BEA WebLogic Integration™

WebLogic Integration チュートリアル

リリース 7.0
マニュアルの日付 : 2002 年 6 月

著作権

限定的権利条項

商標または登録商標

WebLogic Integration チュートリアル

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

3. サンプルについて

A. DTD

このマニュアルの内容

このマニュアルでは、サンプル統合アプリケーションについて説明します。このサンプル アプリケーションでは、サプライチェーン ハブをデプロイして、ビジネス パートナを接続し、複数のビジネス プロセスを自動化し、バックエンドのエンタープライズ情報システムを統合します。このマニュアルでは、サンプル アプリケーションの設定および実行方法と、WebLogic Integration を使用して統合ソリューションを構築および開発する方法について学習します。

このマニュアルは、WebLogic Integration の概要、および WebLogic Integration の機能を統合ソリューションの設計、開発、デプロイメントのさまざまな段階でどのように使用するかについて説明した 4 冊のマニュアル シリーズの 1 つです。まずこれらのマニュアルで WebLogic Integration の機能を包括的に理解してください。他の 3 冊のマニュアルの内容は、以下の通りです。

- 『*WebLogic Integration 入門*』 − WebLogic Integration の概要について説明したマニュアルです。このマニュアルでは、異種のフラグメント化されたビジネス システムが集まっている今日の E ビジネスが直面している統合問題について概説します。次に、WebLogic Integration が E ビジネスの統合問題を解決するために提供する Application Integration、B2B Integration、Business Process Management、および Data Integration の各機能について説明します。
- 『*WebLogic Integration ソリューションの設計*』 − WebLogic Integration ソリューションの設計および構築方法について説明したマニュアルです。このマニュアルでは、推奨されるベスト プラクティスに従った優れた設計原理について学習します。
- 『*WebLogic Integration ソリューションのデプロイメント*』 −統合ソリューションを開発からプロダクション環境に移行する方法について説明したマニュアルです。このマニュアルでは、統合アプリケーションのコンフィグレーション、スケーリング、移植、およびパフォーマンス チューニングについて学習します。

これらのマニュアルの内容を理解したら、WebLogic Integration の機能に関する詳細マニュアルに進んでください。

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、ビジネス上の問題と、それに対して WebLogic Integration がどのように対応するかについて説明します。
- 第 2 章「サンプルの設定と実行」では、サンプル実装を設定および実行する方法について説明します。
- 第 3 章「サンプルについて」では、サンプルの機能、つまり、WebLogic Integration コンポーネントを協調して動作させてエンタープライズ統合の問題を解決する方法について説明します。この章は、サンプルを使用してビジネス パートナを定義および管理し、複数のビジネス プロセスを自動化し、バックエンドのエンタープライズ情報システムを統合する方法についてのチュートリアルです。
- 付録 A「DTD」では、サンプル実装で使用されている文書型定義（DTD）を示します。

対象読者

このマニュアルは、次のユーザを対象としています。

- ビジネス アナリスト－統合ソリューションの目標を定義し、これらの目標を達成するためにテクニカル アナリストと共同で作業します。ビジネス アナリストは、ビジネスおよび経済のエキスパートであり、企業の経営方法に精通しています。ビジネス アナリストは、社内の情報技術の資産を活用して競争で優位に立つための技術をデプロイすることにより、既存のビジネス プロセスを改善します。
- テクニカル アナリスト－ Application Integration、Business Process Management、B2B Integration、および Data Integration を含む WebLogic Integration ソリューションの評価、デプロイメント、および管理を担当します。テクニカル アナリストは通常、システム全体に関するエンド ツー エンドの知識を持ち、統合ソリューションの目標を満たすためにビジネス アナリストと共同で作業します。
- ソフトウェア エンジニア－ WebLogic Integration ソリューションの実装を担当します。ソフトウェア エンジニアは、統合ソリューションを操作するための基盤となるコードに精通し、すべてのソフトウェア コンポーネントがリンクしてソリューションが正常に作動するように確認し、スケーリング、移

植、パフォーマンス チューニングなどのソフトウェア アプリケーションに関する問題に対応します。

このマニュアルは主に、エンタープライズ統合の課題に対応したソリューションを構築するアプリケーション開発者を対象としています。WebLogic Integration 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、WebLogic Integration の Web サイトで入手できます。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

