



# BEA WebLogic Integration™

## B2B Integration サンプルの使い方

## 著作権

Copyright © 2002, BEA Systems, Inc. All Rights Reserved.

## 限定的権利条項

本ソフトウェアおよびマニュアルは、BEA Systems, Inc. 又は日本ビー・イー・エー・システムズ株式会社（以下、「BEA」といいます）の使用許諾契約に基づいて提供され、その内容に同意する場合にのみ使用することができ、同契約の条項通りにのみ使用またはコピーすることができます。同契約で明示的に許可されている以外の方法で同ソフトウェアをコピーすることは法律に違反します。このマニュアルの一部または全部を、BEA Systems, Inc. からの書面による事前の同意なしに、複写、複製、翻訳、あるいはいかなる電子媒体または機械可読形式への変換も行うことはできません。

米国政府による使用、複製もしくは開示は、BEA の使用許諾契約、および FAR 52.227-19 の「Commercial Computer Software-Restricted Rights」条項のサブパラグラフ (c)(1)、DFARS 252.227-7013 の「Rights in Technical Data and Computer Software」条項のサブパラグラフ (c)(1)(ii)、NASA FAR 補遺 16-52.227-86 の「Commercial Computer Software--Licensing」条項のサブパラグラフ (d)、もしくはそれらと同等の条項で定める制限の対象となります。

このマニュアルに記載されている内容は予告なく変更されることがあり、また BEA による責務を意味するものではありません。本ソフトウェアおよびマニュアルは「現状のまま」提供され、商品性や特定用途への適合性を始めとする（ただし、これらには限定されない）いかなる種類の保証も与えません。さらに、BEA は、正当性、正確さ、信頼性などについて、本ソフトウェアまたはマニュアルの使用もしくは使用結果に関していかなる確約、保証、あるいは表明も行いません。

## 商標または登録商標

BEA、Jolt、Tuxedo、および WebLogic は BEA Systems, Inc. の登録商標です。BEA Builder、BEA Campaign Manager for WebLogic、BEA eLink、BEA Manager、BEA WebLogic Commerce Server、BEA WebLogic Enterprise、BEA WebLogic Enterprise Platform、BEA WebLogic Express、BEA WebLogic Integration、BEA WebLogic Personalization Server、BEA WebLogic Platform、BEA WebLogic Portal、BEA WebLogic Server、BEA WebLogic Workshop および How Business Becomes E-Business は、BEA Systems, Inc の商標です。

その他の商標はすべて、関係各社が著作権を有します。

## **BEA Integration サンプルの使い方**

パート番号	日付	ソフトウェアのバージョン
なし	2002年6月	7.0

---

# 目次

## このマニュアルの内容

対象読者.....	viii
e-docs Web サイト.....	viii
このマニュアルの印刷方法.....	viii
関連情報.....	ix
サポート情報.....	ix
表記規則.....	x

## 1. はじめに

サンプルの概要.....	1-1
サンプルの実行前の作業.....	1-3
デフォルト データベースの切り替え.....	1-4
WebLogic Integration サンプルドメイン.....	1-5
ブラウザ プロキシの設定.....	1-5

## 2. Hello Partner サンプル（非推奨）

Hello Partner サンプルの概要.....	2-1
サンプルの内容.....	2-1
Hello Partner サンプルのシナリオのロジック.....	2-2
Hello Partner サンプルを実行する前に.....	2-4
Hello Partner サンプルの実行.....	2-4
サンプルの仕組み.....	2-9
交換されるドキュメント.....	2-9
要求側ロールからの要求メッセージ.....	2-10
応答側ロールからの応答メッセージ.....	2-10
プライベート ワークフローをトリガするためにサーブレットから JMS 経由で送信される XML メッセージ.....	2-10
プライベート ワークフローからサーブレットに JMS 経由で送信さ れる XML メッセージ.....	2-11
応答側パブリック ワークフローから応答側プライベート ワークフ ローに送信される XML イベント.....	2-11
応答側プライベート ワークフローから応答側パブリック ワークフ	

ローに送信される XML イベント .....	2-11
要求側プライベート ワークフロー .....	2-12
要求側パブリック ワークフロー .....	2-13
応答側パブリック ワークフロー .....	2-15
応答側プライベート ワークフロー .....	2-17
<b>3. Channel Master サンプル (非推奨)</b>	
Channel Master サンプルの概要 .....	1-1
Channel Master サンプルを実行する前に .....	1-4
Channel Master サンプルの実行 .....	1-4
Channel Master サンプルのワークフロー .....	1-10
SupplierOnePrivate ワークフローの表示 .....	1-17
マルチキャスト (ブロードキャスト) メッセージ .....	1-26
<b>4. RosettaNet 2.0 Security サンプル</b>	
RosettaNet 2.0 Security サンプルの概要 .....	1-1
RosettaNet 2.0 Security サンプルの概要 .....	1-2
RosettaNet 2.0 Security サンプルを実行する前に .....	1-4
RosettaNet 2.0 Security サンプルの実行 .....	1-4
RosettaNet 2.0 Security サンプルにおけるワークフロー .....	1-7
ワークフローの要点 .....	1-10
<b>5. Messaging API サンプル (非推奨)</b>	
Messaging API サンプルの概要 .....	1-1
Messaging API サンプルを実行する前に .....	1-3
Messaging API サンプルの実行 .....	1-3
実行フローのトレース .....	1-7
<b>6. Trading Partner Zeroweight Client サンプル (非推奨)</b>	
Zeroweight Client サンプルの概要 .....	1-2
サンプルの目的 .....	1-2
Zeroweight Client サンプルのシナリオおよび図 .....	1-3
Zeroweight Client サンプルを実行する前に .....	1-5
Zeroweight Client サンプルの実行 .....	1-6
ゼロウェイト クライアントの作成および使用 .....	1-17
ゼロウェイト クライアントのソース ファイル .....	1-17

JSP タグ ライブラリの使用.....	1-19
Zeroweight Client のコンフィグレーション.....	1-20
ファイル共有クライアントのコンフィグレーション .....	1-20
WebLogic Integration コンフィグレーション ファイルの編集 .....	1-21
ファイル共有コンフィグレーション ファイルの編集.....	1-21
ブラウザ クライアントのコンフィグレーション.....	1-23
HTTP ブラウザ クライアントのコンフィグレーション .....	1-23
HTTPS (SSL) ブラウザ クライアントのコンフィグレーション.....	1-25
サンプルの再コンパイル方法 .....	1-25

## 7. ebXML サンプル

ebXML サンプルの概要.....	1-1
ebXML サンプルを実行する前に .....	1-3
ebXML サンプルの実行.....	1-3
サンプルの仕組み .....	1-7
概要 .....	1-8
リポジトリ データのロード .....	1-8
リポジトリ データについて.....	1-10
ビジネス プロトコル定義.....	1-10
ロジック プラグイン .....	1-11
トレーディング パートナ.....	1-12
会話定義.....	1-12
コラボレーション アグリーメント .....	1-13
ワークフローについて.....	1-14
WebLogic Integration Studio の使い方 .....	1-14
ebXMLConversationInitiator ワークフローについて .....	1-16
ebXMLConversationResponder ワークフローについて .....	1-25

## A. JSP タグ リファレンス

SendmsgTag .....	1-2
ChecknewmsgTag .....	1-3
CheckallmsgTag .....	1-4
ReadmsgTag .....	1-5
DeletemsgTag.....	1-6
DeleteallmsgTag.....	1-7
CreatemboxTag .....	1-8

---

RemovemboxTag ..... 1-9

---

# このマニュアルの内容

このマニュアルでは、**WebLogic Integration** で用意されている企業間（B2B）統合のサンプルについて説明します。コンフィグレーションに関する情報と、各サンプルの使い方および確認方法について説明します。

このマニュアルの内容は以下のとおりです。

- 第1章「はじめに」では、**WebLogic Integration B2B** サンプルの概要と、基本的なインストールおよびコンフィグレーション方法について説明します。
- 第2章「Hello Partner サンプル（非推奨）」では、デフォルトの **XOCP** メッセージング プロトコルを使用した通信を示します。
- 第3章「Channel Master サンプル（非推奨）」では、**WebLogic Integration** トレーディング パートナ間での通信方法として、**XOCP** ビジネスプロトコルを使用するポイント ツー ポイント通信とマルチキャスト（ブロードキャスト）通信の両方を示します。
- 第4章「RosettaNet 2.0 Security サンプル」では、**WebLogic Integration** を使用して **RosettaNet 2.0 PIP 3A2** および **PIP 0A1** をワークフローで実装する方法について示します。
- 第5章「Messaging API サンプル（非推奨）」では、**WebLogic Integration Messaging API** の使い方について説明します。
- 第6章「Trading Partner Zeroweight Client サンプル（非推奨）」では、ブラウザクライアントおよびファイル共有クライアントのサンプルのコンフィグレーション方法と使い方について説明します。
- 第7章「ebXML サンプル」では、デフォルトの **ebXML** メッセージング プロトコルを使用した通信を示します。

---

# 対象読者

このマニュアルは、WebLogic Integration 環境を拡張する独立ソフトウェアベンダ (ISV) を対象としています。BEA WebLogic Integration のプラットフォームおよび Java プログラミングに読者が精通していることを前提として書かれています。

## e-docs Web サイト

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

## このマニュアルの印刷方法

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

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

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



---

## 関連情報

目的にあった BEA WebLogic Integration の使い方について学習するには、次のマニュアルを参照してください。

- 次の URL にある WebLogic Integration のマニュアル  
<http://edocs.beasys.co.jp/e-docs/>
- 次の URL にある Sun Microsystems, Inc の Java サイト  
<http://java.sun.com/>

## サポート情報

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

電子メールのメッセージには、ご使用の WebLogic Integration のリリースをお書き添えください。

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

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

- お名前、電子メール アドレス、電話番号、ファクス番号
- 会社の名前と住所
- お使いの機種とコード番号

- 製品の名前とバージョン
- 問題の状況と表示されるエラー メッセージの内容

## 表記規則

このマニュアルでは、全体を通して以下の表記規則が使用されています。

表記法	適用
太字	用語集で定義されている用語を示す。
[Ctrl] + [Tab]	複数のキーを同時に押すことを示す。
斜体	強調または書籍のタイトルを示す。
等幅テキスト	コード サンプル、コマンドとそのオプション、データ構造体とそのメンバー、データ型、ディレクトリ、およびファイル名とその拡張子を示す。等幅テキストはキーボードから入力するテキストも示す。 <i>例</i> <pre>#include &lt;iostream.h&gt; void main ( ) the pointer psz chmod u+w * \tux\data\ap .doc tux.doc BITMAP float</pre>
太字の等幅 テキスト	コード内の重要な箇所を示す。 <i>例</i> <pre>void <b>commit</b> ( )</pre>
斜体の等幅 テキスト	コード内の変数を示す。 <i>例</i> <pre>String <i>expr</i></pre>

表記法	適用
すべて大文字のテキスト	デバイス名、環境変数、および論理演算子を示す。 <i>例</i> LPT1 SIGNON OR
{ }	構文の中で複数の選択肢を示す。実際には、この括弧は入力しない。
[ ]	構文の中で任意指定の項目を示す。実際には、この括弧は入力しない。 <i>例</i> buildobjclient [-v] [-o name ] [-f file-list]... [-l file-list]...
	構文の中で相互に排他的な選択肢を区切る。実際には、この記号は入力しない。
...	コマンドラインで以下のいずれかを示す。 <ul style="list-style-type: none"> <li>■ 引数を複数回繰り返すことができる</li> <li>■ 任意指定の引数が省略されている</li> <li>■ パラメータや値などの情報を追加入力できる</li> </ul> 実際には、この省略記号は入力しない。 <i>例</i> buildobjclient [-v] [-o name ] [-f file-list]... [-l file-list]...
.	コード サンプルまたは構文で項目が省略されていることを示す。実際には、この省略記号は入力しない。



---

# 1 はじめに

WebLogic Integration では、B2B Integration がどのように機能するかについて理解を深めるため、企業間（B2B）統合のサンプルが用意されています。また、Hello Partner サンプルが正常にインストールおよびセットアップされていれば、WebLogic Integration のインストールとコンフィグレーションが適切に行われたことの確認にもなります。

この章では、以下のトピックを取り上げます。

- サンプルの概要
- サンプルの実行前の作業

## サンプルの概要

この節では、このマニュアルで解説する、Hello Partner、Channel Master、RosettaNet 2.0 Security、Trading Partner Zeroweight Client、Messaging API および ebXML の 7 つのサンプルの概要を説明します。

**注意：** ただし、そのうち 4 サンプルは、今回のリリースから非推奨になった機能に基づいています。Hello Partner、Channel Master、Trading Partner Zeroweight Client の各サンプルはすべて、XOCP プロトコルに基づいていますが、このプロトコルも今回のリリースから非推奨になっています。Messaging API サンプルは Messaging API に基づいていますが、これも既に非推奨になっています。XOCP および Messaging API に代わる機能については、『WebLogic Integration リリース ノート』を参照してください。

Hello Partner と ebXML のサンプルを使用すると、XOCP および ebXML メッセージング プロトコルが正常に使用できるかどうかを確認できます。それ以外のサンプルは、一般的なビジネス上の問題を B2B Integration でいかに解決するか説明するためのものです。

以下は、各サンプルの概要説明です。

- **Hello Partner** サンプルー通信に必要となる基本的なハンドシェークを示します。Hello Partner サンプルでは、**XOCP** ビジネス プロトコルを使用します。
  - **Channel Master** サンプルー大規模トレーディング パートナが、**WebLogic Integration** を使用してサプライ チェーンを自動化する方法を示します。サンプルでは、**WebLogic Integration** トレーディング パートナ間での通信方法として、**XOCP** ビジネス プロトコルを使用するポイント ツー ポイント通信とマルチキャスト（ブロードキャスト）通信の両方が示されています。
  - **RosettaNet 2.0 Security** サンプルー **WebLogic Integration** で、ワークフローを使用して **RosettaNet 2.0 PIP 3A2** および **PIP 0A1** を実装する方法を示します。このサンプルでは、**RosettaNet 2.0** のサポートに必要となる **WebLogic Integration** セキュリティ機能（双方向 **SSL** 認証、デジタル署名、データ暗号化、および否認防止性）を使用します。
  - **Trading Partner Zeroweight Client** サンプルー 2 つのトレーディング パートナ間で、**B2B Integration** メールボックス インタフェースを使用する **B2B Integration** をインストールせずに要求と応答を行うシナリオを示します。このサンプルでは、以下の 2 種類のゼロウェイト クライアント通信がシュミレートされます。
    - ブラウザ ベースのクライアントー情報の準備、配信、収集に **XML** と **JMS** を使用します。
    - ファイル共有クライアントーメッセージ交換にサードパーティのファイル共有サーバを使用します。
- 注意：** **Trading Partner Zeroweight Client** サンプルは、**XOCP** プロトコルに基づいていますが、このプロトコルは、このリリースの **WebLogic Integration** から非推奨になっています。**XOCP** に代わる機能については、『*WebLogic Integration* リリース ノート』を参照してください。
- **Messaging API** サンプルー **WebLogic Integration Messaging API** の使用方法を示します。特に、**Messaging API** で提供される 2 つのメッセージ配信メカニズムの使用方法、および **WebLogic Integration B2B** 統合のロジック プラグイン機能を示します。
  - **ebXML Sample** – **WebLogic Integration** を使用して、それぞれ **WebLogic Integration** をデプロイする 2 つのトレーディング パートナ間の **ebXML** ビジネス トランザクションを実装する方法を示します。具体的には、各トレーディング パートナに対応する 2 つのワークフローの設計と用途を示します。

これらのワークフローは、トレーディング パートナ間の ebXML ベースのビジネス メッセージの交換を管理します。

## サンプルの実行前の作業

B2B サンプルを実行する前に、WebLogic Platform の 2 つのコンポーネントである WebLogic Server および WebLogic Integration と、そのサンプルアプリケーションをインストールする必要があります。また、サンプルのコンフィグレーションも必要です。これらの作業をまだ行っていない場合は以下の手順を実行します。

1. BEA WebLogic Platform をインストールします。

手順説明は、BEA WebLogic Platform マニュアルセットの『*WebLogic Platform インストールガイド*』を参照してください。次の URL にあります。

<http://edocs.beasys.co.jp/e-docs/platform/docs70/install/index.html>

インストール中に、次の 2 つの手順が選択できます。

- [標準インストール] – このオプションを選択すると、インストールプログラムによって、WebLogic Server、WebLogic Integration、WebLogic Portal および WebLogic Workshop の各コンポーネントとそれぞれのサンプルアプリケーションが自動的にインストールされます。
- [カスタム インストール] – このオプションを選択すると、[コンポーネントを選択] ウィンドウが表示されます。少なくとも、以下のコンポーネントを選択する必要があります。

WebLogic Server ([Server] と [Server Examples] を指定)

WebLogic Integration ([Integration Server] と [Integration Examples] を指定)

デフォルトでは、コンフィグレーション済みのサンプルドメインは、PointBase データベースを使用するコンフィグレーションとなっています。

2. コンフィグレーション済みのサンプルドメインで、サーバをコンフィグレーションして起動します。

手順については、『*WebLogic Integration の起動、停止およびカスタマイズ*』、「はじめに」の「サンプルドメインのコンフィグレーションと起動」を参照してください。

「サンプルドメインのコンフィグレーションと起動」に従って `RunSamples` コマンドを使用し、以下を実行します。

- サンプルデータベースが作成されます。
- サンプルリポジトリ データがデータベースにバルク ロードされます。
- **WebLogic Server** が起動し、ワークフローがインポートされます。ブラウザが起動し、サンプル起動ページが表示されます。

**注意：** UNIX システム上でサンプル起動ページを表示するには、**Netscape** の起動ディレクトリ (`netscape`) が `PATH` 環境変数に含まれている必要があります。**Web** ブラウザ コンフィグレーションの要件の詳細については、『*WebLogic Integration の起動、停止およびカスタマイズ*』の「**WebLogic Integration** 管理ツールと設計ツール」の「**Web** ブラウザ コンフィグレーションの要件」を参照してください。

サンプルをコンフィグレーションしてサンプルドメインでサーバを起動し、そのサンプルのドキュメントに記載されている手順に従って特定のサンプルをコンフィグレーションします。**Windows** システム用と **UNIX** システム用の手順が記載されています。

## デフォルト データベースの切り替え

サンプルドメインでサポートされるデータベースはすべて使用できます。**BEA WebLogic Integration** データベース コンフィグレーション ウィザードを使用すると、インストール後にサンプルドメインを新しいデータベースに更新できます。サンプルドメインが使用するデータベースを変更する場合は、次の手順を実行します。

1. データベース ウィザードを使用して、新しいデータベースに切り替えます。

手順については、『*WebLogic Integration の起動、停止およびカスタマイズ*』、「**WebLogic Integration** のカスタマイズ」の「データベース コンフィグレーション ウィザードの使用」および「ドメインに対する新しいデータベースの指定」を参照してください。



2. `RunSamples` コマンドを実行して新しいデータベースをコンフィグレーションするには、サンプルドメインを起動し、Web ブラウザにサンプル起動ページを表示します。

手順については、『*WebLogic Integration の起動、停止およびカスタマイズ*』の「はじめに」の「サンプルドメインのコンフィグレーションと起動」を参照してください。

## WebLogic Integration サンプルドメイン

『*WebLogic Integration の起動、停止およびカスタマイズ*』の「はじめに」で説明されているように、ドメインとは相互に関連した **WebLogic Server** リソースの集まりであり、単一のコンフィグレーション ファイルで定義されます。特別な **WebLogic** サーバドメインは、1 つのサーバのみを必要とするすべての **WebLogic Integration** サンプルを実行するために設定されます。このサンプルドメインのコンフィグレーションはインストール中に完了しますが、リソースはまだ格納されていません。このサンプルドメインは、`BEA_HOME/weblogic700/samples/integration/config/samples` ディレクトリにあります。ここで、`BEA_HOME` は、**BEA** 製品の最上位ディレクトリで、たとえば、**Windows** の場合は、`c:\bea` です。

サンプルドメインは、デフォルトでは **PointBase** データベースを使用します。デフォルト データベースは、「デフォルト データベースの切り替え」の説明にあるとおり、データベース ウィザードを使用して変更できます。

**WebLogic Integration** と共にインストールされるドメインの使用法の詳細については、『*WebLogic Integration の起動、停止およびカスタマイズ*』の「はじめに」を参照してください。

## ブラウザ プロキシの設定

ブラウザからサンプル起動ページの URL に接続できない場合、プロキシ サーバを使用しているためにローカルの **WebLogic Server** に接続できないことが原因である可能性があります。この場合、次のエラー メッセージが表示されます。

```
The requested URL could not be retrieved
```

プロキシサーバを使用しないようにするには、ブラウザのプロキシ設定を変更して、サブレットへのアクセスにプロキシサーバが使用されないようにします。この変更を行う手順は、使用しているブラウザによって異なります。

- **Internet Explorer** では、[ ツール | インターネット オプション | 接続 | LAN の設定 ] を選択します。[ ローカル エリア ネットワーク (LAN) の設定 ] ダイアログボックスが表示されます。[ ローカル アドレスにはプロキシサーバを使用しない ] を選択します。
- **Netscape** では、[ 編集 | プリファレンス | 詳細 | プロキシ | 表示 ] を選択します。[ 例外 ] テキスト フィールドで、`localhost:listening_port` を指定します。`listening_port` には、`config.xml` ファイルで指定されているリスニングポート番号を指定します。デフォルトは **7001** です。

---

## 2 Hello Partner サンプル（非推奨）

Hello Partner サンプルでは、デフォルト メッセージング プロトコルである XOCP を使用した通信が示されています。この章では、以下のトピックについて説明します。

- Hello Partner サンプルの概要
- Hello Partner サンプルを実行する前に
- Hello Partner サンプルの実行
- サンプルの仕組み

**注意：** Hello Partner サンプルは、XOCP プロトコルに基づいていますが、このプロトコルは、このリリースの **WebLogic Integration** から非推奨になっています。XOCP に代わる機能については、『*WebLogic Integration リリース ノート*』を参照してください。

### Hello Partner サンプルの概要

Hello Partner サンプルでは、**WebLogic Integration** を使用した 2 つのトレーディング パートナ間のビジネス通信を示します。

### サンプルの内容

Hello Partner サンプルでは、2 つのトレーディング パートナが **XOCP** プロトコルを使用してビジネス メッセージを送信する方法が示されています。各トレーディング パートナについて以下が示されます。

- パブリック プロセス—トレーディング パートナ間の通信を処理するパブリック プロセスを示します。このサンプルのパブリック プロセスワークフ

ローでは、XOCP プロトコルを使用してトレーディング パートナ間のメッセージを送信します。

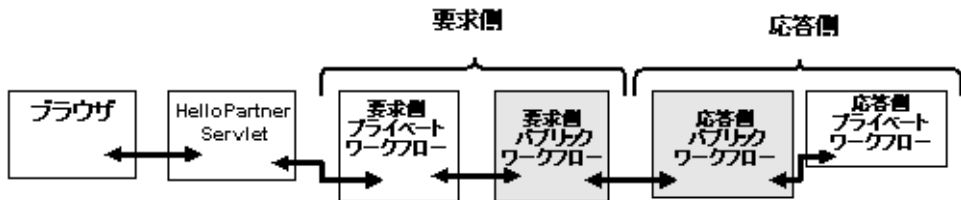
- プライベート プロセサーメッセージの内容を処理するトレーディング パートナごとのプライベート プロセスを示します。

このサンプルでは、トレーディング パートナのメッセージトラフィックを処理するにはどの方法が望ましいかについても示されています。パブリック プロセスはトレーディング パートナのメッセージトラフィックの管理に使用し、プライベート プロセスはメッセージの作成、メッセージの処理、および外部アプリケーションとのリンクに使用します。

## Hello Partner サンプルのシナリオのロジック

Hello Partner サンプルのシナリオでは、図 2-1 に示すように、要求側と応答側の 2 つのトレーディング パートナが設定されています。次の図に、別々の Hello Partner ワークフロー間での上位レベルの会話を示します。

図 2-1 Hello Partner ワークフロー間の会話



**注意：** 上の図では、パブリック ワークフローがグレーで網掛けされています。

このシナリオの主要なイベントをまとめると次のようになります。

1. RunSamples スクリプトが起動され、次の結果が得られます。
  - a. WebLogic Server のサンプル インスタンスが開始されます。
  - b. ブラウザが開き、サンプル起動ページが表示されます。
  - c. サンプル起動ページの [Hello Partner] リンクをクリックすると、Hello Partner のメイン ページが表示されます。

2. メニューから整数の値を選択します。[ サンプルの開始 ] リンクを選択します。その結果、Hello Partner のメインページから HelloPartnerServlet に HTTP 要求が送信されます。HTTP 要求には、選択した整数値が格納されています。
3. HelloPartnerServlet によって、整数値を格納した JMS XML イベントが送信されます。
4. JMS XML イベントによって、RequestorPrivate ワークフローの開始がトリガされます。RequestorPrivate ワークフローは MultiplyRequestXML ワークフロー変数を作成し、MultiplyRequestXML ワークフロー変数を渡して RequestorPublic ワークフローを開始します。その後は、応答側からの応答を待ちます。
5. ReplierPublic ワークフローは、メッセージを受信して MultiplyRequestXML ワークフロー変数から整数値を抽出します。つづいて、これらの値を MultiplyInputsXML ワークフロー変数に格納し、内部 XML イベントをポストします。
6. ReplierPrivate ワークフローは、上記の内部 XML イベントによってトリガされます。ReplierPrivate ワークフローは、2つの整数を乗算した結果を MultiplyOutputsXML ワークフロー変数に格納し、この変数と共に内部 XML イベントをポストします。
7. 上記の内部 XML イベントを待っていた ReplierPublic ワークフローは、MultiplyReplyMessage 入力メッセージ ワークフロー変数を作成し、ビジネスメッセージと共にそれを送信します。
8. 上記のビジネス メッセージを待っていた RequestPublic ワークフローは、MultiplyReplyMessage ワークフロー変数から値を抽出し、MultiplyReplyXML ワークフロー変数を作成して内部 XML イベントと共にポストします。
9. 上記の内部 XML イベントを待っていた RequestPrivate ワークフローは、ResultXMLForJSP ワークフロー変数を作成し、外部 XML イベントと共にポストします。
10. HelloPartnerServlet は外部 XML イベントを受信し、その XML から2つの整数を乗算した結果を抽出してブラウザに表示します。

# Hello Partner サンプルを実行する前に

Hello Partner サンプルを実行する前に、次の手順を実行します。

1. 1-3 ページの「サンプルの実行前の作業」に記載の手順に従います。
2. サンプル WebLogic Server への接続が妨げられないようにブラウザのプロキシ設定を確認します。Web ブラウザ コンフィグレーションの要件の詳細については、『*WebLogic Integration の起動、停止およびカスタマイズ*』の「WebLogic Integration 管理ツールと設計ツール」の「Web ブラウザ コンフィグレーションの要件」を参照してください。

# Hello Partner サンプルの実行

Hello Partner サンプルを実行するには、次の作業が必要です。

1. `WLI_HOME` (WebLogic Integration をインストールしたディレクトリ) に移動します。

```
cd WLI_HOME
```

- Windows の例

WebLogic Platform を `c:\bea` ディレクトリにインストールした場合は、`WLI_HOME` のパスは、`c:\bea\weblogic700\integration` となります。

- UNIX の例

WebLogic Platform を `c:\bea` ディレクトリにインストールした場合は、`WLI_HOME` のパスは、`/home/me/bea/weblogic700/integration` となります。

2. WebLogic Integration の上位レベルの環境変数を設定するには、お使いのプラットフォームに合った `setenv` スクリプトを実行します。

- Windows の場合

```
setEnv
```

- UNIX の場合

```
. setenv.sh
```

3. プラットフォームに合わせて適切な手順を実行し、RunSamples スクリプトを起動します。

- Windows:

[ スタート | プログラム | BEA WebLogic Platform 7.0 | WebLogic Integration 7.0 | Integration Examples | Start Server and Launch Examples (with dataloader) ] を選択します。

- UNIX:

a) PATH 環境変数に、Netscape 実行ファイル (netscape) が格納されたディレクトリが含まれていることを確認します。

b) RunSamples スクリプトを実行します。

```
cd $SAMPLES_HOME/integration/config/samples
RunSamples
```

**警告：** UNIX システムの場合、netscape 実行ファイルが入ったディレクトリが PATH 環境変数に含まれている必要があります。環境変数に含まれていない場合は、RunSamples スクリプトの実行時にサンプル起動ページが表示されません。サンプル起動ページは、現在 RunSamples スクリプトが実行されているマシンで Netscape ブラウザを起動して、次の URL を入力すると起動されます。

```
http://localhost:7001/index.html
```

4. RunSamples スクリプトのコンフィグレーション セクションが実行済みであることが検知されると、次のプロンプトが表示されます。

```
The WebLogic Integration repository has already been
created and populated, possibly from a previous run
of this RunSamples script. Do you want to destroy all the
current data in the repository and create and populate the
WebLogic Integration repository, again? Y for Yes, N for No
```

この質問に N と入力すると、リポジトリの作成および格納を行う手順が省略され、WebLogic Server のサンプル インスタンスを起動する手順のみが実行されます。

この質問に Y と入力すると、リポジトリの作成および格納が改めて行われ、その後で WebLogic Server のサンプル インスタンスを起動する手順が実行されます。Y と入力した場合、その時点でリポジトリに格納されている全データが破棄され、リポジトリにサンプルデータが再ロードされます。現在のサ

## 2 Hello Partner サンプル (非推奨)

サンプルデータが変更または削除され、新規または未変更のサンプルデータをリポジトリに格納する場合にのみ、Yを入力してください。

これで、RunSamples スクリプトは WebLogic Server のインスタンスをバックグラウンド プロセスとして開始し、サンプル起動ページが表示されます。

図 2-2 サンプル起動ページ



5. [Hello Partner] リンクをクリックすると、サンプル起動ページの左ペインの [サンプル アプリケーション] の下にリストが表示されます。右ペインに Hello Partner サンプルが表示されます。



