="StreetLine" type="xsd:string"/>
     </xs:sequence>
  </xs:complexType>
</xs:element>
```

上の例では、Address 要素内で AddressInfo 複合タイプ要素が最大 3 回出現することができることを示しています。このスキーマに対応するビジネス・オブジェクトには、ラッパー・オブジェクトがありません。complexType の AddressInfo 自体が N カーディナリティーの要素になりうるためです。N カーディナリティーの属性には、maxoccurs=3 という ASI が含められます。

図 51 は、これに対応する SOAP ビジネス・オブジェクトを示しています。

Pos	Name	Type	Key	Card	App Spec Info
1	⊟ Address	Address	☑	1	
1.1	⊟ AddressInfo	AddressInfo	☑	N	maxoccurs=3
1.1.1	StreetLine	String	☑		
1.1.2	ObjectEventId	String			
1.2	City	String	☐		
1.3	ObjectEventId	String			
2	ObjectEventId	String			

図 51. SOAP ビジネス・オブジェクトにおける複合タイプ ASI の minOccurs および maxOccurs

図 51 に示したビジネス・オブジェクトに対応する SOAP メッセージは、次のようになります。

```
<Address xsi:type="ns0:Address">
     <AddressInfo xsi:type="ns0:AddressInfo">
        <StreetLine xsi:type="xsd:string">100 Market St.</ StreetLine>
        <StreetLine xsi:type="xsd:string">Apt 15</ StreetLine>
     </AddressInfo>
  <City xsi:type="xsd:string">San Francisco</City>
</Address>
```

all 標識: all 標識は、デフォルトでは、この complexType に関する子要素を任意の順序で指定できること、およびそれぞれの子要素がゼロまたは 1 回出現しなければならないことを指定します。次のスキーマについて考えてみましょう。

```
<complexType name="Item">
     <all>
        <element name="quantity" type="xsd:int"/>
        <element name="product" type="xsd:string"/>
     </all>
</complexType>
```

上記の例は、SOAP メッセージ内で要素 quantity および product を任意の順序で表示できることを示しています。quantity 要素を最初に表示して、product 要素を 2 番目に表示することも、逆の順序で表示することもできます。

図 52 は、このスキーマに対応するビジネス・オブジェクトを表しています。

Pos	Name	Type	Card	App Spec Info
1	⊟ Item	Item	1	all=Item_wrapper
1.1	⊟ Item_wrapper	Item_wrapper	N	wrapper=true
1.1.1	quantity	String		
1.1.2	product	String		
1.1.3	ObjectEventId	String		
1.2	ObjectEventId	String		
2	ObjectEventId	String		

図 52. SOAP ビジネス・オブジェクトにおける all 標識 ASI

対応する SOAP メッセージは次のようになります。

```
<Item xsi:type="ns0:Item">
    <quantity xsi:type="xsd:string">12</quantity>
    <product xsi:type="xsd:string">2</product>
</Item>
```

「all」コンテント・モデルを持つ配列コンテントの処理: このセクションで説明するように、SOAP データ・ハンドラーは、「all」コンテント・モデルを持つ複合タイプ配列コンテントを処理します。例では、ArrayOfSOAPStruct が、「all」コンテント・モデルを持つ SOAPStruct を含んでいます。

```
<complexType name="SOAPStruct">
       <all>
       <element name="varString" type="string" />
       <element name="varInt" type="int" />
       <element name="varFloat" type="float" />
       </all>
</complexType>
<complexType   name ="ArrayOfSOAPStruct'">
  <complexContent>
   <restriction base='SOAP-ENC:Array'>
    <attribute ref='SOAP-ENC:arrayType'
     wsdl:arrayType='typens:SOAPStruct[]'/>
   </restriction>
  </complexContent>
 </complexType>
```

SOAP データ・ハンドラーは、直列化時に次の SOAP データを生成しなければなりません。

```
<?xml version='1.0' encoding='UTF-8'?>
<SOAP-ENV:Envelope
        xmlns:SOAP-ENV = "http://schemas.xmlsoap.org/soap/envelope/"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema">
<SOAP-ENV:Body>
 <ns0:echoStructArray xmlns:ns0="http://soapinterop.org/">
  <inputStructArray SOAP-ENC:arrayType="ns1:SOAPStruct[2]"
           xmlns:ns1="http://soapinterop.org/xsd" xsi:type="SOAP-ENC:Array">
      <item>
                  <ns1:varFloat xsi:type="xsd:string">1.1</ns1:varFloat>
                  <ns1:varInt xsi:type="xsd:string">1</ns1:varInt>
           <ns1:varString xsi:type="xsd:string">hi</ns1:varString>
      <item>
      <item>
           <ns1:varString xsi:type="xsd:string">hello</ns1:varString>
           <ns1:varInt xsi:type="xsd:string">1</ns1:varInt>
           <ns1:varFloat xsi:type="xsd:string">1.1</ns1:varFloat>
```

```
        </item>
     </inputStructArray>
    </ns0:echoStructArray>
  </SOAP-ENV:Body>
    </SOAP-ENV:Envelope>
```

この例において、echoStructArray は操作の名前であり、inputStructArray はタイプ ArrayOfSOAPStruct を持つパラメーター名です。

sequence 標識: sequence 標識は、子要素が、complexType で指定されたとおりの順序で現れなければならないことを示します。

```
<complexType name="Item">
     <sequence>
        <element name="quantity" type="int"/>
        <element name="product" type="string"/>
     </sequence>
</complexType>
```

SOAP データ・ハンドラーは、この標識のための特別な ASI またはラッパー・オブジェクトを必要としません。デフォルトでは、データ・ハンドラーは、ビジネス・オブジェクトで指定されたとおりの順序で SOAP 要素の読み取りおよび書き込みを行います。

choice 標識: choice 標識は、complexType 内の要素のうちの 1 つだけを SOAP メッセージに含めることができることを示します。次のスキーマについて考えてみましょう。

```
<complexType name="Item">
    <choice>
        <element name="quantity" type="int"/>
        <element name="product" type="string"/>
    </choice>
</complexType>
```

SOAP データ・ハンドラーは、この標識のための特別な ASI またはラッパー・オブジェクトを必要としません。ビジネス・オブジェクトを SOAP メッセージに変換するときに、データ・ハンドラーは処理を中断し、SOAP メッセージにどの要素を含めるのかをユーザーが選択できるようにします。SOAP メッセージをビジネス・オブジェクトに変換するときには、データ・ハンドラーは既存の要素を読み取り、それに対応する属性を取り込みます。

sequence、choice、group、および all における maxOccurs 標識: モデル・グループ (sequence、choice、group、および all) は、minOccurs 属性と maxOccurs 属性を持っています。minOccurs および maxOccurs のデフォルト値は 1 です。all グループの場合、maxOccurs は値 1 のみを取ることができます。WSDL ODA および SOAP データ・ハンドラーは、sequence、choice、および group に対するすべての指定可能な maxOccurs の値をサポートします。

SOAP からビジネス・オブジェクトへの変換における ASI

SOAP データ・ハンドラーは、着信 SOAP メッセージを読み取って検証するために、ビジネス・オブジェクトの ASI を使用します。SOAP データ・ハンドラーによる ASI の検証には、以下の規則が適用されます。

- ヘッダーの処理と本文の処理は同じ方法で行われます。

- データ・ハンドラーに下記のセクションで述べる検証を行わせるためには、SOAP 構成 MO プロパティー TypeCheck を strict に、また TypeInfo を true に設定しなければなりません。
- 型検証は一般に type_name プロパティーと type_ns プロパティーの両方に依存するため、これらのプロパティーの検証は並行して行われます。

注: 別の指示が示されていないかぎり、以下のセクションで述べるすべての ASI は属性レベルの ASI です。

elem_name の検証

単純、カーディナリティー 1、およびカーディナリティー n の属性の検証には、以下の規則が適用されます。

1. SOAP メッセージの構文解析中に要素が検出された場合、データ・ハンドラーは最初にビジネス・オブジェクト・レベルのすべての ASI を検索し、その要素の名前と elem_name 値の突き合わせを行います。
2. 要素の名前と elem_name 値が一致しない場合、データ・ハンドラーは、要素の名前を、そのビジネス・オブジェクト・レベルのそれぞれの属性名と突き合わせます。
3. どちらの検索も成功しなかった場合、データ・ハンドラーは失敗します。

elem_ns の検証

単純、カーディナリティー 1、およびカーディナリティー n の属性の検証には、以下のケースが当てはまります。

1. elem_ns ASI も、この要素に関する SOAP メッセージから得られる xmlns も存在しない場合、この要素が適切に検証されます。
2. elem_ns ASI が存在せず、それに対応する SOAP メッセージ内の要素で xmlns が指定されている場合、データ・ハンドラーはデフォルトの elem_ns として、ビジネス・オブジェクトから最後に読み取った、スコープ内にある elem_ns を使用します。データ・ハンドラーはこの値を、SOAP メッセージから得られた xmlns と比較します。両者が一致しない場合、検証は失敗します。
3. elem_ns ASI が存在し、それに対応する SOAP メッセージ内の要素で xmlns が指定されていない場合、データ・ハンドラーは、ASI で指定されている elem_ns が、SOAP メッセージの現行スコープ内のいずれかのネーム・スペースと一致するかどうかを調べます。両者が一致しない場合、検証は失敗します。

type_name および type_ns の検証

以下のセクションでは、type_name および type_ns の検証について説明します。

単純属性: xsdType が true の場合、type_name および type_ns の検証には以下の規則が適用されます。

- **type_name と type_ns の両方が指定されている場合** データ・ハンドラーは、type_name と type_ns のペアを使用して、対応する java Class オブジェクトを作成します。着信 SOAP メッセージの typename と typenamespace を使用して、別の java Class オブジェクトが照会されます。2 つの java Class オブジェクトが一致した場合には、検証が成功します。それ以外の場合、検証は失敗します。
- **type_name も type_ns も指定されていない場合** データ・ハンドラーは、単純ビジネス・オブジェクト属性を java Class オブジェクトにマップします。着信

SOAP メッセージの typename と typenamespace を使用して、別の java Class オブジェクトが照会されます。2 つの java Class オブジェクトが一致した場合には、検証が成功します。それ以外の場合、検証は失敗します。

- **type_name のみが指定されている場合** 単純型検証は失敗します。xsdType が true の場合には、type_name と type_ns がともに指定されているか、あるいはそのいずれも指定されていない必要があります。
- **type_ns のみが指定されている場合** 単純型検証は失敗します。xsdType が true の場合には、type_name と type_ns がともに指定されているか、あるいはそのいずれも指定されていない必要があります。

xsdType が false の場合、type_name および type_ns の検証には以下の規則が適用されます。

- **type_name と type_ns の両方が指定されている場合** データ・ハンドラーは SOAP メッセージの typename および typenamespace のペアと、ASI で指定された type_name 値および type_ns 値のペアを直接に比較します。両方のペアが正確に一致した場合、検証は成功します。それ以外の場合、検証は失敗します。
- **type_name も type_ns も指定されていない場合** データ・ハンドラーは、単純ビジネス・オブジェクト属性を java Class オブジェクトにマップします。着信 SOAP メッセージの typename と typenamespace を使用して、別の java Class オブジェクトが照会されます。2 つの java Class オブジェクトが一致した場合には、検証が成功します。それ以外の場合、検証は失敗します。
- **type_name のみが指定されている場合** type_ns 値のデフォルト値として、ビジネス・オブジェクトの ASI から検出された要素ネーム・スペースが使用されます。データ・ハンドラーは、このデフォルト type_ns と ASI で指定された type_name を使用して、これらの値のペアと SOAP メッセージの typename および typenamespace のペアを、直接比較します。両方のペアが正確に一致した場合、検証は成功します。それ以外の場合、検証は失敗します。
- **type_ns のみが指定されている場合** type_name のデフォルト値として、ビジネス・オブジェクトの属性タイプが使用されます。データ・ハンドラーは、このデフォルト type_name と ASI で指定された type_ns を使用して、これらの値のペアと SOAP メッセージの typename および typenamespace のペアを、直接比較します。両方のペアが正確に一致した場合、検証は成功します。それ以外の場合、検証は失敗します。

複合属性 (カーディナリティー 1 および n): xsdType が true の場合、type_name および type_ns の検証には以下の規則が適用されます。

- **type_name と type_ns の両方が指定されている場合** xsdType は無視されます。データ・ハンドラーは、xsdType が false である場合と同様に処理を行います。
- **type_name も type_ns も指定されていない場合** xsdType は無視されます。データ・ハンドラーは、xsdType が false である場合と同様に処理を行います。
- **type_name のみが指定されている場合** xsdType は無視されます。データ・ハンドラーは、xsdType が false である場合と同様に処理を行います。
- **type_ns のみが指定されている場合** xsdType は無視されます。データ・ハンドラーは、xsdType が false である場合と同様に処理を行います。

xsdType が false の場合、type_name および type_ns の検証には以下の規則が適用されます。

- **type_name と type_ns の両方が指定されている場合** データ・ハンドラーは SOAP メッセージの typename および typenamespace のペアと、ASI で指定された type_name 値および type_ns 値のペアを直接に比較します。両方のペアが正確に一致した場合、検証は成功します。それ以外の場合、検証は失敗します。
- **type_name も type_ns も指定されていない場合** type_name のデフォルト値として、ビジネス属性タイプが使用されます。type_ns 値のデフォルト値として、ビジネス・オブジェクトの ASI から検出された要素ネーム・スペースが使用されます。データ・ハンドラーは、このデフォルトの振る舞いを使用して、これらの値のペアと SOAP メッセージの typename および typenamespace のペアを、直接比較します。両方のペアが正確に一致した場合、検証は成功します。それ以外の場合、検証は失敗します。
- **type_name のみが指定されている場合** type_ns 値のデフォルト値として、ビジネス・オブジェクトの ASI から検出された要素ネーム・スペースが使用されます。データ・ハンドラーは、このデフォルト type_ns と ASI で指定された type_name を使用して、これらの値のペアと SOAP メッセージの typename および typenamespace のペアを、直接比較します。両方のペアが正確に一致した場合、検証は成功します。それ以外の場合、検証は失敗します。
- **type_ns のみが指定されている場合** type_name のデフォルト値として、ビジネス・オブジェクトの属性タイプが使用されます。データ・ハンドラーは、このデフォルト type_name と ASI で指定された type_ns を使用して、これらの値のペアと SOAP メッセージの typename および typenamespace のペアを、直接比較します。両方のペアが正確に一致した場合、検証は成功します。それ以外の場合、検証は失敗します。

attr_name および attr_ns の検証

SOAP メッセージをビジネス・オブジェクトに読み込む際には、SOAP 属性を探すために個々の SOAP 要素が検索されます。SOAP 属性が検出された場合は、対応するビジネス・オブジェクトから得られた attr_name プロパティー値と比較されます。例えば、次の SOAP メッセージについて考えてみましょう。

```
<CustInfo City="4" State="5" Street="2" Zip="6">
   <Name xsi:type="xsd:string">1</Name>
   <Street2 xsi:type="xsd:string">3</Street2>
</CustInfo>
```

ここで、図 53 に示すビジネス・オブジェクト定義構造について考えてみます (各属性の右に属性レベル ASI が指定されています)。

Name	Type	App Spec Info
⊟ CustInfo	CustomerInfo	
Name	String	
Street1	String	attr_name=Street
Street2	String	
City	String	attr_name=City
State	String	attr_name=State
Zip	String	attr_name=Zip

図 53. attr_name および attr_ns の検証

データ・ハンドラーは以下の処理ステップに従います。

1. 要素名 CustInfo を読み取ります。
2. この要素名に対応するビジネス・オブジェクト属性を解決します。
3. SOAP 要素の属性を読み取り、それらを子属性の ASI と突き合わせます。この例では、SOAP メッセージの Street とビジネス・オブジェクト属性 Street1 の突き合わせ、City とビジネス・オブジェクト属性 City の突き合わせ、さらに、以下同様の突き合わせが行われます。
4. CustInfo の子要素が、本文の残りの部分と同じ方法で読み取られ、処理されます。

注: attr_ns の検証は行われません。

データ・ハンドラーは、所定の要素の SOAP 属性について繰り返し処理を行います。属性に出会うたびに、データ・ハンドラーは、対応する属性のためのビジネス・オブジェクトを検索します。ビジネス・オブジェクトが見付かったら、そのビジネス・オブジェクト属性に SOAP 属性の値を取り込みます。対応するビジネス・オブジェクト属性が見付からない場合、データ・ハンドラーは次の SOAP 属性に移って処理を続けます。

SOAP データ・ハンドラー内部からのデータ・ハンドラーの呼び出し

SOAP データ・ハンドラーは、別のデータ・ハンドラーを使用して、エンコードされた要素値を、SOAP メッセージからビジネス・オブジェクトに読み込むことができます。例えば、SOAP 要素の値がエンコードされるフォーマットの 1 つに、RNIF 文書があります。この機能を使用するためには、RNIF ビジネス・オブジェクトを任意のレベルの SOAP 子ビジネス・オブジェクトに追加します。この RNIF エンコード・ストリングを RNIF ビジネス・オブジェクトに変換するときに別のデータ・ハンドラーを使用しなければならないことを SOAP データ・ハンドラーに通知するためには、その属性の ASI に dh_mimetype プロパティーを追加する必要があります。dh_mimetype ASI の値は、MO_DataHandler_Default ビジネス・オブジェクトで指定された適格な mimeType でなければなりません。mimeType は、ストリング上で使用するデータ・ハンドラーを判別するために使用されます。例えば、次の SOAP メッセージでは、RNIFExample が SOAP 要素で、ここには RNIF エンコード・ストリングが含まれています。

```
<CustInfo>
<Name>IBM Corporation</Name>
<CustID>95626</CustID>
<RNIFexample xsi:type="xsd:base64Binary">
1AWERYER238W98EYR923872837487189278 7ASRJK234234JKAWER
J234AWERIJHI423488R4HASF1AWERYER238W98EYR923872837487
1892787ASRJK234234JKAWERJ234AWERIJHI423488R4HASF1AWER
YER238W98EYR923872837487189278 7ASRJK234234JKAWERJ234A
WERIJHI423488R4HASF1AWERYER238W98EYR92387283748718927
87ASRJK234234JKAWERJ234AWERIJHI423488R4HASFWR234
</RNIFexample>
</CustomerInfo>
```

SOAP ビジネス・オブジェクトは、図 54 のようなものとなります。

Name	Type	App Spec Info
⊟ CustInfo	CustomerInfo	
Name	String	
Street1	String	attr_name=Street
Street2	String	
City	String	attr_name=City
State	String	attr_name=State
Zip	String	attr_name=Zip

図 54. RNIFExample ビジネス・オブジェクト

RNIFExample 要素には、その要素値として RNIF エンコード・ストリングが含まれています。また、ASI プロパティー elem_name、elem_ns、type_name、type_ns、および xsdtype は、このビジネス・オブジェクト属性との関係を依然として維持しています。

注: 呼び出し先のデータ・ハンドラーによって戻される要素値がエンコード・テキストである場合には、type_name プロパティーは base64Binary に設定されていなければならず、type_ns は xsd ネーム・スペースに対応していなければならず、また xsdtype は true に設定されていなければなりません。

デフォルトのビジネス・オブジェクト解決

SOAP からビジネス・オブジェクトへの変換では、SOAP データ・ハンドラーと Web サービス・コネクターは、情報を交換してビジネス・オブジェクト名を解決する特殊な契約に準拠しています。このコネクターは、SOAP データ・ハンドラーに、BodyName と BodyNamespace のペアにマップされるビジネス・オブジェクト名のリストを提供します。さらに、TLO に defaultfault ビジネス・オブジェクトが設定されている場合は、この情報がデータ・ハンドラーに渡されます。この情報が渡されると、SOAP データ・ハンドラーは、以下のステップに従って処理します。

1. SOAP メッセージを受け取ります。
2. そのメッセージが SOAP 要求メッセージか応答メッセージか、あるいは障害メッセージかを判断します。
   a. SOAP 要求メッセージまたは応答メッセージである場合は、SOAP-ENV:Body 要素の最初の子要素から、BodyName および BodyNamespace を読み取ります。
   b. SOAP 障害メッセージである場合は、その障害メッセージ内の detail 要素の最初の子要素から、BodyName および BodyNamespace を読み取ります。障害メッセージに detail 要素がない場合は、この変換用の defaultfault ビジネス・オブジェクトを使用します。
3. defaultfault ビジネス・オブジェクトがまだ選択されていない場合、データ・ハンドラーは、ステップ 2 で検出した BodyName および BodyNamespace を、コネクターが提供するリストで検出されるペアと一致させようとします。一致させられれば、ビジネス・オブジェクトの解決は成功したことになります。一致しない場合、データ・ハンドラーは失敗し、失敗を意味するエラー・メッセージが出されます。

プラグ可能な名前ハンドラーの指定

デフォルトのビジネス・オブジェクト解決を使用すると、SOAP メッセージからビジネス・オブジェクトへの変換で使用されるビジネス・オブジェクトを判別するために、プラグ可能な名前ハンドラーを指定することができます。これは、MO_DataHandler_DefaultSOAPConfig 属性を変更することによって行います。

例えば、MO_DataHandler_DefaultSOAPConfig には `string` 型の属性が 2 つあり、以下を指定します。

- **ClassName** SOAP データ・ハンドラーの基本クラスのクラス名。プラグ可能な名前ハンドラーを指定する場合には、この属性値を変更しないでください。
- **SOAPNameHandler** SOAPNameHandler 属性では、どの名前ハンドラーを呼び出すのかを指定します。プラグ可能な名前ハンドラーの値を指定できます。このプロパティーの値は、クラス名でなければなりません。SOAPNameHandler クラスは、以下のシグニチャーを備えた抽象クラスです。