以下の WebLogic Integration のマニュアルには、このマニュアルの内容を補足する情報、および WebLogic Integration アプリケーションを開発およびデプロイする方法について理解するための情報が含まれています。

- *WebLogic Integration 入門*
- *WebLogic Integration ソリューションのデプロイメント*
- *WebLogic Integration ソリューションの設計*

WebLogic Integration の全マニュアルについては、
`http://edocs.beasys.co.jp/e-docs/index.html` を参照してください。

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、ご使用の製品ドキュメントのバージョン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**) 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。
	例
	`#include <iostream.h> void main ( ) the pointer psz`
	`chmod u+w *`
	`\tux\data\ap`
	`.doc`
	`tux.doc`
	`BITMAP`
	`float`

太字の等幅テキスト	コード内の重要な箇所を示す。 *例* `void **commit** ( )`

表記法	適用
斜体の等幅テキスト	コード内の変数を示す。 *例* `String expr`
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。 *例* 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。 *例* `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。 ■ 引数を複数回繰り返すことができる ■ 任意指定の引数が省略されている ■ パラメータや値などの情報を追加入力できる 実際には、この省略記号は入力しない。 *例* `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

この章では、WebLogic Integration サンプルの背景となっているビジネス上の問題例について説明します。この章の内容は以下のとおりです。

- サンプルの範囲
- シナリオの背景
- 統合ソリューションのデプロイ

サンプルの範囲

BEA WebLogic Integration™ は、企業向けにアプリケーション サーバ、アプリケーションの統合、ビジネス プロセス モデル、および企業間（B2B）統合機能を提供する単一のプラットフォームです。

WebLogic Integration サンプルでは、WebLogic Integration プラットフォームを使用して、新しいアプリケーションの開発、既存のシステムとの統合、複雑なビジネス プロセスの効率化、およびビジネス パートナとの連携を実現する方法を示します。子のマニュアルで示したサンプル コードは、WebLogic Platform インストール ディレクトリの *`WL_HOME`*`\samples\integration\samples\wlis` にあります。

上の行の *`SAMPLES_HOME`* は、WebLogic Platform のサンプル ディレクトリです。

サンプル実装では、サプライ チェーンの自動化および統合という課題の一部を扱っているだけですが、柔軟な WebLogic Integration プラットフォームを使用して要件に応じてソリューションを開発し、社内業務とバリュー チェーン全体を効率化する方法を示しています。

注意： このマニュアルで説明する WebLogic Integration サンプルは、WebLogic Integration の本リリースから非推奨になった XOCP ビジネス プロトコルに基づいています。XOCP の代替機能に関する詳細については、『*BEA WebLogic Integration リリース ノート*』を参照してください。

シナリオの背景

General Control Systems (GCS) はイリノイ州を拠点としたある大企業の一部門で、工場およびオフィス ビル向けの各種制御システムを製造しています。

EnergyMiser 76 の歴史

1970 年代の石油危機以降、GCS では、電力消費量を抑制するための監視および制御システムである EnergyMiser 76 を製造してきました。これらのエネルギー制御システムを 3 社に OEM 供給しています。GCS では、EnergyMiser 76 用の金属ボックスをシカゴにある中規模企業の Midwest Metals から調達しています。Midwest Metals は、シルクスクリーン ボックスを 25 年に渡って独占的に供給してきました。

GCS の付加価値再販業者（VAR）はすべて、EDI システムを使用してサプライチェーンを自動化しています。GCS では、15 年間、EDI をサプライ チェーンとデマンド チェーンの両方で使用して、VAR からの発注を処理し、Midwest Metals からの調達を自動処理してきました。シルクスクリーン ボックスの在庫は、GCS および Midwest Metals の EDI システムにリンクした自動補充システムによって管理されています。EnergyMiser 76 モデルの需要が安定していて、在庫計画を簡単に立てられたので、サプライチェーン メカニズムは順調に機能してきました。

最近になって、GCS の親会社では、企業間（B2B）インフラストラクチャとして BEA WebLogic Platform の導入に着手しました。GCS では、従来のシステムが機能している上に、サプライヤからも顧客からも XML ベースの B2B トランザクションの要望がなかったので、このシステムに移行することに納得できませんでした。