図 2-3 Hello Partner サンプル起動ページ



- ラジオ ボタンで、2 つの番号を選択します。[ サンプルの開始 ] をクリックします。

図 2-4 Hello Partner サンプル結果ページ



7. このとき、さらに多くの B2B サンプルを実行するのであれば、サンプル起動ページを開いたままで、WebLogic Server のインスタンスの実行を続行します。

この時点でこれ以上の B2B サンプルを実行しない場合は、ブラウザを終了し、プラットフォームに合わせた適切な手順によって WebLogic Server のインスタンスをシャットダウンします。

- Windows:

```
cd %SAMPLES_HOME%\integration\config\samples
stopWebLogic
```

- UNIX:

```
cd $SAMPLES_HOME/integration/config/samples
stopWebLogic
```

# サンプルの仕組み

このサンプルでは、合計 4 つのワークフローを使用します。2 つのパブリックワークフローは、**XOCP** メッセージのやりとりの要求側と応答側を管理しています。2 つのプライベートワークフローのうち、1 つはサーブレットおよび要求側のパブリックワークフローに接続するため、もう 1 つは応答側の応答データを作成するためのものです。

以下の節では、このプロセスの概要とそれぞれのワークフローについて詳しく説明します。

- 交換されるドキュメント
- 要求側プライベートワークフロー
- 要求側パブリックワークフロー
- 応答側パブリックワークフロー
- 応答側プライベートワークフロー

## 交換されるドキュメント

Hello Partner サンプルでは、以下の XML ドキュメントを使用します。

- 要求側ロールからの要求メッセージ
- 応答側ロールからの応答メッセージ
- プライベートワークフローをトリガするためにサーブレットから **JMS** 経由で送信される **XML** メッセージ
- プライベートワークフローからサーブレットに **JMS** 経由で送信される **XML** メッセージ
- 応答側パブリックワークフローから応答側プライベートワークフローに送信される **XML** イベント
- 応答側プライベートワークフローから応答側パブリックワークフローに送信される **XML** イベント

これらのドキュメントの文書型定義 (DTD) は、Windows システムの場合、`SAMPLES_HOME\integration\samples\common\dtds` ディレクトリに、UNIX システムの場合、`SAMPLES_HOME/integration/samples/common/dtds` ディレクトリに格納されています。いずれの場合も、`SAMPLES_HOME` は、WebLogic Platform のサンプルディレクトリを表しています。

### 要求側ロールからの要求メッセージ

以下は、要求側から送信される XML メッセージです。メッセージには、乗算される 2 つの数字が格納されています。

```
<multiply-request>
<integer-one>5</integer-one>
<integer-two>7</integer-two>
</multiply-request>
```

メッセージは `multiply-request.dtd` に準拠しています。

### 応答側ロールからの応答メッセージ

以下は、応答側から送信される XML メッセージです。生成されたメッセージと共に、乗算の結果が格納されています。

```
<multiply-reply>
<integer-product>35</integer-product>
<note>Dear RequestorPartner: Here is the product of 7 and 5,
  from ReplierPartner to RequestorPartner.</note>
</multiply-reply>
```

メッセージは `multiply-reply.dtd` に準拠しています。

### プライベート ワークフローをトリガするためにサーブレットから JMS 経由で送信される XML メッセージ

以下は、サーブレットが JMS 経由で送信するメッセージです。メッセージを受信すると、要求側のプライベートワークフローがトリガされます。

```
<from-multiply-request-jsp-to-workflow light-weight="false">
<integer-one>5</integer-one>
<integer-two>7</integer-two>
</from-multiply-request-jsp-to-workflow>
```

メッセージは `from-multiply-request-jsp-to-workflow.dtd` に準拠しています。

## プライベート ワークフローからサーブレットに JMS 経由で送信される XML メッセージ

以下は、要求側のプライベート ワークフローから JMS 経由でサーブレットに送信されるメッセージです。メッセージには、テキスト メッセージと共に乗算の結果が格納されています。

```
<from-workflow-to-multiply-request-jsp>
<integer-product>35</integer-product>
<note>Dear RequestorPartner: Here is the product of 7 and 5
  from ReplierPartner to RequestorPartner.</note>
</from-workflow-to-multiply-request-jsp>
```

メッセージは `from-workflow-to-multiply-request-jsp.dtd` に準拠しています。

## 応答側パブリック ワークフローから応答側プライベート ワークフローに送信される XML イベント

次の XML イベントには、4 つのパラメータ（乗算の入力値 2 つ、要求側の名前、および応答側の名前）を格納した要求入力メッセージが含まれています。

```
<multiply-inputs>
<integer-one>5</integer-one>
<integer-two>7</integer-two>
<requestor-name>PartnerRequestor</requestor-name>
<replier-name>PartnerReplier</replier-name>
</multiply-inputs>
```

メッセージは `multiply-inputs.dtd` に準拠しています。

## 応答側プライベート ワークフローから応答側パブリック ワークフローに送信される XML イベント

次の XML イベントには、プライベート ワークフローの応答出力が含まれています。

```
<multiply-outputs>
<integer-product>35</integer-product>
```

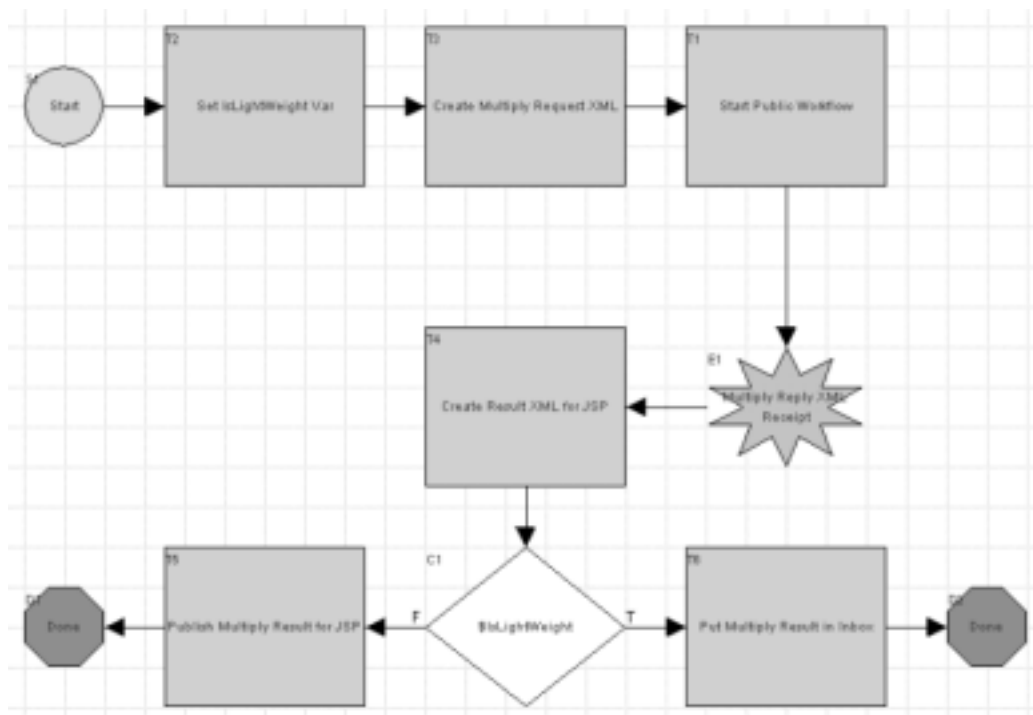
```
<note>Dear RequestorPartner: Here is the product of 7 and 5  
from ReplierPartner to RequestorPartner.</note>  
</multiply-outputs>
```

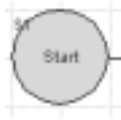
メッセージは multiply-outputs.dtd に準拠しています。

## 要求側プライベート ワークフロー

要求側プライベート ワークフローは、サーブレットから初期要求を受信し、適切なタイプのメッセージを作成してパブリック ワークフローに送信します。応答を受信すると、応答メッセージを処理し、結果をサーブレットに送信します。次の図では、このプロセスをワークフローで示しています。

図 2-5 要求側プライベート ワークフロー





サーブレットから XML イベントを受信すると、ワークフローがトリガされる。「プライベート ワークフローをトリガするためにサーブレットから JMS 経由で送信される XML メッセージ」で説明したように、XML イベントは `<from-multiply-request-jsp-to-workflow>` 形式に準拠している。開始ノードでは、XML から変換文字列を抽出し、`<multiply-request>` ドキュメントを作成してワークフロー変数に格納する。



アクション ノードでは、パブリック ワークフローを開始し、`<multiply-request>` ドキュメントを格納したワークフロー変数を渡す。



イベント ノードでは、`<multiply-reply>` ドキュメントを待つ。`<multiply-reply>` ドキュメントを受信し、変換文字列を抽出する。`<from-workflow-to-multiply-request-jsp>` ドキュメントを作成し、サーブレットに送信する。



完了ノードでワークフローが終了する。

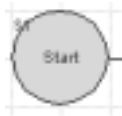
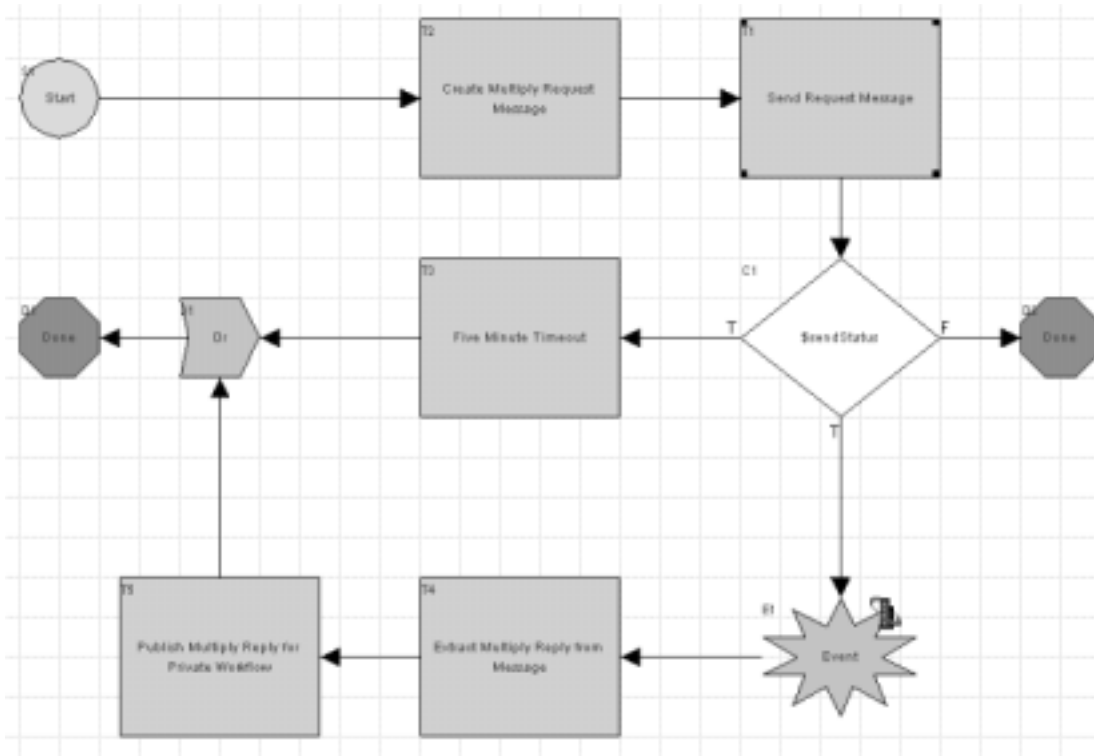
## 要求側パブリック ワークフロー

要求側パブリック ワークフローはプライベート ワークフローによって開始されます。要求側パブリック ワークフローは、プライベート ワークフローからワークフロー変数を受け取ります。次に要求側パブリック ワークフローでは、要求

## 2 Hello Partner サンプル (非推奨)

XML をもとにメッセージを作成し、これを応答側に送信して応答を待ちます。応答メッセージを受信後、応答 XML を抽出し、これをプライベート ワークフローに渡します。次の図に、このプロセスをワークフローで示します。

図 2-6 要求側パブリック ワークフロー



このワークフローはプライベートワークフローから開始される。プライベートワークフローは、このワークフローに <multiply-request> ドキュメントを渡す。開始ノードでは、メッセージ文字列を抽出してワークフロー変数に格納する。





アクション ノードでは、**XOCP** メッセージ内の `<multiply-request>` ドキュメントを応答側ロールに送信する。トレーディング パートナ名には **PartnerReplier** を指定する。



イベント ノードでは、応答側からの **XOCP** 応答メッセージを待つ。メッセージを受信後、`<multiply-reply>` ドキュメントを抽出してワークフロー変数に格納する。



アクション ノードでは、`<multiply-reply>` ドキュメントを **XML** イベントとしてパブリッシュする。

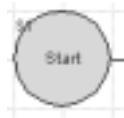
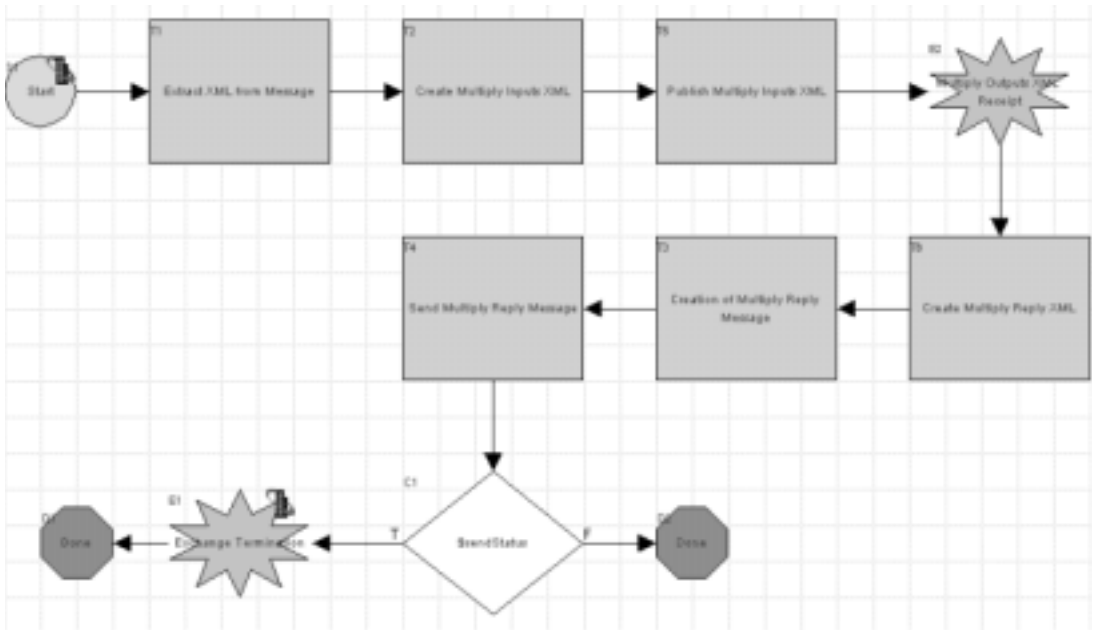


完了ノードでワークフローが終了する。

## 応答側パブリック ワークフロー

応答側パブリック ワークフローは、要求側メッセージの受信と同時に開始されます。要求側メッセージを受信すると、メッセージから要求 **XML** を抽出し、これを格納した **XML** イベントをパブリッシュして応答側プライベート ワークフローを開始します。応答側プライベート ワークフローから **XML** イベントが返信されたら、応答 **XML** を含むメッセージを要求側に返信します。次の図では、このプロセスをワークフローで示しています。

図 2-7 応答側パブリックワークフロー



ワークフローは <multiply-request> ドキュメントを受信すると同時に開始される。開始ノードでは、メッセージから内容を抽出してワークフロー変数に格納する。ワークフロー変数のフォーマットは <multiply-input> で、<multiply-request> ドキュメント内の値と、メッセージ内の送信側と応答側の名前を使用する。



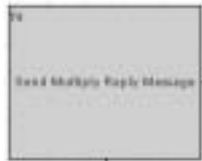
アクションノードでは、<multiply-inputs> ドキュメントを XML イベントとしてパブリッシュし、プライベートワークフローを開始する。



イベント ノードでは、<multiply-outputs>ドキュメントを格納した XML イベントを受信するため応答を待つ。



アクション ノードでは、<multiply-outputs>ドキュメントに基づいて <multiply-reply>ドキュメントを作成する。その結果を XML ワークフロー変数に格納する。



アクション ノードでは、<multiply-reply>ドキュメントを格納したメッセージを XOCF で要求側に送信する。

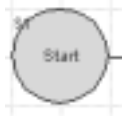
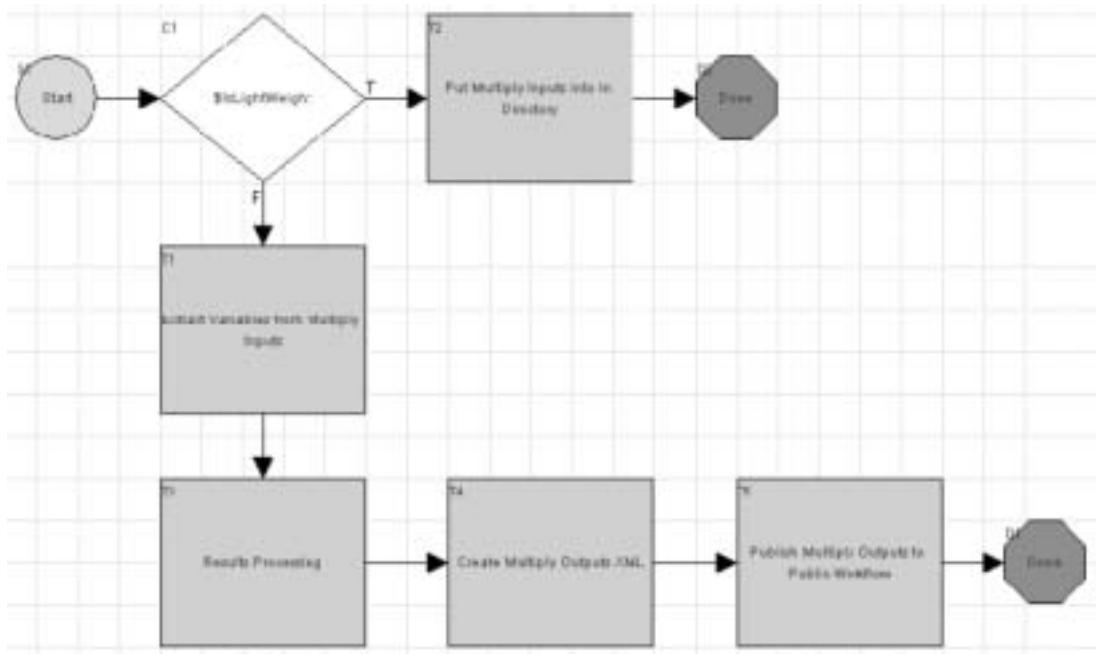


完了ノードでワークフローが終了する。

## 応答側プライベート ワークフロー

応答側プライベート ワークフローは、要求 XML を格納した XML イベントを応答側パブリック ワークフローから受信すると同時に開始されます。要求を受信すると、データを処理して XML ドキュメント内に応答を生成し、XML イベントを使用して応答側パブリック ワークフローに応答 XML を返信します。

図 2-8 応答側プライベート ワークフロー



ワークフローは、<multiply-inputs> 型のドキュメントを格納した XML イベントを受信すると同時に開始される。開始ノードでは、ドキュメントをワークフロー変数に格納する。



このアクション ノードでは、<multiply-inputs> ドキュメントから入力された整数の乗算結果を格納する整数ワークフロー変数を作成する。



このアクション ノードでは、<multiply-outputs>型のドキュメントをワークフロー XML 変数として作成する。ステップ 2 とステップ 3 の整数ワークフロー変数およびノート ワークフロー変数の値はこのドキュメントに格納される。



このアクション ノードでは、<multiply-reply>ドキュメントを XML イベントとしてパブリッシュする。



完了ノードでワークフローが終了する。



---

## 3 Channel Master サンプル（非推奨）

Channel Master サンプルには、大規模トレーディング パートナが WebLogic Integration を使用してそのサプライ チェーンを自動化する方法が示されています。サンプルでは、WebLogic Integration トレーディング パートナ間での通信方法として、XOCP ビジネス プロトコルを使用するポイント ツー ポイント通信とマルチキャスト（ブロードキャスト）通信の両方を示します。

この章では、以下のトピックを取り上げます。

- Channel Master サンプルの概要
- Channel Master サンプルを実行する前に
- Channel Master サンプルの実行
- Channel Master サンプルのワークフロー

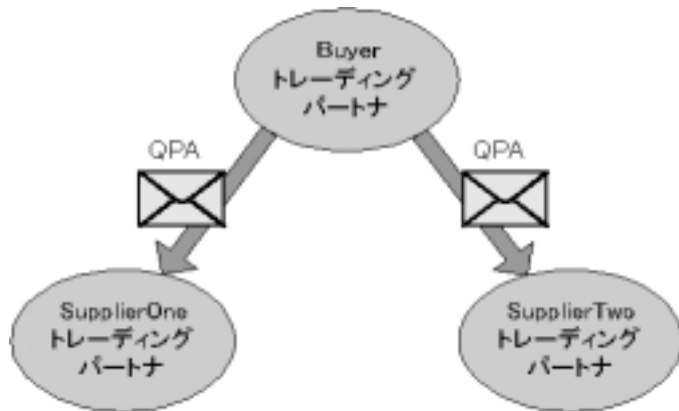
**注意：** Channel Master サンプルは、XOCP プロトコルに基づいていますが、このプロトコルは、このリリースの WebLogic Integration から非推奨になっています。XOCP に代わる機能については、『WebLogic Integration リリース ノート』を参照してください。

## Channel Master サンプルの概要

以下に、このサンプルでのトレーディング パートナ間の通信手順をまとめます。

1. チャネル マスタ バイヤーとなるトレーディング パートナが、特定のアイテムの価格と在庫に関するクエリ（QPA）をブロードキャストします。このサンプルでは、2 社のサプライヤトレーディング パートナがクエリをリスンします。したがって、クエリを受信するサプライヤトレーディング パートナは 2 社ということになります。ブロードキャスト通信は多くのトレーディング

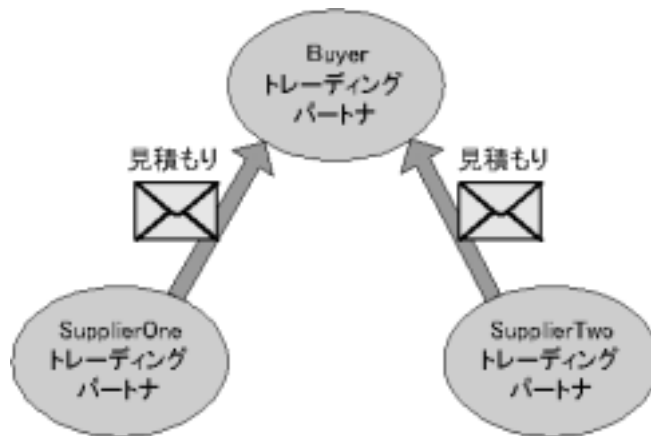
パートナーが受信できますが、このサンプルではリスンしているサプライヤが2社のみであるため、クエリを受信するサプライヤも2社のみということになります。このアクションによって、ブロードキャスト（またはマルチキャスト）通信の例が示されています。次の図では、QPA というエンベロープがXMLメッセージを表しています。



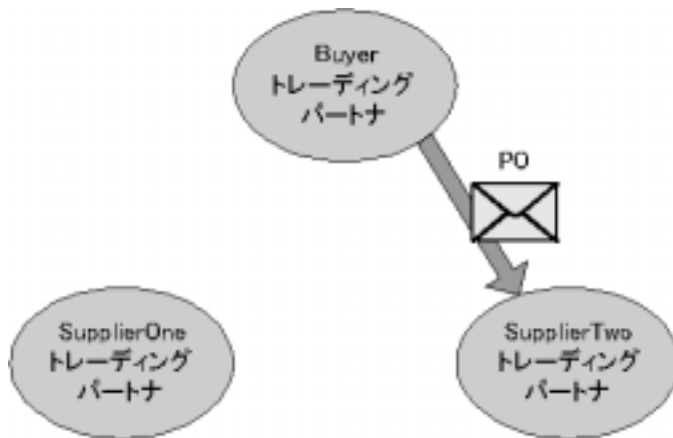
この例では、価格と在庫に関するクエリ（QPA）を格納したXMLメッセージが、トレーディング パートナ間で転送されます。上の図は、これを単純化して表しています。詳細については、3-26 ページの「マルチキャスト（ブロードキャスト）メッセージ」を参照してください。

2. 2社のサプライヤは、バイヤに見積もりを返信します。見積もりには、要求されたアイテムの価格と在庫が記載されています。このステップでの各サプライヤとバイヤとの間の通信はポイント ツー ポイント通信となります。

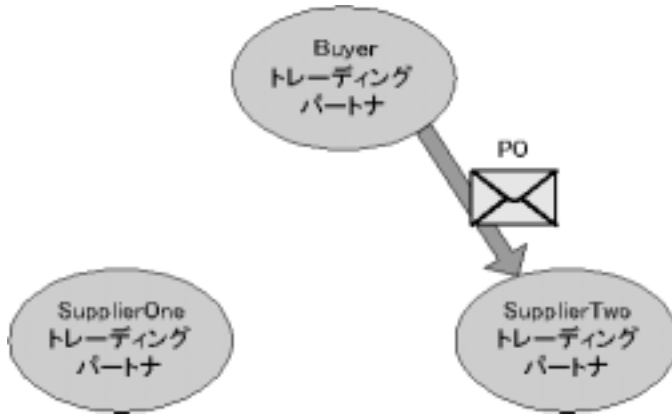




3. バイヤーは、いずれかのサプライヤーを選択して発注書（PO）を送信します。このアクションもポイント ツー ポイント通信です。



4. 選択されたサプライヤは、発注確認メッセージを返信します。



## Channel Master サンプルを実行する前に

Channel Master サンプルを実行する前に、次の手順を実行します。

1. 1-3 ページの「サンプルの実行前の作業」に記載の手順に従います。
2. サンプル WebLogic Server への接続が妨げられないようにブラウザのプロキシ設定を確認します。Web ブラウザ コンフィグレーションの要件の詳細については、『*WebLogic Integration の起動、停止およびカスタマイズ*』の「WebLogic Integration 管理ツールと設計ツール」の「Web ブラウザ コンフィグレーションの要件」を参照してください。

## Channel Master サンプルの実行

Channel Master サンプルを実行するには、次の手順を実行します。

**注意：** RunSamples スクリプトで起動した WebLogic Server のインスタンスが実行中である場合はステップ 4 に進みます。

1. `WLI_HOME` (WebLogic Integration をインストールしたディレクトリ) に移動します。

```
cd WLI_HOME
```

- Windows の例

WebLogic Platform を `c:\bea` ディレクトリにインストールした場合は、`WLI_HOME` は、`c:\bea\weblogic700\integration` というパス名を表します。

- UNIX の例

WebLogic Platform を `/home/me/bea` ディレクトリにインストールした場合は、`WLI_HOME` は、`/home/me/bea/weblogic700/integration` というパス名を表します。

2. WebLogic Integration の上位レベルの環境変数を設定するには、お使いのプラットフォームに合った `setenv` スクリプトを実行します。

- Windows の場合

```
setEnv
```

- UNIX の場合

```
. setenv.sh
```

3. プラットフォームに合わせて適切な手順を実行し、`RunSamples` スクリプトを起動します。

- Windows:

[ スタート | プログラム | BEA WebLogic Platform 7.0 | WebLogic Integration 7.0 | Integration Examples | Start Server and Launch Examples (with dataloader) ] を選択します。

- UNIX:

a) `PATH` 環境変数に、`Netscape` 実行ファイル (`netscape`) が格納されたディレクトリが含まれていることを確認します。

b) `RunSamples` スクリプトを実行します。

```
cd $SAMPLES_HOME/integration/config/samples  
RunSamples
```

**警告：** UNIX システムの場合、`netscape` 実行ファイルが入ったディレクトリが `PATH` 環境変数に含まれている必要があります。含まれていない場合は、`RunSamples` スクリプトの実行時にサン

ブル起動ページが表示されません。サンプル起動ページは、現在 RunSamples スクリプトが実行されているマシンで Netscape ブラウザを起動して、次の URL を入力すると起動されます。  
`http://localhost:7001/index.html`

4. RunSamples スクリプトのコンフィグレーション セクションが実行済みであることが検知されると、次のプロンプトが表示されます。

```
The WebLogic Integration repository has already been created
and populated, possibly from a previous run of this
RunSamples script. Do you want to destroy all the current
data in the repository and create and populate the
WebLogic Integration repository, again?
Y for Yes, N for No
```

この質問に **N** と入力すると、リポジトリの作成および格納を行う手順が省略され、WebLogic Server のサンプル インスタンスを起動する手順のみが実行されます。

この質問に **Y** と入力すると、リポジトリの作成および格納が改めて行われ、その後で WebLogic Server のサンプル インスタンスを起動する手順が実行されます。Y と入力した場合、その時点でリポジトリに格納されている全データが破棄され、リポジトリにサンプル データが再ロードされます。現在のサンプル データが変更または削除され、新規または未変更のサンプル データをリポジトリに格納する場合にのみ、Y を入力してください。

これで、RunSamples スクリプトは WebLogic Server のインスタンスを (バックグラウンド プロセスとして) 開始し、サンプル起動ページが表示されます。

Channel Master サンプルは、その他の WebLogic Integration B2B サンプルと違い、サンプル起動ページからは起動されません。Channel Master サンプルを起動するには、WebLogic Server のサンプル インスタンスが実行されている必要があります。以下の手順を完了するまでは、WebLogic Server のインスタンスとサンプル起動ページが常に実行されている状態にしておいてください。

5. WebLogic Integration Worklist を起動するには、プラットフォームに合わせて適切な手順を実行します。

- Windows:

[ スタート | プログラム | BEA WebLogic Platform 7.0 | Worklist ] を選択します。

- UNIX:

a) \$WLI\_HOME (WebLogic Integration をインストールしたディレクトリ) に移動します。

```
cd $WLI_HOME/bin
```

b) worklist スクリプトを起動します。

```
worklist
```

6. 次の情報を使用して、WebLogic Integration Worklist にログインします。

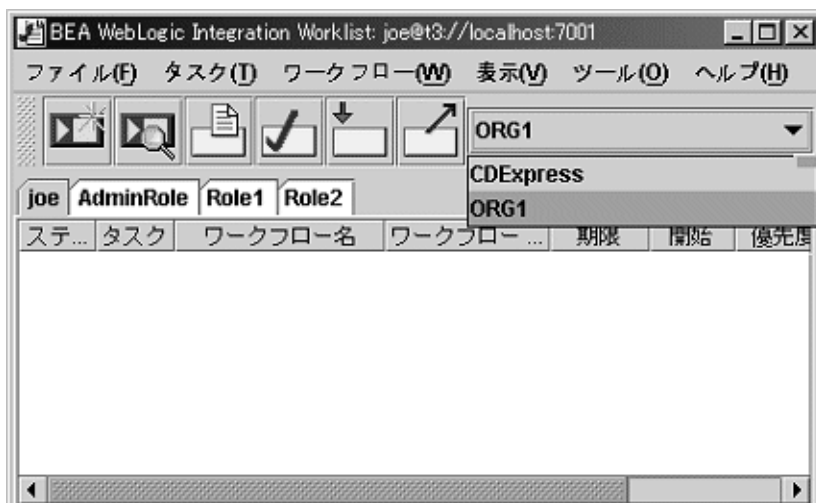
- ログイン : joe
- パスワード : password
- URL : t3://localhost:7001

WebLogic Integration Worklist のメイン ウィンドウが表示されます。

7. WebLogic Integration Worklist を使用してサンプルを実行します。

**注意：** Worklist クライアント アプリケーションは、このリリースの Weblogic Integration から非推奨になっています。Worklist に代わる機能の詳細については、『WebLogic Integration リリース ノート』を参照してください。

a. Worklist ウィンドウの右上のドロップダウン リストを開きます。ORG1 を選択します。



b. [joe] タブを選択します。



- c. Worklist のメニュー バーから、[ワークフロー | ワークフローを開始] を選択します。[ワークフローを開始] ダイアログ ボックスが表示されます。
- d. [BuyerPrivate] ワークフローを選択します。[OK] をクリックします。



- e. サンプルが正常に実行されたかどうかを確認するには、`myserver.log` ファイルの中で次のメッセージを探します。

```
CHANNEL MASTER SAMPLE RAN SUCCESSFULLY!!!
```

`myserver.log` ファイルは、Windows の場合、  
%SAMPLES\_HOME%\integration\config\samples\logs ディレクトリに、  
UNIX の場合、\$SAMPLES\_HOME/integration/config/samples/logs  
ディレクトリに格納されています。

8. **WebLogic Integration Worklist** を閉じます。Worklist のメニューバーから、[ファイル | 終了] を選択します。
9. 別の B2B サンプルを実行したり、3-17 ページの「SupplierOnePrivate ワークフローの表示」で説明する手順を完了する必要がある場合は、サンプル起動ページを開いた状態で、**WebLogic Server** のインスタンスは実行したままにしておきます。

別の B2B サンプルを実行したり、3-17 ページの「SupplierOnePrivate ワークフローの表示」で説明する手順を完了する必要がある場合は、ブラウザと

WebLogic Server を終了します。WebLogic Server のインスタンスを終了する手順は、プラットフォームによって異なります。

- Windows:

```
cd %SAMPLES_HOME%\integration\config\samples
stopWebLogic
```

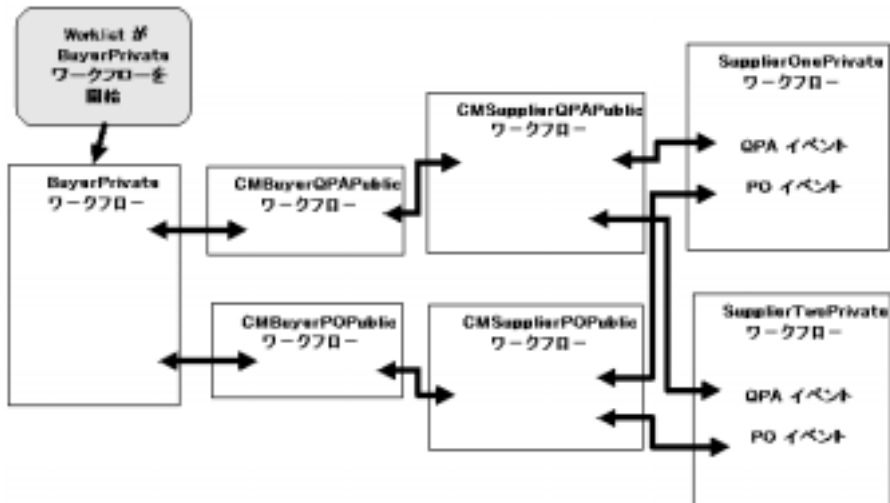
- UNIX:

```
cd $SAMPLES_HOME/integration/config/samples
stopWebLogic
```

## Channel Master サンプルのワークフロー

次の図に、別々の Channel Master ワークフロー間での上位レベルの会話を示します。

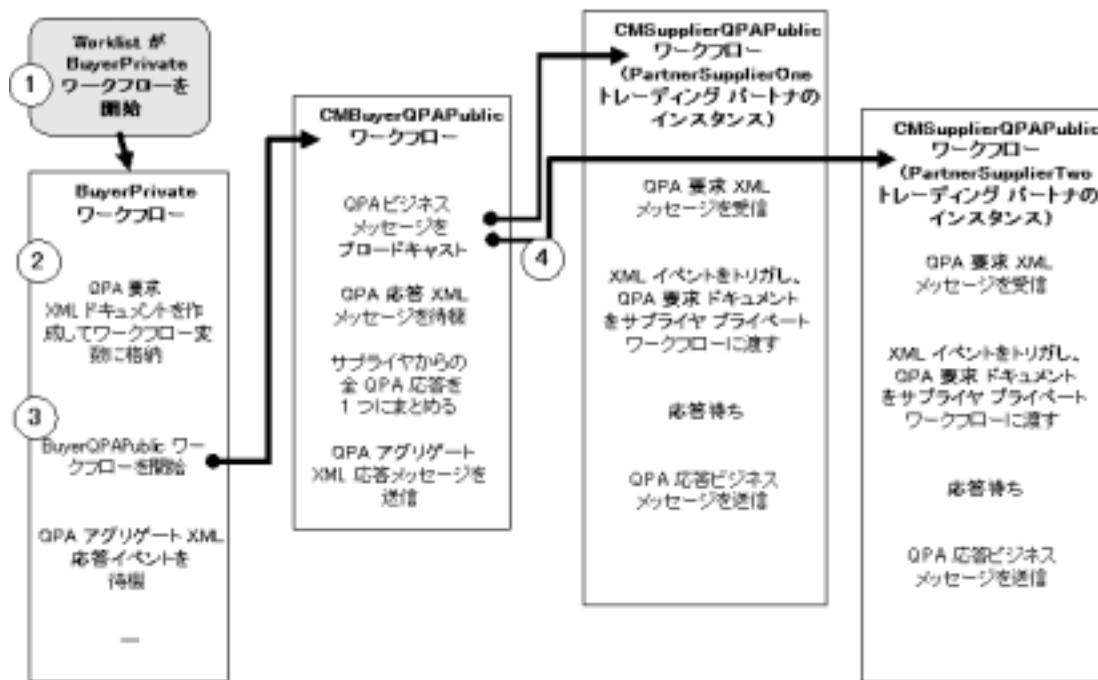
図 3-1 Channel Master ワークフロー間の会話



次の図は、Channel Master サンプルの実行フローをトレースした 1 番目の図です。



図 3-2 ワークフローのトレース : ステップ 1 ~ 4



以下では、各ステップの詳細を説明します。ステップの番号は、図 3-2 の番号に対応しています。

1. ユーザが WebLogic Integration Worklist ユーティリティを呼び出します。これにより、BuyerPrivate ワークフローが開始されます。
2. BuyerPrivate ワークフローは、QPA 要求 XML ドキュメントを作成し、これをワークフロー変数に格納します。
3. BuyerPrivate ワークフローは、CMBuyerQPAPublic ワークフローを開始します。
4. CMBuyerQPAPublic ワークフローは、全サプライヤへの QPA 要求ドキュメントに基づいて、QPA ビジネスメッセージをブロードキャストします。このブロードキャストによって、CMSupplierQPAPublic ワークフローの 2 つのインスタンスが開始されます。1 つは PartnerSupplierOne トレーディング パートナ、もう 1 つは PartnerSupplierTwo トレーディング パートナです。

2つのインスタンスが開始されるのは、各トレーディング パートナが ChannelMasterHub トレーディング パートナとのコラボレーション アグリーメントを持っているためです。これは、ワークフローが CMQPAConversation の会話定義名と 1.1 の会話定義とともにビジネス メッセージを受信した場合に、CMSupplier のロール名を使用する必要があることを指定するためのものです。

次のリストに、ChannelMasterHub と Partner とのコラボレーション アグリーメントを示します。ChannelMasterHub トレーディング パートナが必要となる理由の詳細については、3-26 ページの「マルチキャスト (ブロードキャスト) メッセージ」を参照してください。

#### コード リスト 3-1 インポート リポジトリ データ ファイルのコラボレーション アグリーメント セクション

---

```
<collaboration-agreement
  name="CMQPAConversation|1.1|PartnerSupplierOne|ChannelMasterHub"
  global-identifier="CMQPAConversation|1.1|PartnerSupplierOne|ChannelMasterHub"
  version="1.1"
  status="ENABLED"
  conversation-definition-name="CMQPAConversation"
  conversation-definition-version="1.1">
  <party
    trading-partner-name="PartnerSupplierOne"
    party-identifier-name="PartnerSupplierOnePartyId"
    delivery-channel-name="PartnerSupplierOneDeliveryChannel"
    role-name="CMSupplier"/>
  <party
    trading-partner-name="ChannelMasterHub"
    party-identifier-name="ChannelMasterHubPartyId"
    delivery-channel-name="ChannelMasterHubDeliveryChannel"
    role-name="CMBuyer"/>
</collaboration-agreement>
```

---

このリストは、Channel Master サンプルの BulkLoaderData.xml ファイルからの抜粋です。このファイルは、サンプルに必要なデータを WebLogic Integration リポジトリにインポートするために使用します。Windows システムの場合は %SAMPLES\_HOME%\integration\samples\ChannelMaster\lib ディレクトリに、UNIX システムの場合は \$SAMPLES\_HOME/integration/ChannelMaster/lib にあります。リポジト

リ データは、WebLogic Integration B2B Console から入力することもできます。

これと似ているのが、リポジトリ データ ファイル内の PartnerSupplierTwo トレーディング パートナのコラボレーション アグリーメントです。これは、ワークフローが CMQPAConversation の会話定義名と 1.1 の会話定義とともにビジネス メッセージを受信した場合に、CMSupplier のロール名を使用する必要があります。これを指定するためのものです。

CMQPAConversation の会話定義では、CMSupplier ロール用に CMSupplierQPAPublic ワークフローのインスタンスが開始されている必要があることが指定されています。次のリストを参照してください。

### コード リスト 3-2 インポート リポジトリ データ ファイル内の会話定義

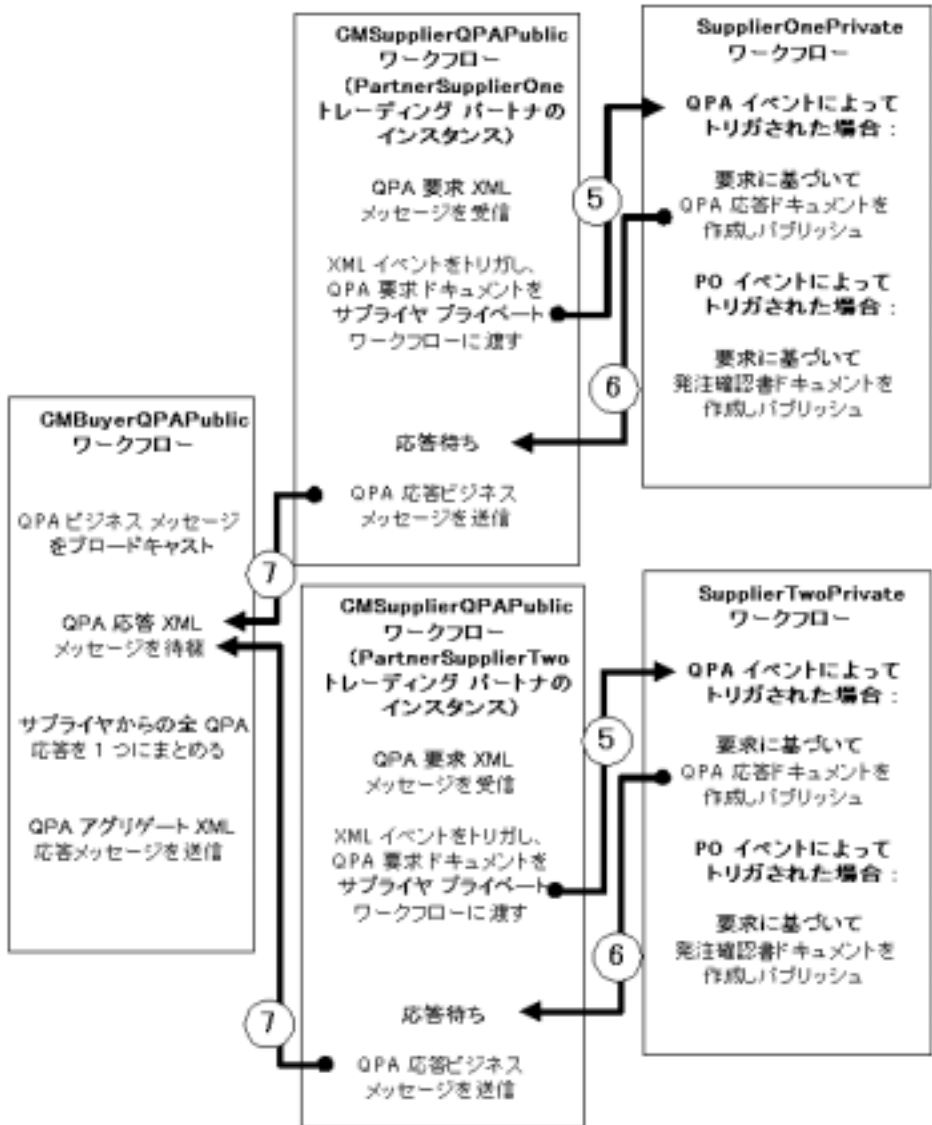
---

```
<conversation-definition
name="CMQPAConversation"
version="1.1"
business-protocol-name="XOCP"
protocol-version="1.1">
  <role
name="CMBuyer"
wlpi-template="CMBuyerQPAPublic">
  <process-implementation wlpi-org="ORG1" />
  </role>
  <role
name="CMSupplier"
wlpi-template="CMSupplierQPAPublic">
  <process-implementation wlpi-org="ORG1" />
  </role>
</conversation-definition>
```

---

次の図に、ステップ 5 ～ 7 の実行フローを示します。各ステップの説明は、図の後にあります。

図 3-3 ワークフローのトレース：ステップ 5 ~ 7



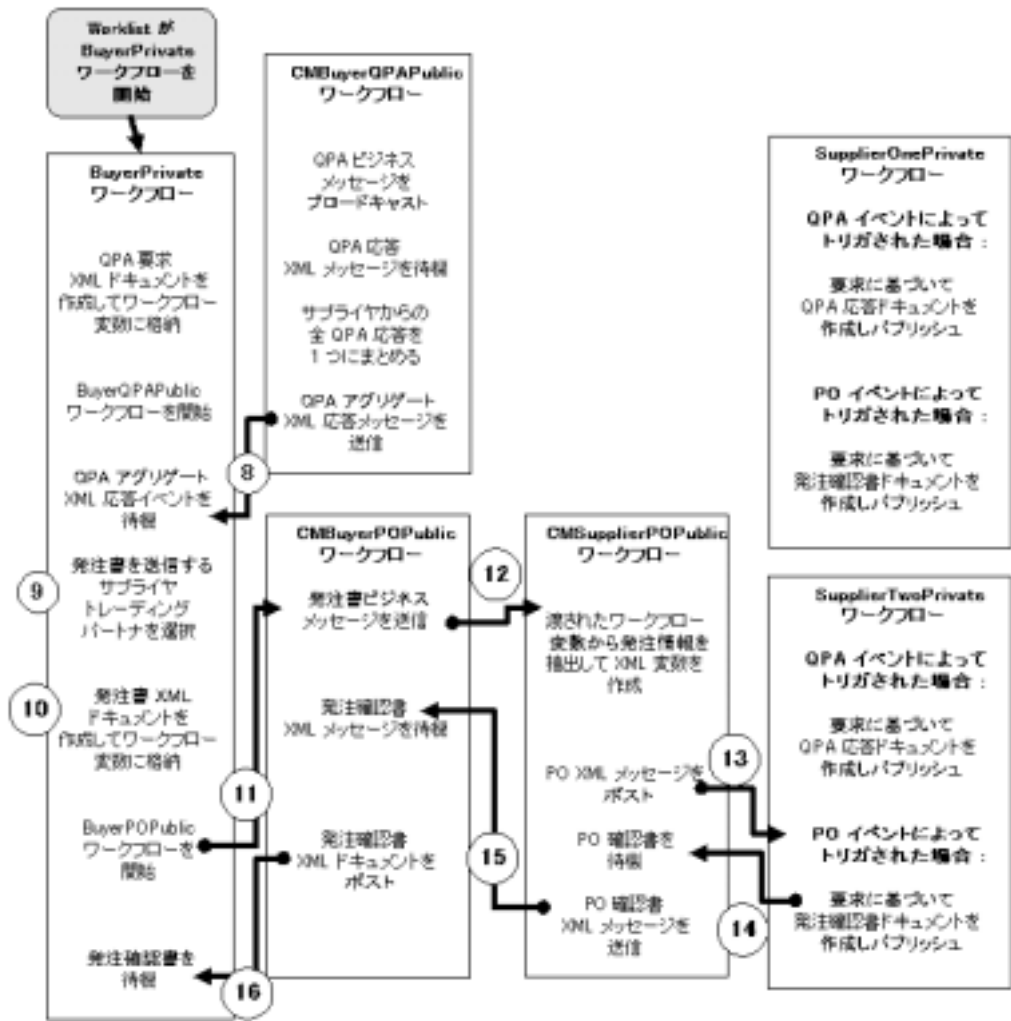
5. **CMBuyerQAPublic** ワークフローの各インスタンスは、内部 XML イベントをポストします。**PartnerSupplierOne** の **CMSupplierQAPublic** ワークフローのインスタンスが内部 XML イベントをポストすると、**SupplierOnePrivate** ワークフローの開始がトリガされます。**PartnerSupplierTwo** の **CMSupplierQAPublic** ワークフローのインスタンスが内部 XML イベントをポストすると、**SupplierTwoPrivate** ワークフローの開始がトリガされます。

**PartnerSupplierOne** トレーディング パートナがトリガするのは、**SupplierOnePrivate** ワークフローのみです。**SupplierTwoPrivate** ワークフローはトリガされません。これは、**SupplierOnePrivate** ワークフローの開始ノードで、**PartnerSupplierOne** からのイベントによってのみトリガされるように指定されているためです。**SupplierOnePrivate** ワークフローの開始方法、および **SupplierOnePrivate** ワークフローを開始するための条件を表示する方法については、3-17 ページの「**SupplierOnePrivate** ワークフローの表示」を参照してください。

6. **SupplierOnePrivate** および **SupplierTwo Private** ワークフローは、QPA 要求から変数情報を抽出して QPA XML 応答を作成し、QPA XML 応答メッセージを内部 XML イベントとしてポストします。
7. **CMSupplierQAPublic** ワークフローの各インスタンスは、QPA XML 応答メッセージを待ちます。応答メッセージを受信すると、QPA XML 応答メッセージからビジネス メッセージを作成して送信します。

次の図に、ステップ 8 ~ 16 の実行フローを示します。各ステップの説明は、図の後にあります。

図 3-4 ワークフローのトレース : ステップ 8 ~ 16



8. CMBuyerQPAPublic ワークフローは、応答を 1 つにまとめた QPA アグリゲート XML 応答メッセージを送信します。メッセージには、サプライヤからの価格と在庫の情報が含まれています。

9. QPA XML 応答を待っていた **BuyerPrivate** ワークフローは、もう一度実行を開始し、QPA XML 応答メッセージに基づいてサプライヤを選択します。
10. **BuyerPrivate** ワークフローは、発注書 XML ドキュメントを作成し、これを POXML XML ワークフロー変数に格納します。
11. **BuyerPrivate** ワークフローは、**CMBuyerPOPublic** ワークフローを開始します。
12. **CMBuyerPOPublic** ワークフローは、発注書ビジネス メッセージを作成して送信します。送信後は、発注確認書が送信されてくるのを待ちます。
13. **CMSupplierPOPublic** ワークフローは、渡された **POMessage** ワークフロー変数から発注情報を抽出し、**POXML XML** ワークフロー変数を構築して XML イベントをポストした後、発注確認書が送信されてくるのを待ちます。**PartnerSupplierTwo** によって一番条件の良い価格と在庫が提示されたため、**PartnerSupplierTwo** がサプライヤとして選択されます。
14. **SupplierTwo Private** ワークフローは、PO 要求から変数情報を抽出して PO XML 応答を作成し、PO XML 応答メッセージを内部 XML イベントとしてポストします。
15. **CMSupplierPOPublic** ワークフローは発注確認書を待ちます。確認書を受信した後、**CMBuyerPOPublic** に XML 発注確認メッセージを返送します。
16. **CMBuyerPOPublic** ワークフローは発注確認書を待ちます。確認書を受信後、受け取った XML ドキュメントから情報を抽出し、発注 XML 確認ドキュメントををポストします。
17. **BuyerPrivate** はこれで終了です（このステップは、図 3-4 には示されていない）。

## SupplierOnePrivate ワークフローの表示

**SupplierOnePrivate** ワークフローは、**PartnerSupplierOne** トレーディング パートナからの XML イベントのみを受け付けます。**SupplierOnePrivate** ワークフローを表示し、受け付ける XML イベントがどう制限されているかを見るには、次の手順を実行します。

**注意：** `RunSamples` スクリプトで起動した **WebLogic Server** のインスタンスが実行中である場合はステップ 4 に進みます。

1. `WLI_HOME` (WebLogic Integration をインストールしたディレクトリ) に移動します。

```
cd WLI_HOME
```

- Windows の例

WebLogic Platform を `c:\bea` ディレクトリにインストールした場合は、`WLI_HOME` は、`c:\bea\weblogic700\integration` というパス名を表します。

- UNIX の例

WebLogic Platform を `/home/me/bea` ディレクトリにインストールした場合は、`WLI_HOME` は、`/home/me/bea/weblogic700/integration` というパス名を表します。

2. WebLogic Integration の上位レベルの環境変数を設定するには、お使いのプラットフォームに合った `setenv` スクリプトを実行します。

- Windows の場合

```
setEnv
```

- UNIX の場合

```
. setenv.sh
```

3. プラットフォームに合わせて適切な手順を実行し、`RunSamples` スクリプトを起動します。

- Windows:

[スタート | プログラム | BEA WebLogic Platform 7.0 | WebLogic Integration 7.0 | Integration Examples | Start Server and Launch Examples (with dataloader)] を選択します。

- UNIX:

a) `PATH` 環境変数に、`Netscape` 実行ファイル (`netscape`) が格納されたディレクトリが含まれていることを確認します。

b) `RunSamples` スクリプトを実行します。

```
cd $SAMPLES_HOME/integration/config/samples  
RunSamples
```

**警告:** UNIX システムの場合、`netscape` 実行ファイルが入ったディレクトリが `PATH` 環境変数に含まれている必要があります。含まれていない場合は、`RunSamples` スクリプトの実行時にサンプル起動ページが表示されませ



ん。サンプル起動ページは、現在 RunSamples スクリプトが実行されているマシンで Netscape ブラウザを起動して、次の URL を入力すると起動されます。

```
http://localhost:7001/index.html
```

4. RunSamples スクリプトのコンフィグレーション セクションが実行済みであることが検知されると、次のプロンプトが表示されます。

```
The WebLogic Integration repository has already been created
and populated, possibly from a previous run of this
RunSamples script. Do you want to destroy all the current
data in the repository and create and populate the
WebLogic Integration repository, again?
Y for Yes, N for No
```

If you answer この質問に N と入力すると、リポジトリの作成および格納を行う手順が省略され、WebLogic Server のサンプル インスタンスを起動する手順のみが実行されます。

この質問に Y と入力すると、リポジトリの作成および格納が改めて行われ、その後で WebLogic Server のサンプル インスタンスを起動する手順が実行されます。Y を入力した場合、RunSamples スクリプトは、現在リポジトリにある全データを破棄し、リポジトリに改変されていないバージョンのサンプルデータをロードします。現在のサンプルデータが変更または削除され、新規または未変更のサンプルデータをリポジトリに格納する場合にのみ、Y を入力してください。

これで、RunSamples スクリプトは WebLogic Server のインスタンスをバックグラウンドプロセスとして開始し、サンプル起動ページが表示されます。

5. WebLogic Integration Studio を起動するには、プラットフォームに合わせて適切な手順を実行します。

- Windows:

```
[ スタート | プログラム | BEA WebLogic E-Business Platform |
WebLogic Integration 7.0 | Studio] を選択します。
```

- UNIX:

```
cd $WLI_HOME/bin
studio
```

6. 次の情報を使用して WebLogic Integration Studio にログインします。

- ログイン : joe
- パスワード : password

### 3 Channel Master サンプル (非推奨)

---

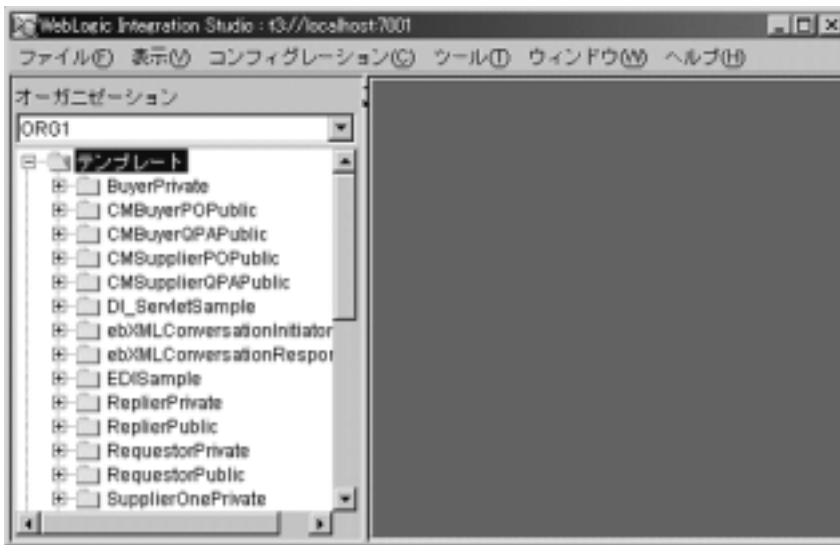
- URL : t3//localhost:7001

WebLogic Integration Studio のメイン ウィンドウが表示されます。

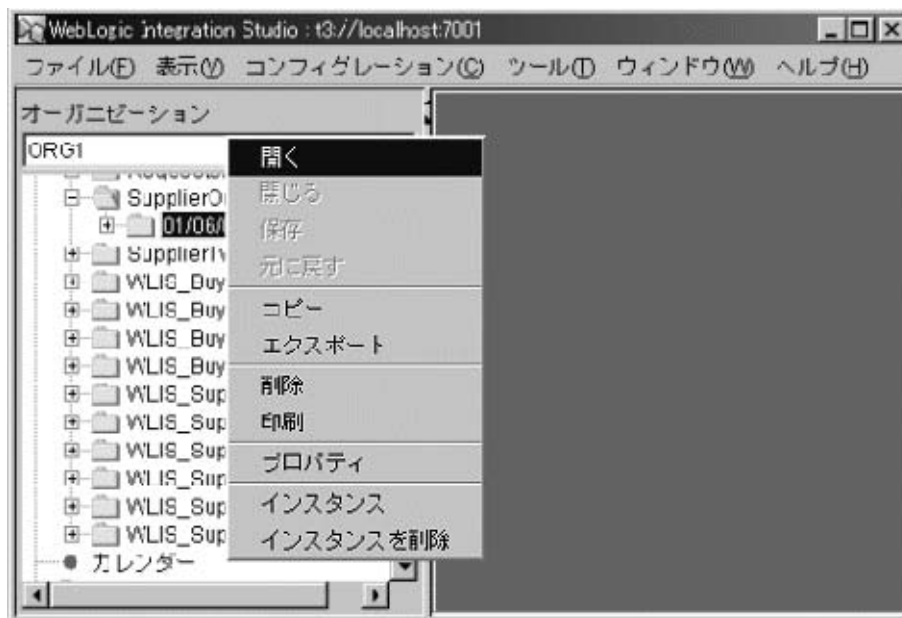
7. 左ペインの [オーガニゼーション] の下にあるドロップダウン リストから、[ORG1] を選択します。