  ```
  public abstract String getBOName(Envelope msgEnv, SOAPProperty prop)
  ```

SOAPNameHandler 属性に値が指定されている場合、SOAP データ・ハンドラーは、指定された名前ハンドラーを呼び出します。この値が存在しない場合、または指定された名前ハンドラーがビジネス・オブジェクト名を入手できなかった場合には、デフォルトのビジネス・オブジェクト解決を行うために、SOAP データ・ハンドラーがデフォルトで呼び出されます。

SOAP DataHandler は MO に指定されている SOAPNameHandler プロパティーを使用して、custom-name-handler クラスのインスタンスを生成します。次に、getBOName を呼び出して、ビジネス・オブジェクト名を解決します。SOAP DataHanlder は、コネクターから受け取った SOAPProperty オブジェクトを custom-name-handler インプリメンテーション・クラスに渡します。

この SOAPProperty オブジェクトには、解決に使用する潜在的な候補ビジネス・オブジェクトの構造化されたリストが含まれています。このリストには、BodyName、BodyNamespace、および BOName トリプレットが含まれています。これらのトリプレットは、SOAP 構成 MO の構成情報に基づいています。デフォルトの名前ハンドラーは、このオブジェクトを使用してビジネス・オブジェクトを解決します。カスタム名前ハンドラー開発者は、自分自身の判断でこのオブジェクトを使用できます。

SOAPProperty オブジェクトの使用

SOAPPropertyUtils クラスを使用して、SOAPProperty からビジネス・オブジェクト名を取り出すことができます。このためには、次のメソッドを使用します。

```
/**
 * Retrieve the business object name based on the body name and the body
* namespace
*.
* @param soapProp top level SOAPProperty object that is passed by the
* connector
* @param name body name from the SOAP message
* @param uri body namespace from the SOAP message
* @return business object name from the SOAPProperty object with the body
```

```
* name and body namespace.
*/
java.lang.String findBOName(SOAPProperty soapProp, String name, String uri);
```

NameHandler のサンプル

NameHandler:package のサンプルを以下に示します。

```
com.ibm.adapters.datahandlers.soap.namehandlers;
// DOM and Parsers
import javax.xml.parsers.DocumentBuilder;
import org.w3c.dom.Document;
import org.w3c.dom.Element;
import org.xml.sax.InputSource;
// Apache SOAP
import org.apache.soap.Envelope;
import org.apache.soap.Header;
import org.apache.soap.Body;
import org.apache.soap.Constants;
import org.apache.soap.util.xml.DOMUtils;
import org.apache.soap.util.xml.XMLParserUtils;
import org.apache.soap.util.xml.QName;
import org.apache.soap.encoding.soapenc.SoapEncUtils;
import org.apache.soap.encoding.soapenc.Base64;
// java
import java.util.Vector;
// SOAP data handler
import com.ibm.adapters.datahandlers.soap.*;
import com.ibm.adapters.datahandlers.soap.exceptions.*;
public class MyCustomNameHandler extends SOAPNameHandler {
  private static final String BOPREFIX = "MyCustomBOPrefix";
  private static final char UNDERSCORE = '_';
  private static final char EMPTY_STRING = "";

  public String getBOName(Envelope msgEnv, SOAPProperty prop)
    throws SOAPNameHandlerException
  {
    // Initialize a String Buffer
    StringBuffer boName = new StringBuffer();
    // Determine the "MyCustomBOPrefix" SOAP data handler
    // MO property. If it exists, and is populated append
    // this prefix to the front of the BOName.
    String pref = dh.getOption(BOPREFIX);
    if (pref != null) {
       boName.append(pref.equals(EMPTY_STRING)
       ? EMPTY_STRING : pref + UNDERSCORE);
  }
    // Begin parsing the SOAP msg envelope.
    Element bodyEl, requestEl;
    Body msgBody = msgEnv.getBody();
    Vector bodyEntries = msgBody.getBodyEntries();
    if((bodyEntries == null) || (bodyEntries.size() <= 0))
       throw new SOAPNameHandlerException("No Body Entries exist
        for this SOAP message. Cannot determine BOName to use.");
    // Grab the first <SOAP-ENV:Body> Element
    bodyEl = (Element) bodyEntries.elementAt(0);
    // Grab the first Child Element of the <SOAP-ENV:Body>
    // Element
    requestEl = (Element) DOMUtils.getFirstChildElement(bodyEl);
    // Read the name and namespace of this first child
    String name = bodyEl.getLocalName();
    String uri = bodyEl.getNamespaceURI();
    if (uri == null)
       uri = Constants.NS_URI_SOAP_ENV;
    // Use the SOAPPropertyUtils findBOName() method to search
    // the SOAPProperty object for this messages first element
    // name and namespace. If no match is found, a
```

```
    // SOAPDataHandlerException will be thrown. If a match is
    // found, and it's not an empty string, append to the boname.
String returnedBOName = SOAPPropertyUtils.findBOName(prop, name, uri);
if (returnedBOName != null &&
       !returnedBOName.equals(EMPTY_STRING))
  boName.append(returnedBOName);
       return boName.toString()
   }
}
```

制約事項

以下のセクションでは、データ・ハンドラーの制約事項について説明します。

SOAP の Style および Use に関するガイドライン

SOAP メッセージは、Web サービスによって定義された Style および Use に基づいて作成されます。SOAP データ・ハンドラーが提供するサポートのレベルは、表 47 に示すとおりです。

表 47. Style および Use のガイドライン

Style	Use	パーツを定義するもの	データ・ハンドラーのサポート
document	literal	element	完全
document	literal	type	限定 (下記を参照)
document	encoded	element	なし
document	encoded	type	限定 (下記を参照)
rpc	literal	element	なし
rpc	literal	type	完全
rpc	encoded	element	なし
rpc	encoded	type	完全

パーツおよびパーツ要素の順序

SOAP データ・ハンドラーが SOAP メッセージをビジネス・オブジェクトに変換する際に、SOAP メッセージが document/literal/type または document/encoded/type のフォーマットに従う場合、メッセージのパーツは、WSDL で記述されたとおりの順序になっていなければなりません。例えば、次の WSDL を検討してください。

```
<operation name="GetQuote"
     style="document" ...>
<input>
   <soap:body parts="Part1 Part2 Part3 Part4" use="literal">
</input>
</operation>

<definitions
 xmlns:stns="(SchemaTNS)"
xmlns:wtns="(WsdlTNS)"
targetNamespace="(WsdlTNS)">

<schema targetNamespace="(SchemaTNS)"
                 elementFormDefault="qualified">
<element name="SimpleElement" type="xsd:int"/>
<element name="CompositElement" type="stns:CompositeType"/>
<complexType name="CompositeType">
```

```
<all>
      <element name='elem_a' type="xsd:int"/>
      <element name='elem_b' type="xsd:string"/>
</all>
</complexType>
</schema>

<message...>
<part name='Part1' type="stns:CompositeType"/>
<part name='Part2' type="xsd:int"/>
<part name='Part3' element="stns:SimpleElement"/>
<part name='Part4' element="stns:CompositeElement"/>
</message>
Ö
</definitions>
```

SOAP メッセージは、パーツによって定義された順序に従うわなければなりません。下の SOAP 例では、Part1 要素が Part2、Part3、および Part4 要素に先行しています。正しいビジネス・オブジェクト解決を行うためには、この順序を維持しなければなりません。