製品需要の増加

第 3 四半期に入って、VAR から EnergyMiser 76 の発注がかつてないほど急増しました。自動補充システムは処理を開始し、いつもより多い数量のシルクスクリーン ボックスの自動 EDI 850 メッセージ（購入注文）を Midwest Metals に送信しました。Midwest Metals の社長は GCS の調達部門に電話をかけて、工場の生産能力が追いつかないので指定納期に注文数量をすべて納品することはできないと伝えました。

さらに、工場の生産能力の拡大には、長期に渡ってそれに見合った発注を見込めない限り、応じられないことも説明しました。GCS のマネージャは、カリフォルニアの顧客から注文が相次いだために需要が急増したことを説明しました。しかし、カリフォルニアのエネルギー状況の見通しがはっきりしないので、現在の注文レベルが長期的な需要増を示しているのか一時的な増加にすぎないのかはわかりません。

状況を理解すると、調達マネージャは Midwest Metals が供給できる最大量を買い付けるとともに、別の調達先を探し始めました。そのマネージャは会社が B2B 統合インフラストラクチャの改良を決定したことを思い出し、IT 部に連絡して、現在の危機的状況を打開する短期的な計画と、サプライ チェーンで管理レベルを上げて現状を一目で把握できるようにして、在庫不足などの不測の事態を避けるための長期的な計画について話し合いました。マネージャはこの状況を、GCS の従来のシステムが現在の自動化されたダイナミックな市場に対応しきれなくなっていることを示すものであると考えるようになりました。

統合ソリューションのデプロイ

このバリュー チェーンでは、GCS がチャネル マスタです。市場での効率性を高め、競争で優位に立つために、GCS は BEA WebLogic Integration を使用してサプライチェーンのハブをデプロイしました。

サンプル アプリケーションは、ビジネス パートナを結ぶサプライチェーンのハブであり、多くのビジネス プロセスを自動化し、バックエンドのエンタープライズ情報システムを統合します。このサンプル アプリケーションでは、GCS デ

マンド チェーンや GCS と VAR との統合については扱いません。EDI 統合アプリケーションのサンプルについては、『*WebLogic Integration EDI ユーザーズ ガイド*』の「EDI のサンプル」を参照してください。

短期的な利点と長期的な利点

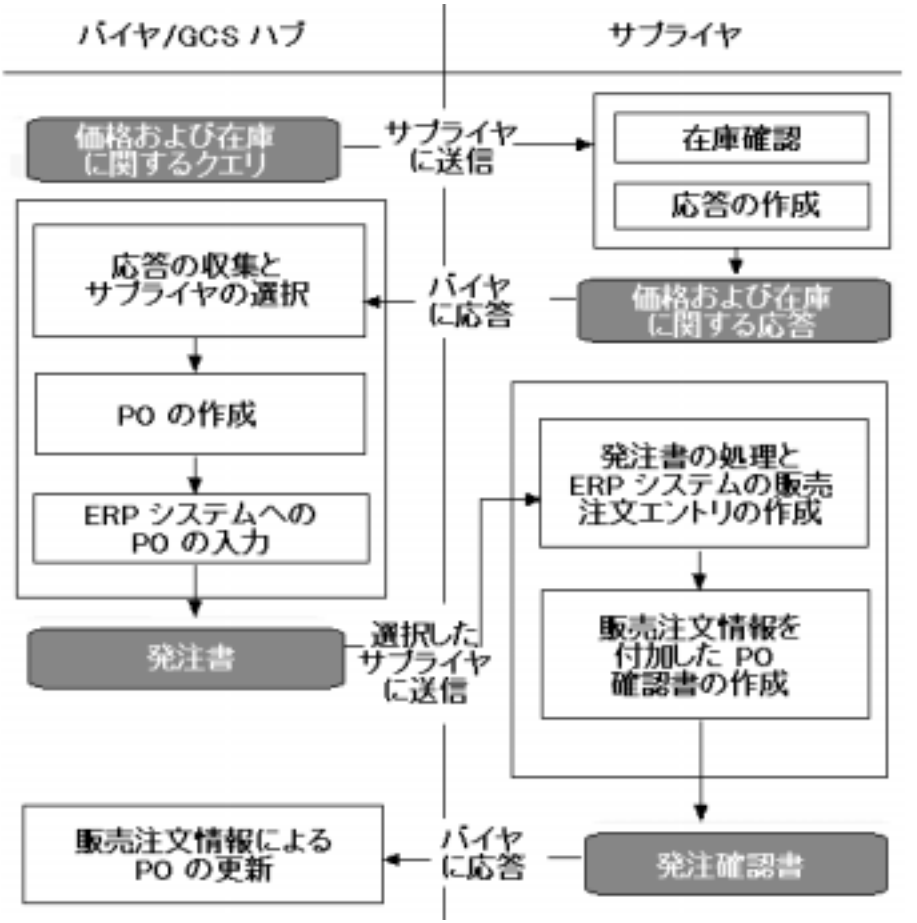
ハブをデプロイすると、短期的な利点と長期的な利点を得られます。ハブにより、Midwest Metals 以外で認可済みベンダ リスト（AVL）に入っているパートナからの供給品の調達を促進することで、現在の供給不足を解決し、今後のビジネスに備えてサプライ チェーンを効率化するシステムを確立できます。