8. 左ペインの [テンプレート] フォルダを展開します。サンプルの全テンプレートのリストが表示されます。

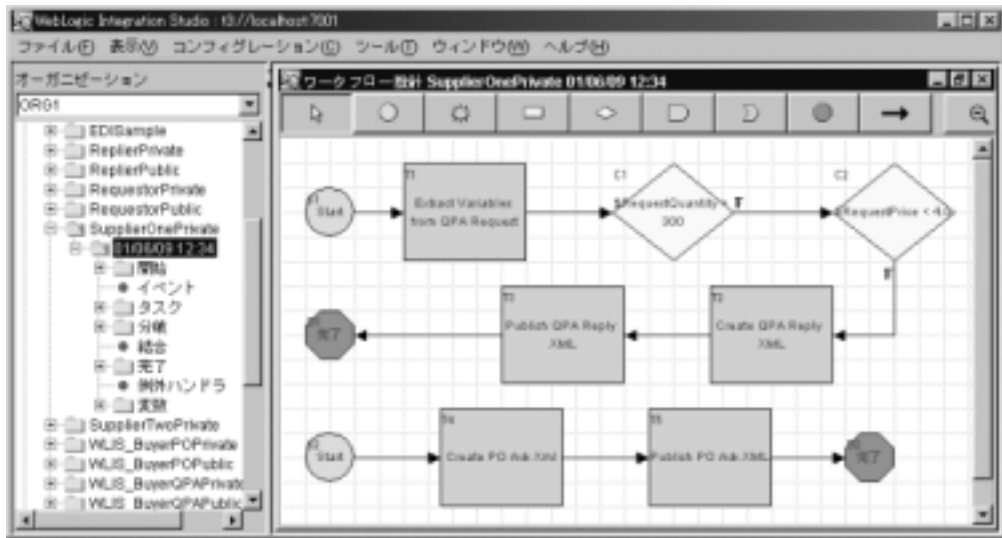


9. 左ペインの [SupplierOnePrivate] フォルダを開きます。
10. SupplierOnePrivate ワークフローのインスタンスを開いて表示するには、次の手順を実行します。
  - a. SupplierOnePrivate の下にリストされている日付のフォルダを右クリックします。



b. [開く] を選択します。

ワークフローが図示されます。**SupplierOnePrivate** ワークフローを構成する開始ノード、タスクノード、分岐ノード、イベントノードが表示されています。

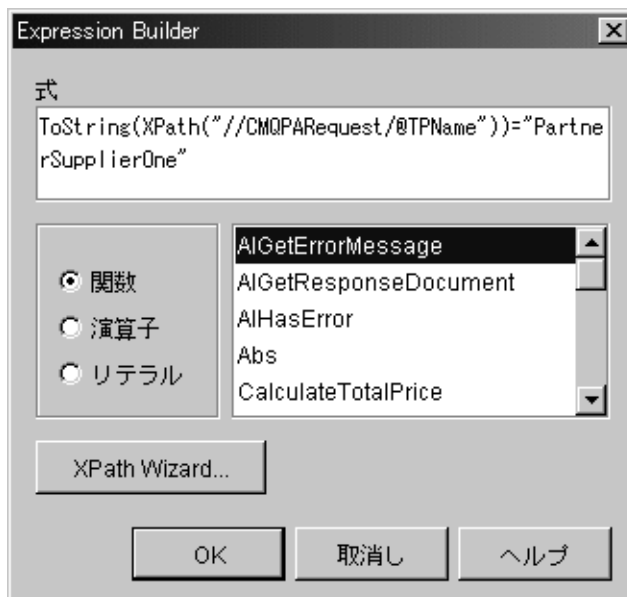


11. 右ペインの左上にある開始ノードをダブルクリックします。  
[開始のプロパティ] ダイアログ ボックスが表示されます。



このワークフローを開始するには、上の図に示した [ 開始のプロパティ ] ダイアログ ボックスの [ 条件 ] フィールドに定義されている式が **true** になっている必要があります。

12. [ 条件 ] フィールドの右にある [ A+B ] オプションをダブルクリックします。 [ Expression Builder ] ダイアログ ボックスが表示されます。



指定した式によって、ワークフローが受信した XML メッセージが評価され、**CMQPARquest TPName** ノードの値が **PartnerSupplierOne** であるかどうか識別されます。値が **PartnerSupplierOne** である場合は、式が **true** となってワークフローが開始されます。

13. B2B サンプルの実行を続けたり、3-17 ページの「SupplierOnePrivate ワークフローの表示」で説明する手順を完了する必要がある場合は、サンプル起動ページを開いた状態で、**WebLogic Server** のサンプル インスタンスは実行したままにしておきます。

B2B サンプルの実行を続ける必要がない場合は、**WebLogic Server** のサンプル インスタンスをシャットダウンし、サンプル起動ページを実行しているブラウザを閉じます。**WebLogic Server** をシャットダウンするには、プラットフォームに合わせて適切な手順を実行します。

- Windows:

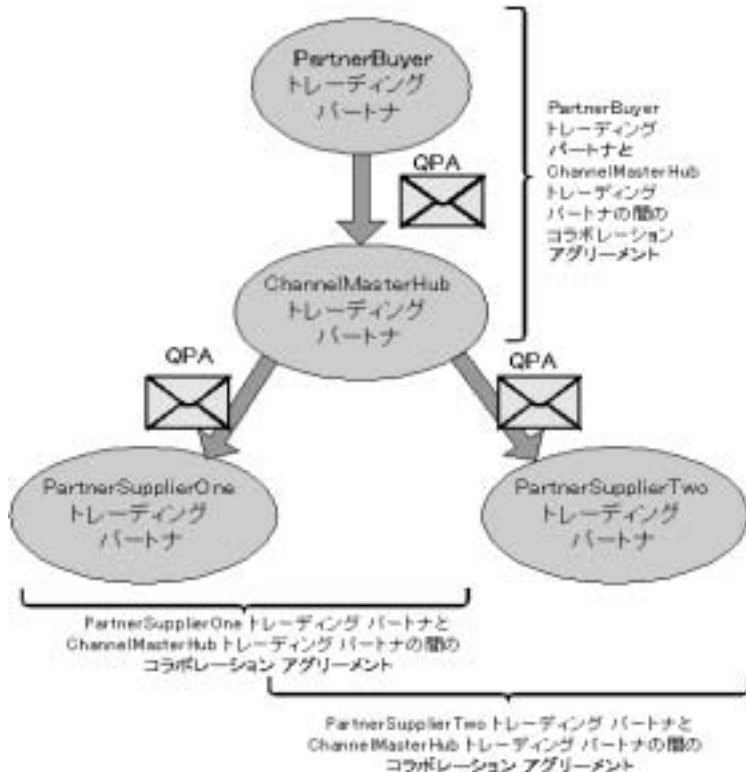
```
cd %SAMPLES_HOME%\integration\config\samples
stopWebLogic
```

- UNIX:

```
cd $SAMPLES_HOME/integration/config/samples
stopWebLogic
```

## マルチキャスト (ブロードキャスト) メッセージ

メッセージのマルチキャストまたはブロードキャストは、XOCP ビジネス プロトコルでのみサポートされています。また、メッセージがルーティング プロキシ 配信チャネル (ハブ) 経由でルーティングされた場合のみサポートされます。直接のポイント ツー ポイント 通信ではサポートされていません。したがって、上記のサンプルでいえば、PartnerBuyer トレーディング パートナから PartnerSupplierOne および PartnerSupplierTwo トレーディング パートナに直接メッセージが送信されるわけではありません。メッセージは、ルーティング プロキシ 配信チャネルと ChannelMasterHub トレーディング パートナを経由して送信されます。次の図は、マルチキャスト メッセージがトレーディング パートナ間でルーティングされる様子を示しています。





**警告：** このサンプルでは、単純化するためにすべてのトレーディング パートナで同じ **WebLogic Server** インスタンスを使用しています。プロダクション環境では、バイヤおよびサプライヤの各トレーディング パートナは、別々の **WebLogic Integration** サーバ インスタンスを実行します。



---

## 4 RosettaNet 2.0 Security サンプル

RosettaNet 2.0 Security サンプルは、WebLogic Integration を使用して RosettaNet 2.0 PIP 3A2 および PIP 0A1 をワークフローで実装する方法を示します。特に、RosettaNet 2.0 PIP 3A2 標準に準拠するビジネス メッセージを交換する 2 社のトレーディング パートナを示しています。

この章のトピックは以下のとおりです。

- RosettaNet 2.0 Security サンプルの概要
- RosettaNet 2.0 Security サンプルの概要
- RosettaNet 2.0 Security サンプルを実行する前に
- RosettaNet 2.0 Security サンプルの実行
- RosettaNet 2.0 Security サンプルにおけるワークフロー

### RosettaNet 2.0 Security サンプルの概要

RosettaNet は、オープンな電子商取引プロセス標準の作成、実装および推進を行う非営利の連合体です。RosettaNet Partner Interface Process (PIP) は、トレーディング パートナ間のビジネス プロセスを定義するものです。PIP 3A2 は、トレーディング パートナが製品の価格と在庫に関する情報の要求と提供を行うために使用できる、自動化されたプロセスです。

RosettaNet 2.0 Security サンプルでは、PIP 3A2 の実装を例示しています。また、障害通知送信メカニズムを提供する PIP 0A1 も実装します。最後に、このサンプルは RosettaNet 2.0 をサポートするために必要な WebLogic Integration のセキュリティ機能である、双方向 SSL 認証、デジタル署名、データ暗号化、および否認防止性を例示します。

WebLogic Integration での RosettaNet の実装の詳細については、『*B2B Integration RosettaNet の実装*』を参照してください。WebLogic Integration の B2B Integration 機能でのセキュリティの使用の詳細については、『*B2B Integration セキュリティの実装*』を参照してください。

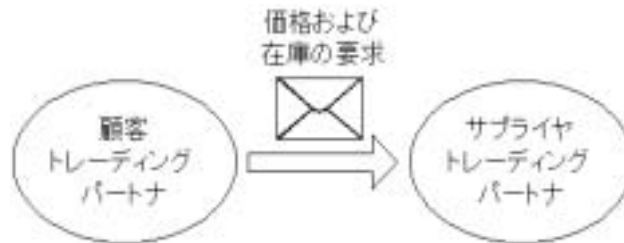
RosettaNet は、このサンプルで使用されている RosettaNet 2.0 PIP の理解に役立つ数点のマニュアルを公開しています。次のマニュアルおよび zip ファイルは、RosettaNet の Web サイト (<http://www.rosettnet.org>) の「Standards」セクションからダウンロード可能です。

- 「Understanding a PIP Blueprint」 – PIP Blueprint の読み方を説明します。「Standards」セクションの「Supporting Documentation」の項目からダウンロードできます。
- PIP 3A2 - Request Quote zip ファイル – PIP 3A2 の仕様情報および DTD が含まれています。[Standard | PIPs | Cluster 3: Order Management | Segment 3A: Quote and Order Entry | PIP 3A2: Request Price and Availability | Version R1.3] を選択するとダウンロードできます。
- PIP 0A1 - Notification of Failure zip ファイル – PIP 0A1 の仕様情報および DTD が含まれています。[Standard | PIPs | Cluster 0: RosettaNet Support | Segment 0A: Administrative | PIP 0A1: Notification of Failure | Version R1.3] を選択するとダウンロードできます。

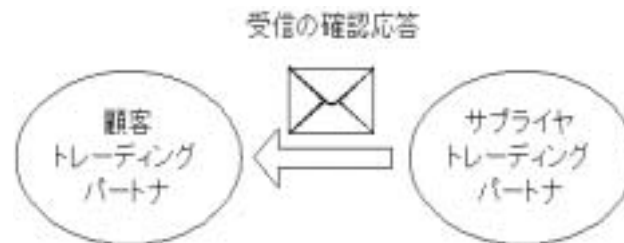
## RosettaNet 2.0 Security サンプルの概要

以下のタスク順序では、このサンプルの PIP 3A2 トレーディング パートナ間通信の高度な概要を示します。

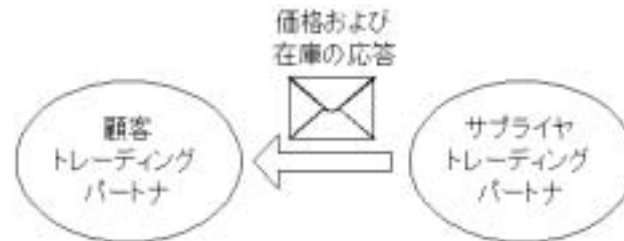
1. 顧客トレーディング パートナは、サプライヤトレーディング パートナに価格および在庫の要求を送信します。このような要求を送信するのはたとえば、サプライヤがある個数のメモリ チップをある特定の価格で提供できるかどうかを調べようとしているコンピュータ メーカー（顧客トレーディング パートナ）などです。次の図は、送信されている要求を示します。図中のエンベロープは、価格と在庫の要求を含む XML メッセージを表します。



2. サプライヤは、価格および在庫の要求を受信したという確認応答を顧客に送信します。



3. サプライヤは、要求された商品を販売するにあたっての個数と価格を含む応答を顧客に送信します。



4. 顧客は、価格および在庫に関する返信を受信したことを示す受領確認書をサプライヤに送信します。



# RosettaNet 2.0 Security サンプルを実行する前に

RosettaNet 2.0 Security サンプルを実行する前に、次の手順を実行します。

- 1-3 ページの「サンプルの実行前の作業」に記載の手順に従います。
2. サンプル WebLogic Server への接続が妨げられないようにブラウザのプロキシ設定を確認します。Web ブラウザ コンフィグレーションの要件の詳細については、『*WebLogic Integration の起動、停止およびカスタマイズ*』の「WebLogic Integration 管理ツールと設計ツール」の「Web ブラウザ コンフィグレーションの要件」を参照してください。

## RosettaNet 2.0 Security サンプルの実行

このサンプルについては、いくつかの制限があります。それらの詳細および回避策については、『*WebLogic Integration リリース ノート*』、「確認済みの制限」にある 2 つの変更要求、CR063709 と CR075768 を参照してください。

WebLogic Integration で提供している他の B2B サンプルとは異なり、RosettaNet 2.0 Security サンプルはサンプルドメインでは実行されず、またサンプル起動ページからも起動されません。

RosettaNet 2.0 Security サンプルを実行するには、次の手順を実行します。

1. 次の例に示すように、適切なコマンドを入力して WebLogic Integration のホーム ディレクトリ (WebLogic Integration をインストールしたディレクトリ) に移動します。

- Windows:

```
cd c:\bea\weblogic700\integration
```

- UNIX:

```
cd /home/me/bea/weblogic700/integration
```

2. WebLogic Integration の上位レベルの環境変数を設定するには、お使いのプラットフォームに合った `setenv` スクリプトを実行します。

- Windows:

```
setEnv
```

- UNIX:

```
./setenv.sh
```

3. プラットフォームに対応したコマンドを入力し、RosettaNet 2.0 Security サンプルの `bin` ディレクトリに移動します。

- Windows:

```
cd %SAMPLES_HOME%\integration\samples\RN2Security\bin
```

- UNIX:

```
cd $SAMPLES_HOME/integration/samples/RN2Security/bin
```

4. `pointbase` オプションを指定して `RunRN2Security` スクリプトを実行します。

```
RunRN2Security pointbase
```

WebLogic Server のインスタンスが両方とも起動を終了するまで待機し、次の手順に進みます。RunRN2Security スクリプトは、バックグラウンド プロセスとして、WebLogic Server の 2 つのインスタンスを開始します。サーバの起動が完了すると、WebLogic Server のコンソール ウィンドウに次のログメッセージが表示されます。

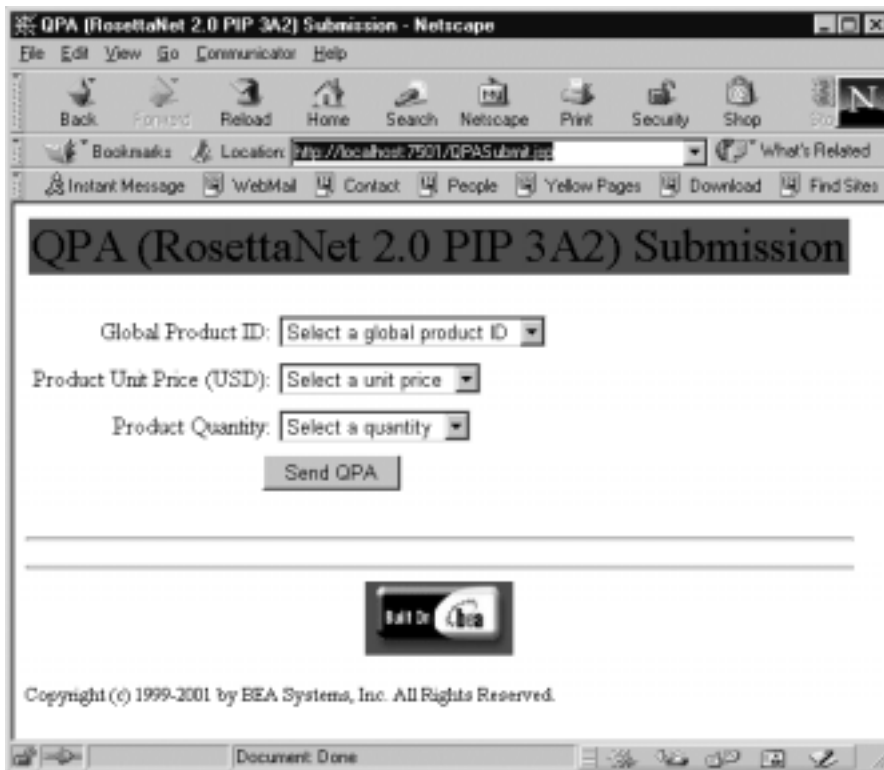
```
RunRN2Security execution successful
```

**警告：** 他の B2B サンプルと異なり、RosettaNet 2.0 Security サンプルでは、サンプルドメインで使用されるデータベースの切り替えを行う WebLogic Integration データベース ウィザードを使用できません。

5. ブラウザを起動します。次の URL を入力して、QPASubmit JSP ページをロードします。

```
http://localhost:7501/QPASubmit.jsp
```

[QPA (RosettaNet 2.0 PIP 3A2) Submission] ページが表示されます。



6. [Global Product ID]、[Product Unit Price]、および [Product Quantity] の各値を選択します。[Send QPA] をクリックします。

次のような送信と応答のステータス情報が、ブラウザに表示されます。



---

### Submit Status

The following QPA entries have been sent successfully:

- Global Product ID = 12345678901234
- Unit Price = 249.95
- Quantity = 1000

---

### Response Status

The following replies have been received:

- Quote from: *Manufacturer(DUNS Number): 987654321*
  - Product ID: 12345678901234
  - Quantity Available: 1000
  - Unit Price: 249.95
- 

7. この時点で 4-10 ページの「ワークフローの要点」に記載の手順を実行する場合は、WebLogic Server のインスタンスを両方とも実行中の状態にします。

この時点で、4-10 ページの「ワークフローの要点」に記載の手順を実行しない場合は、ブラウザを終了し、次のプロシージャによって WebLogic Server のインスタンスを両方ともシャットダウンします。

```
StopRN2Security
```

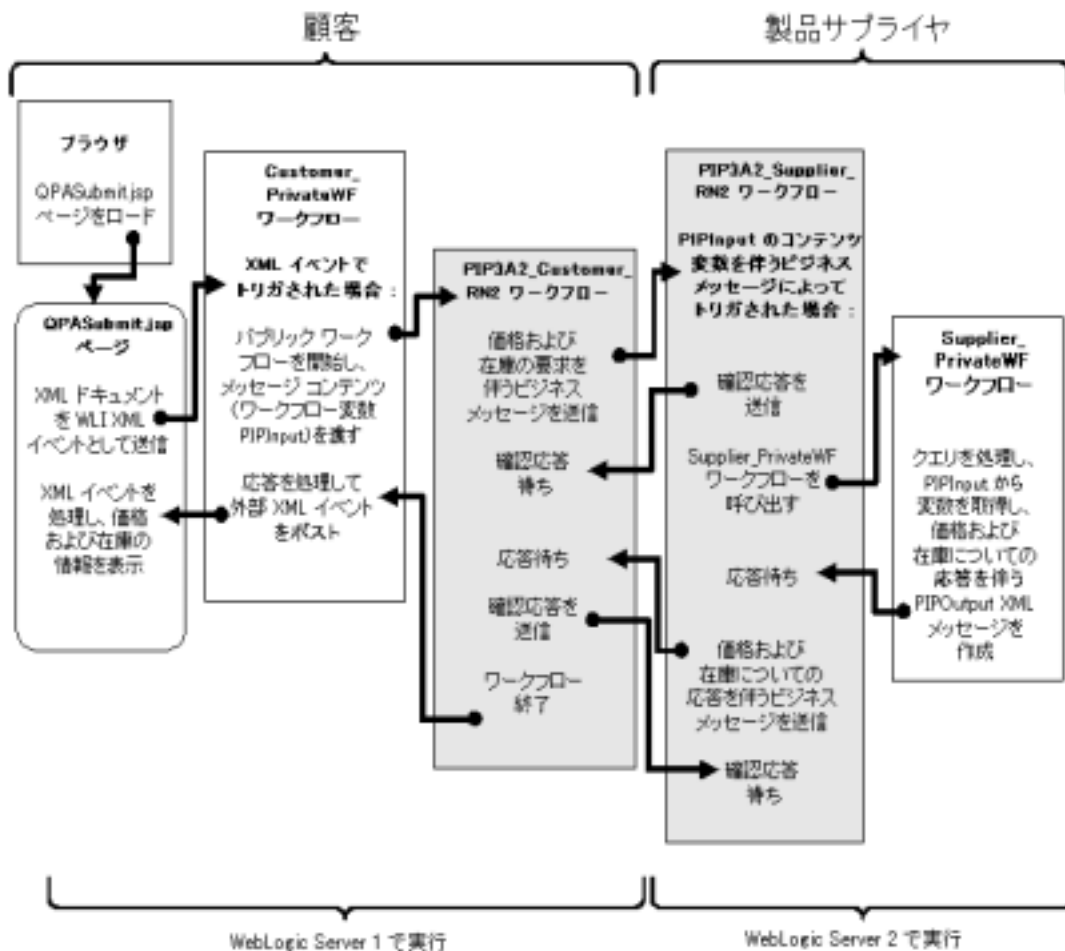
## RosettaNet 2.0 Security サンプルにおけるワークフロー

RosettaNet 2.0 Security サンプルは、WebLogic Integration を使用して RosettaNet 2.0 PIP 3A2 および PIP 0A1 をワークフローで実装する方法を示します。パブリック プロセスは、トレーディング パートナ間の正式な契約です。この契約には、トレーディング パートナ間のメッセージ交換のコンテンツとセマンティクスが規定されます。2つのパブリック ワークフロー (PIP3A2\_Customer\_RN2 お

よび PIP3A2\_Supplier\_RN2) は、PIP 3A2 標準を WebLogic Integration 上で実装したものです。プライベート プロセスは、特定のビジネス組織に固有であり、その組織の外部からは見えません。PIP 3A2 を実装するパブリック ワークフローは、図 4-1 では網掛けされた長方形として表現されています。プライベート プロセスは、網掛けされていない長方形で表しています。図 4-1

次の図は、RosettaNet における各種セキュリティ サンプル ワークフロー間の、さまざまな会話を示します。

図 4-1 RosettaNet 2.0 Security ワークフロー間の会話



**注意:** 図中におけるロジックの流れは、エラーが生じない場合に実行されるワークフローのタスク順序のみを示します。このタスク順序には、エラー処理ロジックは含まれません。たとえば、障害通知を発行するワークフローである PIP 0A1 は、図示されていません。

実行フローを開始する前に、WebLogic Server の 2 つのインスタンスを起動する必要があります。1 つは顧客用、もう 1 つはサプライヤ用です。このサンプルの実行フローは、図 4-1 の左上隅にある「ブラウザ」の四角形から開始されます。次の手順は、実行フローの一部をトレースしています。

1. 実行フローを開始するには、ブラウザを起動し、QPASubmit.jsp ページをロードします。
2. 次に、JSP ページで各値を入力し、[Submit] を選択します。
3. [Submit] を選択すると、XML ドキュメントが WebLogic Integration の XML イベントとして送信されます。
4. このイベントは、Customer\_PrivateWF ワークフローをトリガします。
5. Customer\_PrivateWF ワークフローの最初の項目が実行されます。
6. その結果、PIP3A2\_Cusotmer\_RN2 ワークフローの最初のアクション（価格および在庫の要求を伴うビジネス メッセージの送信）が実行されます。

実行フローの続きについては、図 4-1 を参照してください。

## ワークフローの要点

サンプルのプライベート ワークフローおよびパブリック ワークフローを全体的に調べるには、次の手順を実行します。

1. 次の例に示すように、プラットフォームに合わせて適切なコマンドを入力し、WebLogic Integration のホーム ディレクトリ（WebLobic Integration をインストールしたディレクトリ）に移動します。

- Windows:

```
cd c:\bea\weblogic700\integration
```

- UNIX:

```
cd /home/me/bea/weblogic700/integration
```

2. WebLogic Integration の上位レベルの環境変数を設定するには、お使いのプラットフォームに合った setenv スクリプトを実行します。

- Windows:

```
setenv
```

- UNIX:

```
./setenv.sh
```

3. 4-4 ページの「RosettaNet 2.0 Security サンプルの実行」で説明した RunRN2Security スクリプトを実行して WebLogic Server のインスタンスを 2 つ起動している場合のみ、この手順を実行します。WebLogic Server の 2 つのインスタンスを起動するには、次の例のように使用しているプラットフォームおよびデータベースに対応したプロシージャを実行します。

- Windows:

```
cd %SAMPLES_HOME%\integration\samples\RN2Security\bin  
RunRN2Security pointbase
```

- UNIX:

```
cd $SAMPLES_HOME/integration/samples/RN2Security/bin  
RunRN2Security pointbase
```

WebLogic Server のインスタンスが両方とも起動を終了するまで待機し、次の手順に進みます。サーバの起動が完了すると、WebLogic Server のコンソール ウィンドウに次のログ メッセージが表示されます。

```
RunRN2Security execution successful
```

4. 以下の各プラットフォームに対応した手順を実行して WebLogic Integration Studio を起動します。

- Windows:

[ スタート | プログラム | BEA WebLogic E-Business Platform |  
WebLogic Integration 7.0 | Studio ] を選択します。

- UNIX:

```
cd $WLI_HOME/bin  
studio
```

5. 次の情報を使用して WebLogic Integration Studio にログインします。

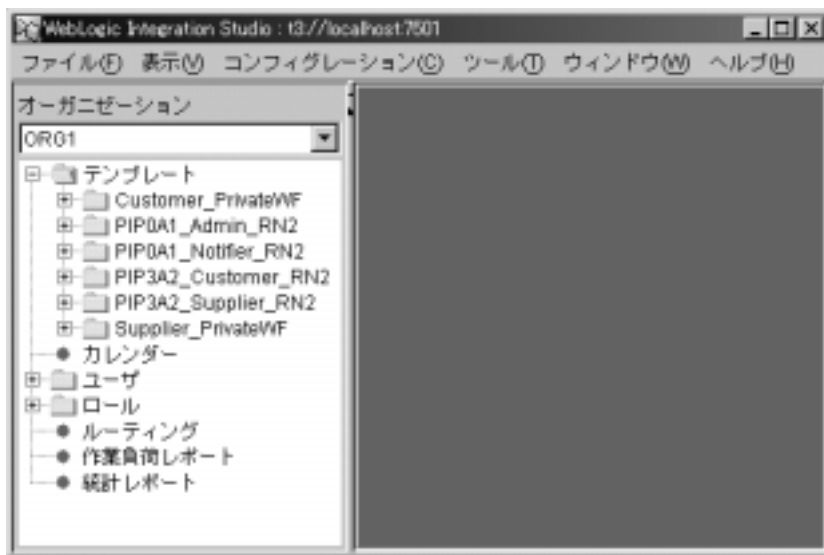
- ログイン: joe
- Password: password
- URL: t3//localhost:7501

WebLogic Integration Studio のメイン ウィンドウが表示されます。

6. 左ペインの [ オーガニゼーション ] の下にあるドロップダウン リストから、[ORG1] を選択します。



7. 左ペインの [テンプレート] フォルダを展開します。このサンプルのテンプレートがすべてリストされます。



8. 左ペインの [Customer\_PrivateWF] フォルダを展開します。

9. Customer\_PrivateWF ワークフローのインスタンスを開いて表示するには、次の手順を実行します。
  - a. Customer\_PrivateWF フォルダ内の、日付と時刻が名前になったフォルダを右クリックします。メニューが表示されます。



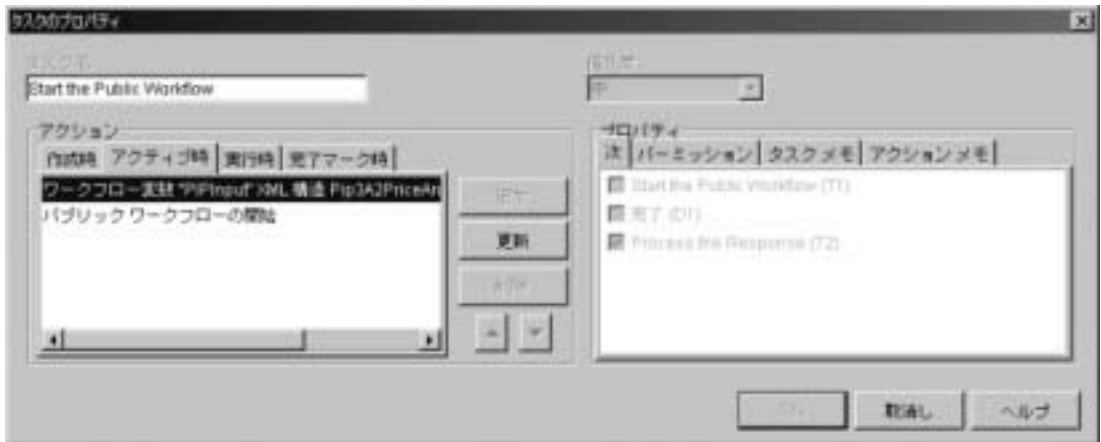
- b. [開く] を選択します。

Customer\_PrivateWF ワークフローを構成する開始ノード、タスク ノード、分岐ノード、およびイベント ノードが表示されます。

10. [パブリックワークフローを開始] タスクをダブルクリックします。[タスクのプロパティ] ウィンドウが表示されます。



11. [ アクティブ時 ] タブを選択します。



[ パブリック ワークフローの開始 ] タスクを構成するアクションが表示されます。

12. [ ワークフロー変数 "PIPInput" XML 構造を設定 ] をダブルクリックし、[ ワークフロー変数を設定 ] ウィンドウを表示します。このウィンドウは、PIPInput XML ワークフロー変数がどのように作成されるかを示します。このサンプルでは、PIPInput は作成済みです。この手順では、これを表示する方法を示します。PIPInput は、必須の WebLogic Integration RosettaNet Input