```
<soapenv:body... xmlns:mns="(MessageNS)"
   xmlns:stns="(SchemaTNS)">
      <stns:elem_a>123</stns:elem_a>
      <stns:elem_b>hello</stns:elem_b>
   <soapenc:int>123</soapenc:int>123</soapenc:int>123</soapenc:int>
   <stns:SimpleElement>123</stns:SimpleElement>
   <stns:CompositeElement>
      <stns:elem_a>123</stns:elem_a>
      <stns:elem_b>hello</stns:elem_b>
   </stns:CompositeElement>
</soapenv:body>
```

SOAP メッセージが document/literal/type フォーマットまたは document/encoded/type フォーマットに従う場合、パーツ要素も定義された順序に従う必要があります。上の例の Part1 で、elem_a タグは elem_b タグの前になければなりません。この制約は、データ・ハンドラーのビジネス・オブジェクト解決プロセスで指示されています。document スタイルの場合のデフォルトのビジネス・オブジェクト解決には、最初の要素の本文の名前およびネーム・スペースが使用されるため、これらを、特定の要求、応答、または障害に関するすべての SOAP メッセージで同じ要素になるようにして、要求、応答、または障害のいずれの場合にも同じビジネス・オブジェクトに解決されるようにする必要があります。

注: SOAP メッセージが document/literal/type フォーマットまたは document/encoded/type フォーマットに従う場合、要素をオプショナルにしてはなりません。

XML の制約事項

以下の XML 構造、機能、および表記はサポートされません。

- 多次元配列
- 部分的に伝送される配列
- 疎配列
- 混合コンテンツ
- maxOccurs が 1 より大きいシーケンス、グループ、および選択モデル・グループ・コンポーネント

第 6 章 要求処理のためのコラボレーションの有効化

- 『要求処理コラボレーションのチェックリスト』

この章では、要求処理のためのコラボレーションを使用可能にする場合に実行する必要のある手順について説明します。コラボレーションは、コネクターを使用して Web サービスを呼び出します。

要求処理コラボレーションのチェックリスト

Business Object Designer Express を使用してビジネス・オブジェクトを生成することは、コラボレーション開発のプロセスの一部として行われます。コラボレーションが Web サービスを呼び出すために使用するビジネス・オブジェクトを生成するには、以下の作業を行う必要があります (これらについては、以下のセクションで説明します)。

1. URL、UDDI、ファイル・システムのいずれかから、WSDL 文書を識別する。このタスクにはサード・パーティー製のツールを使用してください。Web サービス・コネクターではこのタスクのためのツールは提供されません。
2. Business Object Designer Express を開いて WSDL ODA を立ち上げる。詳しくは、180 ページの『WSDL ODA の始動』を参照してください。
3. ODA を構成する。
4. 選択内容を確認する。
5. 要求ビジネス・オブジェクト、および (同期要求の場合に) 応答ビジネス・オブジェクトと障害ビジネス・オブジェクト、SOAP 構成 MO、プロトコル構成 MO、ヘッダー・コンテナー、およびそれぞれのオブジェクトと属性に適した子オブジェクトとアプリケーション固有情報を含む、トップレベルのビジネス・オブジェクトを生成する。このプロセスは、WSDL ODA によって自動化されます。

ビジネス・オブジェクトを生成した後で、コラボレーションがコネクターおよび SOAP データ・ハンドラーを使用して Web サービスを呼び出せるようにするために、いくつかの作業を行う必要があります。コラボレーション・テンプレートやオブジェクトの作成、そのポートのバインディングを含む、コラボレーションの開発手順については、「*コラボレーション開発ガイド*」を参照してください。汎用ビジネス・オブジェクトと WSDL ODA によって生成されるアプリケーション固有のビジネス・オブジェクトとの間のマップ作成の詳細については、「*マップ開発ガイド*」を参照してください。

第 7 章 Web サービスとしてのコラボレーションの公開

この章では、コラボレーションを Web サービスとして公開するための、設計段階における手順について説明します。これにより、Web サービス・クライアントがコラボレーションを呼び出したときにコネクターがイベントを処理できるようになります。

設計ツールが統合されているため、コラボレーションを Web サービスとして公開する作業を容易に行うことができます。Web サービス用にコラボレーションとビジネス・オブジェクトを構成した後で、WSDL 構成ウィザードを使用します。このウィザードは、コラボレーションを Web サービスとして表現する WSDL 文書および XML スキーマを作成します。WSDL 出力は、コラボレーションを記述するだけではなく、Web サービス・クライアントがコラボレーションを呼び出すための基礎を構成します。

手順のチェックリスト

コラボレーションを Web サービスとして公開するには、以下のセクションで説明する作業を行う必要があります。

1. 要求 SOAP メッセージ、およびオプションで (同期イベント処理のために) 応答および障害 SOAP メッセージとして使用されるビジネス・オブジェクトを識別するか、あるいは必要に応じて開発します。それらのオブジェクトを生成するには、以下の 2 つの方法があります。1) Business Object Designer Express を使用して手動で生成する方法。2) Web サービスに対して WSDL インターフェース・ファイルが存在する場合には、WSDL ODA を使用して、要求ビジネス・オブジェクト、その他の (応答または障害) ビジネス・オブジェクトを生成する方法。上記の 2 番目の方法を使用する場合は、以下の手順を実行します。
   a. Collaboration WSDL ODA 構成プロパティーで、コラボレーションの名前を指定します。この値は、TLO の ws_collab ASI を示します。
   b. WSDL インターフェース・ファイルに対して、WSDL_URL または UDDI_InquiryAPI_URL WSDL ODA 構成プロパティーを指定します (使用しているネットワークまたはローカルに WSDL インターフェース・ファイルが常駐する場合は、そのファイルに対するディレクトリー・パスを指定することもできます)。

詳しくは、 180 ページの『WSDL ODA の始動』を参照してください。

2. ビジネス・オブジェクトを使用するには、コラボレーションのテンプレートを開発するか、あるいは既存のテンプレートを選択します。
3. Web サービス用のコラボレーション・オブジェクトとそのポートを作成します。

   最初に、コラボレーション・オブジェクトがビジネス・オブジェクトを正しく取り込むことを確認しておく必要があります。コラボレーション・オブジェクトを作成するための詳しい情報およびステップごとの手順については、「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。

   注: コラボレーション・オブジェクトは、そのマップが適切な変換を実行できるように構成されていることが必要です。マップにより、SOAP 要求メッセージの形で受け取ったビジネス・オブジェクトは、コラボレーションが使用するビジネス・オブジェクトに変換されます。コラボレーションが返すビジネス・オブジェクトを SOAP 応答メッセージに組み込まれたビジネス・オブジェクトに変換するときにも、マップが使用されます。マッピングおよびマッピング手順について詳しくは、「*マップ開発ガイド*」を参照してください。
4. WSDL 構成ウィザードを使用して、WSDL 文書を作成します。このユーティリティーは、Web サービス・コネクターの構成も行います。

   注: WSDL 構成ウィザードは、インプリメンテーション・ファイル、インターフェース・ファイル、および 1 つ以上のスキーマ・ファイルを作成します。本書では、これらの出力を一括して WSDL 文書として参照します。
5. 必要に応じて WSDL 文書を公開します。

   注: コネクターは、WSDL 文書を公開するためのツールを提供しておらず、また公開のサポートもしません。

ビジネス・オブジェクトの識別または開発

ビジネス・オブジェクトを作成するには、Business Object Designer Express を使用し、ビジネス・オブジェクトをサポートするようにコネクターを構成するには、Connector Configurator Express を使用します。

Business Object Designer Express の詳細については、「*ビジネス・オブジェクト開発ガイド* 」を参照してください。Web サービス・ビジネス・オブジェクトについて詳しくは、 27 ページの『第 3 章 ビジネス・オブジェクトの要件』を参照してください。

コラボレーション・テンプレートの選択または開発

選択または開発するコラボレーション・テンプレートは、Web サービスとして公開するためのシナリオを 1 つまたは複数備えていなければなりません。コラボレーション・テンプレートの詳細については、「*コラボレーション開発ガイド*」を参照してください。

新規コラボレーション・オブジェクトのポートのバインディング

ビジネス・オブジェクト・タイプ用にコラボレーション・テンプレートのポートを構成した後で、コラボレーション・オブジェクトを作成し、そのポートを Web サービス・コネクターのインスタンスにバインドする必要があります。

新規コラボレーション・オブジェクトを作成してそのポートを Web サービス・コネクターのインスタンスにバインドするには、次のようにしてください。

1. 「コラボレーション・オブジェクト」フォルダーを右マウス・ボタンでクリックし、「新規コラボレーション・オブジェクトの作成」を選択します。これによって「新規コラボレーションの作成」ウィンドウが現れ、テンプレートのリストが示されます (図 55 を参照)。

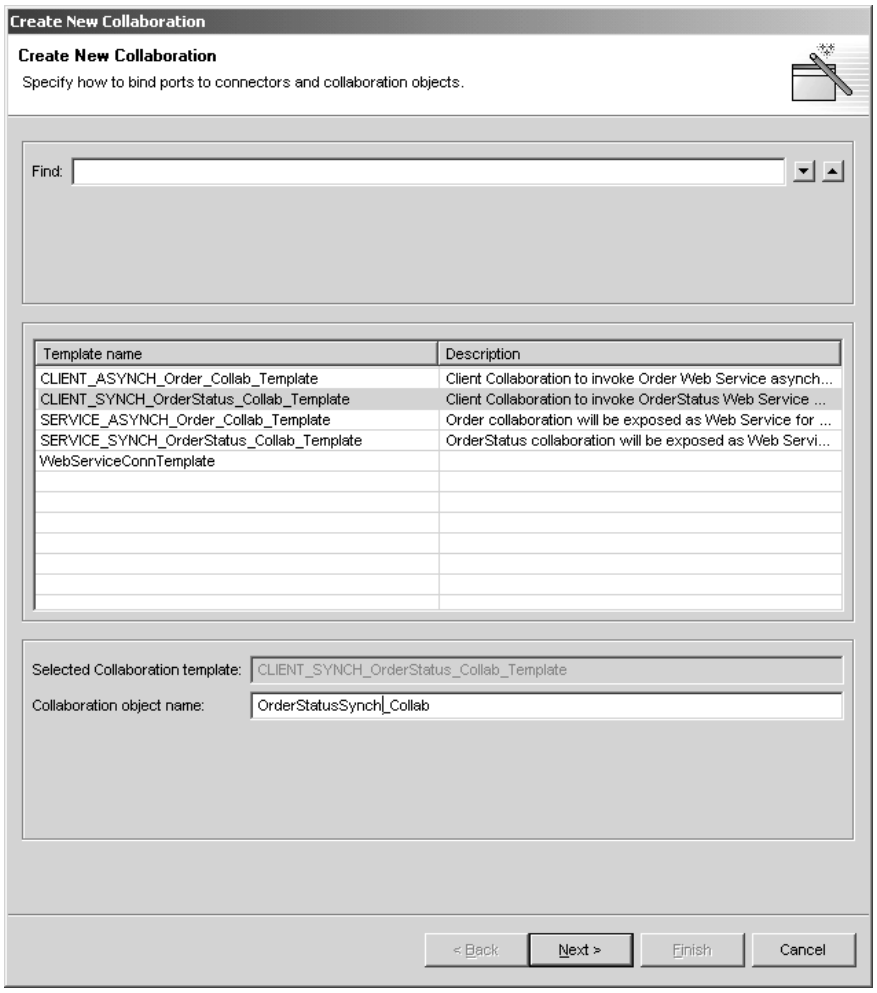


図 55. 「新規コラボレーションの作成」ウィンドウ

2. 「テンプレート名」からコラボレーション・テンプレートを選択し、「コラボレーション・オブジェクト名」フィールドにコラボレーション・オブジェクトの名前を入力します。これにより、図 56 に示すような「ポートをバインド」ウィンドウが表示されます。

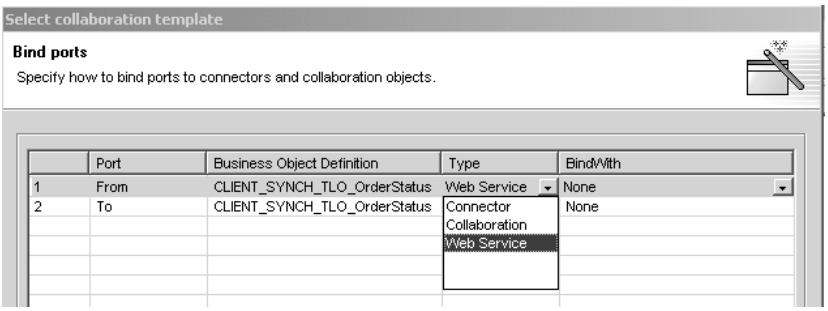


図 56. 「ポートをバインド」ウィンドウ

3. ポートを選択し、「タイプ」の矢印をクリックしてポートのプルダウン・メニューを表示して、「Web サービス」を選択します (図 56 を参照)。

   Web サービス・コネクターのすべてのインスタンスは、アプリケーション固有プロパティー ConnectorType を備えています。デフォルトでは、このプロパティーは `WebService` に設定されています。System Manager の「コラボレーション・ポートのバインド」ウィンドウでは、どのコネクターが Web サービス・コネクターかを判別するために、ConnectorType プロパティーの値が使用されます。
4. 「バインド:」の矢印をクリックしてコネクター・インスタンスのリストを表示します。System Manager は、ConnectorType プロパティーの値が `WebService` に設定されているコネクターのインスタンスを表示します。Web サービス・コネクターのインスタンスを選択してください。(図 57 に例が示されています。)

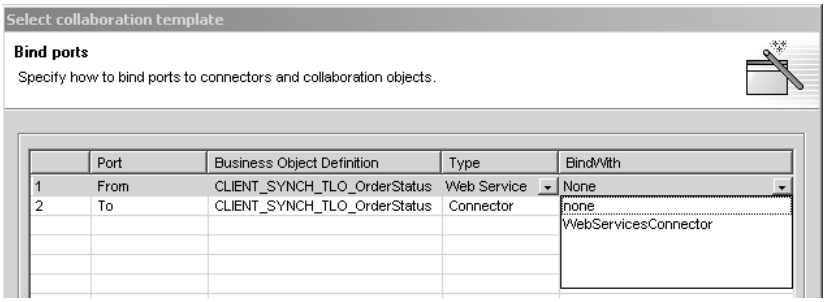


図 57. Web サービス・コネクターのインスタンスの選択

5. 「完了」をクリックします。

これで、WSDL 構成ウィザードを実行する準備ができました。

WSDL 構成ウィザード

コラボレーション・オブジェクトを作成して、そのトリガー・ポートを Web サービス・コネクターのインスタンスにバインドした後は、いつでも WSDL 構成ウィザードを使用することができます。このユーティリティーは、ユーザーがコラボレーション、ビジネス・オブジェクト定義、およびコネクターに関して指定した、バインディング、ポート名、操作、およびその他のデータを使用して、WSDL インプリメンテーション・ファイル (`*.impl.wsdl`)、WSDL インターフェース・ファイル (`*.wsdl`)、および xml スキーマ・ファイル (`*.xsd`) を作成します。これらのファイ

ルは、1 つの Web サービスとして公開されるコラボレーション複合体であり、このユーティリティーでは、これらを複数の別個のファイルとして生成するのか、1 つのファイルとして生成するのかを指定することができます。このユーティリティーは SOAP over HTTP、HTTPS、および JMS の各プロトコルをサポートします。プロトコル・リスナー・フレームワークの構成情報は、コネクター固有プロパティー ProtocolListenerFramework から検索されます。このプロパティーはまた、リスナーのリストを使用できるようにします。

ウィザードの実行

WSDL 構成ウィザードを実行するには、次のようにしてください。

1. Web サービス用に構成したコラボレーション・オブジェクトを右マウス・ボタンでクリックし、ポップアップ・メニューで「Web サービスとして公開」を選択します。図 58 に示すような WSDL 構成ウィザードが表示されます。

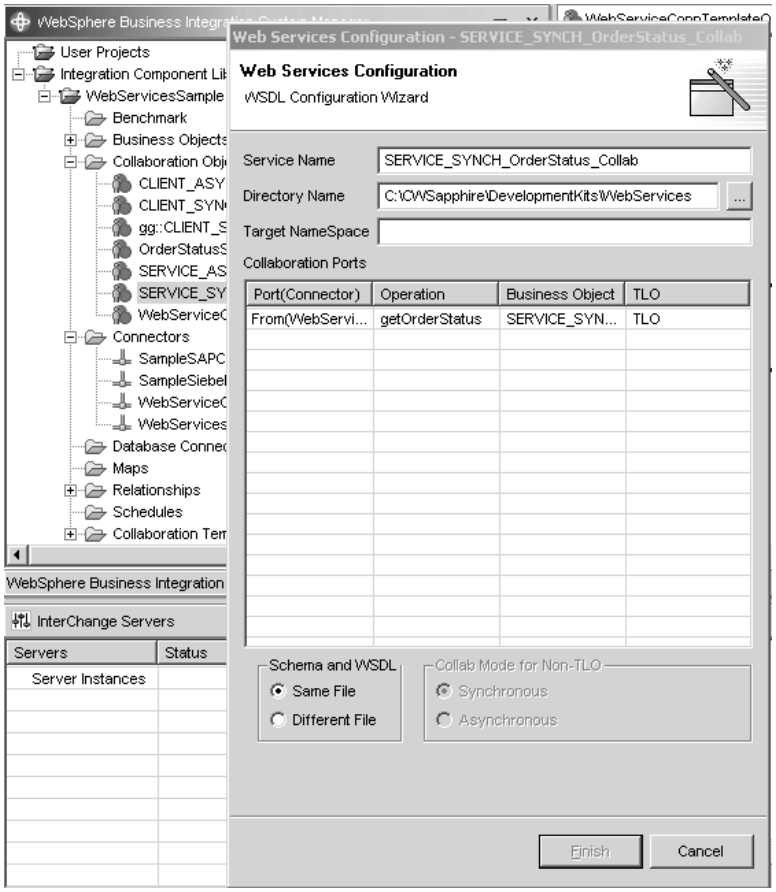


図 58. WSDL 構成ウィザード

図 58 でわかるように、各列は次のようになっています。

- **ポート (コネクター)** コラボレーション・オブジェクト上のトリガー・ポート。Web サービス・コネクターにバインドされています。ウィザードは、コラボレーション・オブジェクトからこの情報を取得します。
- **操作** ビジネス・オブジェクトが TLO である場合、ウィザードは、要求ビジネス・オブジェクトの SOAP 構成 Mo の BodyName 属性からこの情報を取

得します。ビジネス・オブジェクトが TLO ではない場合は、ウィザードはそのビジネス・オブジェクト名とポート名を組み合わせます。

- **ビジネス・オブジェクト** スキーマの作成に使用されます。ウィザードは、コネクターがサポートするこのトリガー・ポート用のビジネス・オブジェクトから、この情報を取得します。

2. 必要に応じて以下の値を入力します。
   - **サービス名** デフォルトでは、このコラボレーション・オブジェクトを記述するためにユーザーが使用した名前。
   - **ディレクトリー名** Web サービスのためのアダプターとコラボレーションのテンプレートおよびオブジェクトが入るディレクトリー。
   - **ターゲット・ネーム・スペース** Web サービスとして公開されるコラボレーションの URL。
   - **コラボレーション・ポート** これらのフィールド内の情報は、コラボレーション・オブジェクト構成手順の「ポートをバインド」ウィンドウで指定されます。
   - **TLO 以外の場合のコラボレーション・モード** これは、TLO を使用している場合には適用されません。あるいは、非 TLO オブジェクトを入力用に使用している場合には、synchronous または asynchronous を指定しなければなりません。
   - **スキーマおよび WSDL** これらの出力を単一ファイルに収めるのか、別個のファイルに収めるのかを指定してください。
3. 「完了」をクリックします。このユーティリティーは、ユーザーが入力した入力データおよび指定内容に基づいて出力を生成します。次のセクションに、これらすべての要約を示します。

TLO フォーマットのビジネス・オブジェクトの WSDL 構成ウィザード処理

構成ウィザードは、Web サービス・コネクターにバインドされるコラボレーション・オブジェクトの各トリガー・ポート用に、WSDL オペレーションを作成します。このオペレーションは、このコラボレーションの呼び出しに関連付けられているビジネス・オブジェクトに基づいて作成されます。

構成ウィザードは、オブジェクト・レベル ASI の `ws_eventtlo` を読み取って、そのビジネス・オブジェクトが TLO フォーマットであるかどうかを判断します。ASI プロパティーが `true` に設定されていれば、そのビジネス・オブジェクトは TLO です。TLO によって、以下の WSDL プロパティーが検出されます。

- **Operation Name および BodyNS** ウィザードは、TLO フォーマットのビジネス・オブジェクトを検出すると、その TLO の SOAP 要求ビジネス・オブジェクト内の SOAP 構成 MO の BodyName プロパティーを使用して、オペレーション名を作成します。同様に、ウィザードは、同じ SOAP 構成 MO 内の BodyNS プロパティーをメッセージのネーム・スペースであると判断します。
- **Execution Mode** TLO のビジネス・オブジェクト・レベル ASI から `ws_mode` プロパティーをインスペクションすることにより、ウィザードは、実行モードが同期か非同期かを判断して、それぞれ REQUEST_RESPONSE または ONE_WAY WSDL を作成します。

TLO に基づいて WSDL オペレーションを作成する場合、コラボレーションの構成には、マップを使用する方法と使用しない方法の 2 つの方法があります。

マップを使用する TLO: コラボレーションは、一般に、汎用ビジネス・オブジェクト (GBO) 要求を受け入れるように構成されています。つまり、コラボレーション・テンプレートのトリガー・ポートは、GBO にサブスクライブしています。このような状況で TLO を使用するには、コラボレーションが Web サービス・コネクターにバインドされていることと、コネクターがマップを介した GBO の TLO への変換をサポートしていることが必要です。図 59 に、この例を示します。

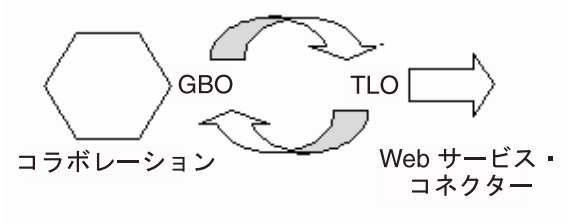


図 59. マップを使用する TLO

コラボレーションとコネクターがこの方法で構成されている場合、ウィザードは、WSDL 文書で記述されているオペレーションの作成に TLO ビジネス・オブジェクトが使用されると判断します。この判断は、コネクターでサポートしているビジネス・オブジェクトとそれに関連したマップをインスペクションすることによって行われます。Web サービス・コネクターのランタイム処理にとっては、構成されたマップが常にコラボレーションの GBO を 1 つの TLO だけに変換することが重要です。また、インバウンド・マップのソース・ビジネス・オブジェクトと宛先ビジネス・オブジェクトが、それぞれ、アウトバウンド・マップの宛先ビジネス・オブジェクトとソース・ビジネス・オブジェクトに変換されることも重要です。

マップを使用しない TLO: ウィザードは、マップを使用しない TLO の処理もサポートしています。この場合、コラボレーション・テンプレートのトリガー・ポートは TLO に直接サブスクライブします。Web サービス・コネクターは TLO をサポートしているため、マップは必要ありません。図 60 は、この例を示したものです。

図 60. マップを使用しない TLO

コラボレーションとコネクターがこの方法で構成されている場合、ウィザードは、コラボレーションで検出される TLO ビジネス・オブジェクトを使用して、WSDL 文書で記述されているオペレーションを作成します。ウィザードは、このポート用に構成されているマップは存在しないと判断します。

非 TLO フォーマットのビジネス・オブジェクトの WSDL 構成ウィザード処理

非 TLO のビジネス・オブジェクトがサポートされていれば、既存のコラボレーションおよびマップを使用して Web サービスとして公開することができます。このため、ウィザードは、TLO フォーマット以外のビジネス・オブジェクトの使用による WSDL オペレーションの作成もサポートしています。

TLO プロセスと同様、ウィザードは、オブジェクト・レベル ASI の `ws_eventtlo` を読み取って、そのビジネス・オブジェクトが非 TLO フォーマットであると判断します。 ASI プロパティーが存在しないか、存在しても `true` 以外の値に設定されている場合は、このビジネス・オブジェクトは非 TLO です。非 TLO は、Web サービスの TLO 構造に準拠していない任意のビジネス・オブジェクトです。ウィザードは、非 TLO を使用して以下のプロパティーを検出します。

- **Operation Name および BodyNS** ウィザードは、非 TLO フォーマットのビジネス・オブジェクトを検出すると、コラボレーション名、ビジネス・オブジェクト名、およびポート名の組み合わせを使用して、オペレーション名を作成します。WSDL オペレーションの Body Namespace は、WSDL 構成ウィザードの「ターゲット・ネーム・スペース」項目を使用して構成されます。
- **WSCollaborations** ウィザードは、Web サービス・コネクターのプロパティーの階層を作成します。この階層には、BO Name、SOAP Body Name、SOAP Body Namespace、および Web サービスとして公開されるコラボレーションのポートにおける各 WSDL オペレーションの Mode などのプロパティーが含まれます。図 61 に、WSCollaborations プロパティーのサンプルを示します。

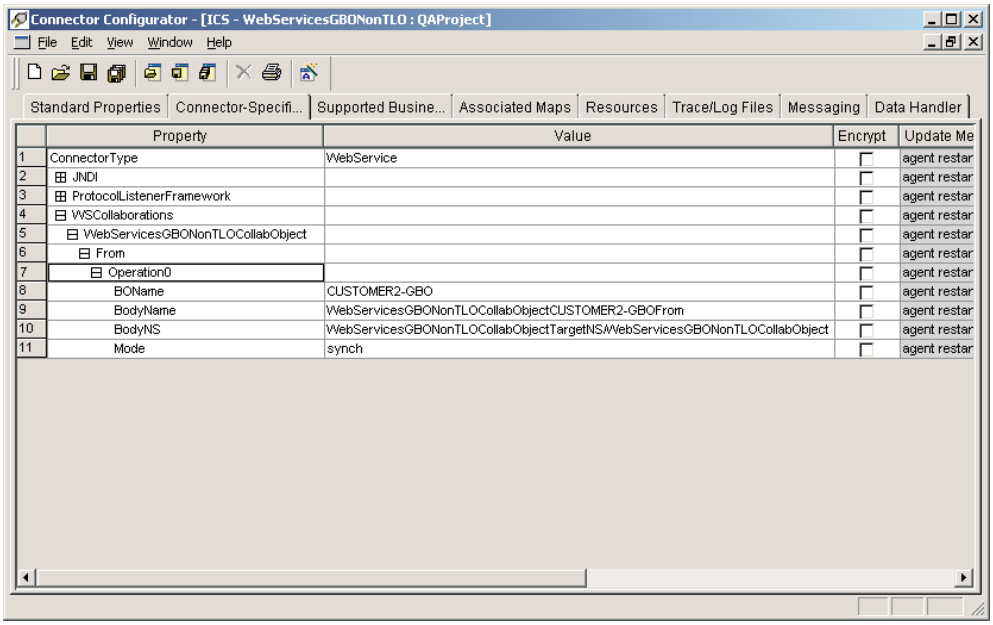


図 61. WSCollaborations

- **Execution Mode** WSDL オペレーションの Execution Mode は、WSDL 構成ウィザードの「TLO 以外の場合のコラボレーション・モード」選択ボタンを使用して構成されます。

非 TLO に基づいて WSDL オペレーションを作成する場合、コラボレーションの構成には、マップを使用する方法と使用しない方法の 2 つの方法があります。

マップを使用する非 TLO: コラボレーションは、一般に、汎用ビジネス・オブジェクト (GBO) 要求を受け入れるように構成されています。また、GBO をコラボレーションから非 TLO ビジネス・オブジェクトに変換する既存のマップが存在する場合があります。図 62 に、この例を示します。

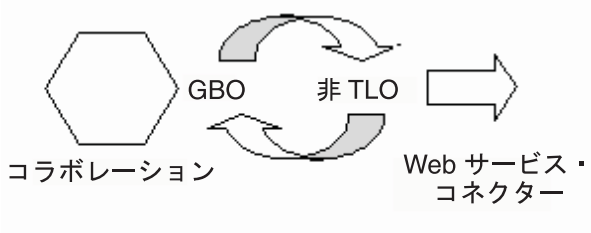


図 62. マップを使用する非 TLO

この場合、ウィザードは非 TLO のビジネス・オブジェクトを使用して、WSDL 文書に記述されている WSDL オペレーションを作成します。Web サービス・コネクターのランタイム処理にとっては、構成されたマップが常にコラボレーションの GBO を 1 つの非 TLO だけに変換することが重要です。また、インバウンド・マップのソース・ビジネス・オブジェクトと宛先ビジネス・オブジェクトが、それぞれ、アウトバウンド・マップの宛先ビジネス・オブジェクトとソース・ビジネス・オブジェクトに正確に変換されることも重要です。

マップを使用しない非 TLO: 高度に専門化されたケースでは、コラボレーションが、GBO 以外のビジネス・オブジェクトからの要求を受け入れるように構成されている場合があります。この場合、非 TLO はコラボレーション用の直接のビジネス・オブジェクトであり、マップは存在しません。図 63 に、この例を示します。

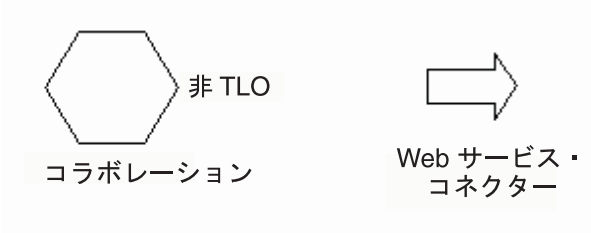


図 63. マップを使用しない非 TLO

この場合、ウィザードは、このポートではマップが構成されていないと判断し、非 TLO のビジネス・オブジェクトを使用して、WSDL 文書に記述されている WSDL オペレーションを作成します。

処理の要件および例外

以下のセクションで説明する WSDL 構成ウィザードの要件は、異なる指示が明記されていないかぎり、すべてのタイプのオブジェクト (TLO および非 TLO) に適用されます。Web サービス TLO に関するビジネス・オブジェクト要件の詳細については、27 ページの『第 3 章 ビジネス・オブジェクトの要件』を参照してください。

注: WSDL ツールが読み取るビジネス・オブジェクト ASI のうち、国際化対応した文字を使用できるのは以下のもののみです。

- elem_name
- elem_ns
- attr_name
- attr_ns
- BodyName
- BodyNS
- type_name
- type_ns

SOAP 構成 MO での Use プロパティーのサポート: WSDL 構成ウィザードは、SOAP 構成 MO の Use プロパティーをサポートしますが、SOAP 要求 BO と対応する SOAP 応答 BO の Use 値が異なる場合はエラーをスローします。Use 値を literal または encoded に設定して、WSDL 文書を生成できます。Use プロパティーとその値に関する詳細については、130 ページの『SOAP メッセージに対する Style と Use の影響』を参照してください。

SOAP 構成 MO でのスタイルのサポート: コラボレーションを Web サービスとして公開する場合は、rpc スタイルのみがサポートされます。Style を SOAP 構成 MO の文書として指定すると、ウィザードはエラーをスローします。

障害処理: SOAP 障害ビジネス・オブジェクト内の details 属性は、子属性を 1 つだけ備えることができます。それ以外の場合、ユーティリティーはエラーを生成します。

ユーティリティーは、障害ビジネス・オブジェクトを受け入れます。複数の障害ビジネス・オブジェクトを検出した場合、ユーティリティーは、最初の、あるいはデフォルトの障害ビジネス・オブジェクトのヘッダー・コンテナーを処理します。処理は次のように行われます。

- バインディング・セクション内の soap:fault 要素については、Namespace は指定されません。
- Fault は常に、document スタイルおよび用途 literal を使用して指定されます。
- Message 部分は、element 属性を使用して指定されます。

ヘッダー障害の処理: ヘッダー障害は、WSDL 文書のバインディング・セクションで、soap:header の子要素である soap:headerfault として処理されます。ヘッダー障害の処理は、次のように、ヘッダー子ビジネス・オブジェクト内で指定された `headerfault` ASI を使用して行われます。

- soap:headerfault 要素については、Namespace は指定されません。
- ヘッダー障害は常に、document スタイルおよび用途 literal を使用して指定されます。
- Message 部分は、type 属性ではなく element 属性を使用して指定されます。

ヘッダーの処理: SOAP ヘッダー・コンテナー・ビジネス・オブジェクト内では、SOAP ヘッダー子ビジネス・オブジェクトとして複数のヘッダー属性が指定されます。ヘッダー・コンテナー・ビジネス・オブジェクトは、その ASI によって、

soap_location=SOAPHeader のように識別されます。ユーティリティーによる処理の際に、バインディング・セクション内で、ヘッダー・コンテナー・ビジネス・オブジェクト内のそれぞれの属性ごとに soap:header 要素が作成されます。この場合、以下の規則が適用されます。

- ヘッダーは常に、document スタイルおよび用途 literal を使用して指定されます。
- Message 部分は、type 属性ではなく element 属性を使用して指定されます。
- elem_ns が指定されていない場合、ヘッダーは Body Namespace に書き込まれます。

注: ヘッダー・コンテナー・ビジネス・オブジェクトは、SOAP 要求、応答、または障害ビジネス・オブジェクトの子です。soap:header 要素については、namespace 属性は指定されません。

elem_ns ASI の処理: このユーティリティーは、メッセージ・パーツ・レベルの elem_ns ASI を無視します。その代わりに、elem_ns は 2 次レベルおよび低レベルの属性で使用されます。elem_ns が指定される場合、2 次レベルのビジネス・オブジェクト属性を別個のネーム・スペース内で定義することができます。

JMS プロトコルの処理: WSDL 文書のポート・セクションにおける SOAP/JMS バインディングには、jms:address 要素が含まれます。jms:address 要素の例を次に示します。("?" という接尾部が付いた属性はオプションです。)