GCS では、サプライチェーン ハブをデプロイすることで、いくつかの処理を自動化できます。同じサンプル アプリケーションを使用して、GCS では選択したビジネス パートナと会話できます。ビジネス パートナ間のトランザクションは、次の一連のイベントにまとめられます。

1. 価格と在庫の照会が GCS から認可済みの二次サプライヤに送られます。
2. GCS では、サプライヤからの応答を取りまとめて発注マネージャに示し、サプライヤを選択します。
3. 選択したサプライヤに発注書が発行されます。
4. 発注書は、社内の ERP システムに自動的に入力されます。
5. 選択されたサプライヤは、発注確認書を GCS に返します。
6. 発注書のレコードは、発注確認書の情報（サプライヤの受注番号など）で自動的に更新されます。

次の図は、GCS のハブとサプライヤがやり取りする価格と在庫の照会の処理フローを示しています。

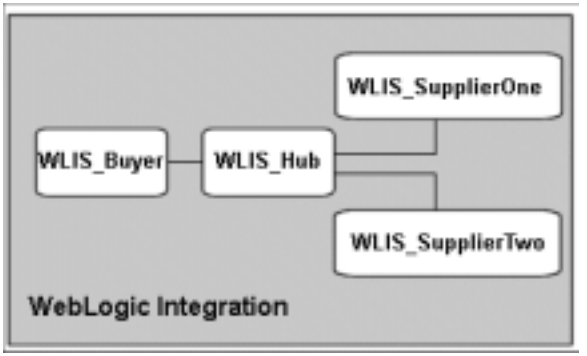
図 1-1　サンプル アプリケーションの処理フロー



ソリューションのアーキテクチャ

WebLogic Integration サンプルは、ハブ、バイヤ、トレーディング パートナである 2 つのサプライヤからなる 4 つのエンティティをデプロイしてソリューションを実装しています。プロダクション環境では、各エンティティは別々の WebLogic Integration サーバ上で動作できます。ただし、サンプルの実行手順を簡略化するために、このシナリオでは、1 つの WebLogic Integration サーバが 4 つのエンティティすべてのホストとなります。次の図は、この実装を示しています。

図 1-2 WebLogic Integration サンプルの実装アーキテクチャ



サンプルを構成し、ビジネス エンティティによって実行される機能について次の表で説明します。

表 1-1 ビジネス エンティティによって実行されるサンプルの機能

ビジネス エンティティ	機能
WLIS_Hub	バイヤとサプライヤ間の通信をルーティングして、企業間統合を実現する。

表 1-1 ビジネス エンティティによって実行されるサンプルの機能

ビジネス エンティティ	機能
WLIS_Buyer	■ ワークフロー テンプレートを使用して、サプライヤおよび内部機能（バックエンド データベースの更新）間のビジネス プロセスを調整する。 ■ アプリケーション ビューを使用して、バイヤのデータベース システムへの接続を提供する。 ■ HTML および JSP ページを使用して、データ表現とユーザ インタフェースを処理する。
WLIS_SupplierOne	■ ワークフロー テンプレートを使用して、バイヤからの要求に対応して内部プログラム（データ トランスフォーメーションや永続性など）を起動する。 ■ アプリケーション間の情報交換を容易にするため、データを変換する。
WLIS_SupplierTwo	■ ワークフロー テンプレートを使用して、バイヤからの要求に対応して内部プログラム（データ トランスフォーメーションや永続性など）を起動する。 ■ アプリケーション間の情報交換を容易にするため、データを変換する。