ワークフロー変数です。これをプライベート ワークフローに設定しないと、パブリック ワークフローを呼び出せません。この変数のフォーマットは、実装されている PIP メッセージの RosettaNet DTD に準拠する XML です。このサンプルでは、XML は

3A2PriceAndAvailabilityQueryMessageGuidline.dtd に準拠している必要があります。この、RosettaNet によって提供される DTD は、顧客トレーディング パートナがサプライヤ トレーディング パートナに渡す最初のメッセージのコンテンツを定義します。4-2 ページの「RosettaNet 2.0 Security サンプルの概要」のステップ 1 を参照してください。

このサンプルでは、プライベート Customer\_PrivateWF ワークフローが PIPInput ワークフロー変数の内容を設定し、PIP3A2\_Customer\_RN2 ワークフローを呼び出します。PIP3A2\_Customer\_RN2 ワークフローは、PIPInput 変数の内容を使用して XML ビジネス メッセージを作成します。このメッセージは、Send Business Message Action が呼び出されると PIP3A2\_Supplier\_RN2 ワークフローに送信されます。RosettaNet テンプレート変数の詳細なリストについては、『*B2B Integration RosettaNet の実装*』の「RosettaNet でのワークフローの使用」の「RosettaNet テンプレート変数」を参照してください。

### 13. XML ツリー内の次の下位ノードを展開します。

- Pip3A2PriceAndAvailabilityQuery
- ProductPriceAndAvailabiltyQuery
- ProductPriceAndAvailability
- ProductLineItem



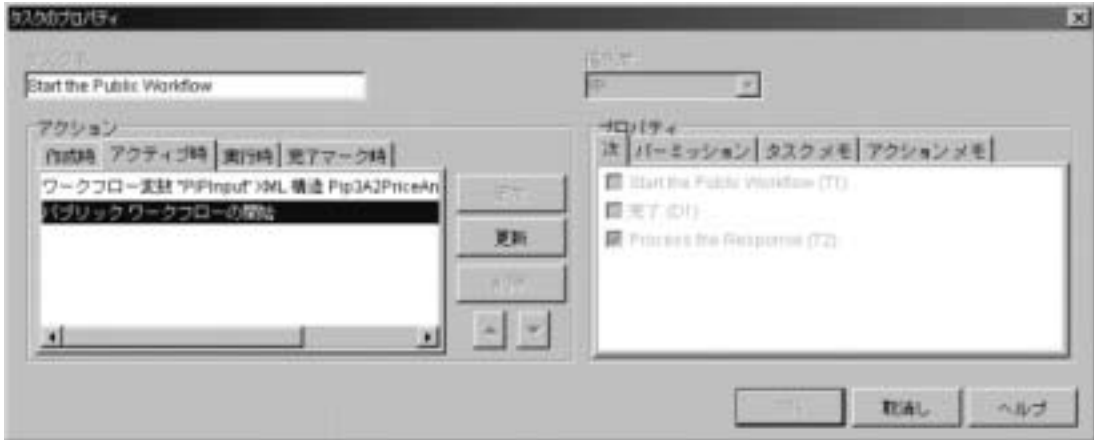
2つの重要なノードの値は、右ペインで設定します。

- **GlobalPartnerRoleClassificationCode XML ノード**は、文字列 **Customer** に設定されます。
- **ProductQuantity ノード**は、**ProductQuantity** ワークフロー変数の値に設定されます。製品の個数（顧客によって要求された商品の数）は [QPASubmit JSP] ページで設定され、**ProductQuantity** ワークフロー変数によってワークフローに渡されます。

**注意：** この手順は、PIPInput 変数に値を入れる方法の1つに過ぎません。PIPInput 変数には、ファイルから読み込んだり、XML ドキュメントのコンテンツ全体を渡すことによっても、値を入れることができません。

14. [ワークフロー変数を設定] ウィンドウで [取消し] をクリックします。

15. [タスクのプロパティ] ウィンドウで、[アクティブ時] タブの [パブリックワークフローの開始] をダブルクリックします。



[パブリックワークフローの開始] ウィンドウが表示されます。



このウィンドウでは、**Start the Public Workflow** アクションに対しパラメータが次のように定義されています。

- [ 会話名 ] が [ 3A2 ] に設定されています。
- [ 会話バージョン ] が [ 1.3 ] に設定されています。
- [ 会話ロール ] が [ カスタマ ] に設定されています。

加えて、[ パーティ ] フィールドでは [ TP 名 ] (トレーディング パートナ名) として [ RNBuyer ] と [ RNSeller ] の 2 つが設定されています。WebLogic Integration では、会話名、会話バージョン、会話ロール、およびパーティと

して指定されたトレーディング パートナにより、あるアクションに使用する適切なコラボレーション アグリーメントを見つけます。

前述のアクションについては、WebLogic Integration はリポジトリ内のアクティブなコラボレーション アグリーメントから、名前が 3A2 でバージョンが 1.3、ロールが顧客である会話を指定する、RNBuyer および RNSeller という名前の 2 つのトレーディング パートナ間におけるコラボレーション アグリーメントを探します。以下の、rn2\_peer1\_sec.xml ファイルからの抜粋では、前述の Start Public Workflow アクションの基準に合致するコラボレーション アグリーメントを定義しています。

#### コード リスト 4-1 インポート リポジトリ データ ファイル内のコラボレーション アグリーメント

```
<collaboration-agreement
  name="RN2|9.9|RosettaNet2|100"
  global-identifier="RN2|9.9|RosettaNet2|RNBuyer|RNSeller|102"
  version="1.0"
  status="ENABLED"
  conversation-definition-name="3A2"
  conversation-definition-version="1.3">
  <party
    trading-partner-name="RNBuyer"
    party-identifier-name="RNBuyerPID"
    delivery-channel-name="RNBuyerChannel"
    role-name="Customer" />
  <party
    trading-partner-name="RNSeller"
    party-identifier-name="RNSellerPID"
    delivery-channel-name="RNSellerChannel"
    role-name="Product Supplier" />
</collaboration-agreement>
```

コラボレーション アグリーメントは、指定したパーティ間で使用される会話定義の名前とバージョンを定義します。コードリスト 4-1 のコラボレーション アグリーメントは、顧客のロールを持つ RNBuyer トレーディング パートナと製品サプライヤのロールを持つ RNSeller トレーディング パートナの間における 会話定義名 3A2 と会話定義バージョン 1.3 の使用を指定します。

WebLogic Integration は会話定義の名前とバージョンを、割り当てられたロールと共に使用して、開始するワークフロー テンプレートを決定できます。コードリスト 4-2 の会話定義では、バージョン番号が 1.3 でトレーディング パートナが顧客ロールを持つ、3A2 という名前の会話に対して、PIP3A2\_Customer\_RN2 ワークフロー テンプレートのインスタンスが開始されることを指定しています。

---

#### コード リスト 4-2 インポート リポジトリ データ ファイル内の会話定義

---

```
<conversation-definition
  name="3A2"
  version="1.3"
  business-protocol-name="RosettaNet"
  protocol-version="2.0">
  <role
    name="Customer"
    wlpi-template="PIP3A2_Customer_RN2">
    <process-implementation wlpi-org="ORG1"/>
  </role>
  <role
    name="Product Supplier"
    wlpi-template="PIP3A2_Supplier_RN2">
    <process-implementation wlpi-org="ORG1"/>
  </role>
</conversation-definition>
```

---

したがって、このサンプルでは、**Start Public Workflow** アクションが PIP3A2\_Customer\_RN2 ワークフローをトリガします。

**Start Public Workflow** で定義された会話名とバージョン番号は、PIP 3A2 について RosettaNet で定義された PIP 名およびバージョンです。これらのパラメータは、ロールおよびトレーディング パートナに指定された値と共に、リポジトリに登録された会話名、会話バージョン、ロール、およびトレーディング パートナに対応しています。

16. [ワークフロー] タブを選択します。



パブリック PIP ワークフローとの間で受け渡しされるテンプレート変数が、[ワークフロー] タブで定義されています。

- パブリック PIP ワークフローに渡されるテンプレート変数は、[パラメータ] の下にリストされます。
- パブリック PIP ワークフローから返されるテンプレート変数は、[結果] の下にリストされます。

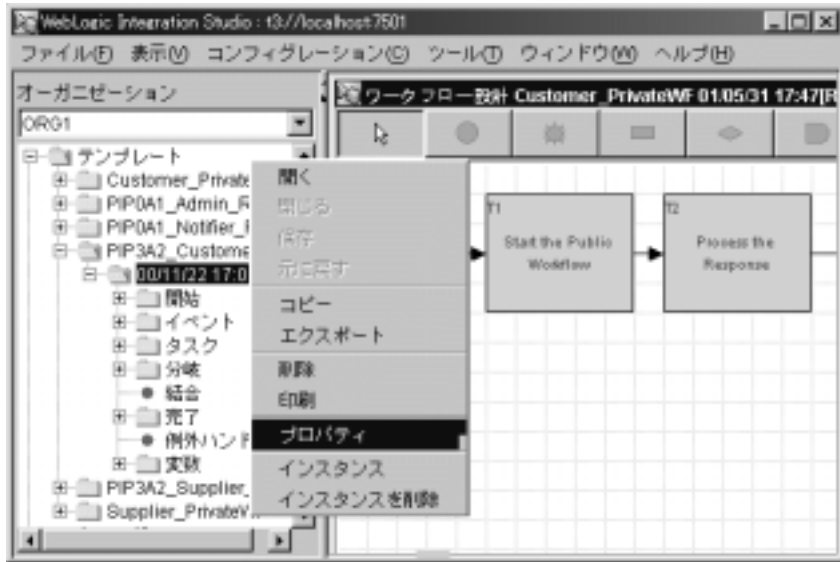
このサンプルでは、テンプレート変数 `fromDUNS` は **Start Public Workflow** アクションの `Customer_PrivateWF` によって設定されます。この変数は `Customer_PrivateWF` ワークフローによって `PIP3A2_Customer_RN2` が呼び出されると `PIP3A2_Customer_RN2` ワークフローに渡されます。**RosettaNet 2.0** のテンプレート変数には、必須のものと、省略可能なものがあります。

送信側の **DUNS** 番号を定義する `fromDUNS` 変数は必須です。**DUNS** 番号は、**Dun & Bradstreet** 社がビジネス エンティティに割り当てた 9 桁の数字による一意の識別子です。`fromDUNS` 変数で指定された **DUNS** 番号は、そのトレーディング パートナのリポジトリで定義されたビジネス ID に一致している必要があります。**RosettaNet** テンプレート変数の詳細なリストについては、『*B2B Integration RosettaNet の実装*』の「**RosettaNet** でのワークフローの使用」の「**RosettaNet** テンプレート変数」を参照してください。

`PIPOutput` は受信したメッセージのサービス コンテンツを含む必須のテンプレート変数です。この変数は `PIP3A2_Customer_RN2` ワークフローにより設定されます。また `PIP3A2_Customer_RN2` ワークフローが、呼び出し元のワークフロー `Customer_PrivateWF` に制御を返すと、`Customer_PrivateWF` ワークフローに渡されます。

17. [パブリック ワークフローの開始] ウィンドウで [取消し] をクリックします。
18. [タスクのプロパティ] ウィンドウで [取消し] をクリックします。
19. Studio のメイン ウィンドウの左ペインで、`[PIP3A2_Customer_RN2]` フォルダを展開します。`PIP3A2_Customer_RN2` ワークフローは、`Customer_PrivateWF` ワークフローから呼び出されます。





20. PIP3A2\_Customer\_RN2 ワークフローのプロパティを表示するには、  
[PIP3A2\_Customer\_RN2] フォルダ内の日付と時刻が名前になったフォルダを  
右クリックします。メニューが表示されます。[プロパティ] を選択します。
21. [Template Definition PIP3A2\_Customer\_RN2] ダイアログ ボックスが表示され  
ます。
22. [B2B Integration] タブを選択します。  
[会話] タブ ([B2B Integration] タブ内) が表示されます。



このタブ上で入力されるデータは、指定した会話およびロールで [パブリックワークフローを開始] アクションが呼び出されると

PIP3A2\_Customer\_RN2 ワークフローが開始されることを指定します。

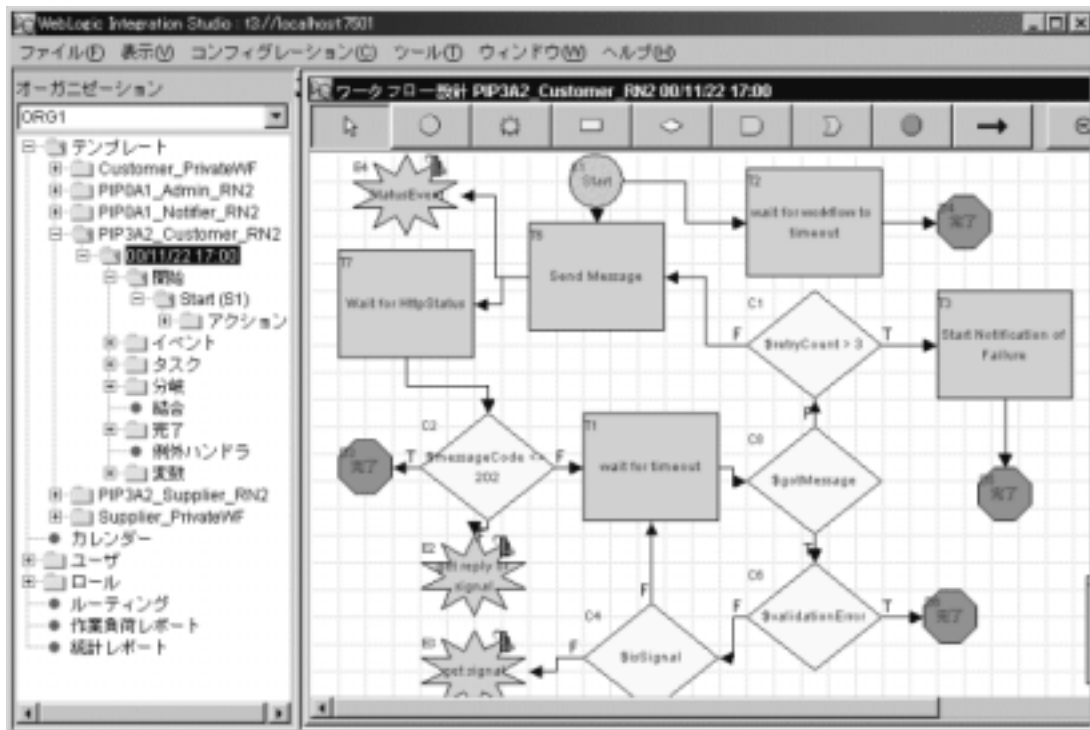
会話名、バージョン、およびロールを、このウィンドウで定義します。

- RosettaNet for PIP 3A2 によって定義された PIP 名、バージョン、およびロールを照合します。
- リポジトリに登録された会話名、バージョン、およびロールに対応させます。

23. [OK] をクリックします。
24. [Workflow Design Customer\_PrivateWF] ウィンドウを閉じます。[ワークフロー設計] ウィンドウの右上の [X] をクリックします。
25. PIP3A2\_Customer\_RN2 ワークフロー インスタンスを開いて表示するには、次の手順を実行します。
  - a. PIP3A2\_Customer\_RN2 フォルダ内の、日付と時刻が名前になったフォルダを右クリックします。メニューが表示されます。

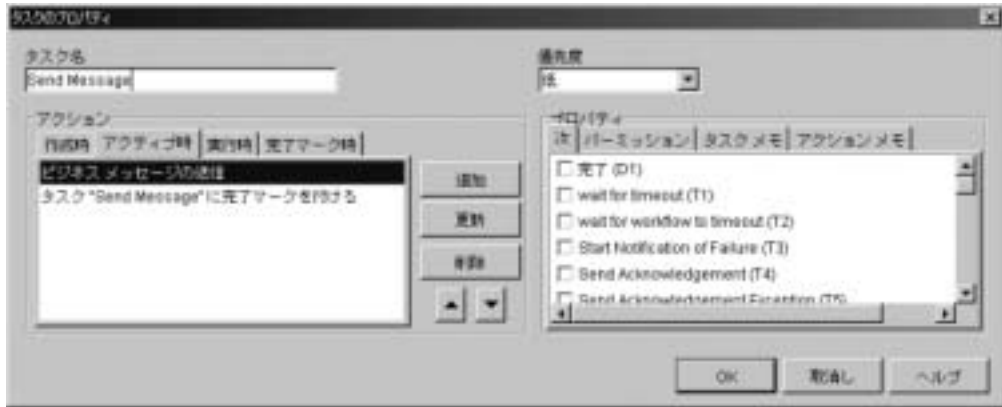


- b. [開く] を選択します。
- PIP3A2\_Customer\_RN2 ワークフローを構成する開始ノード、タスク ノード、分岐ノード、およびイベント ノードが表示されます。
- [開始] ノードは、このウィンドウで実行される最初のタスクです。



26. [メッセージ送信] タスクをダブルクリックします。[タスクのプロパティ] ウィンドウが表示されます。

27. [アクティブ時] タブを選択します。



[メッセージ送信] タスクを構成するアクションが、[アクティブ時] タブにリストされます。

28. [タスクのプロパティ] ウィンドウの左ペインで、[ビジネス メッセージの送信] アクションをダブルクリックします。このアクションは、PIPInput XML ワークフロー変数の内容に基づく XML ビジネス メッセージを送信します。この変数は事前にプライベート Customer\_PrivateWF ワークフローで定義済みです。

**注意：** この手順は、4-2 ページの「RosettaNet 2.0 Security サンプルの概要」のステップ 1 に相当します。

29. WebLogic Integration Studio を閉じます。Worklist のメニュー バーから、[ファイル | 終了] を選択します。



---

# 5 Messaging API サンプル (非推奨)

Messaging API サンプルは、WebLogic Integration Messaging API の使用方法を示します。特に、Messaging API で使用できる 2 つのメッセージ配信メカニズムと、WebLogic Integration B2B のロジック プラグイン機能を例示しています。

この章では、以下のトピックを取り上げます。

- Messaging API サンプルの概要
- Messaging API サンプルを実行する前に
- Messaging API サンプルの実行

**注意：** Messaging API サンプルは WebLogic Integration Messaging API に基づいていますが、この API は、このリリースの WebLogic Integration から非推奨になっています。WebLogic Integration Messaging API に代わる機能については、『*WebLogic Integration リリース ノート*』を参照してください。

## Messaging API サンプルの概要

WebLogic Integration では、ビジネス メッセージを送信する 2 通りの方法をサポートしています。

1. **WebLogic Integration Studio** で作成されるアプリケーション ワークフローを使用する方法。アプリケーション ワークフローには、ビジネス メッセージを送信するアクションが含まれます。**RosettaNet 2.0 Security** サンプルおよび **Channel Master** サンプルは、ビジネス メッセージを送信するワークフロー アプリケーション例です。
2. ビジネス メッセージを送信するために **WebLogic Integration Messaging API** を呼び出す **Java** アプリケーションを使用する方法。

このサンプルは、2つ目の方法を使用します。

WebLogic Integration Messaging API では、2つのメッセージ配信メカニズムをサポートしています。

- 同期-送信側アプリケーションは、公開されたメッセージが宛先に配信されるまで待機します。メッセージングシステムは、メッセージ公開処理の結果が判明した後、アプリケーションに制御を返します。アプリケーションは、タイムアウトが発生するか、処理ステータスが判明するかのどちらかが起こるまで待機します。
- 遅延同期-メッセージ公開後に、制御がアプリケーションに戻されます。アプリケーションには `XOCPMessageToken` オブジェクトが返されます。アプリケーションは後でそのオブジェクトにアクセスしてメッセージ配信のステータスをチェックできます。

このサンプルでは、同期および遅延同期の双方のメッセージ配信メカニズムの使い方を例示しています。

Messaging API サンプルには、ビジネスメッセージを送信する3社のトレーディングパートナー (Partner1、Partner2、および Partner3) が含まれます。Messaging API サンプルには、4つの Java ソースコードファイル (`MdmTp1Servlet.java`、`MdmTp2Servlet.java`、`MdmTp3Servlet.java`、および `WaiterPlugin.java`) が含まれます。

ファイル	含まれるソースコード
<code>MdmTp1Servlet.java</code>	Partner1 トレーディング パートナ
<code>MdmTp2Servlet.java</code>	Partner2 トレーディング パートナ
<code>MdmTp3Servlet.java</code>	Partner3 トレーディング パートナ
<code>WaiterPlugIn.java</code>	ハブ フィルタ ロジック プラグイン

`WaiterPlugIn.java` コードの詳細については、5-7 ページの「実行フローのトレース」を参照してください。



# Messaging API サンプルを実行する前に

Messaging API サンプルを実行する前に、次の手順を実行します。

- 1-3 ページの「サンプルの実行前の作業」に記載の手順に従います。
2. サンプル WebLogic Server への接続が妨げられないようにブラウザのプロキシ設定を確認します。Web ブラウザ コンフィグレーションの要件の詳細については、『*WebLogic Integration の起動、停止およびカスタマイズ*』の「WebLogic Integration 管理ツールと設計ツール」の「Web ブラウザ コンフィグレーションの要件」を参照してください。

## Messaging API サンプルの実行

Messaging API サンプルを実行するには、次の手順を実行します。

1. `WLI_HOME` (WebLogic Integration をインストールしたディレクトリ) に移動します。

```
cd WLI_HOME
```

- Windows の例

WebLogic Platform を `c:\bea` ディレクトリにインストールした場合は、`WLI_HOME` のパスは、`c:\bea\weblogic700\integration` となります。

- UNIX の例

WebLogic Platform を `/home/me/bea` ディレクトリにインストールした場合は、`WLI_HOME` のパスは、`/home/me/bea/weblogic700/integration` となります。

2. WebLogic Integration の上位レベルの環境変数を設定するには、お使いのプラットフォームに合った `setenv` スクリプトを実行します。

- Windows の場合

```
setEnv
```

- UNIX の場合
  - `setenv.sh`
- 3. プラットフォームに合わせて適切な手順を実行し、`RunSamples` スクリプトを起動します。
  - Windows:
    - [スタート | プログラム | BEA WebLogic Platform 7.0 | WebLogic Integration 7.0 | Integration Examples | Start Server and Launch Examples (with dataloader)] を選択します。
  - UNIX:
    - a) `PATH` 環境変数に、`Netscape` 実行ファイル (`netscape`) が格納されたディレクトリが含まれていることを確認します。
    - b) `RunSamples` スクリプトを実行します。

```
cd $SAMPLES_HOME/integration/samples/bin
RunSamples
```

**警告：** UNIX システムの場合、`netscape` 実行ファイルが入ったディレクトリが `PATH` 環境変数に含まれている必要があります。環境変数に含まれていない場合は、`RunSamples` スクリプトの実行時にサンプル起動ページが表示されません。サンプル起動ページは、現在 `RunSamples` スクリプトが実行されているマシンで `Netscape` ブラウザを起動して、次の URL を入力すると起動されます。

```
http://localhost:7001/index.html
```

4. `RunSamples` スクリプトのコンフィグレーションセクションが実行済みであることが検知されると、次のプロンプトが表示されます。

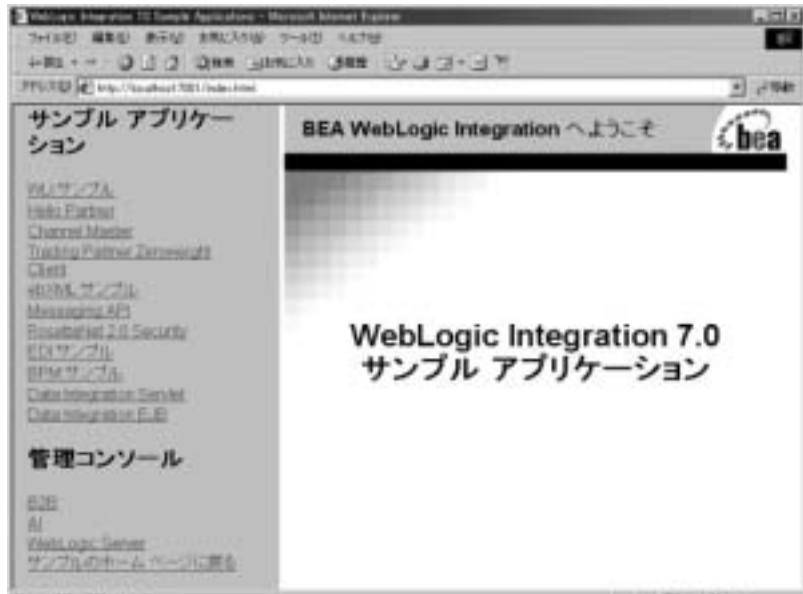
```
The WebLogic Integration repository has already been
created and populated, possibly from a previous run
of this RunSamples script. Do you want to destroy all the
current data in the repository and create and populate the
WebLogic Integration repository, again? Y for Yes, N for No
```

この質問に `N` と入力すると、リポジトリの作成および格納を行う手順が省略され、`WebLogic Server` のサンプル インスタンスを起動する手順のみが実行されます。

この質問に `Y` と入力すると、リポジトリの作成および格納が改めて行われ、その後で `WebLogic Server` のサンプル インスタンスを起動する手順が実行されます。`Y` と入力した場合、その時点でリポジトリに格納されている全デー

タが破棄され、リポジトリにサンプルデータが再ロードされます。現在のサンプルデータが変更または削除され、新規または未変更のサンプルデータをリポジトリに格納する場合にのみ、Yを入力してください。

これで、RunSamples スクリプトは WebLogic Server のインスタンスをバックグラウンドプロセスとして開始し、サンプル起動ページが表示されます。



5. サンプル起動ページの左ペインで、[ サンプルアプリケーション ] の下の [ Messaging API ] のリンクをクリックします。Messaging API サンプルのオプション メニューは、右ペインに表示されています。



必要に応じて、3つのオプションをすべて表示できるようにブラウザの画面をサイズ変更してください。

6. 1つ目のオプション ([Partner3 スポークを参加させる]) をクリックします。このオプションについてのメッセージが表示されるまで待機し、次のステップに進んでください。
7. 2つ目のオプション ([Partner2 スポークを参加させる]) をクリックします。このオプションについてのメッセージが表示されるまで待機し、次のステップに進んでください。
8. 3つ目のオプション ([Partner1 スポークを参加させてリクエストを送信]) をクリックします。

Messaging API サンプルが正常に実行されると、右ペインの下部に次の出力結果が表示されます。

```
Partner1 process flow:
Starting XOCPApplication... done.
Creating conversation : verifierConversation:1.0:
requestor_Partner1_1001029696695_341001029696695...done.
send string for Message 1 = FIRST MESSAGE
Sending message 1 using synchronous deferred delivery method to Partner 2
Sending a second message before checking for acknowledgment on the first
send string for Message 2 = SECOND MESSAGE
```

```
Sending message 2 using synchronous delivery method to Partner 3
success status for message 2
Waiting for Message 2 response... done.
Processing reply for Message 2:
Received string for Message 2 = partner3 -- second message
Verification for Message 2 SUCCESS

Doing something else... done
Waiting acknowledgment for Message 1... Acknowledgment received
Success status for message 1
Waiting for Message 1 response... done
Processing reply:
Received string for Message 1 = partner2 -- first message
Verification for Message 1 SUCCESS

Terminating conversation:verifierConversation:1.0:
requestor_Partner1_1001029696695_341001029696695
success
Shutting down session... done.
```

9. このとき、さらに多くの B2B サンプルを実行する場合は、サンプル起動ページを開いた状態で、WebLogic Server のサンプル インスタンスの実行を続行します。

この時点で、これ以上 B2B サンプルを実行しない場合は、各プラットフォームに合わせて適切な手順を実行し、WebLogic Server の現在のインスタンスをシャットダウンします。

- Windows:

```
cd %SAMPLES_HOME%\config\integration\samples
stopWebLogic
```

- UNIX:

```
cd $SAMPLES_HOME/config/integration/samples
stopWebLogic
```

## 実行フローのトレース

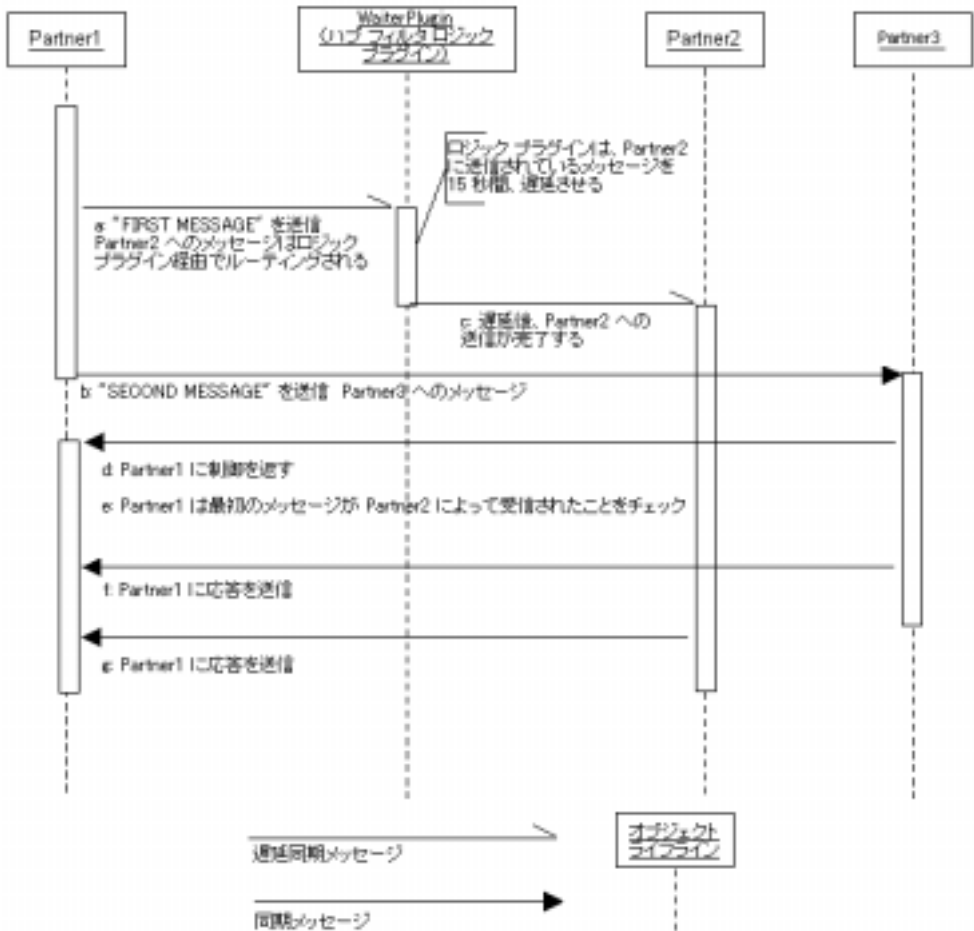
以下は、Messaging API サンプルの実行フローをトレースする手順です。