```
    <jms:address
                    destinationStyle = "queue"
jmsVendorURI = "http://ibm.com/ns/mqseries"?
initialContextFactory = "com.ibm.NamingFactory"?
jndiProviderURL = "iiop://something:900/wherever"?
        jndiConnectionFactoryName = "orange"
        jndiDestinationName = "fred"

        jmsProviderDestinationName="trash" />
```

LookupQueuesUsingJNDI コネクター・プロパティーが true に設定されている場合、InputQueue プロパティーの値は、SOAP/JMS バインディングの jms:address 要素の jndiDestinationName 属性に対応します。jms:address 要素は、wsdl:port セクションで指定されています。LookupQueueUsingJNDI の値が false に設定されている場合には、jmsProviderDestinationName 属性の値は InputQueue に設定されます。InputQueue は、Listener_JMS 階層プロパティーの下で利用できるコネクター・プロパティーです。initialContextFactory、jndiProviderURL、および jndiConnectionFactoryName の各プロパティーは、同期処理用にのみ指定されます。

HTTP プロトコルの処理: WSDL 文書のサンプル・ポート・セクションを以下に示します。

```
<service name="StockQuoteWebService">
<port name="StockQuoteWebServicePort" binding="intf:StockQuoteBinding">
<soap:address location="http://localhost:8080/wbia/webservices/stockquoteservice"/>
</port>
</service>
```

WSDL 構成ウィザードは、コンテキスト・パス内のホスト名とポートの値を使用します。コンテキスト・パスに含まれているのがホスト名もポートもない相対パスの

みである場合は、Listener_HTTP 構成プロパティー下にあるホスト名およびポート・プロパティーの値が、soap:address xml 要素内で location 属性を指定するのに使用されます。

第 8 章 WSDL ODA の使用

注: Web Services Description Language (WSDL) Object Discovery Agent (ODA) は、要求処理用のビジネス・オブジェクトおよび、WSDL インターフェース・ファイルが使用可能な場合に、イベント処理用のビジネス・オブジェクトを生成するために使用されます。

コラボレーションは、コネクターを使用して Web サービスを呼び出します。あるいは、コラボレーションは、Web サービスとして公開することができます。Web サービスは WSDL (Web Services Description Language) を使用して記述されています。この章では、Web Services Description Language (WSDL) Object Discovery Agent (ODA) を使用してビジネス・オブジェクトを生成する方法について説明します。コラボレーションが Web サービスを呼び出すとき、およびコラボレーションを Web サービスとして公開するときに、コネクターおよび SOAP データ・ハンドラーは、これらのビジネス・オブジェクトを使用します。

以下の 2 つの目的のために、WSDL ODA を使用して、ビジネス・オブジェクトを生成します。

1. WSDL ODA は、WSDL インプリメンテーション・ファイルを使用して、コラボレーション用にビジネス・オブジェクトを生成して、外部 Web サービスを起動することができます。
2. WSDL ODA は、WSDL インターフェース・ファイルを使用して、Web サービスとして公開されるコラボレーション用にビジネス・オブジェクトを生成することができます。

WSDL ODA は、Business Object Designer Express を使用するときに立ち上げることができます。WSDL ODA は WSDL 文書を読み取り、コネクターおよび SOAP データ・ハンドラーによって必要とされるビジネス・オブジェクトを作成します。WSDL ODA を使用すると、ビジネス・オブジェクト開発の作業が簡略化されます。

注: WSDL ODA は、WSDL における SOAP/HTTP バインディングおよび SOAP/JMS バインディングを扱います。

WSDL ODA の始動

WSDL ODA は、以下のスクリプトのいずれかを使用して始動することができます。

- Windows
  - start_WSDLODA.bat

    注: WSDL ODA は、インストーラーが Windows 環境用に自動的に作成するショートカットを使用して始動することもできます。
- Linux
  - start_WSDLODA.sh
- i5OS の場合、次のいずれかの方法を使用します。
  - WBI SE Console for i5OS がインストールされている Windows システムから、「**プログラム**」>「**IBM Websphere Business Integration Server Express Console**」>「**コンソール**」を選択します。続いて、OS/400 システム名および i5/OS システム名、または IP アドレスと、*JOBCTL の特殊権限を持つユーザー・プロファイルおよびパスワードを指定します。ODA のリストから ODA を選択して、「ODA を始動」ボタンを選択します。
  - ODA をバッチ・ジョブとして開始するために、i5/OS コマンド行から CL コマンド QSH を実行し、QSHELL 環境から次のコマンドを実行します。
    ```
    /QIBM/ProdData/WBIServer44/bin/submit_oda.sh pathToODAStartScript
    jobDescriptionName
    ```

    ここで、`ToODAStartScript` は ODA 始動スクリプトの絶対パスで、*jobDescriptionName* は QWBISVR44 ライブラリーで使用するジョブ記述の名前です。
  - ODA を非バッチ・ジョブとして開始するために、i5/OS コマンド行から CL コマンド QSH を実行して、QSHELL コマンド入力から ODA 始動スクリプトを直接実行します。
    ```
    start_ODAName.sh
    ```

WSDL ODA の選択、構成、および実行は、Business Object Designer Express を使用して行います。Business Object Designer Express は、個々の ODA を各 Linux スクリプト・ファイル (start_WSDLODA.sh) または Windows バッチ・ファイル (start_WSDLODA.bat) の AGENTNAME 変数で指定された名前で探し出します。

WSDL ODA の実行

Object Discovery Agent (ODA) は、要求処理のためにビジネス・オブジェクトを構築する作業を簡略化します。Business Object Designer Express は、使用可能なすべての ODA に対するグラフィカル・インターフェースを備えているため、ユーザーは、必要なエージェントを見付けやすくなります。WSDL ODA は、デフォルトでは WSDLODA という名前になっています。WSDL Wizard で表示される名前は、`start_WSDLODA.bat` または `start_WSDLODA.sh` ファイル内の `AGENTNAME` 変数の値によって決まります。ODA およびビジネス・オブジェクト定義の詳細、および ODA

の構成方法、開始方法、使用法については、「*ビジネス・オブジェクト開発ガイド*」を参照してください。下記の手順を実行する際に、必要に応じてこの資料を参照することをお勧めします。

Object Discovery Agent を始動した後で、以下のステップに従って WSDL ODA を立ち上げてください。

1. Business Object Designer Express を開きます。
2. 「ファイル」メニューから「ODA を使用して新規作成...」サブメニューを選択します。Business Object Designer Express が Business Object Wizard で「エージェントの選択」ダイアログ・ボックスを表示します。図 64 はこのウィンドウを示しています。
3. 「エージェントの検索」ボタンをクリックして実行中のすべてのエージェントを表示し、WSDL ODA を選択します。

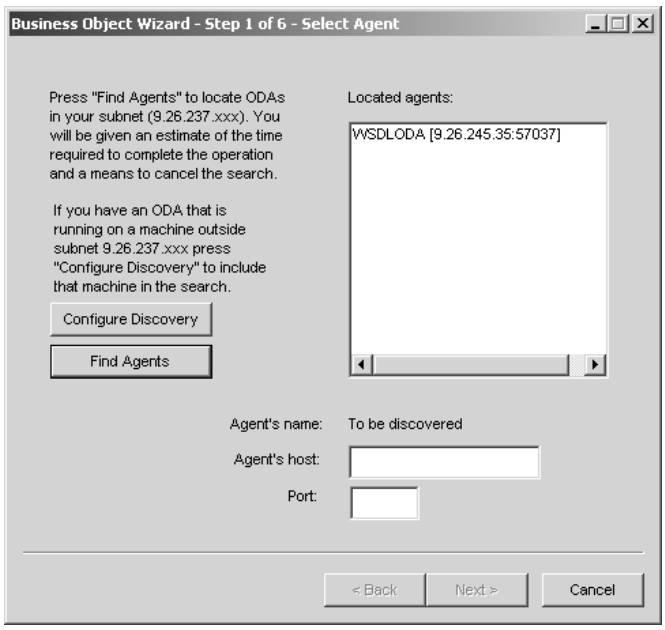


図 64. 「エージェントの選択」ウィンドウ

Business Object Designer Express で WSDL ODA が見付からない場合には、ODA のセットアップを見直してください。

4. 「検索されたエージェント」ペインのリストから WSDL ODA を選択し、「次へ」をクリックします。

これによって「エージェントの構成」ウィザード・ウィンドウが表示され、ユーザーが指定する必要のある構成プロパティーが示されます。

エージェントの構成

エージェントは、WSDL ODA Business Object Wizard の「エージェントの構成」ウィンドウで構成されます。表 48 には、WSDL ODA のために構成する必要のあるプロパティーがリストされています。

注: 初めて WSDL ODA を使用する場合には、それぞれの各構成プロパティーごとに値を指定する必要があります。それらの値を指定した後で、「保管」ボタンをクリックして、プロパティー値をプロファイルに保管することができます。次回 WSDL ODA を使用するときには、「現在のプロファイル」ボックスから保管したプロファイルを選択することができます。

表 48. WSDL ODA 構成プロパティー

プロパティー	型	必須	デフォルト	説明
WSDL_URL	String	UDDI を指定していないときには必須	なし	WSDL 文書の URL。この値として、ローカル WSDL ファイルへの絶対パスを指定することもできます。URL はネイティブ言語で指定することができます。このプロパティーは、双方向言語の変換を行う場合に使用可能にします。
UDDI_InquiryAPI_URL	String	UDDI の場合には必須	なし	UDDI 照会 API の URL。
WebServiceProvider	String	UDDI の場合には必須	なし	ターゲット Web サービス・プロバイダーの名前。これは通常、UDDI レジストリーで公開された Business 名です。この項目では大/小文字の区別があり、英字のみを使用することができます。
WebService	String	UDDI の場合には必須	なし	Web サービスの名前。この項目では大/小文字の区別があり、英字のみを使用することができます。
MimeType	String	いいえ	xml/soap	コネクターが呼び出すデータ・ハンドラーの MIME タイプ。これはビジネス・オブジェクト TLO でデフォルト値として設定される値であり、英字のみで指定しなければなりません。
BOPrefix	String	いいえ	SOAP_	これは、作成されるすべてのビジネス・オブジェクトの前に付加されます。ユーザー構成可能 (英字のみ)。8 文字以下で指定します。

表 48. WSDL ODA 構成プロパティー (続き)

プロパティー	型	必須	デフォルト	説明
BOVerb	String	はい	Create	要求ビジネス・オブジェクト (およびオプションにより応答ビジネス・オブジェクトと障害ビジネス・オブジェクト) の SOAP 構成 MO で設定される動詞。
Collaboration	String	いいえ	なし	この値は、TLO の ws_collab ASI を示し、イベント処理用にオブジェクトを生成する場合には必須となります。
GenerateUniqueBOs	String	いいえ	なし	このプロパティーが true の場合、ビジネス・オブジェクト名はすべての Web サービス全体で固有になります。このプロパティーが false の場合、同じパーツ・タイプを持つ操作の間でビジネス・オブジェクトを再利用できます。
SOAPVersion	String	いいえ	なし	ビジネス・オブジェクトの生成に使用する SOAP 標準を指定します。指定可能な値は 1.1 および 1.2 です。
BiDi.ExtApplicationMetaData	String	はい	ILYNN	必要な場合に、WSDL_URL プロパティーの双方向の形式を指定します。このプロパティーを指定しない場合、デフォルト値では、WSDL_URL の双方向処理はアクティブになりません。

次のセクションでは、「エージェントの構成」ウィンドウで WSDL 文書を指定する方法を説明します。

WSDL 文書の指定

Web サービス・ビジネス・オブジェクトは WSDL 文書から生成されます。このセクションでは、ODA の「エージェントの構成」ウィンドウで WSDL 文書のソースを選択および指定する方法を示します。

WSDL 文書はローカル・ファイル・システムに置かれることも、Web の URL ロケーションまたは UDDI レジストリーに置かれることもあります。ユーザーが WSDL 文書の置かれた場所を指定すると、WSDL ODA がそれを検索します。(完全な WSDL サービスは、複数の文書にわたって記述されることがあります。)

URL ロケーションからの WSDL 文書の入手

1. 構成プロパティー WSDL_URL で WSDL 文書の URL を指定します。

   これにより、ODA が WSDL 文書から Web サービスのリストを検索し、インポートされる文書の URL を解決します。WSDL_URL プロパティーでは、URL 構文 (例えば `file:///C:/test.wsdl`) または絶対パス (例えば `C:¥test.wsdl`) を使用して、ローカル・ファイル・システムにある WSDL ファイルのロケーションを指定することもできます。ODA がこの文書およびその依存関係 (インポートされるすべてのファイル) にアクセスできるようになっていなければなりません。

   WSDL_URL プロパティーは、双方向言語の変換を行う場合に使用可能にします。
2. 「次へ」をクリックします。

   ODA は、Web サービス・プロバイダーの URL を照会し、この URL ロケーションから WSDL で定義されるサービスのリストを取得して、図 65 に示すような展開されたポートの操作のリストを表示します。

   注: WSDL ODA は、SOAP/JMS または SOAP/HTTP バインディングを備えたポートのみを表示し、それ以外のタイプのバインディングは除外します。

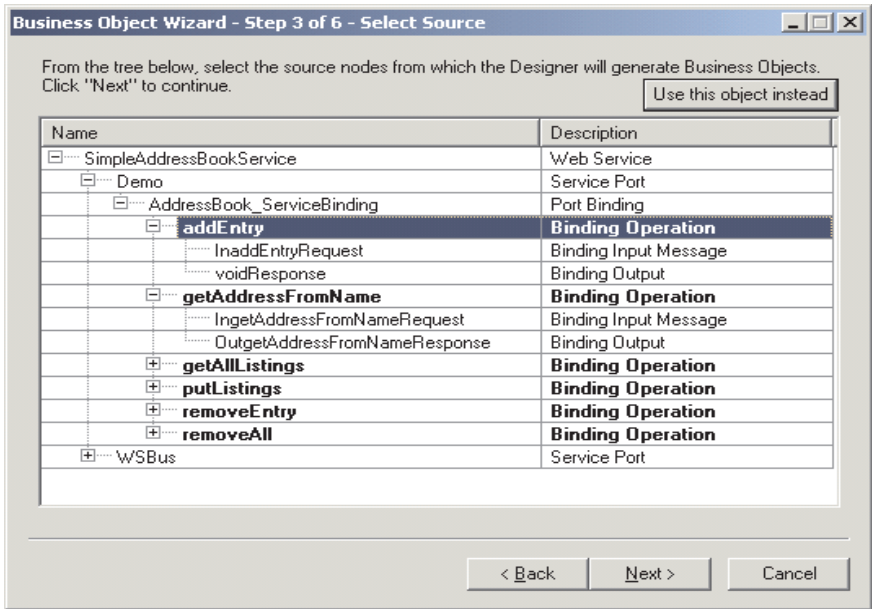


図 65. 「ソースの選択」ウィンドウ

3. ポートのリストから、1 つだけオペレーションを選択します (選択可能なオペレーションが強調表示されています)。サービスまたはポート・ノードを選択する

ことはできません。これらは表示専用です。WSDL オペレーションのタイプとしては、ONE_WAY、REQUEST_RESPONSE、SOLICIT_RESPONSE、および NOTIFICATION があります。 WSDL ODA がサポートおよび表示するのは、REQUEST_RESPONSE オペレーションと ONE_WAY オペレーションのみです。

4. 「次へ」をクリックして 186 ページの『選択内容の確認』に進みます。

UDDI レジストリーからの WSDL 文書の入手

ODA は、URL ロケーションからではなく、UDDI レジストリーからも WSDL 文書を検索することができます。そのためには、次のようにする必要があります。

1. 「エージェントの構成」ウィンドウで、「検索キー」として以下のプロパティーを指定します。
   - UDDI_InquiryAPI_URL (例えば、https://uddi.ibm.com/ubr/inquiryapi)
   - WebServiceProvider (例えば、IBM Corporation)
   - WebService (例えば、StockQuoteService)
   - WSDL ODA は、UDDI レジストリー内で完全に一致する名前 (findQualifier) を照会に使用します。パラメーターの値が正しく入力されていることを確認してください。正規の UDDI ブラウザーを使用すると、サービス・プロバイダーが提供するサービスを検出できます。

   WSDL ODA はこれらのプロパティー (表 48 で説明されています) を使用して UDDI レジストリーに接続します。
2. 「次へ」をクリックします。

   ODA は、UDDI レジストリーに対して Web サービス・プロバイダーに関する照会を行い、ユーザーが指定した Web サービス・パラメーターに一致するサービスのリストを検索します。WSDL ODA は、Web サービス・プロバイダーによって提供されているサービスのリストを、図 65 に示すようなウィンドウに表示します。UDDI 照会が複数の一致を戻す場合、WSDL ODA は、アンダースコアー (_) とシーケンス番号を付加してそれらを表示します。例えば、StockQuoteService_1、StockQuoteService_2、などのようになります。

   注: WSDL ODA は、SOAP/JMS または SOAP/HTTP バインディングを備えたポートのみを表示します。
3. ポートのリストから、1 つだけオペレーションを選択します。サービスまたはポート・ノードを選択することはできません。これらは表示専用です。WSDL オペレーションのタイプとしては、ONE_WAY、REQUEST_RESPONSE、SOLICIT_RESPONSE、および NOTIFICATION があります。 WSDL ODA がサポートおよび表示するのは、REQUEST_RESPONSE オペレーションと ONE_WAY オペレーションのみです。
4. 「次へ」をクリックして 186 ページの『選択内容の確認』に進みます。

注: このコネクターがサポートしているのは UDDI バージョン 2 の API のみです。したがって UDDI バージョン 2 をサポートしていない UDDI レジストリーから WSDL の検索はできません。

選択内容の確認

Web サービス・オペレーションのソースを選択すると、WSDL ODA Business Object Wizard は図 66 に示すような確認画面を表示します。

1. 選択内容を確認します。
2. 「次へ」をクリックして 『オブジェクトの生成』に進みます。

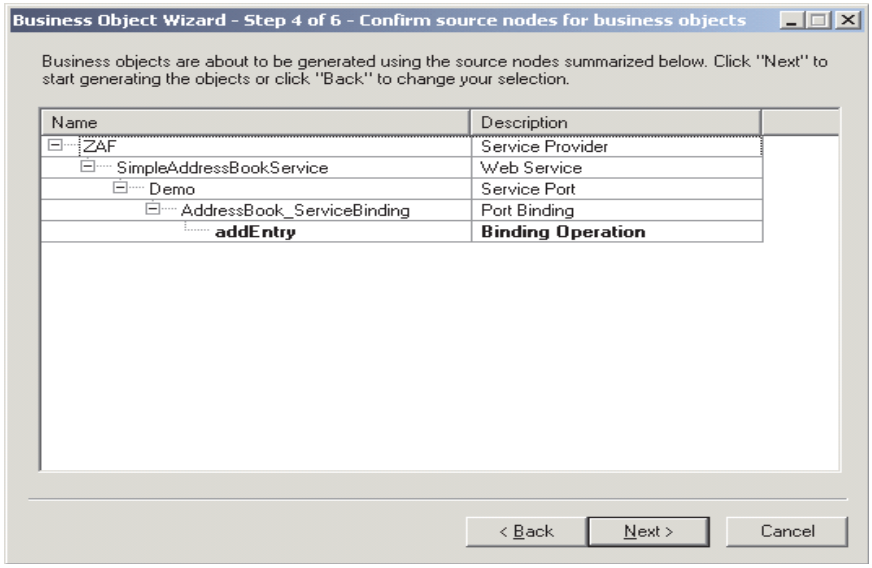


図 66. 「確認」ウィンドウ

オブジェクトの生成

ユーザーが、WSDL 文書ソースを確認すると、起動する必要のある Web サービスまたは Web サービスとして公開する必要のあるコラボレーション用に、ビジネス・オブジェクトとメタオブジェクトが、WSDL ODA によって生成されます。オブジェクトの保管については、 187 ページの図 67 を参照し、以下のステップを実行してください。

注: WSDL ODA は、トップレベルのビジネス・オブジェクトのキー属性を自動的に選択することはできません。それ以外のレベルのビジネス・オブジェクトでは、WSDL ODA は最初の属性をキーに設定します。したがって、Business Object Designer Express で WSDL ODA が生成したオブジェクトを保管すると、トップレベルのオブジェクトにキー属性が欠落していることを知らせるエラー・メッセージが出されます。ビジネス・データとビジネス・オブジェクトの要件を満たすキー属性を割り当ててから、オブジェクトを再度保管してください。キー属性を選択する際は注意が必要です。キー属性はイベントの順序付けに使用されるので、慎重に選択しないと、パフォーマンス上の問題が発生します。

1. 「ビジネス・オブジェクトをファイルに保管」、または「別のウィンドウで新規ビジネス・オブジェクトを開く」にチェックマークを付けます。後者を選択すると、Business Object Designer Express が立ち上がり、そのアプリケーションでビジネス・オブジェクトが開きます。
2. ODA の実行を続けたくない場合は、「ODA をシャットダウン」にチェック・マークを付け、「完了」をクリックします。それ以外の場合は、単に「完了」をクリックします。ODA は次のビジネス・オブジェクトを生成できる状態になります。

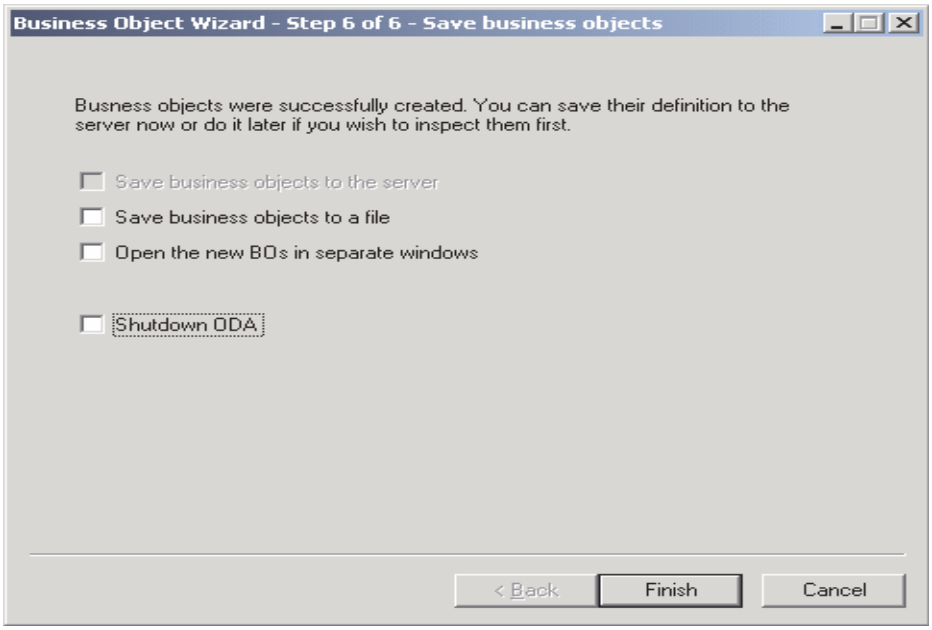


図 67. 「保管」ウィンドウ

要求処理の場合、Web サービスに対する呼び出しには、要求を設定する必要があり、また、同期処理では、応答メッセージと障害メッセージも設定する必要があります。イベント処理の場合、公開されるコラボレーションには要求を設定する必要があり、また、同期処理では、応答メッセージと障害メッセージも設定する必要があります。WSDL ODA は、これらのそれぞれについて、各レベルのアプリケーション固有情報 (ASI)、SOAP データ・ハンドラー、およびプロトコル構成 MO を含む、ビジネス・オブジェクトを生成します。WSDL 文書における SOAP バインディングにより、SOAP メッセージの構造が決定します。ビジネス・オブジェクト構造の詳細については、27 ページの『第 3 章 ビジネス・オブジェクトの要件』を参照してください。

制約事項

表 49 は、WSDL および XML スキーマでの、スタイル、使用法、および型または要素を使用したパーツ定義のさまざまな組み合わせに対する、WSDL ODA のサポート状況を示しています。

表 49. WSDL ODA の制約事項

Style/Use/Parts	説明
rpc/encoded/type	サポートあり
rpc/encoded/element	サポートあり
rpc/literal/type	サポートあり
rpc/literal/element	サポートあり
doc/encoded/type	サポートなし
doc/encoded/element	サポートなし
doc/literal/type	サポートあり
doc/literal/element	サポートあり

WSDL ODA は (1 つのファイルで) 完全に自己完結した WSDL ファイルか、または、サービス要素を含むインプリメンテーション・ファイル、他のすべての WSDL 要素 (型、メッセージ、ポート・タイプ、バインディング) を含むインターフェース・ファイル、およびスキーマ用の 1 つ以上のファイルに分離された WSDL ファイルを検索できます。WSDL ODA は、複数のインターフェース・ファイル (例えば、メッセージとポート・タイプが 1 つのファイルにあり、バインディングが別のファイルにある場合など) を含む WSDL ファイルを正常に検索できません。

WSDL 文書内の <schema> 要素は、ネーム・スペース・プレフィックスに関しては自己完結型でなければなりません。<types> 要素の子である <schema> 要素内の WSDL 文書の <definitions><types>...</types></definitions> 要素に定義されたネーム・スペース・プレフィックスを使用することはできません。ネーム・スペース・プレフィックスを <schema> 要素のサブエレメントで使用する場合は、<schema> 要素に対してネーム・スペース・プレフィックスを再定義する必要があります。次の例では、自己完結型でないため、正しくないスキーマを示しています。