2　サンプルの設定と実行

この章では、環境を設定し、WebLogic Integration サンプル アプリケーションを実行するステップについて説明します。それぞれのステップは以下の節に分かれています。

- サンプルの実行前の作業
- サンプルの実行

このサンプルのアーキテクチャの詳細については、1-6 ページの「ソリューションのアーキテクチャ」を参照してください。サンプル アプリケーションの処理の概要については、図 1-1 の「サンプル アプリケーションの処理フロー」を参照してください。

サンプルの実行前の作業

サンプルを実行する前に、次のステップを実行します。

1. [WebLogic Integration Full Installation with Samples] オプションを指定して、WebLogic Integration をインストールします。インストール手順については、『*Installing BEA WebLogic Integration*』を参照してください。

WebLogic Integration サンプルは *SAMPLES_HOME*`\integration\samples\wlis` ディレクトリにインストールされています。*SAMPLES_HOME* は WebLogic Platform のサンプル ディレクトリを示します。

2. ローカルの WebLogic Server に接続できるように、ブラウザのプロキシ設定を確認します。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。

注意： サンプルでは、WebLogic Integration をインストールするときに指定したデータベースを使用します。他のデータベースを使用する場合は、[WebLogic Integration Database Configuration] ウィザードを実行します。このウィザードの実行手順については、『*WebLogic Integration の起動、停止およびカスタマイズ*』の「データベース コンフィグレーション ウィザードの使い方」を参照してください。

サンプルの実行

注意： このサンプルは、WebLogic Integration の各インスタンスにつき一度に 1 ユーザだけが実行できます。

WebLogic Integration サンプルを実行するには、次の 8 つのステップをすべて実行します。

- ステップ 1: 起動用 Web ページをコンフィグレーションして呼び出す
- ステップ 2: WebLogic Integration サンプルを選択する
- ステップ 3: サンプルを起動する
- ステップ 4: QPA 要求を送信する
- ステップ 5: QPA 要求を確認する
- ステップ 6: 発注書を作成する
- ステップ 7: 発注書を確認する
- ステップ 8: 発注確認書を確認する

ステップ 1: 起動用 Web ページをコンフィグレーションして呼び出す

WebLogic Platform のインストール ディレクトリの *SAMPLES_HOME*\integration\samples\bin ディレクトリには、WebLogic Integration 製品のすべてのサンプルをコンフィグレーションおよび実行するためのスクリプトが入っています（*SAMPLES_HOME* は WebLogic Platform のサンプル ディレクトリを示す)。

スクリプトについて

RunSamples スクリプトは、起動用 Web ページが表示されるまで、適切な順序で他のスクリプトを起動するドライバ スクリプトです。起動用 Web ページには、複数の WebLogic Integration サンプルおよびコンソールを起動するためのリンクが含まれています。RunSamples スクリプトの詳細については、『*WebLogic Integration の起動、停止およびカスタマイズ*』の「はじめに」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。

このステップ（起動用 Web ページをコンフィグレーションして呼び出す）は、2 つのサブステップに分けて説明します。この 2 つのサブステップでは、起動用 Web ページをコンフィグレーションして呼び出すための別のステップを説明します。

- ステップ 1A : RunSamples スクリプトを起動する
- ステップ 1B : Start Server および Launcher スクリプトを起動する

サンプル ドメインでサンプルを初めて実行する場合は、ステップ 1A で説明するように、RunSamples スクリプトを実行しなければなりません。RunSamples スクリプトはサンプル データベースをコンフィグレーションするだけでなく、サンプル ドメインで WebLogic Integration を起動します。

一度 RunSamples スクリプトを実行し、データベースを正しくコンフィグレーションしたら、次にサンプル ドメインを起動するときには、RunSamples スクリプトをもう一度実行すると、サンプルを選択して起動できるようになります。ステップ 1A を参照してください。

注意： サンプル データベースをコンフィグレーションした後に RunSamples コマンドを実行すると、次のメッセージが表示されます。

```
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
```

N を入力すると、データベースのコンフィグレーション タスクが省略されます。コマンドは、サンプル ドメインで WebLogic Integration を起動し、[Samples Launcher] ページを Web ブラウザに表示します。