1. RunSamples スクリプトが起動され、次の結果が得られます。
  - a. WebLogic Server のサンプル インスタンスが開始されます。
  - b. ブラウザが開き、サンプル起動ページが表示されます。

- c. サンプル起動ページで [Messaging API] リンクをクリックすると、[Messaging API] サンプル ページが表示されます。このページには、[Partner3 スポークを参加させる]、[Partner2 スポークを参加させる]、および [Partner1 スポークを参加させてリクエストを送信] の3つのオプションが表示されます。
2. [Partner3 スポークを参加させる] オプションを選択すると、次の結果が得られます。
  - a. HTTP 要求が MdmTp3 サーブレットにポストされます。
  - b. MdmTp3 は、MdmTp3Servlet.java から doPost メソッドを呼び出します。
  - c. doPost メソッドは、トレーディング パートナ Partner3 の XOCApplication を起動します。
3. [Partner2 スポークを参加させる] オプションが選択されると、次の結果が得られます。
  - a. HTTP 要求が MdmTp2 サーブレットにポストされます。
  - b. MdmTp2 は、MdmTp2Servlet.java から doPost メソッドを呼び出します。
  - c. doPost メソッドは、トレーディング パートナ Partner2 の XOCApplication を起動します。
4. [Partner1 スポークを参加させてリクエストを送信] オプションを選択すると、次の結果が得られます。
  - a. HTTP 要求が MdmTp1 サーブレットにポストされます。
  - b. MdmTp1 は、MdmTp1Servlet.java から doPost メソッドを呼び出します。
  - c. doPost メソッドは、トレーディング パートナ Partner1 の XOCApplication を起動します。
  - d. doPost は 1 つ目のメッセージを送信し、3 つのトレーディング パートナ (Partner1、Partner2、および Partner3) 間の一連のメッセージをトリガします。このサンプルの 3 つのトレーディング パートナはすべて、VerifierHubChannel 配信チャネルの使用と verifierConversation 会話への参加のために、WebLogic Integration リポジトリに登録します。Partner1 は、要求側のロールでリポジトリに登録されます。Partner2 および Partner3 は応答側のロールで登録されます。

次の図は、これらのビジネス メッセージの送受信が最も行われやすいタスク順序を示します。正確なタスク順序はタイミングに関係しており、Java 仮想マシンのスレッドのスケジューリングに応じて異なります。

図 5-1 トレーディング パートナ間のメッセージの流れを示す会話図



次のタスク順序では、図 5-1 内の対応する文字で示される各手順の詳細について説明します。



- a. **Partner1** は **Partner2** へテキスト **FIRST MESSAGE** を伴うメッセージを送信します。このメッセージは、遅延同期送信されます。したがって、**Partner1** は **Partner2** からの返信を待機もブロックもせず、引き続きタスクの実行を行います。

メッセージの、ブロックを行わないという局面は、**Partner1** のオブジェクトライフラインによって、会話図（図 5-1）に示されています。**Partner1** が初めてアクティブになったとき、**Partner1** のライフラインは、破線（非アクティブ状態を示す）表示から細い長方形（アクティブ状態を示す）の表示に変化します。ライフラインは、**Partner1** が最初のメッセージを送信した後も、引き続きアクティブです。アクティブであるため、**Partner1** は、次の手順で示すように、別のメッセージの送信など、他のタスクを実行できます。

このサンプルで送信されるメッセージはすべて、ハブ経由でルーティングされます。**WaiterPlugIn** というロジックプラグインがハブのフィルタチェーンに追加されています。ハブを経由してメッセージがルーティングされると、**WaiterPlugIn** クラスの **process** メソッドが実行されます。**process** メソッドは、送信されるメッセージの対象となる受信側をチェックします。対象となる受信側が **Partner2** の場合、**process** は 15 秒間スリープします。それ以外の場合は、直ちにメッセージを送信します。図 5-1 に示す例では、**Partner1** から **Partner2** へ送信された 1 つ目のメッセージは、15 秒間遅延されます。

- b. 1 つ目のメッセージがまだ処理されている間に、**Partner1** はテキスト **SECOND MESSAGE** と共に 2 つ目のメッセージを **Partner3** へ送信します。このメッセージは、同期送信されます。したがって、**Partner1** は **Partner3** が返るまでブロックまたは待機してからでなければ、他のタスクを処理できません。

- c. 図 5-1 で、この **Partner1** のブロックがどのように表現されているかに注意してください。2 つ目のメッセージが同期しているため、**Partner1** のライフラインは、**Partner1** が 2 つ目のメッセージを送信後に、細い長方形（アクティブ状態を示す）の表示から破線（非アクティブ状態を示す）表示に変化します。非アクティブ状態への変化は、メッセージが送信され、**Partner3** によって確認応答があるまで **Partner1** は、再びアクティブになって他のタスクを実行できるようにならないということです。

このサンプルの 2 つ目のメッセージと、すべての応答メッセージは、ハブを経由してルーティングされます。このハブで、**WaiterPlugIn** の **process** メソッドがメッセージに対して実行されます。宛先が **Partner2** ではないため、これらのメッセージは各々、遅延することなくハブを通

過します。これらのメッセージの、ハブ フィルタ ロジック プラグインを介してのルーティングは、図 5-1 には示されていません。

- d. 1 つ目のメッセージはハブで 15 秒間遅延されたあと、Partner2 にルーティングされます。
  - e. Partner3 への同期送信が完了すると、Partner1 に制御が返されます。この時点で Partner1 は再びアクティブになります。これは図 5-1 において、ライフラインの表示が破線（非アクティブ状態を表す）から細い長方形（アクティブ状態を表す）に変化することによって示されます。
  - f. Partner1 は、1 つ目のメッセージが Partner2 によって受信されたことを検証します。
  - g. Partner3 は、2 つ目のメッセージのテキストを取得し、それを小文字に変換して、プレフィクス `partner 3--` を付加します。その後、修正したメッセージを Partner1 に返します。
  - h. Partner2 は、1 つ目のメッセージのテキストを取得し、それを小文字に変換して、プレフィクス `partner 2--` を付加します。その後、修正したメッセージを Partner1 に返します。
5. 結果は、Partner1 によって HTML のページに表示されます。この手順は、図 5-1 には示されていません。

---

## 6 Trading Partner Zeroweight Client サンプル（非推奨）

Trading Partner Zeroweight Client サンプルは、BEA ソフトウェアのホストではないトレーディング パートナが 1 つ以上ある場合に使用する、ブラウザおよびファイル共有という 2 通りの通信方法を例示します。このサンプルは、ブラウザクライアントまたはファイル共有クライアントを介しての要求側と応答側の通信のための、ビジネス実施機能およびビジネス プロセスに基づいています。

この章では、以下のトピックを取り上げます。

- Zeroweight Client サンプルの概要
- Zeroweight Client サンプルを実行する前に
- Zeroweight Client サンプルの実行
- ゼロウェイト クライアントの作成および使用
- サンプルの再コンパイル方法

**注意：** Zeroweight サンプルは、WebLogic Integration Messaging API、B2B ブラウザ クライアント、B2B ファイル共有クライアント用の JSP タグ ライブラリに基づいていますが、そのすべてが、このリリースの WebLogic Integration から非推奨になっています。WebLogic Integration Messaging API、B2B ブラウザ クライアント、B2B ファイル共有クライアントの JSP タグ ライブラリに代わる機能の詳細については、『*WebLogic Integration リリース ノート*』を参照してください。

## Zeroweight Client サンプルの概要

B2B Integration のほとんどの会話には、共に BEA WebLogic Integration がインストールされている 2 社のトレーディング パートナが関与しています。ブラウザ クライアントおよびファイル共有クライアントは、BEA ソフトウェアをインストールしていないトレーディング パートナと通信する手段を提供します。Zeroweight Client サンプルでは、ブラウザ クライアントを使用する要求側トレーディング パートナと、ファイル共有クライアントを使用する応答側トレーディング パートナの間での通信を例示しています。

### サンプルの目的

Zeroweight Client サンプルには、WebLogic Integration がインストールされていない要求側と 1 社以上の応答側の間でどのようにビジネス通信を行うことができるかが示されています。この通信は、B2B Integration がインストールされたリモートまたはホストにおいてコンフィグレーションされた 2 種類のゼロウェイトクライアントを使用して行われます。

- ブラウザ クライアント – 事前に定義された WebLogic Integration ビジネスプロセスによって処理される JSP で会話を開始し、メッセージの送受信を行います。Web ホストは、要求に応じてブラウザ クライアントに配信される JSP 経由の通信に役立ちます。ブラウザ クライアントは WebLogic Integration JSP タグ ライブラリを使用し、B2B Integration メールボックス インタフェースを介してメッセージを送信およびチェックします。
- ファイル共有クライアント – 要求側は Web ホスト上に置かれた、事前にコンフィグレーションされたメールボックスにメッセージを入れます。応答側は WebLogic Integration により提供されるファイル共有クライアントを使用して、メールボックスとファイル共有ディレクトリの間でメッセージを転送します。会話においてファイル共有クライアントを使用するパーティには、あらかじめ FTP がインストールされている必要があります。

このサンプルでは、応答側のプライベート ワークフローで呼び出されたビジネス オペレーションが、事前にコンフィグレーションされたディレクトリから要求を取得し、この要求に基づく応答を作成し、応答を事前にコンフィグレーション

ンされた出力ディレクトリに入れます。実際の FTP サーバは使用されません。このサンプルでは、ビジネス オペレーションを使って FTP サーバをシミュレートしています。

## Zeroweight Client サンプルのシナリオおよび図

Zeroweight Client サンプルのシナリオには、2社のトレーディング パートナ（一方は要求側、もう一方は応答側）が関与しています。これらのトレーディング パートナは、ゼロウェイト クライアントを処理するように事前にコンフィグレーションされているリモートの **WebLogic Integration** を通じて通信します。要求側は、2つの整数の乗算要求を送信します。応答側は乗算を実行し、積を要求側に返します。

**WebLogic Integration** のサンプル インスタンスを実行しているマシン以外マシン上の任意のブラウザを使用します。そうすることで、要求側にローカルの **WebLogic Integration** インストールが存在しないという状況が発生します。その代わりに、要求側はブラウザを使って、リモート マシンにインストールされた **WebLogic Integration** に存在する **JSP** にアクセスします。

**JSP** はメールボックスを作成し、**WebLogic Integration** によって提供される **JSP** タブ ライブラリで **XML** 要求を送信します。また同じ **JSP** が、要求側のメールボックス内のメッセージのチェック、および要求側と応答側のメールボックスからのメッセージの削除にも使用されます。

**注意：** Zeroweight Client サンプルのデフォルトのコンフィグレーションでは、**WebLogic Integration** のサンプル インスタンス、ブラウザ クライアント、および **FTP** クライアントを同じマシン上で実行する指定になっています。ただし Zeroweight Client サンプルのプロダクション デプロイメントでは、**WebLogic Integration**、ブラウザ クライアント、および **FTP** クライアントは3つの異なるマシンで実行されます。

応答側にも **WebLogic Integration** はインストールされていません。一部のトレーディング パートナとの通信には、サード パーティ製の **FTP** サーバ アプリケーションを使用します。また、メッセージの処理には独自のメカニズムを使用することもあります。応答側は、ファイル共有クライアントを通じて要求側と通信します。

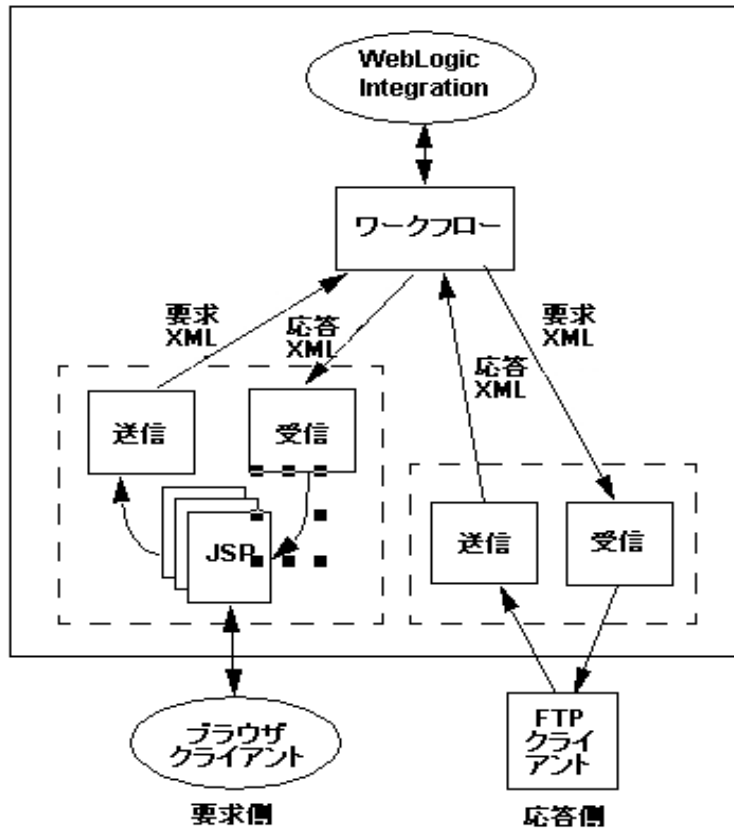
Zeroweight Client サンプルでは、次のイベントを例示しています。

1. 要求側のプライベート ワークフローが、ブラウザ クライアントの **JSP** タグから送信された **XML** イベントによってトリガされます。
2. 応答側のパブリック ワークフローは、ファイル共有クライアントが応答側のメールボックスに入れた応答によってトリガされます。
3. 要求側のメッセージおよび応答側のメッセージはそれぞれ適切なメールボックスに入れられます。

**Zeroweight Client** サンプルでは、あらかじめコンフィグレーションされた 4 つのメールボックスを使用します。各トレーディング パートナは、次のように受信ボックスと送信ボックスを 1 つずつ使用します。

- ブラウザ クライアントを使っている要求側の送信ボックスと受信ボックスは **BrowserTP1\_Inbox** および **BrowserTP1\_Outbox** です。
- ファイル共有クライアントを使っている応答側の受信ボックスと送信ボックスは **FtpTP1\_Inbox** および **FtpTP1\_Outbox** です。

図 6-1 Zeroweight Client のデプロイメント



## Zeroweight Client サンプルを実行する前に

Zeroweight Client サンプルを実行する前に、次の手順を実行します。

1. 1-3 ページの「サンプルの実行前の作業」に記載の手順に従います。

2. サンプル WebLogic Server への接続が妨げられないようにブラウザのプロキシ設定を確認します。Web ブラウザ コンフィグレーションの要件の詳細については、『*WebLogic Integration の起動、停止およびカスタマイズ*』の「WebLogic Integration 管理ツールと設計ツール」の「Web ブラウザ コンフィグレーションの要件」を参照してください。

## Zeroweight Client サンプルの実行

Zeroweight Client サンプルを実行するには、次の手順を実行します。

1. `WLI_HOME` (WebLogic Integration をインストールしたディレクトリ) に移動します。

```
cd WLI_HOME
```

- Windows の例

WebLogic Platform を `c:\bea` ディレクトリにインストールした場合は、`WLI_HOME` のパスは、`c:\bea\weblogic700\integration` となります。

- UNIX の例

WebLogic Platform を `/home/me/bea` ディレクトリにインストールした場合は、`WLI_HOME` のパスは、`/home/me/bea/weblogic700/integration` となります。

2. WebLogic Integration の上位レベルの環境変数を設定するには、お使いのプラットフォームに合った `setenv` スクリプトを実行します。

- Windows の場合

```
setEnv
```

- UNIX の場合

```
. setenv.sh
```

3. プラットフォームに合わせて適切な手順を実行し、`RunSamples` スクリプトを起動します。

- Windows:



[ スタート | プログラム | BEA WebLogic Platform 7.0 | WebLogic Integration 7.0 | Integration Examples | Start Server and Launch Examples (with dataloader) ] を選択します。

- UNIX:

a) PATH 環境変数に、Netscape 実行ファイル (netscape) が格納されたディレクトリが含まれていることを確認します。

b) RunSamples スクリプトを実行します。

```
cd $SAMPLES_HOME/integration/samples/bin
RunSamples
```

**警告:** UNIX システムの場合、netscape 実行ファイルが入ったディレクトリが PATH 環境変数に含まれている必要があります。環境変数に含まれていない場合は、RunSamples スクリプトの実行時に、サンプル起動ページが表示されません。サンプル起動ページは、現在 RunSamples スクリプトが実行されているマシンで Netscape ブラウザを起動して、次の URL を入力すると起動されます。  
http://localhost:7001/index.html

4. RunSamples スクリプトのコンフィグレーション セクションが実行済みであることが検知されると、次のプロンプトが表示されます。

```
The WebLogic Integration repository has already been
created and populated, possibly from a previous run
of this RunSamples script. Do you want to destroy all the
current data in the repository and create and populate the
WebLogic Integration repository, again? Y for Yes, N for No
```

この質問に N と入力すると、RunSamples スクリプトはリポジトリを作成してデータを入れる手順を省略し、WebLogic Server のサンプル インスタンスを起動する手順のみを実行します。

この質問に Y と入力すると、RunSamples スクリプトはリポジトリを再度作成してデータを入れてから、WebLogic Server のサンプル インスタンスを起動します。Y を入力した場合、RunSamples スクリプトは、現在リポジトリにある全データを破棄し、リポジトリに改変されていないバージョンのサンプルデータをロードします。現在のサンプルデータが変更または削除され、新規または未変更のサンプルデータをリポジトリに格納する場合にのみ、Y を入力してください。

これで、RunSamples スクリプトは WebLogic Server のインスタンスをバックグラウンドプロセスとして開始し、サンプル起動ページが表示されます。



5. 左ペインで、[ サンプル アプリケーション ] の下の [ Trading Partner Zeroweight Client ] アプリケーションのリンクをクリックします。右ペインに [ Zeroweight Client Main Page ] ページが表示されます。



6. 以下の各プラットフォームに対応した手順を実行してファイル共有クライアントを起動します。

- Windows:

```
cd %SAMPLES_HOME%\integration\samples\bin
startFSCClient.cmd
```

- UNIX:

```
cd $SAMPLES_HOME/integration/samples/bin
startFSCClient
```

**注意：** JSP のロード前にファイル共有クライアントを起動した場合、Mailbox not found 例外を繰り返し受け取ることになります。メールボックスの作成が行われるのは最初に JSP がロードされたときなので、これらの例外は無視してかまいません。メールボックスの作成後、これらの例外は表示されなくなります。


## 6 Trading Partner Zeroweight Client サンプル (非推奨)

---

7. [ サンプル開始 ] をクリックします。次のページが表示されます。



8. [Go to the request input page] オプションを選択します。[Request for Multiplication] ページが表示されます。

Trading Partner Zeroweight Client Sample 

## Request for Multiplication

Select 2 integers for the multiplication.

The 1st integer:  1  2  3  4  5  6  7  8  9

The 2nd integer:  1  2  3  4  5  6  7  8  9

Request for the multiplication of an integer 2 and an integer 4 has been sent.

[Go to the request input page](#)   
 [Check all replies](#)   
 [Delete all messages in all mailboxes](#)

9. ここで、次の手順を実行して、BrowserTP1\_Outbox へ手動でメッセージを送信します。
- a. 2つの整数を選択します。
  - b. [Send Multiplication Request] をクリックします。

**注意：** この手順の副次的作用として、SendmsgTag およびラッパー Mailbox API がテストされます。

次の図のように、画面が更新されています。

Trading Partner Zeroweight Client Sample 

---

## Request for Multiplication

Select 2 integers for the multiplication.

The 1st integer:  1  2  3  4  5  6  7  8  9

The 2nd integer:  1  2  3  4  5  6  7  8  9

Request for the multiplication of an integer 2 and an integer 3 has been sent.

---

[Go to the request next page](#)   [Check all replies](#)   [Delete all messages in all multi-cases](#)

10. 受信メールボックスに届いた応答をチェックするには、[Check all replies] をクリックします。応答のリストが表示されます。

Trading Partner Zeroweight Client Sample 

## Check All Replies

This page will automatically refresh every 10 seconds

Sender	MessageId	Date	Status
FtpTP1	6734169416441006243	Tue Jan 15 13:26:20 PST 2002	New Delete
FtpTP1	673457913647485161	Tue Jan 15 13:27:58 PST 2002	New Delete
FtpTP1	5715198116733518832	Tue Jan 15 14:53:25 PST 2002	New Delete

Total 3 replies in the mailbox.

[Go to the request input page](#) [Check all replies](#) [Delete all messages in all mailboxes](#)

**注意：** 応答のリストがチェックされるときに、CheckallmsgTag もチェックされます。

応答がなければ、次のメッセージが表示されます。

Trading Partner Zeroweight Client Sample 

## Main Page

The requester and replies' IN and OUT mailboxes are now ready.  
Please click on Go to the request input page link to start.

[Go to the request input page](#) [Check all replies](#) [Delete all messages in all mailboxes](#)

11. 参加しているすべてのトレーディング パートナの受信メールボックスと送信メールボックス（このサンプル内では、BrowserTP1 および FtpTP1 のメールボックス）からメッセージをすべて削除するには、次の手順を実行します。

- a. [ Delete all messages in all mailboxes ] をクリックします。B2B Integration のすべてのメールボックスに入っている全メッセージのリストが表示されます。



- b. [Delete all messages in all mailboxes] を再度クリックします。削除されたメッセージの総数が表示されます。  
**注意：** この手順の副次的作用として、DeleteallmsgTag のテストも行われます。
12. メールボックスから特定のメッセージを削除するには、次の手順を実行します。
- a. [Check All Replies] ウィンドウ (ステップ 6 で表示) で、メッセージを選択します。
  - b. [Delete] をクリックします。  
[Delete One Reply] ウィンドウで、メッセージが正常に削除できたことが確認できます。



Trading Partner Zeroweight Client Sample



## Delete Single Reply

Message <7948526373375796071> deleted successfully

[Go to the request input page](#)

[Check all replies](#)

[Delete all messages in all mailboxes](#)

**注意：** この手順の副次的作用として、DeleteMsgTag のテストも行われ  
ます。

13. メッセージの詳細を表示するには、次の手順を実行します。
  - a. [Check all replies] と表記されたリンクをクリックします。
  - b. [Message ID] 欄から表示するメッセージを選択します。[View Reply Detail] ウィンドウが表示されます。



14. このとき、さらに多くの B2B サンプルを実行する場合は、サンプル起動ページを開いた状態で、WebLogic Server のインスタンスの実行を続行します。この時点で、これ以上の B2B サンプルを実行しない場合は、ブラウザを終了し、プラットフォームに合わせた適切な手順によって WebLogic Server のインスタンスをシャットダウンします。

- Windows:

```
cd %SAMPLES_HOME%\integration\config\samples
stopWebLogic
```

- UNIX:

```
cd $SAMPLES_HOME/integration/config/samples
stopWebLogic
```

# ゼロウェイト クライアントの作成および使用

この節では、ゼロウェイト クライアントの作成とコンフィグレーションに関する次の情報を提供します。

- ゼロウェイト クライアントのソース ファイル
- JSP タグ ライブラリの使用
- Zeroweight Client のコンフィグレーション

## ゼロウェイト クライアントのソース ファイル

WebLogic Integration は、ユーザ独自のゼロウェイト クライアントの作成に必要なすべてのソース ファイルを提供しています。Zeroweight Client サンプルは B2B Integration の情報を Hello Partner サンプルと共有します。

次の表に、Zeroweight Client サンプル専用提供される全ファイルをリストしています。いずれのパス名でも、`SAMPLES_HOME` は、WebLogic Platform のサンプルディレクトリを表しています。

ソース ディレクトリ	サブディレクトリおよびファイル	内容
<code>SAMPLES_HOME\integration\samples\lightweightClient\src\wlcsamples\lightweightClient</code>		ビジネス オペレーションおよびそのヘルパーのためのソース ファイルが格納されている。どちらもワークフロー インスタンスによって呼び出される。
<code>SAMPLES_HOME\integration\config\samples</code>		

## 6 Trading Partner Zeroweight Client サンプル (非推奨)

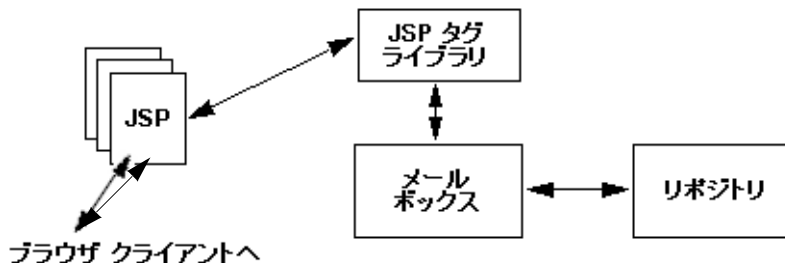
ソース ディレク トリ	サブディレクトリ およびファイル	内容
	LwcFileSync.dtd	LwcFileSync.xml のための DTD ファイル
	LwcFileSync.xml	ファイル共有クライアントによって読み取られる ファイル。使用している環境に合わせて、次の個所 を修正する必要がある。 <ul style="list-style-type: none"> <li>■ url</li> <li>■ wlcfilesync name</li> <li>■ directoryin</li> <li>■ directoryout</li> </ul>
<i>SAMPLES_HOME</i> \integration\samples\lightweightClient\lib		
<i>SAMPLES_HOME</i> \integration\config\samples 内にある LwcFileSync.dtd および LwcFileSync.xml の元のファイルが格納されている。		
<i>SAMPLES_HOME</i> \integration\samples\bin		
	<ul style="list-style-type: none"> <li>■ startFSClient.cmd</li> <li>■ startFSClient</li> </ul>	以下のタスクを担当するファイル共有クライアント を起動するスクリプト。 <ul style="list-style-type: none"> <li>■ 応答側の受信メールボックスからファイル共有 ディレクトリへ要求を転送するタスク</li> <li>■ 応答側の送信ディレクトリから送信メールボック スへ応答を転送するタスク</li> </ul>
<i>SAMPLES_HOME</i> \integration\config\samples\lightweightClient\		
	ftpDir\	<b>Zeroweight Client</b> サンプルの実行用に作成された ディレクトリ。ファイル共有クライアント上の受信 ディレクトリと送信ディレクトリをシミュレートす る、2つのディレクトリ (FtpTP1_indir および FtpTP1_outdir) が含まれている。名前は、このサ ンプル用にハードコード化されている。  <b>注意：</b> このディレクトリ、およびそのサブディレク トリは、実行時に作成される。
integration\samples\lightweightClient\		

ソース ディレクトリ	サブディレクトリ およびファイル	内容
	web\ web\lwcWebApp\  <i>SAMPLES_HOME</i> \integration\samples\lightweightClient\src\wlcsamples\lightweightClient\ tags\ 	*.jpg ファイルや *.jsp ファイルなど、Web アプリケーション用のファイルが格納されている。  Zeroweight Client Web アプリケーションで使用される JSP タグ ライブラリ記述ファイル、および web.xml ファイルが格納されている。  B2B Integration JSP タグ ライブラリの参照実装が格納されている。com.bea.lwclient.LwcMailbox によるメッセージ管理に使用する。

## JSP タグ ライブラリの使用

JSP タグ ライブラリは、WebLogic Integration Messaging API の処理にラッパーを使用します。JSP タグ ライブラリは、図 6-2 に示すように、あらかじめ定義されたワークフローで、メールボックスとの会話に使用されます。

図 6-2 JSP タグ ライブラリを使用するメールボックスのシナリオ



この図に示したメールボックスは、リポジトリを補助するものです。タグ ライブラリは必要に応じてメールボックスにアクセスします。リポジトリにも、メールボックス メッセージは保存されます。

エラー処理は、Mailbox API クラスによる標準の Java 例外とエラー処理、およびブラウザ クライアントに渡される JSP に基づいています。

JSP タグの詳細なリストについては、付録 A 「JSP タグ リファレンス」を参照してください。

**注意：** WebLogic Integration Messaging API の B2B JSP タグ ライブラリは、このリリースの WebLogic Integration から非推奨になっています。このタグ ライブラリに代わる機能については、『WebLogic Integration リリース ノート』を参照してください。

## Zeroweight Client のコンフィグレーション

この節は、WebLogic Integration のサンプル インスタンスが実行されているマシン以外のマシンにブラウザ クライアントまたはファイル共有クライアントを移動しようとしているユーザのみを対象としています。クライアントをこのように移動する場合は、ここに記載した適切な手順を実行して、リモート ゼロウェイト クライアントのコンフィグレーションを行ってください。

Zeroweight Client サンプルと、WebLogic Integration のサンプル インスタンスを同じマシン上で実行している場合は、この節を読む必要はありません。

この節では、次のタスクのための手順を説明します。

- ファイル共有クライアントのコンフィグレーション
- ブラウザ クライアントのコンフィグレーション

## ファイル共有クライアントのコンフィグレーション

ゼロウェイト クライアントとは、WebLogic Integration が専用の Java 仮想マシンで実行するプロセスです。したがって、セキュリティ メカニズムは不要です。

WebLogic Integration は、LwcFileSync.xml コンフィグレーション ファイルで定義されているあらゆるゼロウェイト クライアントにサービスを提供することができます。詳細については、この節で後述する「ファイル共有コンフィグレーション ファイルの編集」を参照してください。