```
<definitions xmlns="http://schemas.xmlsoap.org/wsdl/" xmlns:NS="NS">
    <types>
         <schema xmlns="http://www.w3.org/1999/XMLSchema">
        <element name="NSElem" type="NS:NSType"/>
         </schema>
   </types>
</definitions>
```

ネーム・スペース・プレフィックス NS が <definitions> 要素に定義され、<schema> 要素に再定義されることなく使用されます。したがって、WSDL ODA はエラーをスローします。この制限を回避するには、ネーム・スペース・プレフィックス NS を <schema> 要素に再定義します。以下に例を示します。

```
<definitions xmlns="http://schemas.xmlsoap.org/wsdl/" xmlns:NS="NS">
    <types>
         <schema xmlns="http://www.w3.org/1999/XMLSchema" xmlns:NS="NS">
        <element name="NSElem" type="NS:NSType"/>
         </schema>
   </types>
</definitions>
```

第 9 章 トラブルシューティング

この章では、コネクターの始動または実行時に検出される問題について説明します。

始動時の問題

問題	可能なソリューション/説明
アルゴリズムがサポートされていない/アルゴリズム「SSL」が使用できない。	このエラーは、Connector Configurator Express で指定された SSL バージョンが JSSE プロバイダーによってサポートされない場合に発生します。ソリューション: サポートされる SSL バージョンについての JSSE プロバイダーの資料を調べます。IBM JSSE の場合は、*ProductDir*/lib/security ディレクトリー内の java.security ファイルに、以下の項目が含まれていることを確認します。 `security.provider.<number>=com.ibm.jsse.IBMJSSEProvider` ここで、<number> は、セキュリティー・プロバイダーをロードする際の優先順位です。
鍵ストアのロード中にエラーが発生する。鍵ストア・ファイル・パス「*<path>*」の指定が誤っている。鍵ストアが見つからない。	このエラーは、鍵ストアおよび/またはトラストストア・ファイルに間違ったパスを指定した場合に発生します。ソリューション: Connector Configurator Express 内の「SSL」->「鍵ストア」プロパティーで指定された鍵ストア・ファイル・パスを調べます。また、トラストストアを使用している場合は、Connector Configurator Express 内の「SSL」->「トラストストア (TrustStore)」プロパティーで指定されたトラストストア・ファイル・パスを調べます。
KeyManagementError: 鍵ストアが改ざんされる。KeyManagement エラー。	このエラーは、鍵ストアおよび/またはトラストストアが改ざんされたか、あるいは破壊された場合に発生します。また、このエラーは、パスワードに間違った値を指定した場合にも発生します。ソリューション: 鍵ストアが改ざんされていないことを確認します。鍵ストアの再作成を試みます。また、正しいパスワードを「SSL」->「KeyStorePassword」および「SSL」->「TrustStorePassword」コネクター・プロパティーに入力したことを確認します。
鍵ストアから証明書をロード中にエラーが発生する。	このエラーは、証明書および/または鍵ストア、トラストストアが改ざんされた場合に発生します。また、このエラーは、パスワードに間違った値を指定した場合にも発生します。ソリューション: 証明書、鍵ストア、またはトラストストアが改ざんされているかどうかを調べます。また、正しいパスワードを「SSL」->「KeyStorePassword」および「SSL」->「TruststorePassword」コネクター・プロパティーに入力したことを確認します。

問題	可能なソリューション/説明
サーバー・ソケットの作成中にエラーが発生し、終了する: エラー。	このエラーは、SOAP/HTTP または SOAP/HTTPS プロトコル・リスナーが、コネクター・プロパティーで指定されたポートにバインドできない場合に発生します。ソリューション: すべての SOAP/HTTP および SOAP/HTTPS プロトコル・リスナーに対して指定したポートを調べます。同一のポートが複数のリスナーに対して指定されている場合は、リスナーの 1 つだけが始動できます。さらに、そのポート上で実行中のほかのサービスがあるかどうか調べます。そのポート上で実行中のほかのサービスがある場合には、別のポートをプロトコル・リスナー用に選択することもできます。
KeyManagementError:UnrecoverableKeyException、鍵が回復できなかった。	このエラーは、鍵ストアまたはトラストストアを使用できない場合に発生します。ソリューション: 鍵ストアを新規に作成します。
SSL ハンドシェーク例外: 不明な CA。	これは、CA 証明書がトラストストア内にない場合に発生します。ソリューション: CA の証明書のほか、その自己署名証明書がトラストストア内に存在しているかどうかを調べます。さらに、証明書の DN にホスト名 (なるべく IP アドレス) が含まれていることを確認します。
ログ・ファイル内の過剰な JSSE ロギングに気付く。	基盤となる JSSE の詳細をすべてコンソール上に表示する必要がない場合は、Connector Configurator Express 内の「SSL」->「SSLDebug」プロパティーの値を `false` に設定します。
プロトコル・リスナーを指定したが、そのリスナーが初期化されないで、以下の警告メッセージがコネクターに表示される。 `Skipping Protocol Listener Property Set "SOME_LISTENER_NAME" with protocol property "": unable to determine the protocol listener class.]`	コネクターが、プロトコル・リスナーの「プロトコル (Protocol)」プロパティーに対して有効な値を取り出せませんでした。有効な値は、`soap/http`、`soap/https`、または `soap/jms` です。ソリューション: これはエラー状態ではありません。しかし、コネクターにこのリスナーを使用させる場合は、有効な値を「プロトコル (Protocol)」プロパティーに指定します。
プロトコル・ハンドラーを指定したが、そのハンドラーが初期化されないで、以下の警告メッセージがコネクターに表示される。 `Unable to determine the type of the handler; skipping initializing of current handler. Handler property details: Name: <Handler Name>; Value: Name: Protocol; Value: Name: ResponseWaitTimeout; Value: Name: ReplyToQueue; Value: .]`	コネクターが、ハンドラーの「プロトコル (Protocol)」プロパティーに対して有効な値を取り出せませんでした。有効な値は、`soap/http` または `soap/jms` です。ソリューション: これはエラー状態ではありません。しかし、コネクターにこのハンドラーを使用させる場合は、有効な値を「プロトコル (Protocol)」プロパティーに指定します。
`java.lang.NoClassDefFoundError: Javax/jms/JMSException...`	コネクターが `jms.jar` を検出できません。ソリューション: `jms.jar` がコネクターのクラスパス内にあることを確認します。

問題	可能なソリューション/説明
Fail to lookup, queue: "InProgressQueue" for specified queue name: "*<queue name>*" queue using JNDI "*<queue name>*"" javax.naming.NameNotFoundException: *<queue name>*	SOAP/JMS Web サービスをコネクターとともに使用している場合は、キューを作成しないと、この問題が発生します。また、このエラーは、「JNDI」->「LookupQueuesUsingJNDI」を true に設定していて、コネクターが JNDI を使用してキューを検索できない場合にも発生することがあります。ソリューション: コネクターに必要なキューを作成します。「JNDI」->「LookupQueuesUsingJNDI」が true に設定されている場合は、コネクターに必要なキューを、JNDI を使用して検索できることを確認します。
初期化時にエラーが発生する。JNDI コンテキストは初期化されず、ユーザーは JMS プロトコルを使用できない。	SOAP/JMS プロトコル・リスナーまたは SOAP/JMS プロトコル・ハンドラーを使用するようにコネクターを構成した場合は、JNDI プロパティーを指定する必要があります。ソリューション: 必要な JNDI コネクター固有プロパティーを指定したことを確認します。JNDI プロバイダーの資料を参照して、JNDI プロバイダーに接続するために必要なライブラリーおよび JAR ファイルを確認します。必要な JAR ファイルがすべてコネクターのクラスパス内にあることを確認します。さらに、必要なライブラリーがすべてコネクターのパス内にあることを確認します。
初期コンテキストを取得中にエラーが発生する。	SOAP/JMS プロトコル・リスナーまたは SOAP/JMS プロトコル・ハンドラーを使用するようにコネクターを構成した場合は、JNDI プロパティーを指定する必要があります。また、このエラーは、JNDI プロパティーを正しく指定しなかった場合にも発生します。ソリューション: JNDI プロパティーを調べます。JNDI が適切に構成されていることを確認します。JNDI プロバイダーの資料を参照して、JNDI プロバイダーに接続するために必要なライブラリーおよび JAR ファイルを確認します。必要な JAR ファイルがすべてコネクターのクラスパス内にあることを確認します。さらに、必要なライブラリーがすべてコネクターのパス内にあることを確認します。

ランタイム・エラー

問題	可能なソリューション/説明
HTTP 応答を解析中にエラーが発生する。HTTP 応答ヘッダーの読み取り中にストリームの終了に到達する。	このエラーは、コネクターが SOAP/HTTP Web サービスを呼び出すときに発生します。これは、ターゲット Web サービスが、間違った HTTP 応答を送信したために発生します。ソリューション: ターゲット SOAP/HTTP Web サービスのエンドポイント・アドレスが正しいことを確認します。
指定された URL でエラーが発生する。ホストおよびポートの詳細を抽出できない。宛先が誤っている。 <destination URL>	このエラーは、コネクターが SOAP/HTTP Web サービスを呼び出すときに発生します。これは、SOAP/HTTP Web サービスに間違ったエンドポイント・アドレスを指定したために発生します。ソリューション: Web サービスに正しいエンドポイント・アドレスを指定したことを確認します。

問題	可能なソリューション/説明
イベント・ビジネス・オブジェクト *<BO Name>* を動詞 *<Verb>* を指定してブローカーに送信中に障害が発生する。実行状況「-1」と以下のエラー・メッセージを受け取る。 マップ例外: ビジネス・オブジェクト <BO Name> をコネクター・コントローラー WebServicesConnector にマップするマップが見つかりません。 (MapException: Unable to find the map to map business objects <BO Name> for the connector controller WebServicesConnector) .	このエラーが発生するのは、コネクターによるイベントの同期送信先のコラボレーションが存在していないか、あるいはビジネス・オブジェクトの動詞を受け入れないかのいずれかであるため、統合ブローカーがそのイベントの処理に失敗する場合です。ソリューション: イベント通知のために Web サービス TLO を使用している場合は、TLO の `ws_collab` オブジェクト・レベルの ASI を調べます。(TLO の名前は、エラー・メッセージに示されます。) `ws_collab` ASI の値を調べます。このコラボレーションが存在して、稼働していることを確認します。`ws_mode` BO レベルの ASI が `synch` に設定されている場合は、`ws_collab` ASI が必要です。`ws_verb` オブジェクト・レベルの ASI の値を調べます。`ws_collab` ASI で指定したコラボレーションを、`ws_verb` ASI で指定した動詞によって起動できることを確認します。イベント通知のために非 TLO を使用している場合は、WSCollaborations コネクター・プロパティーを調べます。このビジネス・オブジェクトによって同期的に呼び出されるコラボレーションを見つけます。このコラボレーションが存在して、稼働していることを確認します。
SOAP 要求を要求ビジネス・オブジェクトに変換できなかった。SOAP 障害: 要求オブジェクトの生成時に障害が発生しました。動詞を要求ビジネス・オブジェクトに設定できませんでした。 (Failure in generating request object - no verb could be set on the request bo)	このエラーは、コネクターが統合ブローカーに送信しようとしているビジネス・オブジェクトの動詞を判別できない場合に、イベント通知の際に発生します。ソリューション: イベント通知のために Web サービス TLO を使用している場合は、この TLO に対して `ws_verb` オブジェクト・レベルの ASI を指定したことを確認します。動詞をこの ASI の値として指定します。イベント通知のために非 TLO を使用している場合は、Web サービス・クライアントによって送信された SOAP メッセージに動詞要素が含まれていなければなりません。SOAP データ・ハンドラーは、SOAP メッセージ内の動詞要素の値を使用して、ビジネス・オブジェクトの動詞を設定します。Web サービス・クライアントが SOAP メッセージ内の動詞を送信しない場合、SOAP データ・ハンドラーは、ビジネス・オブジェクト上で動詞を設定できません。この場合、コネクターはビジネス・オブジェクトを統合ブローカーに渡すことができません。Web サービス・クライアントで SOAP メッセージ内に動詞要素が含まれていない可能性がある場合は、このビジネス・オブジェクトに対して `DefaultVerb` 動詞レベルの ASI を指定できます。そのようにする場合、コネクターは、この動詞をビジネス・オブジェクト上で設定してから、それを統合ブローカーに送信します。

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
- BiDi.Application
- BiDi.Broker
- BiDi.Metadata
- BiDi.Transformation
- 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。コネクターを Linux システム上で稼働している場合でも、これらのツールがインストールされた Windows マシンが必要です。

Linux 上で動作するコネクターのコネクター・プロパティーを設定する場合は、Windows マシン上で System Manager を起動し、Linux の統合ブローカーに接続してから、コネクター用の Connector Configurator Express を開く必要があります。

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、195 ページの表 50 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 50 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 50 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 50. 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	

表 50. 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
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	
BiDi.Application	以下の双方向属性の 任意の有効な 組み合わせ 最初の文字: I、V 2 番目の文字: L、R 3 番目の文字: Y、N 4 番目の文字: S、N 5 番目の文字: H、C、N	ILYNN (5 文字)	コンポーネント 再始動	このプロパティーは、 BiDi.Transformation の値が true の 場合のみ有効です。
BiDi.Broker	以下の双方向属性の 任意の有効な 組み合わせ 最初の文字: I、V 2 番目の文字: L、R 3 番目の文字: Y、N 4 番目の文字: S、N 5 番目の文字: H、C、N	ILYNN (5 文字)	コンポーネント 再始動	このプロパティーは、 BiDi.Transformation の値が true の場合のみ有効で す。BrokerType の値が ICS の場合、プロパティーは読 み取り専用です。
BiDi.Metadata	以下の双方向属性の 任意の有効な 組み合わせ 最初の文字: I、V 2 番目の文字: L、R 3 番目の文字: Y、N 4 番目の文字: S、N 5 番目の文字: H、C、N	ILYNN (5 文字)	コンポーネント 再始動	このプロパティーは、 BiDi.Transformation の値が true の 場合のみ有効です。
BiDi.Transformation	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 WAS でない場合のみ 有効です。
BrokerType	ICS	ICS	コンポーネント 再始動	
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。
CommonEventInfrastruc ture	true または false	false	コンポーネント 再始動	
CommonEventInfrastruc tureURL	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。

表 50. 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
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。
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。

表 50. 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.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.NumConcurrentRequests	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	
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。

表 50. 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
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 に値がない場合にのみ有効です。
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	
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	

表 50. 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
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。

BiDi.Application

BiDi.Application プロパティーは、このアダプターがサポートする任意のビジネス・オブジェクトの形式で、外部アプリケーションからアダプターに入ってくるデータの双方向フォーマットを指定します。このプロパティーは、アプリケーション・データの双方向属性を定義します。これらの属性は以下のとおりです。

- テキストのタイプ: 暗黙または可視 (I または V)
- テキストの方向: 左から右または右から左 (L または R)
- 対称スワッピング: オンまたはオフ (Y または N)
- 成形 (アラビア語): オンまたはオフ (S または N)

- 数字成形 (アラビア語): ヒンディ語、コンテキスト、または標準 (H、C、または N)

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

デフォルト値は ILYNN (暗黙、左から右、オン、オフ、標準) です。

BiDi.Broker

BiDi.Broker プロパティーは、サポートされる任意のビジネス・オブジェクトの形式で、アダプターから統合ブローカーに送信されるデータの双方向フォーマットを指定します。データの双方向属性を定義します。属性は、前述の BiDi.Application の下にリストされています。

このプロパティーは、BiDi.Transformation プロパティーの値が true に設定されている場合のみ有効です。BrokerType プロパティーが ICS の場合、プロパティー値は読み取り専用です。

デフォルト値は ILYNN (暗黙、左から右、オン、オフ、標準) です。

BiDi.Metadata

BiDi.Metadata プロパティーは、メタデータの双方向フォーマットまたは属性を定義します。メタデータは、外部アプリケーションへのリンクを確立および保守するために、コネクターが使用します。属性の設定は、双方向機能を使用する各アダプターに固有です。アダプターが双方向処理をサポートする場合、詳細についてはアダプター固有のプロパティーに関するセクションを参照してください。

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

デフォルト値は ILYNN (暗黙、左から右、オン、オフ、標準) です。

BiDi.Transformation

BiDi.Transformation プロパティーは、システムが実行時に双方向変換を実行するかどうかを定義します。

プロパティー値が true に設定されている場合、BiDi.Application、BiDi.Broker、および BiDi.Metadata プロパティーが使用可能です。プロパティー値が false に設定されている場合は、それらは非表示になります。

デフォルト値は false 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。詳しくは、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・サイドの) 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 768MB 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)*」、「*WebSphere Business Integration Server Express インストール・ガイド (Linux 版)*」、または「*WebSphere Business Integration Server Express インストール・ガイド (OS/400 および i5/OS 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、次のようにします。
```
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。詳しくは、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 によりトリガーされる 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)*、「*WebSphere Business Integration Server Express インストール・ガイド (Linux 版)*」、または「*WebSphere Business Integration Server Express インストール・ガイド (OS/400 および i5/OS 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・ランタイムが以下のことを検出した場合、

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

PollFrequency プロパティーに構成された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、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・ランタイムが以下のことを検出した場合、