ステップ 1B で説明するように、Start Server および Launcher スクリプトを使用することもできます。

ステップ 1A：RunSamples スクリプトを起動する

注意： RunSamples スクリプトを一度も起動していない場合、またはサンプルで操作するデータベースを変更した場合、このステップで説明するタスクを実行する必要があります。

次の節では、Windows および UNIX システム上で RunSamples スクリプトを実行するステップを示します。

Windows 上での RunSamples スクリプトの実行

Windows システム上で RunSamples スクリプトを実行するには、次のいずれかを実行します。

- メニューから RunSamples スクリプトを実行する場合

  [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。

- コマンド ラインから RunSamples スクリプトを実行する場合

  a. コマンド ウィンドウを開きます。

b. setenv スクリプトを実行して WebLogic Integration 環境変数を設定します。

```
cd WLI_HOME
setenv
```

WLI_HOME は WebLogic Integration がインストールされているディレクトリを表します。

c. サンプルドメインの \bin ディレクトリに移動します。たとえば、

```
cd SAMPLES_HOME\integration\samples\bin
```

上の行の *SAMPLES_HOME* は、WebLogic Platform のサンプルディレクトリです。

d. 次のように入力して、RunSamples スクリプトを実行します。

```
RunSamples
```

RunSamples スクリプトの実行が終了したら、サンプルに付属の起動用 Web ページが Web ブラウザに表示されます。

UNIX 上での RunSamples スクリプトの実行

UNIX システム上で RunSamples スクリプトを実行するには、次のステップを実行します。

1. setenv スクリプトを実行して WebLogic Integration 環境変数を設定します。

```
cd WLI_HOME
setenv
```

WLI_HOME は WebLogic Integration がインストールされているディレクトリを表します。

2. サンプルドメインの bin ディレクトリに移動します。たとえば、

```
cd SAPLES_HOME/integration/samples/bin
```

上の行の *SAMPLES_HOME* は、WebLogic Platform のサンプルディレクトリです。

3. 次のように入力して、RunSamples スクリプトを実行します。

```
. ./RunSamples
```

RunSamples スクリプトの実行が終了したら、サンプルに付属の起動用 Web ページが Web ブラウザに表示されます。

ステップ 1B : Start Server および Launcher スクリプトを起動する

ステップ 1A で説明したように、既に RunSamples スクリプトを実行した場合、データベースが作成され、サンプル データが入力されています。この場合、サンプルを実行するときには、ステップ 1A の代わりにこのステップのタスクを実行してもかまいません。ステップ 1B の手順では、Windows および UNIX システム上で WebLogic Server を起動し、起動用 Web ページを呼び出します。

Windows 上での Start Server および Launcher スクリプトの起動

Windows システム上でスクリプトを実行するには、次のいずれかを実行します。

- メニューから Start Server スクリプトに次いで Samples Launcher スクリプトを起動する場合

  a. [スタート | プログラム | BEA WebLogic Platform 7.0 | WebLogic Integration 7.0 | Integration Examples | Start Server] を選択します。

     次のメッセージが表示された場合は、Start Server スクリプトは正しく終了しています。

     ```
     StartServer execution successful
     ```

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

     サンプルに付属の起動用 Web ページが Web ブラウザに表示されます。

- コマンド ラインから Start Server スクリプトに次いで Samples Launcher スクリプトを起動する場合

1. コマンド ウィンドウを開きます。
2. setenv スクリプトを実行して WebLogic Integration 環境変数を設定します。

   ```
   cd WLI_HOME
   setenv
   ```

   WLI_HOME は WebLogic Integration がインストールされているディレクトリを表します。

   c. サンプル ドメインの bin ディレクトリに移動します。たとえば、

   ```
   cd SAMPLES_HOME\integration\samples\bin
   ```

上の行の *SAMPLES_HOME* は、WebLogic Platform のサンプル ディレクトリです。

d. 次のように入力してサーバを起動します。

```
startServer
```

次のメッセージが表示された場合は、startServer スクリプトは正しく終了しています。

```
StartServer execution successful
```

e. 次のように入力して Web ブラウザを起動します。

```
launchBrowser
```

サンプルに付属の起動用 Web ページが Web ブラウザに表示されます。

UNIX 上での Start Server および Launcher スクリプトの起動

UNIX システム上でスクリプトを実行するには、次の手順を実行します。

1. setenv スクリプトを実行して WebLogic Integration 環境変数を設定します。

```
cd WLI_HOME
setenv
```