## WebLogic Integration コンフィグレーション ファイルの編集

コード リスト 5-1 のゼロウェイト トレーディング パートナ コンフィグレーション情報を、B2B Integration のコンフィグレーション ファイル config.xml に追加します。

### コード リスト 6-1 ファイル共有クライアントのコンフィグレーション

```
<StartupClass
  ClassName="com.bea.lwclient.Startup"
  Name="LwcStartup"
  Targets="myserver"
/>
```

コンフィグレーション ファイルは、以下のコンフィグレーション 済みサンプルのドメイン (*SAMPLES\_HOME* は WebLogic Integration のサンプル ディレクトリ) で編集できます。

*SAMPLES\_HOME*\integration\config\samples\config.xml

## ファイル共有コンフィグレーション ファイルの編集

ゼロウェイト クライアントをコンフィグレーションするには、LwcFileSync.xml で定義します。LwcFileSync.xml のファイル共有クライアントのコンフィグレーションは、LwcFileSync.dtd に準拠している必要があります。

**注意：** LwcFileSync.dtd は、ファイル共有クライアントが起動するディレクトリ内に置く必要があります。

コードリスト 6-2 は、サンプル DTD ファイルを示します。太字の値を、ご使用のゼロウェイト クライアントに適した値に置換してください。

### コード リスト 6-2 サンプル LwcFileSync.dtd

---

```
<!-- この DTD は、ファイル共有クライアントのコンフィグレーション ファイルを説明する -->
<!ELEMENT wlcfilesynconfig (wlcfilesync*) >

<!-- maxThreads 所定の時点のスレッド プール内における -->
<!-- スレッドの最小許容数 -->
<!ATTLIST wlcfilesynconfig maxThreads CDATA #REQUIRED>

<!-- minThreads 所定の時点のスレッド プール内における -->
<!-- スレッドの最小許容数 -->
<!ATTLIST wlcfilesynconfig minThreads CDATA #REQUIRED>

<!-- maxIdleTime The maximum time interval thread could be idle -->
<!-- それ以外の場合は、プールから削除される -->
<!ATTLIST wlcfilesynconfig maxIdleTime CDATA #REQUIRED>

<!-- pollInterval ポーリング前の待機間隔を定義する時間間隔 -->
<!ATTLIST wlcfilesynconfig pollInterval CDATA #REQUIRED>

<!-- url Weblogic JNDI ルックアップに使用される URL -->
<!ATTLIST wlcfilesynconfig url CDATA #REQUIRED>

<!-- デバッグ処理のオン / オフを切り替えるデバッグ フラグ -->
<!ATTLIST wlcfilesynconfig debug (TRUE | FALSE) "FALSE">

<!-- 次の要素は、LWTP の詳細のコンフィグレーションに使用される -->
<!ELEMENT wlcfilesync EMPTY>

<!-- LWTP name -->
<!ATTLIST wlcfilesync name CDATA #REQUIRED>

<!-- ローカル ファイル システム上の受信ディレクトリのパス -->
<!ATTLIST wlcfilesync directoryin CDATA #REQUIRED>

<!-- ローカル ファイル システム上の送信ディレクトリのパス -->
<!ATTLIST wlcfilesync directoryout CDATA #REQUIRED>
```

---

コードリスト 6-3 は、ファイル共有クライアントがゼロウェイト トレーディング パートナとして定義されるサンプル XML ファイルを示します。太字の文字列を、ご使用のゼロウェイト クライアントに適した値に置換してください。



## コード リスト 6-3 サンプル LwcFileSync.xml

```
<?xml version="1.0"?>
<!DOCTYPE wlcfilesynconfig SYSTEM "LwcFileSync.dtd">
<wlcfilesynconfig maxThreads="9"
    minThreads="3"
    maxIdleTime="5000"
    pollInterval="2000"
    url="t3://localhost:7001"
    debug="FALSE">
<wlcfilesync name="FtpTP1"

directoryin="<SAMPLES_HOME>/samples/integration/config/samples
    /lightweightClient/ftpDir/FtpTP1_indir"

directoryout="<SAMPLES_HOME>/samples/integration/config/samples
    /lightweightClient/ftpDir/FtpTP1_outdir" />
</wlcfilesynconfig>
```

## ブラウザ クライアントのコンフィグレーション

WebLogic Integration では、使用されるセキュリティの枠組みに応じて、HTTP と HTTPS、または SSL という、ブラウザ クライアントをコンフィグレーションする 2 つの方法をサポートしています。以下の各節では、これらの方法のための手順を説明します。

- HTTP ブラウザ クライアントのコンフィグレーション
- HTTPS (SSL) ブラウザ クライアントのコンフィグレーション

## HTTP ブラウザ クライアントのコンフィグレーション

HTTP ゼロウェイト クライアントをコンフィグレーションするには、web.xml ファイルを編集する必要があります。ブラウザ クライアントのセキュリティをオンにするには、次の手順を実行します。

1. トレーディング パートナ ゼロウェイト クライアント Web アプリケーションの HTML ユーザ インタフェースを担当する JSP 開発者は、sendmsg タグを使用し、引数 (必須) security = ON または OFF を指定します。

ワークフローテンプレートの開発者も、

wlcsamples.zeroClient.LwcBizOp.putMessage() メソッドを呼び出すこ

とで、セキュリティのオンとオフを切り替えることができます。このメソッドは、**SECURITY** の最後の引数として文字列 (ON または OFF) を受け付けます。セキュリティのオン、オフを切り替えるには、開発者は以下の引数のうち 1 つを指定します。

2. ホストされた **Web** アプリケーションをデプロイする担当者は、`web.xml` ファイルを修正し、このアプリケーションを使用するためのパーミッションを与えられたすべてのユーザに対し、セキュリティ上の制約を付加します。このファイルのパス名は、`SAMPLES_HOME\samples\lightweightClient\web\lwcWebApp\web.xml` で、`SAMPLES_HOME` は **WebLogic Platform** サンプルディレクトリを表しています。

たとえば、**Web** アプリケーション `lwcWebApp.war` を使用するために必要なセキュリティ権限を新規ユーザに付与するには、次のコードを追加します。

```
<security-constraint>
  <web-resource-collection>
    <web-resource-name>lwcWebApp</web-resource-name>
    <url-pattern>/*</url-pattern>
  </web-resource-collection>
  <auth-constraint>
    <role-name>newpartner</role-name>
  </auth-constraint>
</security-constraint>

<login-config>
  <auth-method>BASIC</auth-method>
</login-config>

<security-role>
  <role-name>newpartner</role-name>
</security-role>
```

3. `newpartner` という名前を付けられたユーザが、有効な **WebLogic Server** ユーザであることを確認します。このユーザ名は、**WebLogic Server Administration Console** または **WebLogic Integration B2B Console** によって割り当てることができます。詳細については、それぞれ『*WebLogic Server 管理者ガイド*』または『*B2B Integration 管理ガイド*』を参照してください。
4. `newpartner` ユーザは、**B2B Console** を使って、次の 2 つの手順によって有効なトレーディング パートナ ゼロウェイト クライアントにマップする必要があります。
  - a. トレーディング パートナ ゼロウェイト クライアントを作成する。
  - b. ユーザの ID を適切なトレーディング パートナのレコードにマップする。

詳細については、『*B2B Integration 管理ガイド*』を参照してください。

5. **WebLogic Integration Studio** を使って、*newpartner* と **B2B Integration** ハブの間のコラボレーション アグリーメントを作成します。詳細については、『*WebLogic Integration Studio ユーザーズ ガイド*』を参照してください。

## HTTPS (SSL) ブラウザ クライアントのコンフィグレーション

ゼロウェイト HTTPS (SSL) クライアントをコンフィグレーションするには、次の手順を実行します。

1. ブラウザと **B2B Integration** ノードの双方に対する証明書をコンフィグレーションします。
2. *newpartner* という **WebLogic Server** ユーザを作成します。このユーザは、トレーディング パートナ ゼロウェイト クライアントの代わりにブラウザを実行できます。このユーザは、**WebLogic Server Administration Console** または **WebLogic Integration B2B Console** で作成することができます。
3. **B2B Console** を使って、トレーディング パートナ ゼロウェイト クライアントを作成します。
4. 適切なトレーディング パートナ レコードに *newpartner* をマップします。
5. **WebLogic Integration B2B Console** を使って、ゼロウェイト クライアントと **B2B Integration** ハブの間のコラボレーション アグリーメントを作成します。詳細については、『*B2B Integration 管理ガイド*』を参照してください。

## サンプルの再コンパイル方法

サンプルに何らかの変更を加えた場合は、実行前に再コンパイルする必要があります。修正したサンプルを再コンパイルするには、次の手順を実行します。

1. *WLI\_HOME* (**WebLogic Integration** をインストールしたディレクトリ) に移動します。

```
cd WLI_HOME
```

- Windows の例

WebLogic Platform を c:\bea ディレクトリにインストールした場合は、`WLI_HOME` のパスは、`c:\bea\weblogic700\integration` となります。

- UNIX の例

WebLogic Platform を /home/me/bea ディレクトリにインストールした場合は、`WLI_HOME` のパスは、`/home/me/bea/weblogic700/integration` となります。

2. プラットフォームに対応した手順を実行することによって `setenv` スクリプトを実行し、最上位の **WebLogic Integration** 環境変数を設定します。

- Windows

```
setEnv
```

- UNIX

```
. setenv.sh
```

3. `project` ディレクトリに移動します。

- Windows:

```
cd $SAMPLES_HOME  
cd integration\samples\lightweightClient\project
```

- UNIX:

```
cd $SAMPLES_HOME  
cd integration/samples/lightweightClient/project
```

4. 次のコマンドを実行します。

```
ant all
```

---

## 7 ebXML サンプル

ebXML サンプルは、ebXML ビジネス プロトコルを使用して 2 つのトレーディング パートナがビジネス メッセージを交換する方法を示します。ebXML サンプル アプリケーションは、`SAMPLES_HOME\integration\samples\ebxml` ディレクトリにあります (`SAMPLES_HOME` は WebLogic Platform サンプル ディレクトリを表している)。

この章では、以下のトピックを取り上げます。

- ebXML サンプルの概要
- ebXML サンプルを実行する前に
- ebXML サンプルの実行
- サンプルの仕組み

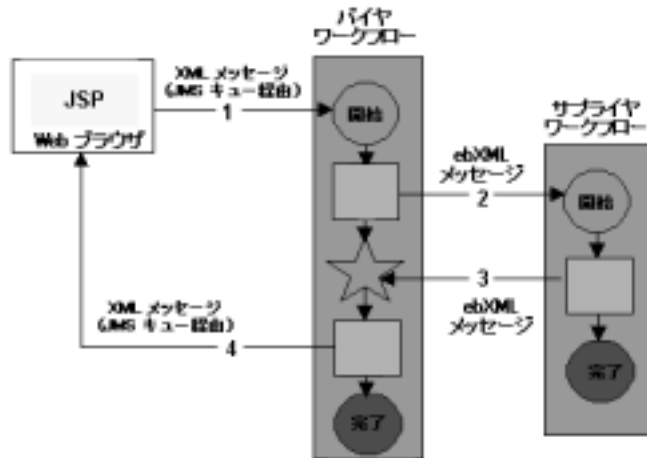
### ebXML サンプルの概要

WebLogic Integration は、E ビジネス トランザクションにおけるビジネス メッセージの交換のための ebXML ビジネス プロトコルをサポートします。ebXML サンプルは、それぞれが WebLogic Integration をデプロイする 2 つのトレーディング パートナ間の ebXML ベースの会話を、2 つのワークフローを使用して管理する方法を例示します。

このサンプル中のワークフローは、`ebXMLConversationInitiator` および `ebXMLConversationResponder` と名付けられています。これらのワークフローは、トレーディング パートナに代わって、QPA (価格および在庫) 会話で、開始者と参加者のロールを通じて ebXML メッセージの交換を管理します。このサンプルでは、QPA プロセスの開始と QPA 要求データおよび応答データの表示に使用できる Java Server Page (JSP) を示します。

次の図は、QPA ビジネス トランザクションに 関与するトレーディング パートナ間のデータ フローを示しています。

図 7-1 QPA ビジネス プロセスのデータ フロー



以下のタスク順序では、このサンプルのトレーディング パートナ間通信の概要を示します。

1. バイヤは、製品、その単価、および購入する数量を選択するために提供された Web フォームを使用します。その Web フォームが掲載されている JSP は、JMS キューに XML メッセージを送信し、バイヤの (ebXMLConversationInitiator) ワークフローをトリガします。

2. バイヤのワークフローは、QPA メッセージおよび選択された製品の詳細情報をサプライヤトレーディング パートナに送信します。QPA メッセージは ebXML フォーマットです。

バイヤのワークフロー (ebXMLConversationInitiator) から送信された ebXML メッセージは、サプライヤのワークフロー (ebXMLConversationResponder) をトリガします。

3. サプライヤのワークフローは、QPA を処理して、同じく ebXML フォーマットの応答をバイヤトレーディング パートナに送信します。

4. バイヤのワークフロー (ebXMLConversationInitiator) は、QPA 応答 ebXML メッセージを受け取って、XML ファイルに書き込みます。JSP は XML を解析し、QPA 応答をバイヤの Web ブラウザに表示します。

このステップは、QPA ビジネス プロセスが終了したことを示します。

## ebXML サンプルを実行する前に

ebXML サンプルを実行する前に、次の手順を実行します。

- 1-3 ページの「サンプルの実行前の作業」に記載の手順に従います。
- サンプル WebLogic Server への接続が妨げられないようにブラウザのプロキシ設定を確認します。Web ブラウザ コンフィグレーションの要件の詳細については、『*WebLogic Integration の起動、停止およびカスタマイズ*』の「WebLogic Integration 管理ツールと設計ツール」の「Web ブラウザ コンフィグレーションの要件」を参照してください。

## ebXML サンプルの実行

ebXML サンプルを実行するには、次の手順を実行します。

1. `WLI_HOME` (WebLogic Integration をインストールしたディレクトリ) に移動します。

```
cd WLI_HOME
```

- Windows の例

WebLogic Platform を `c:\bea` ディレクトリにインストールした場合は、`WLI_HOME` のパスは、`c:\bea\weblogic700\integration` となります。

- UNIX の例

WebLogic Platform を `/home/me/bea` ディレクトリにインストールした場合は、`WLI_HOME` のパスは、`/home/me/bea/weblogic700/integration` となります。

2. WebLogic Integration の上位レベルの環境変数を設定するには、お使いのプラットフォームに合った `setenv` スクリプトを実行します。

- Windows の場合

```
setEnv
```

- UNIX の場合

```
. setenv.sh
```

3. プラットフォームに合わせて適切な手順を実行し、RunSamples スクリプトを起動します。

- Windows:

[スタート | プログラム | BEA WebLogic Platform 7.0 | WebLogic Integration 7.0 | Integration Examples | Start Server and Launch Examples (with dataloader)] を選択します。

- UNIX:

a) PATH 環境変数に、Netscape 実行ファイル (netscape) が格納されたディレクトリが含まれていることを確認します。

b) RunSamples スクリプトを実行します。

```
cd $SAMPLES_HOME/integration/samples/bin
RunSamples
```

**警告：** UNIX システムの場合、netscape 実行ファイルが入ったディレクトリが PATH 環境変数に含まれている必要があります。環境変数に含まれていない場合は、RunSamples スクリプトの実行時にサンプル起動ページが表示されません。サンプル起動ページは、現在 RunSamples スクリプトが実行されているマシンで Netscape ブラウザを起動して、次の URL を入力すると起動されます。  
<http://localhost:7001/index.html>

4. RunSamples スクリプトのコンフィグレーション セクションが実行済みであることが検知されると、次のプロンプトが表示されます。

```
The WebLogic Integration repository has already been
created and populated, possibly from a previous run
of this RunSamples script. Do you want to destroy all the
current data in the repository and create and populate the
WebLogic Integration repository, again? Y for Yes, N for No
```

この質問に N と入力すると、RunSamples スクリプトはリポジトリを作成してデータを入れる手順を省略し、サンプルドメインで WebLogic Server のインスタンスを起動する手順のみを実行します。

この質問に Y と入力すると、RunSamples スクリプトはリポジトリを再度作成してデータを入れてから、サンプルドメインで WebLogic Server のインスタンスを起動します。Y と入力した場合、その時点でリポジトリに格納され



ている全データが破棄され、リポジトリにサンプルデータが再ロードされます。以下の状況では、Yと入力します。

- サービス パック アップグレード インストーラを使用して **WebLogic Integration** インストールをアップグレードし、**ebXML** を初めて実行する場合（**ebXML** サンプルは、このサービス パックの新規サンプルです。**RunSamples** スクリプトを使用して、リポジトリに、サンプル実行に必要なデータを入れる必要がある）。
- 現在のサンプル データが変更または削除され、新規または未変更のサンプル データをリポジトリに格納する場合。

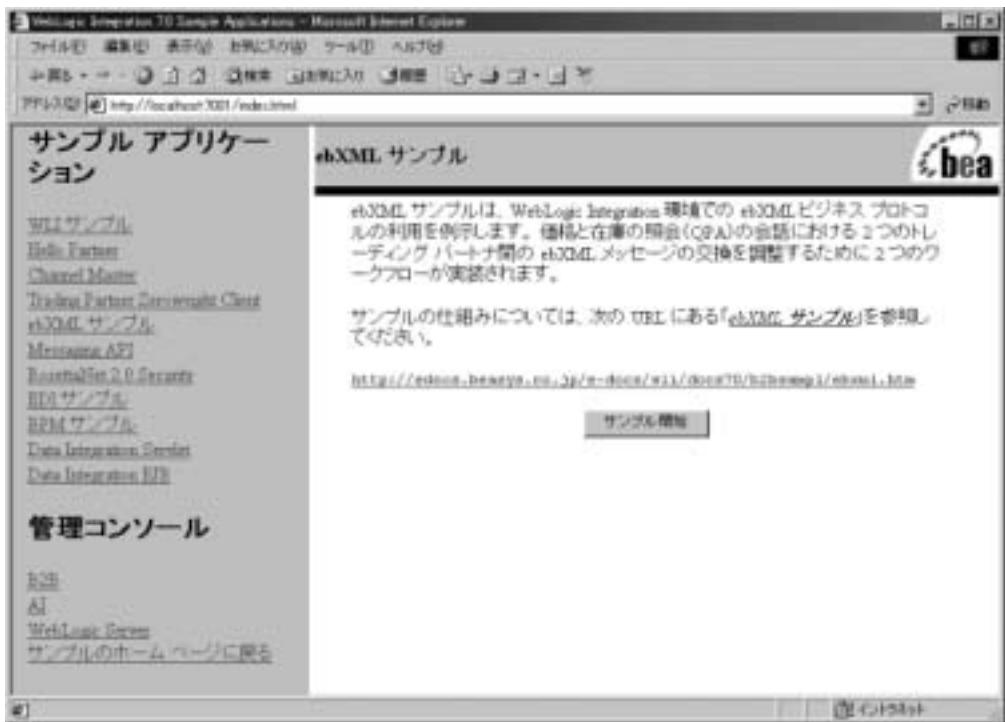
これで、**RunSamples** スクリプトは **WebLogic Server** のインスタンスをバックグラウンドプロセスとして開始し、サンプル起動ページが表示されます。

図 7-2 サンプル起動ページ



5. [ebXML サンプル] リンクをクリックすると、サンプル起動ページの左ペインの [サンプルアプリケーション] の下にリストが表示されます。右ペインに ebXML サンプルが表示されます。

図 7-3 ebXML サンプル起動ページ



6. このフォームの [Product ID]、[Product Unit Price]、および [Product Quantity] の各フィールドに値を入力して、[Submit] をクリックします。

ebXML ベースの QPA メッセージがサプライヤ トレーディング パートナに送信されます。

7. サプライヤ トレーディング パートナは、QPA メッセージを処理して、応答 ebXML メッセージをバイヤ トレーディング パートナに送信します。

応答はバイヤの Web ブラウザに表示されます。

図 7-4 応答メッセージ



8. このとき、さらに多くの **WebLogic Integration** サンプルを実行する場合は、サンプル起動ページを開いた状態で、**WebLogic Server** のインスタンスの実行を続行します。

この時点で、これ以上のサンプルを実行しない場合は、ブラウザを終了し、プラットフォームに合わせた適切な手順によって **WebLogic Server** のインスタンスをシャットダウンします。

- **Windows:**

```
cd %SAMPLES_HOME%\integration\config\samples
stopWebLogic
```

- **UNIX:**

```
cd $SAMPLES_HOME/integration/config/samples
stopWebLogic
```

## サンプルの仕組み

この節では、以下のトピックを取り上げます。

- 概要
- リポジトリ データのロード
- リポジトリ データについて

- ワークフローについて

## 概要

このサンプルアプリケーションをサポートするのに必要なデータは、サンプルの設定（7-3 ページの「ebXML サンプルの実行」参照）時に `RunSamples` スクリプトを実行したときに **WebLogic Integration** リポジトリにバルク ロードされます。**WebLogic Integration** では、コンフィグレーション データをバルク ロードすることも、**WebLogic Integration B2B Console** を使用して入力することもできます。**ebXML** サンプルを実行する場合に **B2B Console** を実行する必要はありませんが、実行すると、サンプル用にバルク ロードされたリポジトリ データを表示できます。

このサンプルで使用する 2 つのワークフローは、QPA 会話における ebXML ベースのビジネス メッセージの交換を管理します。これらのワークフローは、**ebXML** メッセージの送信側および受信側を管理します。

**B2B** 統合環境のコンフィグレーション方法を詳しく説明することは、このマニュアルの範囲を越えますが、この節では、このサンプルアプリケーションで使用されている **WebLogic Integration** リポジトリ データについて簡単に説明します。この節ではまた、サンプルの **ebXML** ビジネス トランザクションの送信側および受信側でのワークフローの実装についても説明します。ワークフローの主要な設計要素、タスク、およびイベントは、特に詳しく説明されています。

## リポジトリ データのロード

トレーディング パートナを統合するためにサンプルに必要なデータは、サンプルの設定（7-3 ページの「ebXML サンプルの実行」参照）時に `RunSamples` スクリプトを実行したときに、**WebLogic Integration** リポジトリにバルク ロードされます。

`RunSamples` スクリプトは、以下の XML ファイルに格納された **B2B** コンフィグレーション データをリポジトリにロードします。

- `SystemRepData.xml` – **WebLogic Integration** のインストール ディレクトリ の `integration\dbscripts` ディレクトリにあります。次に例を示します。

```
c:\bea\weblogic700\integration\dbscripts
```

SystemRepData.xml ファイルには、システム データが格納されています。  
このサンプルで使用する要素は、次のとおりです。

- ビジネス プロトコル定義
- ロジック プラグイン
- BulkLoaderData.xml – `SAMPLES_HOME\integration\samples\ebxml\lib` ディレクトリにあります。`SAMPLES_HOME` は WebLogic Platform のサンプル ディレクトリを表しています。

この BulkLoaderData.xml ファイルには、ebXML サンプルに固有のデータが格納されています。このファイルでは、次の要素を記述します。

- トレーディング パートナ
- 会話定義
- コラボレーション アグリーメント

## リポジトリ データについて

この節では、ebXML サンプルアプリケーション用として WebLogic Integration リポジトリにバルク ロードされるデータ要素に関して特に重要な情報を取り上げます。

- ビジネスプロトコル定義
- ロジック プラグイン
- トレーディング パートナ
- 会話定義
- コラボレーション アグリーメント

SystemRepData.xml ファイルおよび BulkLoaderData.xml ファイルのデータは、サンプル アプリケーションをサポートするため、WebLogic Integration リポジトリにインポートされます。ebXML アプリケーションをコンフィグレーションする際、コンフィグレーションデータは、バルク ロードすることも、WebLogic Integration B2B Console を使用して入力することもできます。また、B2B Console を使用して、バルク ロードしたデータにアクセスしたり、コンフィグレーションしたりすることができます。E ビジネストランザクションに必要な WebLogic Integration データのコンフィグレーションの詳細については、『*B2B Integration ebXML の実装*』の「ebXML の管理」を参照してください。

## ビジネス プロトコル定義

SystemRepData.xml ファイルには、ebXML をはじめとする、WebLogic Integration がサポートするすべてのビジネスプロトコルの定義が格納されています。SystemRepData.xml ファイルの以下の抜粋に、ebXML ビジネスプロトコル定義が示されています。

### コード リスト 7-1 ebXML ビジネス プロトコル 定義

```
<business-protocol-definition
  name="ebXML"
  business-protocol-name="ebXML"
  protocol-version="1.0"
  endpoint-type="PEER">
```

```

        <java-class>com.bea.b2b.protocol.ebxml.EBXMLProtocol
    </java-class>

    <decoder>EBXML-Decoder</decoder>
        <encoder>EBXML-Encoder</encoder>
</business-protocol-definition>

```

---

## ロジック プラグイン

ロジック プラグインは、実行時にビジネス メッセージをインターセプトして処理する Java クラスです。各ビジネス プロトコルは、標準のルータおよびフィルタ ロジック プラグインに関連付けられています。SystemRepData.xml ファイルには、XOCP、RosettaNet、cXML の各ビジネス プロトコル用のロジック プラグインが格納されています。このサンプルでは、次の ebXML ロジック プラグインのみを使用します。

- ebXML エンコーダーメッセージを B2B 転送サービスに転送する。
- ebXML デコーダー ebXML ヘッダーを処理し、送信側トレーディング パートナを識別し、送信側トレーディング パートナを会話に参加させ、送信元への応答を準備し、メッセージを B2B スケジューリング サービスに転送する。

SystemRepData.xml ファイルの以下の抜粋には、ebXML エンコーダおよびデコーダのロジック プラグインが示されています。

### コード リスト 7-2 ebXML ロジック プラグイン定義

```

<logic-plugin
    name="EBXML-Decoder"
    type="decoder">
    <java-class>com.bea.b2b.protocol.ebxml.EBXMLDecoder
    </java-class>
</logic-plugin>
<logic-plugin
    name="EBXML-Encoder"
    type="encoder">
    <java-class>com.bea.b2b.protocol.ebxml.EBXMLEncoder
    </java-class>
</logic-plugin>

```

---

## トレーディング パートナ

ebXML サンプルでは、バイヤとサプライヤという 2 つのビジネス パートナが設定されています。WebLogic Integration 環境では、各ビジネス パートナごとにトレーディング パートナをコンフィグレーションする必要があります。この場合、BulkLoaderData.xml ファイルで ebXML - sender と ebXML - receiver という 2 つのトレーディング パートナがコンフィグレーションされています。コンフィグレーション要素、属性、およびこれらのトレーディング パートナに関連付けられた下位要素については、以下のファイルを参照してください。

```
SAMPLES_HOME\samples\ebxml\lib\BulkLoaderData.xml
```

`SAMPLES_HOME` は、WebLogic Platform のサンプル ディレクトリを表しています。

## 会話定義

BulkLoaderData.xml ファイルには、ebXML ベースの Query Price and Availability (QPA) 会話の会話定義が格納されています。その会話定義の名前は、ebxmlQPA です。この定義には、開始者と参加者という、2 つのロールが含まれています。ebXML ビジネス プロトコルでは、すべての会話定義は、開始者と参加者という定義済みの名前を持つ 2 つのロールを参照します。

次のリストは、BulkLoaderData.xml ファイルの抜粋です。ここでは、ebxmlQPA 会話を定義しています。

### コード リスト 7-3 BulkLoaderData.xml ファイルの会話定義

---

```
<conversation-definition
  name="ebxmlQPA"
  version="1.0"
  business-protocol-name="ebXML"
  protocol-version="1.0">
  <role
    name="Initiator">
  </role>
  <role
    name="Participant">
  </role>
</conversation-definition>
```

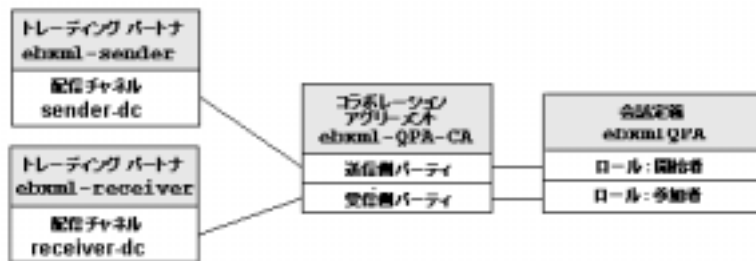
---



## コラボレーション アグリーメント

BulkLoaderData.xml ファイルには、このサンプル用の ebxml-sender トレーディング パートナおよび ebxml-receiver トレーディング パートナが使用するコラボレーション アグリーメントが格納されています。次の図は、このサンプルアプリケーションでのトレーディング パートナ間の関係、コラボレーション アグリーメントにかかわるパーティ、会話に対して定義されているロールを示しています。

図 7-5 QPA 会話のトレーディング パートナ間のコラボレーション アグリーメント



次のリストは、BulkLoaderData.xml ファイルの抜粋です。ebxml-QPA-CA コラボレーション アグリーメントの要素を示しています。

### コード リスト 7-4 BulkLoaderData.xml ファイルのコラボレーション アグリーメント

```
<collaboration-agreement
  name="ebxml-QPA-CA"
  version="1.0"
  status="ENABLED"
  global-identifier="sachin/172.16.15.113:2423d2:
    ea8fe66b8f:-8000"
  conversation-definition-name="ebxmlQPA"
  conversation-definition-version="1.0">
  <party
    trading-partner-name="ebxml-sender"
    party-identifier-name="sender-party"
    delivery-channel-name="sender-dc"
    role-name="Initiator"/>
  <party
    trading-partner-name="ebxml-receiver"
    party-identifier-name="receiver-party"
```

```
delivery-channel-name="receiver-dc"  
role-name="Participant" />  
</collaboration-agreement>
```

---

## ワークフローについて

このサンプル QPA 会話では、2つのワークフローが、開始者と参加者のロールで2つのトレーディング パートナによる ebXML メッセージの交換を管理します。これらのワークフロー (ebXMLConversationInitiator および ebXMLConversationResponder) は、サンプルの設定 (7-3 ページの「ebXML サンプルの実行」参照) 時に RunSamples スクリプトを実行したときに、WebLogic Integration リポジトリにバルク ロードされます。