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

PollFrequency プロパティーに構成された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。詳しくは、 203 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、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**を作成する。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最初から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、219 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。

Linux でのコネクターの実行

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

Connector Configurator Express 内のいくつかのプロパティーはディレクトリー・パスを使用します。このパスは、Windows のディレクトリー・パスの規則がデフォルトになっています。Linux 環境で構成ファイルを使用する場合、これらのパスの Linux の規則に対応するように、ディレクトリー・パスを修正する必要があります。正しいオペレーティング・システム規則が拡張検証に使用されるように、ツールバー・ドロップ・リストでターゲット・オペレーティング・システムを選択します。

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 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 (225 ページの『構成ファイルの完成』を参照)。

System Manager からの 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、『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。入力した情報が、Connector Configurator Express によって、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 XML 文書として保管されます。

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 および Linux 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。
  Linux では、最初の文字は / でなければなりません。
- 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。選択可能なオプションは、「Windows」、「Linux」、および「i5/OS」、「その他」 (Windows でも Linux でもない場合)、および「なし (拡張検証なし)」 (拡張検証をオフに切り替え) です。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 (Select Connector-Specific Property Template)**

     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、XML コネクターの場合は CN_XML.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 に保管したりできます。
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 229 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」、「Linux」、および「i5/OS」、または「その他」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、または既存のコネクター構成ファイルを開くと、Connector Configurator Express に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. 227 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 193 ページの『標準コネクター・プロパティーの概要』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 つのマップを見付けて、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`
- Linux および i5/OS 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、そのファイルに含まれるロギングとトレースに関する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>
```

付録 C. Adapter for Web Services チュートリアル

この付録では、以下の手順を段階的に説明します。

- 要求処理とイベント処理の両方の場合の非同期および同期イベント伝送
- SOAP/HTTPS のサンプルに対して Web サービス・コネクターを構成する方法
- SOAP/HTTP のサンプルに対して Web サービス・コネクターを構成する方法
- SOAP/JMS のサンプルに対して Web サービス・コネクターを構成する方法

チュートリアルの概要

このチュートリアルは、Adapter for Web Services の要求処理とイベント処理の両方の場合の非同期および同期イベント伝送について、サポートされる各プロトコル (SOAP/HTTP、SOAP/HTTPS、および SOAP/JMS) 別に説明することを目的としています。それぞれのシナリオにおけるアダプターの役割は以下のとおりです。

- Web サービスを呼び出すコラボレーションの Web サービス・クライアント
- InterChange Server Express コラボレーションを Web サービスとして公開するプロキシー

このチュートリアルは、以下のサンプル・シナリオを使用してアダプターの基本的な機能を示すことを目的としています。

- **非同期シナリオ** このシナリオでは、コネクターが処理する非同期 (要求専用) Web サービスとそのクライアントについて説明します。このシナリオには 2 つのサンプルがあります。構成を単純にするため、コラボレーションを Web サービスとして公開するときと Web サービスをクライアントとして呼び出すときに、同じ Web サービス・コネクターを使用します。
  - **Web サービスとして公開されるコラボレーション**: このサンプルでは、Web サービスは単にコネクターによって Web サービスとして公開されている InterChange Server Express 内のコラボレーション `SERVICE_ASYNCH_Order_Collab` です。この Web サービスは Asynch Order Service と呼ばれます。コネクターが正しく構成されている場合は、任意の (1 つの) Web サービス・プロトコル (SOAP/HTTP、 SOAP/HTTPS または SOAP/JMS) を使用してこの Web サービスを呼び出すことができます。`SERVICE_ASYNCH_Order_Collab` は `SERVICE_ASYNCH_TLO_Order` を取る単純パススルー・コラボレーションです。このコラボレーションのトリガー・ポート (元) は Web サービス・コネクターにバインドされています。サービス・ポート (先) は SampleSiebelConnector にバインドされています。
  - **Web サービス・クライアントによって呼び出されるコラボレーション**: このサンプルでは、Web サービス・クライアントは、Web サービス・コネクターを

使用して Asynch Order Service Web サービスを呼び出す InterChange Server Express 内のもう 1 つのコラボレーション CLIENT_ASYNCH_Order_Collab です。コネクターが正しく構成されている場合は、この Web サービス・クライアントは任意の (1 つの) Web サービス・プロトコル (SOAP/HTTP、SOAP/HTTPS または SOAP/JMS) を使用して Web サービスを呼び出すことができます。CLIENT_ASYNCH_Order_Collab は CLIENT_ASYNCH_TLO_Order を取る単純パススルー・コラボレーションです。このコラボレーションのトリガー・ポート (元) は SampleSAPConnector にバインドされています。サービス・ポート (先) は Web サービス・コネクターにバインドされています。

非同期シナリオのいずれのサンプルにも以下の 2 つのアプリケーションが含まれています。

- SampleSiebel: クライアントのためにオーダーを作成します。
- SampleSAP: オーダーを作成します。

- **同期シナリオ** このシナリオでは、コネクターが処理する同期 (要求/応答) Web サービスとそのクライアントについて説明します。このシナリオには 2 つのサンプルがあります。構成を単純にするため、コラボレーションを Web サービスとして公開するときと Web サービスをクライアントとして呼び出すときに、同じ Web サービス・コネクターを使用します。
  - Web サービスとして公開されているコラボレーション: このサンプルでは、Web サービスは単にコネクターによって Web サービスとして公開されている InterChange Server Express 内のコラボレーション SERVICE_SYNCH_OrderStatus_Collab です。このサンプルでは、この Web サービスは Synch OrderStatus Service と呼ばれます。コネクターが正しく構成されている場合は、任意の Web サービス・プロトコル (SOAP/HTTP、SOAP/HTTPS または SOAP/JMS) を使用してこの Web サービスを呼び出すことができます。SERVICE_SYNCH_OrderStatus_Collab は SERVICE_SYNCH_TLO_OrderStatus を取る単純パススルー・コラボレーションです。このコラボレーションのトリガー・ポート (元) は Web サービス・コネクターにバインドされています。サービス・ポート (先) は SampleSiebelConnector にバインドされています。
  - **Web サービス・クライアントによって呼び出されるコラボレーション**: このサンプルでは、Web サービス・クライアントは、Web サービス・コネクターを使用して Synch OrderStatus Service Web サービスを呼び出す InterChange Server Express 内のコラボレーション CLIENT_SYNCH_OrderStatus_Collab です。コネクターが正しく構成されている場合は、この Web サービス・クライアントは任意の Web サービス・プロトコル (SOAP/HTTP、SOAP/HTTPS、または SOAP/JMS) を使用して Web サービスを呼び出すことができます。CLIENT_SYNCH_OrderStatus_Collab は CLIENT_SYNCH_TLO_OrderStatus を取る単純パススルー・コラボレーションです。このコラボレーションのトリガー・ポート (元) は SampleSAPConnector にバインドされています。サービス・ポート (先) は Web サービス・コネクターにバインドされています。

同期シナリオのいずれのサンプルにも以下の 2 つのアプリケーションが含まれています。

- SampleSiebel: クライアントのためにオーダーの状況を検索します。
- SampleSAP: オーダーの状況を要求します。

いずれのシナリオでも、2 つの Test Connector を使用して SampleSiebelConnector および SampleSAPConnector をシミュレートします。

はじめに

チュートリアルを開始する前に、以下のことを確認してください。

- InterChange Server Express 4.2.x 以降をインストール済みで、その運用経験を持っていること。
- InterChange Server Express ホーム・ディレクトリーに WebSphere Business Integration Adapter For Web Services がインストール済みであること。
- Web サービス・テクノロジーの運用経験があること。
- SOAP テクノロジーの運用経験があること。

インストールと構成

以下のセクションでは、*WBI_folder* は現在インストールされている InterChange Server Express が格納されているフォルダーを指します。すべての環境変数およびファイル分離文字は Windows NT/2000 の形式で記述されます。Linux および i5/OS で実行する場合は、適宜変更してください (例えば、WBI_folder¥connectors は WBI_folder/connectors になります)。

サーバーとツールの始動

1. ショートカットから InterChange Server Express を始動します。
2. WebSphere Business Integration Server Express System Manager を始動して、Component Navigator のパースペクティブを開きます。
3. サーバーをサーバー・インスタンスとして Interchange Server Express ビューに登録および接続します。

サンプル・コンテンツのロード

Component Navigator のパースペクティブから以下を実行します。

1. 新規統合コンポーネント・ライブラリーの作成
2. WBI_folder¥connectors¥WebServices¥samples¥WebSphereICS¥ にある WebServicesSample.jar というリポジトリー・ファイルのインポート

コラボレーション・テンプレートのコンパイル

WebSphere Business Integration System Manager を使用して以下の作業を実行します。

- リポジトリー・ファイル WebServicesSample.jar からインポートされたコラボレーション・テンプレートを**すべてコンパイル**する。

コネクターの構成

1. コネクターをまだ構成していない場合は、このガイドの説明に従ってご使用のシステムに応じて構成してください。

2. WebSphere Business Integration System Manager を使用して、Connector Configurator Express の WebServicesConnector を開きます。
3. また、このサンプルで使用するプロトコルの WebServicesConnector も構成する必要があります。
   - SOAP/HTTP を使用する場合は、『SOAP/HTTP プロトコル・シナリオ用の構成』を参照して SOAP/HTTP 用にコネクターを構成してください。
   - SOAP/HTTPS を使用する場合は、241 ページの『SOAP/HTTPS プロトコル・シナリオ用の構成』を参照して SOAP/HTTPS 用にコネクターを構成してください。
   - SOAP/JMS を使用する場合は、243 ページの『SOAP/JMS プロトコル・シナリオ用の構成』を参照して SOAP/JMS 用にコネクターを構成してください。

SOAP/HTTP プロトコル・シナリオ用の構成

このセクションでは、SOAP/HTTP サンプル・シナリオ用にコネクターを構成する方法について説明します。本書の本文に記載されているとおり、コネクターには SOAP/HTTP プロトコル・リスナーと SOAP/HTTP-HTTPS プロトコル・ハンドラーが組み込まれています。このサンプル・シナリオでは、`SERVICE_ASYNCH_Order_Collab` コラボレーションと `SERVICE_SYNCH_OrderStatus_Collab` コラボレーションを SOAP/HTTP Web サービスとして公開します。コラボレーションを SOAP/HTTP Web サービスとして公開するため、コネクターは SOAP/HTTP プロトコル・リスナーを使用します。このサンプル・シナリオには `CLIENT_ASYNCH_Order_Collab` コラボレーションと `CLIENT_SYNCH_OrderStatus_Collab` コラボレーションがあります。これらは SOAP/HTTP Web サービスの SOAP/HTTP クライアントです。SOAP/HTTP Web サービスを呼び出すため、コネクターは SOAP/HTTPHTTPS プロトコル・ハンドラーを使用します。

以下のステップと説明では、階層コネクター構成プロパティーが ” 記号で示されています。例えば、A” B は、A が階層プロパティーであり、B が A の子プロパティーであることを示します。

このサンプルに対して SOAP/HTTP プロトコル・リスナーを構成するには、以下の手順に従います。

1. Connector Configurator Express で、WebServicesConnector の「**コネクター固有プロパティー**」をクリックします。
2. **ProtocolListenerFramework** プロパティーを展開して、ProtocolListeners 子プロパティーを表示します。
3. **ProtocolListeners** 子プロパティーを展開して、**SOAPHTTPListener1** 子プロパティーを表示します。
4. **SOAPHTTPListener1”Host** プロパティーと **SOAPHTTPListener1”Port** プロパティーの値を確認します。ホストで他のプロセスが実行中でないこと、この TCP/IP ポートで listen していないことを確認してください。オプションで、**SOAHTTPListener1”Host** の値をコネクターを実行するマシン名に設定することもできます。

このサンプルに SOAP/HTTP-HTTPS プロトコル・ハンドラーを構成する必要はありません。

SOAP/HTTPS プロトコル・シナリオ用の構成

このセクションでは、SOAP/HTTPS サンプル・シナリオ用にコネクターを構成する方法について説明します。コネクターには SOAP/HTTPS プロトコル・リスナーと SOAP/HTTP-HTTPS プロトコル・ハンドラーが組み込まれています。このサンプル・シナリオでは、SERVICE_ASYNCH_Order_Collab コラボレーションと SERVICE_SYNCH_OrderStatus_Collab コラボレーションを SOAP/HTTPS Web サービスとして公開します。コラボレーションを SOAP/HTTPS Web サービスとして公開するため、コネクターは SOAP/HTTPS プロトコル・リスナーを使用します。このサンプル・シナリオには CLIENT_ASYNCH_Order_Collab コラボレーションと CLIENT_SYNCH_OrderStatus_Collab コラボレーションがあります。これらは SOAP/HTTPS Web サービスの SOAP/HTTPS クライアントです。SOAP/HTTPS Web サービスを呼び出すため、コネクターは SOAP/HTTPHTTPS プロトコル・ハンドラーを使用します。

以下のステップと説明では、階層コネクター構成プロパティーが ” 記号で示されています。例えば、A” B は、A が階層プロパティーであり、B が A の子プロパティーであることを示します。

注: 239 ページの『はじめに』にリストされたプリインストール項目の他に、鍵および証明書管理ソフトウェアを使用して、鍵ストアとトラストストアを作成およびテストしておく必要があります。

SSL コネクター固有プロパティーの構成: SOAP/HTTPS の場合、コネクターに SSL コネクター固有の階層プロパティーを構成する必要があります。

1. Connector Configurator Express で、WebServicesConnector の「**コネクター固有プロパティー**」タブをクリックします。
2. **SSL** 階層プロパティーを展開して、すべての子プロパティーを表示します。さらに、階層 SSL コネクター固有プロパティーの以下の子プロパティーを確認または変更します。
   - **SSL” KeyStore** このプロパティーは、鍵ストア・ファイルへの完全パスに設定します。このファイルは鍵および証明書管理ソフトウェアを使用して作成する必要があります。
   - **SSL”KeyStorePassword** このプロパティーは、KeyStore にアクセスするために必要なパスワードに設定します。
   - **SSL”KeyStoreAlias** このプロパティーは、KeyStore の秘密鍵の別名に設定します。
   - **SSL”TrustStore** このプロパティーは、鍵および証明書管理ソフトウェアを使用して作成したトラストストア・ファイルへの完全パスに設定します。
   - **SSL”TrustStorePassword** このプロパティーは、TrustStore にアクセスするために必要なパスワードに設定します。

注: 必ず Connector Configurator Express で変更内容を保管するようにしてください。

SOAP/HTTPS プロトコル・リスナーの構成:

1. Connector Configurator Express で、WebServicesConnector の「**コネクター固有プロパティー**」をクリックします。

2. **ProtocolListenerFramework** プロパティーを展開して、**ProtocolListeners** 子プロパティーを表示します。
3. **ProtocolListeners** 子プロパティーを展開して、**SOAPHTTPSListener1** 子プロパティーを表示します。**SOAPHTTPSListener1"Host** プロパティーと **SOAPHTTPSListener1"Port** プロパティーの値を確認します。ホストで他のプロセスが実行中でないこと、この TCP/IP ポートで listen していないことを確認してください。オプションで、**SOAHTTPSListener1"Host** の値をコネクターを実行するマシン名に設定することもできます。

このサンプルに SOAP/HTTP-HTTPS プロトコル・ハンドラーを構成する必要はありません。

KeyStore および TrustStore のセットアップ: サンプル・シナリオで使用できるように KeyStore と TrustStore を素早くセットアップできます。実動システムの場合は、鍵ストア、証明書、および鍵生成をセットアップおよび管理するために、サード・パーティーのソフトウェアを使用する必要があります。これらのリソースをセットアップおよび管理するためのツールは Adapter for Web Services には含まれていません。

このセクションは、Java 仮想マシンがシステムにインストール済みであること、JVM (Java 仮想マシン) に付属の鍵ツールについて十分な知識と経験があることを前提としています。鍵ツールの詳細と問題のトラブルシューティングについては、JVM に付属の資料を参照してください。

KeyStore をセットアップするには、以下の手順に従います。

1. 鍵ツールを使用して KeyStore を作成します。KeyStore に鍵ペアを作成する必要があります。これを実行するには、コマンド行に以下を入力します。

   ```
   keytool -genkey -alias wsadapter -keystore c:¥security¥keystore
   ```

2. これにより、鍵ツールからパスワードを入力するようプロンプトが出ます。SSL"KeyStorePassword コネクター・プロパティーの値として入力したパスワードを指定します。
   上記の例で、コマンド行に `-keystore c:¥security¥keystore` を指定した場合は、`c:¥security¥keystore` を SSL"KeyStore プロパティーの値として入力することができます。また、コマンド行に `-alias wsadapter` を指定した場合は、`wsadapter` を SSL"KeyStoreAlias コネクター・プロパティーの値として入力することができます。次に、鍵ツールから、証明書の詳細についてプロンプトが出されます。以下は、各プロンプトとその入力例を示しています。ただし、以下に示すのは単なる例にすぎません。必ず鍵ツールの資料を参照してこれに従ってください。

   ```
   What is your first and last name?
        [Unknown]:  HostName
   What is the name of your organizational unit?
        [Unknown]:  myunit
   What is the name of your organization?
        [Unknown]:  myorganization
   What is the name of your City or Locality?
        [Unknown]:  mycity
   What is the name of your State or Province?
        [Unknown]:  mystate
   ```