WLI_HOME は WebLogic Integration がインストールされているディレクトリを表します。

2. サンプル ドメインの bin ディレクトリに移動します。たとえば、

```
cd SAMPLES_HOME/integration/samples/bin
```

上の行の *SAMPLES_HOME* は、WebLogic Platform のサンプル ディレクトリです。

3. 次のように入力してサーバを起動します。

```
. ./startServer
```

次のメッセージが表示された場合は、startServer スクリプトは正しく終了しています。

```
StartServer execution successful
```

4. Web ブラウザを起動して次の URL を入力します。

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

サンプルに付属の起動用 Web ページが表示されます。

ステップ 2: WebLogic Integration サンプルを選択する

起動用 Web ページからは、複数のサンプル アプリケーションおよび管理コンソールを起動できます。この Web ページの [サンプル アプリケーション] の下にある [WLI Sample] リンクをクリックして、第 1 章「はじめに」で説明したサプライ チェーンの問題を解決するサンプル アプリケーションを起動します。

ブラウザには、次の図のようなサンプル概要ページが表示されます。

図 2-1　WebLogic Integration サンプルの [概要] ウィンドウ

概要ウィンドウでは、ビジネス上の問題と、WebLogic Integration サンプルによるソリューションが簡単に説明されています。

ステップ 3: サンプルを起動する

次のいずれかの方法でサンプルを起動します。

- [概要] ウィンドウ上部にある水平の黒いバーの [実行] をクリックします。
- [概要] ウィンドウ下部にある [サンプルの実行] をクリックします。

次の図に示す [QPA リクエストの送信] ウィンドウが表示されます。

図 2-2 [QPA リクエストの送信] ウィンドウ

このウィンドウでは、次の部分に注目してください。

- ウィンドウには、[価格と在庫の照会（QPA）] フォームが表示されています。このフォームには、バイヤ（GCS）から 2 つのサプライヤに送信する QPA 要求データが格納されます。

  このフォームではフィールドのデータを変更できません。

- ウィンドウの右側には、ステータス バーが表示されています。このバーは小さなフロー チャートです。各ボックスは、実行中の手順のステップを表し、

サンプル アプリケーションの段階に合わせてそのステップに対応するウィンドウも表します。サンプルの段階に合わせて、該当するステータス ボックスが各ウィンドウで強調表示されます。このステータス バーを見ると、どの段階の処理なのかがわかります。