この節では、以下のトピックを取り上げます。

- WebLogic Integration Studio の使い方
- ebXMLConversationInitiator ワークフローについて
- ebXMLConversationResponder ワークフローについて

## WebLogic Integration Studio の使い方

WebLogic Integration Studio を使用すると、新しいワークフローを設計したり、見慣れたフローチャートを使用して進行中のワークフローをモニタしたりすることができます。ebXML サンプルを実行する場合に Studio を実行する必要はありませんが、ワークフローまたはワークフロー ノードの詳細を表示したり、このサンプルに関するノードの定義およびコンフィギュレーションを調べる場合には Studio が役立ちます。

Studio を起動するには、プラットフォームに合わせて適切な手順を実行します。

- Windows システム上で Studio を実行するには、次のいずれかを実行します。
  - メニューを使用する方法は次のとおりです。

[スタート | プログラム | BEA WebLogic Platform 7.0 | WebLogic Integration 7.0 | Studio] を選択します。

b. Studio にログオンします (ユーザ名:wlpisystem、パスワード:wlpisystem)。

- 次のように、コマンドラインから Studio スクリプトを起動します。
  - a. コマンド ウィンドウを開きます。
  - b. **WebLogic Integration** ディレクトリ (**WebLogic Integration** をインストールしたディレクトリ) に移動し、**setenv** スクリプトを実行して、最上位の **WebLogic** 環境変数を設定します。たとえば、**WebLogic Integration** が `c:\bea` ディレクトリにインストールされている場合は、以下のように入力します。

```
cd c:\bea\weblogic700\integration
setEnv
```

- c. **WebLogic Integration** インストールディレクトリの下 `bin` ディレクトリに移動します。

```
cd bin
```

- d. 次のように入力して **Studio** を開きます。

```
studio
```

- e. ユーザ名およびパスワードとして、`wlpisystem` を使用して **Studio** にログオンします。

- UNIX システム上で **Studio** を起動するには、次のタスクを実行します。

- a. **WebLogic Integration** をインストールしたディレクトリに移動し、**setenv** スクリプトを実行して、最上位の **WebLogic** 環境変数を設定します。たとえば、**WebLogic Integration** が `/home/me` ディレクトリにインストールされている場合は、以下のように入力します。

```
cd /home/me/bea/integration
. ./setenv.sh
```

- b. **WebLogic Integration** インストールディレクトリの下 `bin` ディレクトリに移動します。

```
cd bin
```

- c. 次のように入力して **Studio** アプリケーションを起動します。

```
./studio
```

- d. ユーザ名およびパスワードとして、`wlpisystem` を使用して **Studio** にログオンします。

**Studio** を起動し、以下の手順を実行することで、ワークフロー テンプレートおよびそのプロパティを表示できます。

1. Studio の左ペインで、**ORG1** が [ オーガニゼーション ] フィールドで選択されていることを確認します。
2. 左ペインで、[ テンプレート ] フォルダをダブルクリックしてワークフローテンプレートのリストを表示します。
3. [ テンプレート ] フォルダを展開して、ワークフロー テンプレートの定義リストを表示します。このサンプル アプリケーションに関連するテンプレート定義は ebXMLConversationInitiator および ebXMLConversationResponder です。これらの定義は、サンプルをコンフィグレーションするときに workflow.jar ファイルでインポートされます。7-3 ページの「ebXML サンプルの実行」を参照してください。
4. テンプレート 定義を右クリックして [ 開く ] を選択すると、ワークフロー テンプレートが Studio に表示されます。

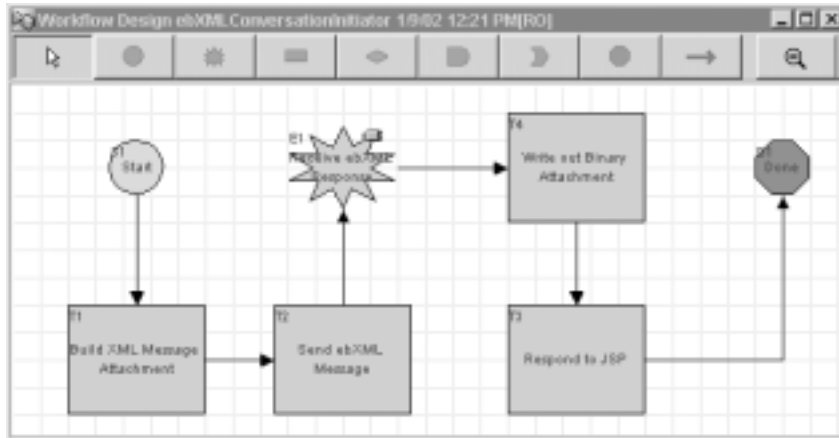
**注意：** また、特定のワークフロー テンプレート 定義を展開すると、そのワークフロー テンプレート 定義の [ タスク ]、[ 分岐 ]、[ イベント ]、[ 結合 ]、[ 開始 ]、[ 完了 ]、および [ 変数 ] を格納するフォルダが表示されます。
5. Studio でノードをダブルクリックすると、そのノードの [ プロパティ ] ダイアログ ボックスが表示されます。

Studio のツールおよび機能の詳細については、『*WebLogic Integration Studio ユーザーズガイド*』を参照してください。

## ebXMLConversationInitiator ワークフローについて

この節では、ebXML サンプル シナリオにおける QPA 会話のバイヤ側を管理するワークフローについて説明します。次の図は、Studio での ebXMLConversationInitiator ワークフロー テンプレートを示しています。

図 7-6 ebXMLConversationInitiator ワークフロー テンプレート



以下の節では、上の図に示した ebXMLConversationInitiator ワークフロー テンプレートの以下のノードの主要なタスクおよびイベントについて定義します。

- 開始
- Build XML Message Attachment
- ebXML メッセージの送信
- Receive ebXML Response
- Write Out Binary Attachment
- Respond to JSP
- 完了

## 開始

サンプル実行時に、QPA 要求が、HTML Web フォームの入力（図 7-3 参照）に基づいて、XML として作成されます。XML が、JSP（ebXML\_Sample.jsp）を使用して、JMS キューに送信されます。

ebXML\_Sample.jsp ファイルは、次のディレクトリ（SAMPLES\_HOME は WebLogic Platform のサンプルディレクトリ）にあります。

```
SAMPLES_HOME\integration\config\samples\applications\DefaultWebApp
p_myserver
```

開始ノードは、JMS キューから XML イベントを受信したときに、この ebXMLConversationInitiator ワークフローを開始する設計になっています。次の図に、開始ノードの [開始のプロパティ] ダイアログ ボックスを示します。

図 7-7 [開始のプロパティ] ダイアログ ボックス



この図の以下のプロパティ設定に注意してください。

- 開始メソッドとして、[ イベント | XML イベント ] が選択されている。この開始ノードを [ イベント | XML イベント ] オプションを指定して設計されているため、XML ドキュメントが JMS キューに到着した時点でワークフローが開始することを保証できる。

**注意：** この開始メソッドを、ebXMLConversationResponder ワークフローの開始ノードに対して定義されている開始メソッド（7-26 ページの「開始」参照）と対比してください。

- [ドキュメントタイプ/ルート要素] フィールドの QPARoot は、このワークフローをトリガする XML メッセージのルート要素である。
- [変数] タブには、着信イベント データから初期化される変数が格納されている。  
開始ノードは、XPath 式を使用しているメッセージを抽出し、データをワークフロー変数 (ProductID、ProductQuantity、および ProductUnitPrice) に格納します。

## Build XML Message Attachment

このタスク ノードでは、outXMLAttach ワークフロー変数が設定されます。この変数には、ebXML メッセージ ペイロードの一環として、サプライヤのワークフロー (ebXMLConversationResponder) に送信される QPA 要求 XML メッセージ (ebXMLQPAREquest) が保持されています (詳細は、このワークフローの次のノードを参照)。

## ebXML メッセージの送信

このタスク ノードで定義されているアクションが、QPA 会話の最初の ebXML メッセージを送信します。ebXMLConversationInitiator ワークフローは、ebXML ベースの会話用に設計されているので、このタスク ノードでは ebXML メッセージのアクション (BPM への ebXML プラグインが提供) を使用します。

Studio で Send ebXML Message アクションを定義するには、次の手順に従います。

1. タスク ノードをダブルクリックして、[タスクのプロパティ] ダイアログ ボックスを表示します。
2. [アクション | 追加 | ebXML アクション | ebXML メッセージの送信] を選択します。  
[ebXML メッセージの送信] ダイアログ ボックスが表示されます。

このサンプル ノードに対して指定されている [ebXML メッセージの送信] プロパティを表示するには、次の手順に従います。

1. タスク ノードをダブルクリックして、[タスクのプロパティ] ダイアログ ボックスを表示します。
2. [アクション | アクティブ時] を選択します。

3. [ebXML メッセージの送信] をダブルクリックして、次の図に示す [ebXML メッセージの送信] ダイアログ ボックスを表示します。

図 7-8 [ebXML メッセージの送信] ダイアログ ボックス



この図の以下の [ebXML メッセージの送信] プロパティに注意してください。

- [新規または関連する会話] - [新規の会話] または [関連する会話] のいずれかを選択する必要があります。この場合、このアクションは新規の会話を開始するための ebXML メッセージを送信する設計となっているため [新規の会話] が選択されています。[関連する会話] オプションの使用法の詳細については、7-28 ページの「ebXML 応答の送信」を参照してください。
- [会話名] - [新規の会話] オプションを選択した後、メッセージが送信される会話を選択する必要があります。この場合、ebxmlQPA 会話を選択されています。ebxmlQPA 会話定義は、このサンプルのコンフィグレーション時 (7-12 ページの「会話定義」参照) に WebLogic Integration リポジトリにロードされます。
- [送信側ビジネス ID] - [新規の会話] オプションを選択した後、メッセージの送信者のビジネス ID を選択する必要があります。この場合、値 (ebxml-sender-id) は静的ですが、Expression Builder を使用して動的に評価される値とすることもできます。  
**注意：** このフィールドの値は、引用符で囲む必要があります。
- [受信側ビジネス ID] - [新規の会話] オプションを選択した後、メッセージの受信者のビジネス ID を選択する必要があります。この場合、値



(`ebxml-receipient-id`) は静的ですが、**Expression Builder** を使用して動的に評価される値にすることもできます。

**注意：** このフィールドの値は、引用符で囲む必要があります。

- [メッセージペイロード]-各メッセージ添付ファイルの型(XML またはバイナリ) および関連付けられている値を指定します。ゼロ以上のペイロードに対するエントリを入力できます。この場合、添付ファイルは1つ、型は **XML**、値は上のノード (**Build XML Message Attachment**) で設定された `outXMLAttach` 変数です。

メッセージは、このノードでは非同期的に送信されます。すなわち、[タスクのプロパティ] ダイアログ ボックスの [アクション | アクティブ時] タブでは、以下のアクションは記載の順序で指定されています。

1. Send ebXML Message
2. Mark task \Send ebXML Message\ done

## Receive ebXML Response

このイベント ノードのワークフローでは、`ebXMLConversationResponder` ワークフローからの指定した **ebXML** イベントを待機します。次の図に示す [イベントのプロパティ] ダイアログ ボックスでは、[**ebXML イベント**] が選択されています。

図 7-9 [ イベントのプロパティ ] ダイアログ ボックス



ebXML イベントが受信されると、メッセージエンベロープは、エンベロープ変数（この場合は inEnvelope）に格納され、ペイロードは、XML 変数またはバイナリ変数のいずれか適切な方に格納されます。この場合、ペイロードには、添付ファイルが2つ（XML とバイナリの添付ファイルが1つずつ）あります。それらは、それぞれ、inXMLAttach 変数、inBinAttach 変数に割り当てられています。

## Write Out Binary Attachment

このタスク ノードでは、着信 QPA 応答 XML ドキュメント (ebXMLQPAREsponse) からイメージファイルの名前を抽出する **Set Workflow Variable** アクションを最初に定義し、その名前を imageFileName ワークフロー変数に割り当てます。**Set Workflow Variable** アクションは、次の XPath 式を使用してこれらのタスクを実行します。

```
XPath("/ebXMLQPAREsponse/ImageFileName/text()", $inXMLAttach))
```

以降は、ビジネスオペレーション (ebXMLSavePictureToWebApp) を使用するアクションがこのノードで定義されます。このアクションは、着信 ebXML メッセージのバイナリ添付ファイルを次のディレクトリに保存します (SAMPLES\_HOME は WebLogic Platform サンプルディレクトリを表す)。

```
SAMPLES_HOME\integration\config\samples\applications\DefaultWebApp_p_myserver\
```

このローカル ファイルは **JSP** によって処理されます（この節で次に説明する **Respond to JSP** ノードを参照）。

このサンプルのビジネス オペレーションを確認するには、**Studio** のタスク メニューから [ **コンフィグレーション | ビジネス オペレーション** ] を選択します。[ **ビジネス オペレーション** ] ダイアログ ボックスにビジネス オペレーションのリストが表示されます。ビジネス オペレーションをダブルクリックすると、詳細を確認できます。ebXMLSavePictureToWebApp ビジネス オペレーションの **Java** クラスは、次の場所にあります（*SAMPLES\_HOME* は **WebLogic Platform** のサンプル ディレクトリを表す）。

```
SAMPLES_HOME\integration\samples\ebxml\src\ebxmlsamples\util\EBXMLBizOp.java
```

## Respond to JSP

このタスク ノードでは、**Post XML Event** アクションを定義します。このイベントをノードに対して定義するには、ノードをダブルクリックして [ **タスクのプロパティ** ] ダイアログ ボックスを表示し、[ **追加 | 統合アクション | XML イベントをポスト** ] を選択します。

このアクションは、応答メッセージからの **XML** メッセージ（前のノードで定義された `inXMLAttach` 変数）を内部 **XML** イベントとしてポストします。この **XML** イベントのポスト先である **JMS** キューは、[ **XML イベントをポスト** ] ダイアログ ボックスにある [ **送り先** ] タブで定義されます。

このノードの [ **XML イベントをポスト** ] ダイアログ ボックスを次の図に示します。

図 7-10 [XML イベントをポスト] ダイアログ ボックスの [送り先] タブ



上の図の [送り先] で、[JMS キュー] が選択されている点に注意してください。これは、メッセージが **WebLogic Server** でコンフィグレーションされた内部キューにポストされることを意味しています。引用符で囲まれたキューに対して、静的 JNDI 名または実行時にキュー名を決定する式を入力できます。この場合、内部 JMS キューの JNDI 名は、`com.bea.wlpi.EventQueueExt` です。

JMS キューからの XML メッセージは、`ebXML_Sample.jsp` ファイルによって使用されます。これは、図 7-4 に示したように、サンプル実行時に、QPA 応答メッセージの内容をブラウザに表示するファイルです。

`ebXML_Sample.jsp` ファイルは、次のディレクトリ (`SAMPLES_HOME` は **WebLogic Platform** のサンプルディレクトリ) にあります。

```
SAMPLES_HOME\integration\config\samples\applications\DefaultWebApp_myserver
```

## 完了

ワークフローの終了を示します。**ebXML** ベースの会話は、両方のトレーディング パートナにとって **ebXML** メッセージの交換が完了した時点で終了します。

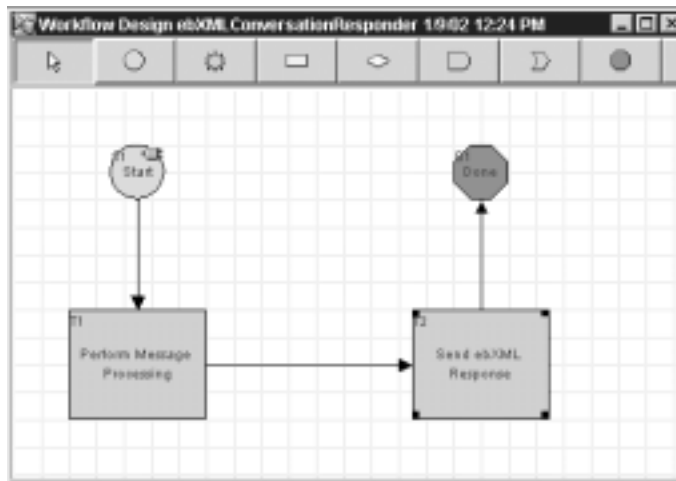
この動作を、XOCP ビジネス プロトコルの場合と比較してください。WebLogic Integration では、XOCP 会話管理サービスをサポートしています。すなわち、会話を開始するワークフローは、会話の終了および会話に参加している各ワークフローへの会話終了メッセージの送信も担当します（『B2B Integration ワークフローの作成』の「協調的ワークフローの終了」を参照）。

XOCP 会話の場合は、会話終了プロパティを定義するには、会話を開始したワークフローの完了ノードで、[カスタム] オプションを選択しますが、ebXML ベースの会話はこの方法では終了しません。たとえば、この完了ノードに対して [カスタム] オプションは選択されていません。

## ebXMLConversationResponder ワークフローについて

この節では、この ebXML サンプル シナリオにおける QPA 会話のサプライヤ側を管理するワークフローについて説明します。次の図は、Studio での ebXMLConversationResponder ワークフロー テンプレートを示しています。

図 7-11 ebXMLConversationResponder ワークフロー テンプレート



以下の節では、上の図に示した ebXMLConversationResponder ワークフロー テンプレートの以下のノードの主要なタスクおよびイベントについて定義します。

- 開始
- Perform Message Processing

- ebXML 応答の送信
- 完了

## 開始

この ebXMLConversationResponder ワークフローは、ebXML ベースの会話での参加者ロールを担当するトレーディング パートナ用に設計されています。このワークフローは、トレーディング パートナから ebXML メッセージを受信した時点で開始します。Studio では、ebXML 会話における参加者ロールのワークフローの設計をサポートするため、開始ノードに ebXML 固有のオプションが用意されています。次の図に [開始のプロパティ] ダイアログ ボックスを示します。

図 7-12 [開始のプロパティ] ダイアログ ボックス



この図の以下のプロパティ設定に注意してください。

- ebXML ビジネス プロトコル固有の開始メソッドが選択されている。[ イベント ] オプションが選択され、[ イベント ] として、[ ebXML メッセージ ] が指定されている。この開始ノードを [ ebXML メッセージ ] を指定して設計されているため、ワークフローは ebXML メッセージをトレーディング パートナから受信した時点で開始する。

**注意：** この開始メソッドを、ebXMLConversationInitiator ワークフローの開始ノードに対して定義されている開始メソッド（7-17 ページの「開始」参照）と対比してください。

- [ 会話名 ] フィールドでは、ebxmlQPA が指定されている。この会話では、ワークフローは、ebXML メッセージの受信時に開始する。
- [ エンベロープ変数 ] フィールドでは、inEnvelope が指定されている。ebXML メッセージが受信されると、メッセージ エンベロープはこのワークフロー変数に格納される。
- 各メッセージ添付ファイルの型（XML またはバイナリ）および関連付けられている値を指定する。ゼロ以上のペイロードに対するエントリを入力できる。この場合、添付ファイルは 1 つで、型は XML、関連付けられた変数は inXMLAttach である。

## Perform Message Processing

このタスク ノードは、着信 ebXML メッセージを処理します。ワークフロー変数を設定するアクションおよびビジネス オペレーションを実行するアクションが定義されています。

XPath 式により、製品 ID、数量、単価が着信 QPA ebXML メッセージから抽出され、ProductID、ProductQuantity、および ProductUnitPrice の各ワークフロー変数に対して日付が割り当てられます。

このノードでは以下のビジネス オペレーションが実行されます。

1. ebXMLGetQPAREply – このノードで、製品 ID、数量、単価に対して抽出された値を使用して XML QPA 応答ドキュメントを作成します。応答ドキュメントは、XML 変数 outXMLAttach に割り当てられます。
2. ebXMLGetPictureForProductID – 入力として製品 ID を受け取り、指定された製品のピクチャの入ったイメージ ファイルのファイル名を返します。
3. file to binary – ebXMLGetPictureForProductID ビジネス オペレーションから返されたファイルの場所にあるイメージ ファイル データからバイナリ データを読み出します。読み出されたデータは、outBinAttach 変数に割り当てられます。

このサンプルのビジネス オペレーションを確認するには、**Studio** のタスク メニューから [ **コンフィグレーション | ビジネス オペレーション** ] を選択します。[ **ビジネス オペレーション** ] ダイアログ ボックスが表示されます。ビジネス オペレーションのリストが表示されます。ビジネス オペレーションをダブルクリックすると、詳細が確認できます。

ebXMLGetQPAREply および ebXMLGetPictureForProductID ビジネス オペレーションの Java クラスは、次の場所にあります (`SAMPLES_HOME` は **WebLogic Platform** のサンプルディレクトリを表す)。

```
SAMPLES_HOME\samples\integration\samples\ebxml\src\ebxmlsamples\util\EBXMLBizOp.java
```

file to binary ビジネス オペレーションの Java クラスは、次の場所にありません。 `SAMPLES_HOME` は **WebLogic Platform** のサンプルディレクトリを表します。

```
SAMPLES_HOME\samples\integration\samples\wlis\src\examples\wlis\common\util\Utils.java
```

### ebXML 応答の送信

このタスク ノードでは、このトレーディング パートナが **QPA** 会話で受信した **ebXML** メッセージに応答する **ebXML** メッセージを送信するアクションが定義されています。 `ebXMLConversationResponder` ワークフローは、 **ebXML** ベースの会話用に設計されているので、このタスク ノードでは **ebXML** メッセージのアクション (**BPM** への **ebXML** プラグインが提供) を使用します。

**Studio** で [ **ebXML メッセージの送信** ] アクションを定義するには、次の手順に従います。

1. タスク ノードをダブルクリックして、[ **タスクのプロパティ** ] ダイアログ ボックスを表示します。
2. [ **アクション | 追加 | ebXML アクション | ebXML メッセージの送信** ] を選択して [ **ebXML メッセージの送信** ] ダイアログ ボックスを表示します。

このサンプル ノードに対して指定されている [ **ebXML メッセージの送信** ] プロパティを表示するには、次の手順に従います。

1. タスク ノードをダブルクリックして、[ **タスクのプロパティ** ] ダイアログ ボックスを表示します。
2. [ **アクション | アクティブ時** ] を選択します。



3. [ebXML メッセージの送信] をダブルクリックして、次の図に示す [ebXML メッセージの送信] ダイアログ ボックスを表示します。

図 7-13 [ebXML メッセージの送信] ダイアログ ボックス



上の図で、ebXML サンプルのタスク ノードの以下のプロパティに注目してください。

- [新規または関連する会話] – [新規の会話] または [関連する会話] のいずれかを選択する必要があります。この場合、このアクションはトレーディング パートナが受信したメッセージに応答して ebXML メッセージを送信するように設計されているため [関連する会話] が選択されています。

[関連する会話] が選択されている場合、会話名、送信者のビジネス ID、または受信者のビジネス ID などの属性は、システムによって前のメッセージ交換から取得されるため、それらを指定する必要はありません。[新規の会話] オプションの使用法の詳細については、7-19 ページの「**ebXML メッセージの送信**」を参照してください。

- [メッセージペイロード] – 各メッセージ添付ファイルの型 (XML またはバイナリ) および関連付けられている値を指定します。ゼロ以上のペイロードに対するエントリを入力できます。この場合、添付ファイルは 2 つあります。1 つ目の添付ファイルは XML ファイルで、関連付けられた変数は outXMLAttach です。2 つめの添付ファイルはバイナリ ファイルで、関連付けられた変数は outBinAttach です。データは前のノードで両方の変数に割り当てられます (Perform Message Processing 参照)。

## 完了

ワークフローの終了を示します。ebXML メッセージを交換するワークフローにおける完了ノードの詳細については、7-24 ページの「完了」の ebXMLConversationInitiator ワークフロー用の完了ノードについての説明を参照してください。

---

# A JSP タグ リファレンス

Java Server Pages (JSP) タグ ライブラリは、トレーディング パートナ ゼロウェイ トクライアント用に提供されています。WebLogic Integration Messaging API の指定にはラッパーを使用します。エラー処理には、Java の標準の例外処理およびエラー処理 (WebLogic Integration Messaging API クラス経由) と、ゼロウェイ トクライアントに配信された JSP ページを使用します。

この章では、以下の JSP タグのリファレンス情報を提供します。

- SendmsgTag
- ChecknewmsgTag
- CheckallmsgTag
- ReadmsgTag
- DeletemsgTag
- DeleteallmsgTag
- CreatemboxTag
- RemovemboxTag

**注意：** WebLogic Integration Messaging API および JSP タグ ライブラリは、このリリースの WebLogic Integration から非推奨になっています。WebLogic Integration Messaging API および JSP タグ ライブラリに代わる機能については、『*WebLogic Integration リリース ノート*』を参照してください。

## SendmsgTag

ビジネス メッセージを送信メール用のメールボックスに渡します。信頼性を高めるため、メッセージの永続性を保証します。

**構文** `SendMsgTag (String mboxName, String sender, String message, String URL, String security)`

**戻り値** メッセージが、`SendMsgTag` が埋め込まれた **JSP** ページに正常に送信された場合は、`Message has been sent successfully` と返します。

### 変数

変数	説明
<code>String mboxName</code>	メールボックスの名前。サンプルを実行するには、適切なメールボックスに以下の名前を使用する。 <ul style="list-style-type: none"><li>■ 受信メール用：<code>trading_partner_name_Inbox</code></li><li>■ 送信メール用：<code>trading_partner_name_Outbox</code></li></ul>
<code>String message</code>	メッセージの内容。
<code>String sender</code>	送信側のメールボックス アドレス。電子メール アドレスまたは <b>FTP</b> サーバ アドレスを指定する場合もある。
<code>String URL</code>	トレーディング パートナをホスティングする <b>URL</b> 。
<code>String security</code>	値は <b>ON</b> または <b>OFF</b> 。

**例**

```
sendmsg mboxname="<%=outboxName_browserTP1%>"  
sender="<%=SENDER%>" message="<%=domAsStr%>" url="<%=url%>"  
security="ON"/>
```

---

## ChecknewmsgTag

受信メールおよび送信メールのメールボックスに、新しいメッセージが入っているかどうかを確認します。格納されているメッセージは確認されません（「CheckallmsgTag」を参照）。

**構文**      ChecknewmsgTag (String mboxName)

**戻り値**      メールボックスが空の場合は、No new message found in mailbox と返します。メールボックスに 1 つ以上の新しいメッセージが入っていた場合は、それらのメッセージが **HTML** フォーマットで表示されます。

**変数**

---

変数	説明
mboxName	メールボックスの名前。サンプルを実行するには、適切なメールボックスに以下の名前を使用する。 <ul style="list-style-type: none"><li>■ 受信メール用: <i>trading_partner_name_Inbox</i></li><li>■ 送信メール用: <i>trading_partner_name_Outbox</i></li></ul>

---

### CheckallmsgTag

格納されているメッセージも含め、メールボックス内のすべてのメッセージを確認します。

**構文**      `CheckallmsgTag (String mboxName)`

**戻り値**      メールボックスにメッセージが入っていた場合は、それらが **HTML** フォーマットで表示されます。メールボックスが空の場合は、`No message found in mailbox` と返します。

**変数**

変数	説明
<code>mboxName</code>	メールボックスの名前。サンプルを実行するには、適切なメールボックスに以下の名前を使用する。 <ul style="list-style-type: none"><li>■ 受信メール用: <code>trading_partner_name_Inbox</code></li><li>■ 送信メール用: <code>trading_partner_name_Outbox</code></li></ul>

---

## ReadmsgTag

メールボックスから特定のメッセージの詳細を取得します。

**構文** `ReadmsgTag (String mboxName, String msgId)`

**戻り値** メッセージの詳細が、HTML フォーマットで表示されます。

**変数**

変数	説明
mboxName	メールボックスの名前。サンプルを実行するには、適切なメールボックスに以下の名前を使用する。 <ul style="list-style-type: none"><li>■ 受信メール用：<code>trading_partner_name_Inbox</code></li><li>■ 送信メール用：<code>trading_partner_name_Outbox</code></li></ul>
msgID	ユニークなメッセージ ID。

### DeletemsgTag

メールボックスから特定のメッセージを削除します。

**構文** DeletemsgTag (String mboxName, String msgID)

**戻り値** Message with messageID msgID deleted successfully を返します。

**変数**

変数	説明
mboxName	メールボックスの名前。サンプルを実行するには、適切なメールボックスに以下の名前を使用する。 <ul style="list-style-type: none"><li>■ 受信メール用: <code>trading_partner_name_Inbox</code></li><li>■ 送信メール用: <code>trading_partner_name_Outbox</code></li></ul>
msgID	ユニークなメッセージ ID。



---

## DeleteallmsgTag

メールボックスからすべてのメッセージを削除します。

**構文** DeleteallmsgTag (String mboxName)

**戻り値** 正常に終了すると、All messages were deleted successfully と返します。

**変数**

変数	説明
mboxName	メールボックスの名前。サンプルを実行するには、適切なメールボックスに以下の名前を使用する。 <ul style="list-style-type: none"><li>■ 受信メール用 : <i>trading_partner_name_Inbox</i></li><li>■ 送信メール用 : <i>trading_partner_name_Outbox</i></li></ul>

## CreatemboxTag

メールボックスを作成します。

構文 `CreatemboxTag (String mboxName)`

戻り値 なし。

変数

変数	説明
<code>mboxName</code>	メールボックスの名前。サンプルを実行するには、適切なメールボックスに以下の名前を使用する。 <ul style="list-style-type: none"><li>■ 受信メール用: <code>trading_partner_name_Inbox</code></li><li>■ 送信メール用: <code>trading_partner_name_Outbox</code></li></ul>

---

## RemovemboxTag

特定のメールボックスを削除します。

**構文**     `RemovemboxTag (String mboxName)`

**戻り値**     `Mailbox removed successfully` と返します。

**変数**

変数	説明
<code>mboxName</code>	メールボックスの名前。サンプルを実行するには、適切なメールボックスに以下の名前を使用する。 <ul style="list-style-type: none"><li>■ 受信メール用：<code>trading_partner_name_Inbox</code></li><li>■ 送信メール用：<code>trading_partner_name_Outbox</code></li></ul>