```
What is the two-letter country code for this unit?
     [Unknown]:  mycountryIs <CN=HostName, OU=myunit, O=myorganization,
     L=mycity, ST=mystate, C=mycountry> correct?
     [no]:  yes
```

3. 「What is your first and last name?」というプロンプトが表示されたら、コネクターを実行しているマシンの名前を入力します。次に、鍵ツールから次のプロンプトが出されます。

   ```
   Enter key password for <wsadapter> (RETURN if same as keystore password):
   ```

4. 同じパスワードを使用するには「**Return**」を選択します。自己署名証明書を使用する場合は、上記で作成済みの証明書のエクスポートが可能です。これを実行するには、コマンド行に以下を入力します。

   ```
   C:¥security>keytool -export -alias wsadapter -keystore c:¥security¥keystore
   -file c:¥security¥wsadapter.cer
   ```

5. ここで、鍵ツールから、鍵ストア・パスワードを入力するよう求めるプロンプトがでます。上記で入力したパスワードを入力します。

TrustStore をセットアップするには、以下の手順に従います。

1. TrustStore にトラステッド証明書をインポートするには、次のコマンドを入力します。

   ```
   keytool -import -alias trusted1 -keystore c:¥security¥truststore
   -file c:¥security¥wsadapter.cer
   ```

2. ここで、鍵ツールから、鍵ストア・パスワードを入力するよう求めるプロンプトがでます。`-keystore c:¥security¥truststore` と入力した場合は、SSL”TrustStore プロパティーが `c:¥security¥truststore` に設定されていることを必ず確認してください。また、SSL”TrustStorePassword プロパティーの値を、上記で入力したパスワードに設定する必要があります。

SOAP/JMS プロトコル・シナリオ用の構成

このセクションでは、SOAP/JMS サンプル・シナリオ用にコネクターを構成する方法について説明します。このサンプル・シナリオでは、`SERVICE_ASYNCH_Order_Collab` コラボレーションと `SERVICE_SYNCH_OrderStatus_Collab` コラボレーションを SOAP/JMS Web サービスとして公開します。コラボレーションを SOAP/JMS Web サービスとして公開するため、コネクターは SOAP/JMS プロトコル・リスナーを使用します。このサンプル・シナリオには `CLIENT_ASYNCH_Order_Collab` コラボレーションと `CLIENT_SYNCH_OrderStatus_Collab` コラボレーションがあります。これらは SOAP/JMS Web サービスの SOAP/JMS クライアントです。SOAP/JMS Web サービスを呼び出すため、コネクターは SOAP/JMS プロトコル・ハンドラーを使用します。

以下のステップと説明では、階層コネクター構成プロパティーが ” 記号で示されています。例えば、A” B は、A が階層プロパティーであり、B が A の子プロパティーであることを示します。

注: 239 ページの『はじめに』にリストされたプリインストール項目の他に、JMS サービス・プロバイダーをインストールし、JNDI をインストールおよび構成しておく必要があります。

JNDI プロパティーの構成: SOAP/JMS の場合は、以下の JNDI コネクター構成プロパティーを構成する必要があります。

1. Connector Configurator Express で、WebServicesConnector の「**コネクター固有プロパティー**」をクリックします。
2. JNDI 階層プロパティーを展開して、子プロパティーを表示します。次に、子プロパティーを確認または変更して、以下にリストされた値に一致するようにします。
   - **JNDI"JNDIProviderURL** このプロパティーは、JNDI サービス・プロバイダーの URL に設定します。詳しくは、JNDI プロバイダーの資料を参照してください。
   - **JNDI"InitialContextFactory** このプロパティーは、JNDI 初期コンテキストを作成するファクトリー・クラスの完全修飾クラス名に設定します。詳しくは、JNDI プロバイダーの資料を参照してください。
   - **JNDI"JNDIConnectionFactoryName** このプロパティーは、JNDI コンテキストを使用して検索する、接続ファクトリーの JNDI 名に設定します。この名前は、JNDI を使用して検索できることを確認してください。
   - 以下のどのプロパティーが JNDI プロバイダーに必要かを確認するには、該当する JNDI の資料を参照してください。
     - **JNDI"CTX_ObjectFactories**
     - **JNDI"CTX_ObjectFactories**
     - **JNDI"CTX_StateFactories**
     - **JNDI"CTX_URLPackagePrefixes**
     - **JNDI"CTX_DNS_URL**
     - **JNDI"CTX_Authoritative**
     - **JND"CTX_Batchsize**
     - **JNDI"CTX_Referral**
     - **JNDI"CTX_SecurityProtocol**
     - **JND"CTX_SecurityAuthentication**
     - **JNDI"CTX_SecurityPrincipal**
     - **JNDI"CTX_SecurityCredentials**
     - **JNDI"CTX_Language**
3. Connector Configurator Express で変更内容を保管します。

JMS キューおよび SOAP/JMS プロトコル・リスナーの構成: このシナリオでは、JMS サービス・プロバイダーに 6 つのキューを定義する必要があります。キューを定義する前に、ご使用の JMS プロバイダーの資料を確認してください。キューの定義方法はプロバイダーによって異なります。

1. 以下のキューを定義 (または JNDI 検索によって有効化) します。
   - `ORDER_INPUT`
   - `ORDER_INPROGRESS`
   - `ORDER_ERROR`
   - `ORDER_ARCHIVE`
   - `ORDER_UNSUBSCRIBED`

- ORDER_REPLYTO

2. CSM から Connector Configurator Express の **WebServicesConnector** を開きます。コネクターをまだ構成していない場合は、ご使用のシステムのインストール・ガイドの説明に従って構成してください。
3. Connector Configurator Express で「**アプリケーション構成プロパティー**」をクリックします。
4. **ProtocolListenerFramework** プロパティーを展開して、ProtocolListeners 子プロパティーを表示します。
5. **ProtocolListeners** プロパティーを展開して、SOAPJMSListener1 子プロパティーを表示します。
6. **SOAPJMSListner1** 子プロパティーの値を確認または変更して、以下にリストされた値に一致するようにします。
   - **SOAPJMSListener"Protocol** このプロパティーは、soap/jms に設定します。
   - **SOAPJMSListener1"Protocol** このプロパティーは、soap/jms に設定します。
   - **SOAPJMSListener1"InputQueue** このプロパティーは、ORDER_INPUT に設定します。
   - **SOAPJMSListener1"InProgressQueue** このプロパティーは、ORDER_INPROGRESS に設定します。
   - **SOAPJMSListener1"ArchiveQueue** このプロパティーは、ORDER_ARCHIVE に設定します。
   - **SOAPJMSListener1"UnsubscribedQueue** このプロパティーは、ORDER_UNSUBSCRIBED に設定します。
   - **SOAPJMSListener1"ErrorQueue** このプロパティーは、ORDER_ERROR に設定します。
   - **SOAPJMSListener1"ReplyToQueue** このプロパティーは、ORDER_REPLYTO に設定します。
7. Connector Configurator Express で変更内容を保管します。

SOAP/JMS プロトコル・ハンドラーの構成:

1. System Manager から Connector Configurator Express の **WebServicesConnector** を開きます。コネクターをまだ構成していない場合は、ご使用のシステムのインストール・ガイドの説明に従って構成してください。
2. Connector Configurator Express で「**コネクター構成プロパティー (Connector-Config Properties)**」をクリックします。
3. **ProtocolHandlerFramework** プロパティーを展開して、ProtocolHandlers 子プロパティーを表示します。
4. **ProtocolHandlers** 子プロパティーを展開して、SOAPJMSHandler 子プロパティーを表示します。SOAPJMSHandler 子プロパティーの値を確認または変更して、以下にリストされた値に一致するようにします。
   - **SOAPJMSHandler"Protocol** このプロパティーは、soap/jms に設定します。
   - **SOAPJMSHandler"ReplyToQueue** このプロパティーは、ORDER_REPLYTO_HANDLER に設定します。
5. Connector Configurator Express で変更内容を保管します。

ユーザー・プロジェクトの作成

- WebSphere Business Integration System Manager を使用して、新規**ユーザー・プロジェクト**を作成します。 239 ページの『サンプル・コンテンツのロード』で作成した統合コンポーネント・ライブラリーから、すべてのコンポーネントを選択します。

プロジェクトの追加と配置

1. サーバー・インスタンス・ビューから、『ユーザー・プロジェクトの作成』で作成した**ユーザー・プロジェクト**を InterChange Server Express に追加します。
2. このユーザー・プロジェクトのすべてのコンポーネントを InterChange Server Express に配置します。

InterChange Server Express のリブート

1. すべての変更内容を有効にするため、InterChange Server Express をリブートします。
2. System Monitor ツールを使用して、コラボレーション・オブジェクト、コネクター・コントローラー、およびマップがすべて正常であることを確認します。

非同期シナリオの実行

このシナリオでは、Asynch Order Service Web サービスを呼び出します。シナリオを実行する前に、シナリオに関するデータの流れを段階を追って説明します。

1. `CLIENT_ASYNCH_TLO_Order.Create` イベントが、Test Connector の 1 つのインスタンスで動作する `SampleSAP` という名前のアプリケーションで発生します。
2. イベントが `SampleSAP` から `CLIENT_ASYNCH_Order_Collab` という名前のコラボレーションに送信されます。
3. 続いて、イベントがコラボレーションから Web サービス・コネクターに送信されます。
4. Web サービス・コネクターが、`CLIENT_ASYNCH_TLO_Order` オブジェクトの子である `CLIENT_ASYNCH_Order` オブジェクトを検出します。
5. 要求ビジネス・オブジェクトが、SOAP データ・ハンドラーを使用して SOAP メッセージに変換されます。
6. Web サービス・コネクターは、SOAP メッセージを Web サービス Asynch Order Service のエンドポイント (宛先) に送信します。エンドポイントは、プロトコル構成メタオブジェクト (MO) の Destination 属性によって指定されます。コネクターが使用するプロトコル構成 MO は、`CLIENT_ASYNCH_TLO_Order` の Handler 属性の値によって決まります。この属性の値が `soap/http` に設定されている場合、`CLIENT_ASYNCH_Order_SOAP_HTTP_CfgMO` の Destination 属性は、エンドポイントを Web サービスの URL として指定します。また、Handler 属性が `soap/jms` に設定されている場合は、`CLIENT_ASYNCH_Order_SOAP_JMS_CfgMO` の Destination 属性はエンドポイントを宛先キュー名として指定します。
7. Asynch Order Service Web サービスが SOAP 要求を受信します。前述のように、Web サービス・コネクターはこの Web サービスのエンドポイントになります。要求の送信先であるエンドポイントで listen しているコネクターのプロトコル・リスナーが SOAP メッセージを受信します。

8. コネクターは SOAP メッセージを SERVICE_ASYNCH_Order に変換してから、SERVICE_TLO_Order オブジェクトを作成します。SERVICE_ASYNCH_Order オブジェクトが SERVICE_TLO_Order オブジェクトの子として設定されます。
9. 次に、Web サービス・コネクターが SERVICE_TLO_Order オブジェクトを InterChange Server Express に非同期的に通知します。これにより、非同期 Web サービスの呼び出しが完了します。

これは非同期 Web サービス (要求専用) であるため、Web サービス・クライアントに応答は戻されません。SERVICE_ASYNCH_Order_Collab がこのオブジェクトを受信すると、コラボレーションがビジネス・オブジェクトを Test Connector の 2 番目のインスタンスとして実行中の SampleSiebel というアプリケーションに送信します。オブジェクトが Test Connector に表示されます。SampleSiebel アプリケーションから「正常に応答」を選択すると、イベントが SERVICE_ASYNCH_Order_Collab に戻されます。

非同期シナリオを実行するには、以下の手順を実行します。

1. InterChange Server Express 統合ブローカーがまだ稼働していない場合は始動します。
2. Web サービス・コネクターを始動します。
3. Test Connector の 2 つのインスタンスを開始します。
4. Test Connector を使用して、SampleSAPConnector および SampleSiebelConnector のプロファイルを定義します。
5. エージェントのシミュレートを開始するため、それぞれの「Test Connector」メニューから「**ファイル**」”「**エージェントの接続**」を選択します。
6. Test Connector を使用して SampleSAPConnector をシミュレートしながら、メニューから「**編集**」” 「**ビジネス・オブジェクトのロード**」を選択します。以下のファイルをロードします。

   WBI_folder¥connectors¥WebServices¥samples¥WebSphereICS¥OrderStatus¥CLIENT_ASYNCH_TLO_Order.bo

   Test Connector は CLIENT_ASYNCH_TLO_Order がロードされたというメッセージを表示します。
7. Web サービスのエンドポイント・アドレスを確認します。
   - **SOAP/HTTP Web サービスの場合** SOAP/HTTP を使用する場合は、以下の作業を実行します。
     a. Web サービス・コネクターを SOAP/HTTP 用に構成済みであることを確認します。Test Connector で、CLIENT_ASYNCH_TLO_Order ビジネス・オブジェクトの Handler 属性の値が soap/http に設定されていることを確認します。この値に引用符を含めることはできません。
     b. CLIENT_ASYNCH_TLO_Order の Request 属性を展開します。この属性のタイプは CLIENT_ASYNCH_Order ビジネス・オブジェクトです。
     c. CLIENT_ASYNCH_Order の SOAPHTTPCfgMO 属性を展開します。この属性のタイプは CLIENT_ASYNCH_Order_SOAP_HTTP_CfgMO です。
     d. CLIENT_ASYNCH_Order_SOAP_HTTP_CfgMO の Destination 属性の値が http://localhost:8080/wbia/webservices/samples に設定されていることを確認します。この値に引用符を含めることはできません。

- **SOAP/HTTPS Web サービスの場合** SOAP/HTTPS を使用する場合は、以下の作業を実行します。
  a. Web サービス・コネクターを SOAP/HTTPS 用に構成済みであることを確認します。Test Connector で、CLIENT_ASYNCH_TLO_Order ビジネス・オブジェクトの Handler 属性の値が soap/http に設定されていることを確認します。この値に引用符を含めることはできません。
  b. CLIENT_ASYNCH_TLO_Order の Request 属性を展開します。この属性のタイプは CLIENT_ASYNCH_Order ビジネス・オブジェクトです。
  c. CLIENT_ASYNCH_Order の SOAPHTTPCfgMO 属性を展開します。この属性のタイプは CLIENT_ASYNCH_Order_SOAP_HTTP_CfgMO です。
  d. CLIENT_ASYNCH_Order_SOAP_HTTP_CfgMO の Destination 属性の値が https://localhost:8443/wbia/webservices/samples に設定されていることを確認します。この値に引用符を含めることはできません。
- **SOAP/JMS Web サービスの場合** SOAP/JMS を使用する場合は、以下の作業を実行します。
  a. Web サービス・コネクターを SOAP/JMS 用に構成済みであることを確認します。Test Connector で、CLIENT_ASYNCH_TLO_Order ビジネス・オブジェクトの Handler 属性の値が soap/jms に設定されていることを確認します。この値に引用符を含めることはできません。
  b. CLIENT_ASYNCH_TLO_Order の Request 属性を展開します。この属性のタイプは CLIENT_ASYNCH_Order ビジネス・オブジェクトです。
  c. CLIENT_ASYNCH_Order の SOAPJMSCfgMO 属性を展開します。この属性のタイプは CLIENT_ASYNCH_Order_SOAP_JMS_CfgMO です。
  d. CLIENT_ASYNCH_Order_SOAP_JMS_CfgMO の Destination 属性の値が ORDER_INPUT に設定されていることを確認します。この値に引用符を含めることはできません。

8. Test Connector を使用して SampleSAPConnector をシミュレートしながら、ロードした**テスト・ビジネス・オブジェクト**をクリックします。メニューから「**要求**」>「**送信**」を選択します。イベントの流れについての詳細は、このセクションで前述した段階的な説明を参照してください。
9. Test Connector を使用して SampleSiebelConnector をシミュレートしながら、「**要求**」>「**要求の受付**」を選択します。Test Connector の右パネルに、SERVICE_ASYNCH_TLO_Order.Create というラベルの付いたイベントが表示されます。
10. ビジネス・オブジェクトをダブルクリックします。ウィンドウ内でビジネス・オブジェクトが開きます。
11. ビジネス・オブジェクトの Request 属性を展開します。Request 属性のタイプは SERVICE_ASYNCH_Order です。SERVICE_ASYNCH_Order の OrderId、Customarily、およびその他の属性をインスペクションして、受信したオーダーを検査します。これにより、非同期シナリオの実行が完了します。
12. ビジネス・オブジェクトのインスペクションが完了したら、ウィンドウを閉じます。「**要求**」>「**返信**」>「**成功**」を選択します。

同期シナリオの実行

このシナリオでは、Synch OrderStatus Service Web サービスを呼び出します。シナリオを実行する前に、シナリオに関するデータの流れを段階を追って説明します。

1. CLIENT_SYNCH_TLO_OrderStatus.Retrieve イベントが、Test Connector の 1 つのインスタンスで動作する SampleSAP という名前のアプリケーションで発生します。
2. イベントが SampleSAP から CLIENT_SYNCH_OrderStatus_Collab という名前のコラボレーションに送信されます。
3. 続いて、イベントがコラボレーションから Web サービス・コネクターに送信されます。
4. Web サービス・コネクターが、CLIENT_SYNCH_TLO_OrderStatus オブジェクトの子である CLIENT_SYNCH_OrderStatus_Request オブジェクトを検出します。
5. Web サービス・コネクターが、SOAP データ・ハンドラーを呼び出して、CLIENT_SYNCH_OrderStatus_Request ビジネス・オブジェクトを SOAP メッセージに変換します。
6. Web サービス・コネクターが、SOAP メッセージを Web サービス Synch OrderStatus Service のエンドポイント (宛先) に送信します。エンドポイントは、プロトコル構成メタオブジェクト (MO) の Destination 属性によって指定されます。コネクターが使用するプロトコル構成 MO は、CLIENT_SYNCH_TLO_OrderStatus の Handler 属性の値によって決まります。この属性の値が soap/http に設定されている場合は、CLIENT_SYNCH_OrderStatus_Request_SOAP_HTTP_CfgMO の Destination 属性はエンドポイントを Web サービスの URL として指定します。また、Handler 属性が soap/jms に設定されている場合は、CLIENT_SYNCH_OrderStatus_Request_SOAP_JMS_CfgMO の Destination 属性はエンドポイントを Web サービスの宛先キュー名として指定します。
7. Web サービス Synch OrderStatus Service が SOAP 要求を受信します。前述のように、Web サービス・コネクターは宛先エンドポイントになります。要求の送信先であるエンドポイントで listen しているコネクターのプロトコル・リスナーが SOAP メッセージを受信します。
8. コネクターが SOAP メッセージによって SOAP データ・ハンドラーを呼び出します。SOAP データ・ハンドラーによって SOAP メッセージが SERVICE_SYNCH_OrderStatus_Request オブジェクトに変換されます。次に、Web サービス・コネクターが SERVICE_TLO_OrderStatus オブジェクトを作成します。SERVICE_SYNCH_OrderStatus_Request オブジェクトは SERVICE_TLO_OrderStatus オブジェクトの子として設定されます。
9. 次に、Web サービス・コネクターが SERVICE_TLO_OrderStatus オブジェクトを InterChange Server Express で実行中の SERVICE_SYNCH_OrderStatus_Collab コラボレーションに同期的に通知します。これは同期実行であるため、コラボレーションが実行されて応答が戻るまで、Web サービス・コネクターはブロックされたままになります。
10. SERVICE_SYNCH_OrderStatus_Collab が SERVICE_TLO_OrderStatus オブジェクトを受信します。次に、コラボレーションがビジネス・オブジェクトを Test Connector の 2 番目のインスタンスとして実行中の SampleSiebel というアプリケーションに送信します。

11. SampleSiebel アプリケーションから「正常に応答」を選択すると、イベントが SERVICE_SYNCH_OrderStatus_Collab コラボレーションに戻されます。
12. SERVICE_SYNCH_OrderStatus_Collab が SERVICE_TLO_OrderStatus オブジェクトを受信します。次に、コラボレーションがビジネス・オブジェクトを Web サービス・コネクターに送信します。
13. Web サービス・コネクターが、SERVICE_SYNCH_OrderStatus_TLO の子である SERVICE_SYNCH_OrderStatus_Response ビジネス・オブジェクト (あるいは、SERVICE_SYNCH_OrderStatus_Fault が取り込まれている場合はこのオブジェクト) を検出します。このビジネス・オブジェクトは、SOAP データ・ハンドラーによって SOAP 応答メッセージ (または SOAP 障害メッセージ) に変換されます。
14. Web サービス・コネクターが SOAP 応答メッセージ (または SOAP 障害メッセージ) を Web サービス・クライアントに戻します。
15. Web サービス・クライアント (この場合はコネクター) が応答を受信します。コネクターが応答メッセージによって SOAP データ・ハンドラーを呼び出します。
16. SOAP データ・ハンドラーが応答メッセージを CLIENT_SYNCH_OrderStatus_Response ビジネス・オブジェクトまたは CLIENT_SYNCH_OrderStatus_Fault ビジネス・オブジェクトに変換します。どちらのビジネス・オブジェクトに変換されるかは、Order Synch Service によって戻されるものによって決まります。Web サービス・コネクターがこのオブジェクトを CLIENT_SYNCH_OrderStatus_TLO の子として設定します。CLIENT_SYNCH_OrderStatus_TLO が CLIENT_SYNCH_OrderStatus_Collab コラボレーションに戻されます。

次に、CLIENT_SYNCH_OrderStatus_Collab が CLIENT_SYNCH_OrderStatus_TLO を Test Connector の 1 番目のインスタンスとして実行中の SampleSAP というアプリケーションに送信します。Test Connector がこのオブジェクトを表示します。

同期シナリオを実行するには、以下の手順を実行します。

1. InterChange Server Express 統合ブローカーがまだ稼働していない場合は始動します。
2. Web サービス・コネクターを始動します。
3. Test Connector の 2 つのインスタンスを開始します。
4. Test Connector を使用して、SampleSAPConnector および SampleSiebelConnector のプロファイルを定義します。
5. エージェントのシミュレートを開始するため、それぞれの「Test Connector」メニューから「**ファイル**」”「**エージェントの接続**」を選択します。
6. Test Connector を使用して SampleSAPConnector をシミュレートしながら、メニューから「**編集**」”「**ビジネス・オブジェクトのロード**」を選択します。以下のファイルをロードします。

   WBI_folder¥connectors¥WebServices¥samples¥WebSphereICS¥OrderStatus ¥CLIENT_SYNCH_TLO_OrderStatus.bo

   Test Connector は CLIENT_SYNCH_TLO_OrderStatus がロードされたというメッセージを表示します。
7. Web サービスのエンドポイント・アドレスを確認します。

- **SOAP/HTTP Web サービスの場合** SOAP/HTTP を使用する場合は、以下の作業を実行します。
  a. Web サービス・コネクターを SOAP/HTTP 用に構成済みであることを確認します。Test Connector で、CLIENT_SYNCH_TLO_OrderStatus ビジネス・オブジェクトの Handler 属性の値が soap/http に設定されていることを確認します。この値に引用符を含めることはできません。
  b. CLIENT_SYNCH_TLO_OrderStatus の Request 属性を展開します。この属性のタイプは CLIENT_SYNCH_OrderStatus ビジネス・オブジェクトです。
  c. CLIENT_SYNCH_OrderStatus の SOAPHTTPCfgMO 属性を展開します。この属性のタイプは CLIENT_SYNCH_OrderStatus_SOAP_HTTP_CfgMO です。
  d. CLIENT_SYNCH_OrderStatus_SOAP_HTTP_CfgMO の Destination 属性の値が http://localhost:8080/wbia/webservices/samples に設定されていることを確認します。この値に引用符を含めることはできません。
- **SOAP/HTTPS Web サービスの場合** SOAP/HTTPS を使用する場合は、以下の作業を実行します。
  a. Web サービス・コネクターを SOAP/HTTPS 用に構成済みであることを確認します。Test Connector で、CLIENT_SYNCH_TLO_OrderStatus ビジネス・オブジェクトの Handler 属性の値が soap/http に設定されていることを確認します。この値に引用符を含めることはできません。
  b. CLIENT_SYNCH_TLO_OrderStatus の Request 属性を展開します。この属性のタイプは CLIENT_SYNCH_OrderStatus ビジネス・オブジェクトです。
  c. CLIENT_SYNCH_OrderStatus の SOAPHTTPCfgMO 属性を展開します。この属性のタイプは CLIENT_SYNCH_OrderStatus_SOAP_HTTP_CfgMO です。
  d. CLIENT_SYNCH_OrderStatus_SOAP_HTTP_CfgMO の Destination 属性の値が https://localhost:8443/wbia/webservices/samples に設定されていることを確認します。この値に引用符を含めることはできません。
- **SOAP/JMS Web サービスの場合** SOAP/JMS を使用する場合は、以下の作業を実行します。
  a. Web サービス・コネクターを SOAP/JMS 用に構成済みであることを確認します。Test Connector で、CLIENT_SYNCH_TLO_OrderStatus ビジネス・オブジェクトの Handler 属性の値が soap/jms に設定されていることを確認します。この値に引用符を含めることはできません。
  b. CLIENT_SYNCH_TLO_OrderStatus の Request 属性を展開します。この属性のタイプは CLIENT_SYNCH_OrderStatus ビジネス・オブジェクトです。
  c. CLIENT_SYNCH_OrderStatus の SOAPJMSCfgMO 属性を展開します。この属性のタイプは CLIENT_SYNCH_OrderStatus_SOAP_JMS_CfgMO です。
  d. CLIENT_SYNCH_OrderStatus_SOAP_JMS_CfgMO の Destination 属性の値が ORDER_INPUT に設定されていることを確認します。この値に引用符を含めることはできません。

8. Test Connector を使用して SampleSAPConnector をシミュレートしながら、ロードした**テスト・ビジネス・オブジェクト**をクリックします。メニューから「**要求**」”「**送信**」を選択します。データの流れについての詳細は、このセクションで前述した段階的な説明を参照してください。

9. SampleSiebelConnector をシミュレートしている Test Connector インスタンスの右パネルに、SERVICE_SYNCH_TLO_OrderStatus.Retrieve というラベルの付いたイベントが表示されます。ビジネス・オブジェクトをダブルクリックしてウィンドウに表示します。
10. ビジネス・オブジェクトの Request 属性を展開します。Request 属性のタイプは SERVICE_SYNCH_OrderStatus_Request です。SERVICE_ASYNCH_Order の属性 OrderId をインスペクションして、状況が必要なオーダーであることを確認します。
    - オーダーの状況がわかっている場合は、以下の手順を実行します。
      a. SERVICE_SYNCH_TLO_OrderStatus の **Response** 属性をクリックします。Response 属性のタイプは CLIENT_SYNCH_OrderStatus_Response です。
      b. **Response** 属性を右マウス・ボタンでクリックします。
      c. 「**インスタンスを追加**」オプションをクリックします。CLIENT_SYNCH_OrderStatus_Response ビジネス・オブジェクトの新規インスタンスが作成されます。
      d. **OrderId**、**CustomerId** に値を入力します。さらに、このオーダーについてわかっているその他の詳細すべてに値を入力します。このオーダーの詳細をすべて入力したら、このウィンドウを閉じます。
    - オーダーの状況がわからない場合は、以下の手順を実行します。
      a. SERVICE_SYNCH_TLO_OrderStatus の **Fault** 属性をクリックします。Fault 属性のタイプは CLIENT_SYNCH_OrderStatus_Fault です。
      b. **Fault** 属性を右マウス・ボタンでクリックします。
      c. 「**インスタンスを追加**」オプションをクリックします。CLIENT_SYNCH_OrderStatus_Fault の新規インスタンスが作成されます。
      d. faultcode、faultstring、および SOAP 障害メッセージで送信するその他の詳細すべてに値を入力します。この障害の値をすべて入力したら、このウィンドウを閉じます。
11. 「**要求**」”「**返信**」”「**成功**」を選択します。SampleSAPConnector をシミュレートしている Test Connector の右パネルに、SERVICE_SYNCH_TLO_OrderStatus.Retrieve というラベルの付いたイベントが表示されます。
12. **SERVICE_SYNCH_TLO_OrderStatus.Retrieve** ビジネス・オブジェクトをダブルクリックします。ダブルクリックすると、ビジネス・オブジェクトがウィンドウに表示されます。
    - SampleSiebelConnector によってオーダーの状況が戻された場合は、取り込まれたビジネス・オブジェクトの Response 属性が表示されます。**Response** 属性を展開してオーダーの状況を確認します。
    - SampleSiebelConnector によって障害が戻された場合は、取り込まれたビジネス・オブジェクトの Fault 属性が表示されます。**Fault** 属性を展開して障害を判別します。
13. ビジネス・オブジェクトのインスペクションが完了したら、ウィンドウを閉じます。「**要求**」”「**返信**」”「**成功**」を選択します。

これにより、同期シナリオの実行が完了します。

付録 D. 3.0.x へのマイグレーション

- 『後方互換性』
- 『アップグレード作業』
- 『Web サービス生成ユーティリティー』
- 254 ページの『SOAP データ・ハンドラー』
- 255 ページの『SOAP コネクター』
- 255 ページの『Web サービス・コネクター』

この付録では、Adapter for Web Services バージョン 3.0.x の後方互換性、および Web Services Adapter バージョン 1.x および 2.x からのマイグレーション方法について説明します。

後方互換性

Adapter for Web Services バージョン 3.0.x には、後方互換性はありません。

- どの新規コンポーネント (Web サービス・コネクター、SOAP データ・ハンドラー、WSDL ODA) も、この製品の従来のバージョンでリリースされたコンポーネントと一緒に (共同でも、個別でも) 使用することはできません。
- この製品の従来のバージョンでリリースされたどのコンポーネント (SOAP コネクター、SOAP データ・ハンドラー、Web サービス生成ユーティリティー) も、バージョン3.0.x と一緒に (共同でも、個別でも) 使用することはできません。
- この製品ソリューションの従来のバージョンで作成または使用された成果物は、バージョン 3.0.x では使用できません。
- この製品のバージョン 3.0.x は、バージョン 4.2 よりも前のリリースの WebSphere InterChange Server では使用できません。

アップグレード作業

以下のセクションでは、この製品のバージョン 1.x および 2.x からのコンポーネントをアップグレードする方法について説明します。

Web サービス生成ユーティリティー

Web サービス生成ユーティリティーは利用できなくなりました。これに代えて、コラボレーションを Web サービスとして公開するためには System Manager ツールを使用します。Web サービス生成ユーティリティーによって生成された成果物は、今回のリリースで使用することはできません。

- **プロキシー・クラス** Web サービス・コネクターはイベント通知をサポートするようになりました。したがって、コラボレーションを Web サービスとして公開するためには、プロキシー・クラスが不要になりました。Web サービス生成ユーティリティーで生成されたプロキシー・クラスは、System Manager バージョン 4.2 のツールを使用して Web サービスとして公開されたコラボレーションを呼び出すことはできません。

- **WSDL 文書** Web サービス生成ユーティリティーを使用して InterChange Server バージョン 4.2 のコラボレーション用に WSDL 文書を生成することはできません。これに代えて、System Manager のツールを使用して WSDL 文書を生成する必要があります。詳しくは、 171 ページの『ウィザードの実行』 を参照してください。
  コラボレーションを要求処理に使用できるようにする場合、Web サービス生成ユーティリティーを使用して生成した WSDL 文書は、3.0.x のコネクターで使用可能な WSDL ODA では使用できない可能性があります。

SOAP データ・ハンドラー

SOAP データ・ハンドラーと一緒に使用できるのは Web サービス・コネクターのみです。このデータ・ハンドラーは、他のコネクターで使用することも、Server Access Interface で使用することもできません。以下のセクションでは、追加サポートについて説明します。

メタオブジェクト

従来のリリースで使用されていたトップレベルの SOAP データ・ハンドラー・メタオブジェクトは、サポートされなくなりました。これに代えて、バージョン 3.0.x の SOAP データ・ハンドラーで使用する、新しいトップレベル・メタオブジェクトを作成する必要があります。このメタオブジェクトは、Classname 属性と SOAPNameHandler 属性のみを備えている必要があります。

新規メタオブジェクトでは、SOAP メッセージからビジネス・オブジェクトへの変換およびビジネス・オブジェクトから SOAP メッセージへの変換のための子メタオブジェクトは不要になりました。したがって、トップレベル・メタオブジェクトでは SOAPToBOConfigMO 属性と BOToSOAPConfigMO 属性を指定しないようにしてください。

これまで SOAP メッセージからビジネス・オブジェクトへの変換とビジネス・オブジェクトから SOAP メッセージへの変換を記述していた子メタオブジェクトは、SOAP 要求、SOAP 応答、および SOAP 障害ビジネス・オブジェクトの子として追加することが必要になりました。詳しくは、 125 ページの『第 5 章 SOAP データ・ハンドラー』および 27 ページの『第 3 章 ビジネス・オブジェクトの要件』を参照してください。

アプリケーション固有情報

新規 SOAP データ・ハンドラーの重要な特徴として、新しいアプリケーション固有情報 (ASI) の機能があります。特定の ASI を SOAP ビジネス・オブジェクトに追加することによっても、この拡張機能を利用することができますが、そのようなことを行う必要はありません。子 SOAP 構成 MO をビジネス・オブジェクトに追加するだけで、従来のリリースのコネクターで作成された SOAP ビジネス・オブジェクトを、バージョン 3.0.x で使用するために配備することができます。

コネクターの互換性

新規 SOAP データ・ハンドラーと一緒に使用できるのは、3.0.x の Web サービス・コネクターのみです。新規 SOAP データ・ハンドラーは、以前のリリースで提供された、SOAP コネクターや Server Access Interface などのコンポーネントで使用することはできません。

古い SOAP データ・ハンドラーを 3.0.x の Web サービス・コネクターで使用することはできません。

SOAP コネクター

SOAP コネクターは、リリース 3.0.x ではサポートされません。Web サービスを呼び出すためには、これに代えて Web サービス・コネクターを使用しなければなりません。

Web サービス・コネクター

リリース 3.0.x では、コラボレーションを Web サービスとして公開する場合にも、Web サービスを呼び出す場合にも、Web サービス・コネクターを使用してください。コラボレーションを Web サービスとして公開するためには、新規のイベント通知機能が使用されます。新規の要求処理機能が Web サービスの呼び出しに使用されるようになりました。以下のセクションでは、Web サービス・コネクターを使用するために行う必要のあるマイグレーション作業を中心に説明します。

注: 以下で述べるマイグレーション作業は、網羅的なものではありません。また、これらの作業は、任意の順序で行うことができます。

イベント通知

3.0.x コネクターはコラボレーションを同期的に呼び出すことも、非同期的に呼び出すこともできます。この場合、Web サーバーにプロキシー・クラスを作成し、配備する必要はありません。コネクターが、コネクターにイベントを通知するリスナー・フレームワークを備えるようになりました。リスナー・フレームワークは SOAP/HTTP、SOAP/HTTPS、および SOAP/JMS バインディングをサポートします。リスナーを正しく構成すると、コネクターは、Web サービスとして公開されているコラボレーションのために、Web サービス・クライアントを検出し、それに応答できるようになります。リスナー・フレームワークとその構成方法の詳細については、70 ページの『プロトコル・リスナー』を参照してください。

イベント通知用のビジネス・オブジェクト: イベント通知のトップレベル・オブジェクト (TLO) を作成する必要があります。TLO は、SOAP 要求ビジネス・オブジェクト用、かつ (同期要求処理の場合の) オプションで SOAP 応答ビジネス・オブジェクトと SOAP 障害ビジネス・オブジェクト用のコンテナーです。TLO の構造コンポーネントは、単一 Web サービス・オペレーションを想定しています。SOAP 要求ビジネス・オブジェクトは SOAP 要求メッセージに対応し、SOAP 応答ビジネス・オブジェクトは SOAP 応答メッセージに対応し、SOAP 障害ビジネス・オブジェクトは SOAP 障害メッセージに対応します。詳しくは、28 ページの『同期イベント処理 TLO』を参照してください。

イベント通知と SOAP ビジネス・オブジェクト: 従来のリリースで Server Access Interface とともに使用されていた SOAP ビジネス・オブジェクトは、上記 254 ページの『SOAP データ・ハンドラー』で述べたように変更することにより、3.0.x リリースで使用することができます。イベント通知 TLO で SOAP ビジネス・オブジェクトを子として指定しなければならないことに注意してください。

要求処理

従来のリリースにおける SOAP コネクターと同じように、3.0.x の Web サービス・コネクターは Web サービスを呼び出すことができます。さらに、新規コネクターは SOAP/JMS バインディングによる Web サービスの呼び出しもサポートします。以下のセクションでは、コネクター要求処理における変更点についてさらに説明します。

トップレベル・オブジェクトの要求処理: 要求処理用の TLO を作成する必要があります。TLO は、SOAP 要求ビジネス・オブジェクト用、かつ (同期要求処理の場合の) オプションで SOAP 応答ビジネス・オブジェクトと SOAP 障害ビジネス・オブジェクト用のコンテナーです。TLO の構造コンポーネントは、単一 Web サービス・オペレーションを想定しています。SOAP 要求ビジネス・オブジェクトは SOAP 要求メッセージに対応し、SOAP 応答ビジネス・オブジェクトは SOAP 応答メッセージに対応し、SOAP 障害ビジネス・オブジェクトは SOAP 障害メッセージに対応します。この意味で、3.0.x の要求処理 TLO は、従来のリリースの SOAP コネクターとともに使用されていた TLO に対応します。要求処理 TLO に関する詳細については、47 ページの『同期要求処理 TLO』を参照してください。

SOAP ビジネス・オブジェクト: 従来のリリースで SOAP コネクターとともに使用されていた SOAP ビジネス・オブジェクトは、254 ページの『SOAP データ・ハンドラー』で述べたように変更することにより、使用することができるようになります。これらのビジネス・オブジェクトは、要求処理 TLO の子として指定する必要があります。従来のリリースでは、これらのビジネス・オブジェクトは SOAP コネクターで使用される TLO の子でした。

3.0.x の Web サービス・コネクターには、SOAP 要求ビジネス・オブジェクトに関する追加要件があります。それぞれの SOAP 要求ビジネス・オブジェクトには、プロトコル構成 MO (メタオブジェクト) タイプの属性が備わっていなければなりません。コネクターはプロトコル構成 MO を使用して、要求メッセージの宛先を判別します。それぞれのプロトコル構成 MO には、ターゲット Web サービスのアドレスを表す Destination 属性が指定されています。ターゲット Web サービスを呼び出すために SOAP/HTTP または SOAP/HTTPS を使用している場合には、Destination 属性は、従来のリリースの SOAP コネクターで使用される TLO の URL 属性に対応します。詳しくは、35 ページの『プロトコル構成 MO』を参照してください。

付録 E. HTTPS/SSL の構成

SSL の使用を計画している場合は、鍵ストア、証明書、および鍵生成を管理するために、サード・パーティーのソフトウェアを使用する必要があります。Web サービス・コネクターにはこれらの作業用のツールは備わっていません。ただし、IBM JRE に同梱の Keytool の使用を選択して、自己署名証明書を作成し、鍵ストアを管理することもできます。

鍵および証明書管理ユーティリティー、すなわち鍵ツールにより自己所有の公開鍵/秘密鍵の鍵ペアおよび関連証明書の管理が可能です。これらは、自己認証 (他のユーザーまたはサービスに対して自分自身を認証させる)、またはデータ保全性もしくはディジタル署名を使用する認証サービス用に使用されます。さらに鍵ツール・ユーティリティーでは、対等の通信対象者の公開鍵を証明書の形態で保管することが可能です。

この付録では、鍵ツールを使用して鍵ストアのセットアップを行う方法について説明します。この付録は、具体的な例の説明のみを意図しており、鍵ツールまたは関連製品の資料を置き換えるものではありませんので、ご注意ください。鍵ストアのセットアップでツールを使用する際は、必ずソース資料を参照してください。鍵ツールの詳細については、次の web サイトを参照してください。

- http://java.sun.com/j2se/1.3/docs/tooldocs/tools.html#security

鍵ストアのセットアップ

鍵ツールを使用して鍵ストアを作成するには、まず最初に鍵ストアに鍵ペアを作成する必要があります。例えば、コマンド行に次のように入力します。

```
keytool -genkey -alias wsadapter -keystore c:¥security¥keystore
```

これにより、鍵ツールからパスワードを入力するよう求めるプロンプトが出されます。鍵ツール・パラメーター内で選択したパスワードを入力することも可能ですが、鍵ツールで入力したパスワードは、SSL ” KeyStorePassword コネクター・プロパティーの値として指定する必要があります。詳しくは、 115 ページの『KeyStorePassword』を参照してください。

このコマンド例は、`c:¥security¥keystore` ディレクトリーで命名された `keystore` の鍵ストアを作成します。したがって、`c:¥security¥keystore` を SSL ” KeyStore コネクター階層プロパティーの値として入力することができます。また上記例のように、コマンド行から `-alias wsadapter` を SSL ” KeyStoreAlias コネクター階層プロパティーの値として入力することもできます。次に、鍵ツール・ユーティリティーから、証明書の詳細についてプロンプトが出されます。以下は、各プロンプトとその入力例を示しています。(鍵ツール文書を参照してください。)

```
What is your first and last name?
     [Unknown]:  HostName
What is the name of your organizational unit?
     [Unknown]:  wbi
What is the name of your organization?
     [Unknown]:  IBM
What is the name of your City or Locality?
     [Unknown]:  Burlingame
What is the name of your State or Province?
     [Unknown]:  CA
What is the two-letter country code for this unit?
    [Unknown]:  US
Is <CN=HostName, OU=wbi, O=IBM, L=Burlingame,
ST=CA, C=US> correct?
    [no]:  yes
```

鍵ツールからパスワードを入力するよう求める次のプロンプトが出されます。

```
Enter key password for <wsadapter> (RETURN if same as keystore password):
```

同じパスワードを使用するには「Return」を選択します。自己署名証明書を使用する場合は、上記で作成済みの証明書のエクスポートが可能です。その場合は、コマンド行で次のように入力します。

```
keytool -export -alias wsadapter -keystore c:¥security¥keystore -file wsadapter.cer
```

ここで、鍵ツールから、鍵ストア・パスワードを入力するよう求めるプロンプトがでます。上記で入力したパスワードを入力します。

トラストストアのセットアップ

トラストストアを次のようにセットアップしたい場合について説明します。SOAP/HTTPS プロトコル・リスナーを web サービス・クライアントに認証したい場合は、SSL ” UseClientAuth コネクターの構成プロパティーを `true` に設定します。このケースでは、SOAP/HTTPS プロトコル・リスナーは、すべてのトラステッド web サービス・クライアントに対応した証明書を、トラストストアが収容するよう要求します。コネクターは、JSSE デフォルト・メカニズムを使用してクライアントを信頼することに注意してください。SOAP/HTTPS web サービスを呼び出す場合は、SOAP/HTTP-HTTPS プロトコル・ハンドラーはトラストストアが web サービスを信頼することを要求します。これは、トラストストアがトラステッド web サービスの証明書をすべて収容していなければならないことを意味しています。コネクターは、JSSE デフォルト・メカニズムを使用してクライアントを信頼することに注意してください。トラストストアにトラステッド証明書をインポートするには、次のようにコマンドを入力します。

```
keytool -import -alias trusted1 -keystore c:¥security¥truststore -file
 c:¥security¥trusted1.cer
```

ここで、鍵ツールから、鍵ストア・パスワードを入力するよう求めるプロンプトがでます。`-keystore c:¥security¥truststore` と入力する場合は、SSL ” TrustStore 階層プロパティーが `c:¥security¥truststore` に設定されていることを必ず確認してください。また、SSL ” TrustStorePassword 階層プロパティーの値を、前に入力したパスワードに設定する必要があります。

公開鍵証明書用の証明書署名要求 (CSR) の生成

ユーザーの身元を信頼しているトラステッド・パートナーの中での SSL データ交換では、自己署名証明書は適切と考えられます。ただし、証明書が証明機関 (CA) により署名されていれば、より他者から信頼されやすくなります。

鍵ツール・ユーティリティーを使用して CA で署名された証明書を取得するには、まず最初に Certificate Signing Request (CSR) を生成し、その CSR を CA に渡します。次に、CA は証明書に署名してユーザーに戻します。

次のコマンドの入力により CSR が生成されます。

```
keytool -certreq -alias wsadapter -file wsadapter.csr
 -keystore c:¥security¥keystore
```

このコマンドで、alias は、秘密鍵用に作成した鍵ストアの別名です。鍵ツール・ユーティリティーは CA に提供する CSR ファイルを生成します。次に CA より署名済みの証明書が提供されます。この証明書を鍵ストアにインポートする必要があります。そのためには、次のコマンドを入力します。

```
keytool -import -alias wsadapter -keystore c:¥security¥keystore -trustcacerts
-file casignedcertificate.cer
```

一度インポートすると、鍵ストアの自己署名証明書は CA 署名済み証明書で置き換えられます。

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。一般使用プログラミング・インターフェースにより、お客様はこのプログラム・ツール・サービスを含むアプリケーション・ソフトウェアを書くことができます。ただし、この情報には、診断、修正、および調整情報が含まれている場合があります。診断、修正、調整情報は、お客様のアプリケーション・ソフトウェアのデバッグ支援のために提供されています。

警告: 診断、修正、調整情報は、変更される場合がありますので、プログラミング・インターフェースとしては使用しないでください。

商標

以下は、IBM Corporation の商標です。
IBM
IBM ロゴ
AIX
CICS
CrossWorlds
DB2
DB2 Universal Database
IMS
Informix
i5/OS
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.ecclipse.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

索引

日本語, 数字, 英字, 特殊文字の順に配列されています。なお, 濁音と半濁音は清音と同等に扱われています。

[ア行]

[カ行]

[サ行]

[タ行]

[ハ行]

[マ行]

[ヤ行]

[ラ行]

[数字]

A

C

D

E

H

J

K

N

S

W

X

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