ステップ 4: QPA 要求を送信する

[QPA リクエストの送信] をクリックして、QPA プロセスを開始します。次の図に示す [QPA 応答のチェック] ウィンドウが表示されます。

図 2-3 [QPA 応答のチェック] ウィンドウ

このウィンドウは、QPA 要求がサプライヤに送信され、バイヤが応答を待っていることを示します。

[QPA 応答のチェック] ウィンドウの [QPA 応答のチェック] をクリックして、次のウィンドウを表示します。

ステップ 5: QPA 要求を確認する

次に表示されるウィンドウは、サプライヤからの QPA 応答がバイヤに届いているかどうかによって異なります。

- QPA 応答がサプライヤからバイヤに届いていない場合、次のダイアログ ボックスが表示されます。

図 2-4 [QPA 応答] ダイアログ ボックス

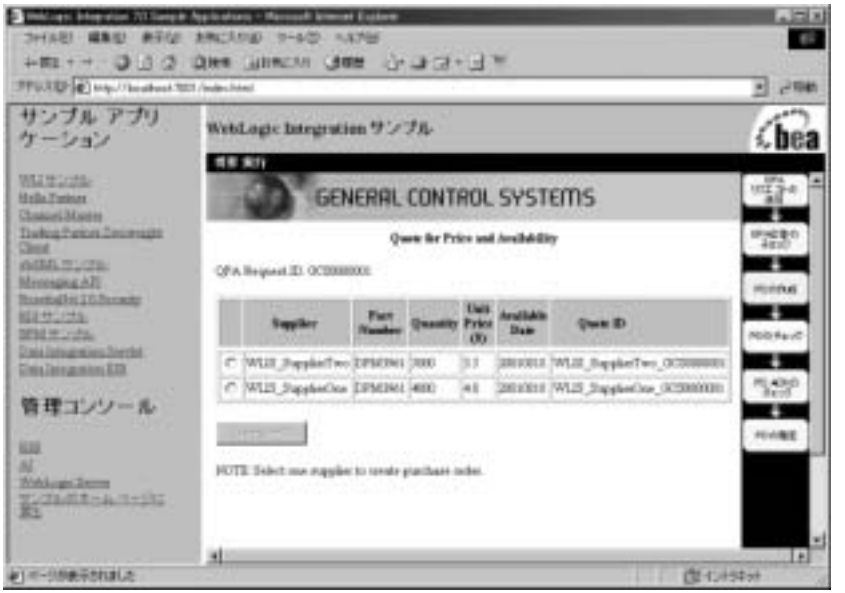
a. [OK] をクリックして、ダイアログ ボックスを閉じます。[QPA 応答のチェック] ウィンドウは、ブラウザに表示されたままとなります。

b. しばらくしてから、[QPA 応答のチェック] をもう一度クリックします。

c. [QPA Response] ダイアログ ボックスが表示されなくなるまで同じステップを繰り返します。表示されなくなったら、サプライヤからの応答がバイヤに届いたことになります。応答が届くと、図 2-5 に示す [PO の作成] ウィンドウが表示されます。応答がない場合は、WebLogic Server ログでエラー情報を確認します。

- QPA 応答がサプライヤからバイヤに届いた場合、次の図の [PO の作成] ウィンドウが表示されます。

図 2-5　[PO の作成] ウィンドウ



このウィンドウには、価格と在庫の照会に対する各サプライヤからの応答をまとめたフォームが表示されます。

ステップ 6: 発注書を作成する

次の手順を実行して、シルクスクリーンの金属ボックスの発注書を作成します。

1. 図 2-5 に示したように、フォーム上で WLIS_SupplierOne または WLIS_SupplierTwo を選択します。
2. [PO の作成] をクリックして、バックエンドの ERP システムに要求を送信します。このシステムでは、指定した発注書が生成され、選択したサプライヤに送信されます。

   [PO のチェック] ウィンドウが表示されます。このウィンドウには、発注要求が送信されたことを示すメッセージが表示されます。次の図を参照してください。

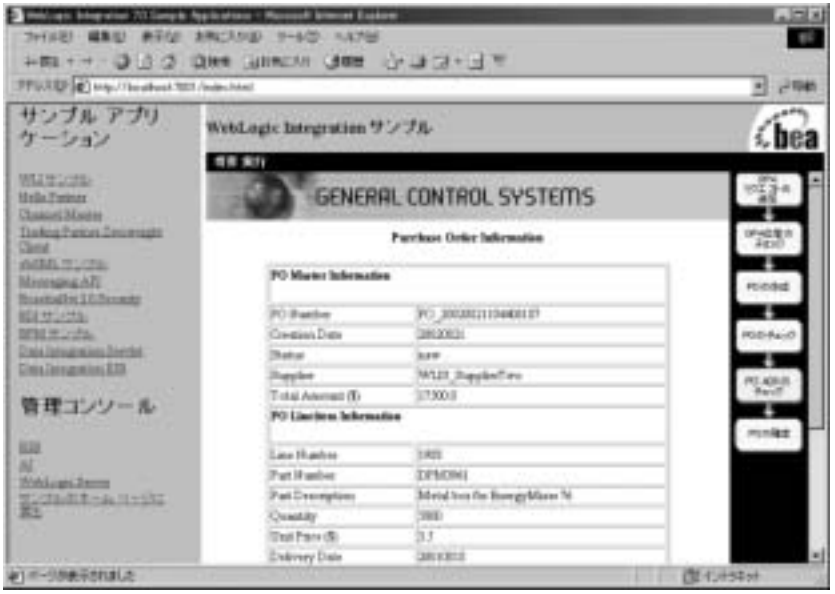
図 2-6 [PO のチェック] ウィンドウ

提出済みの発注書のステータスを確認するには、[発注書のチェック] をクリックします。

ステップ 7: 発注書を確認する

選択したサプライヤに発注書が送信されると、次の図のように、注文内容が表示されます。

図 2-7 [PO Ack のチェック] ウィンドウ

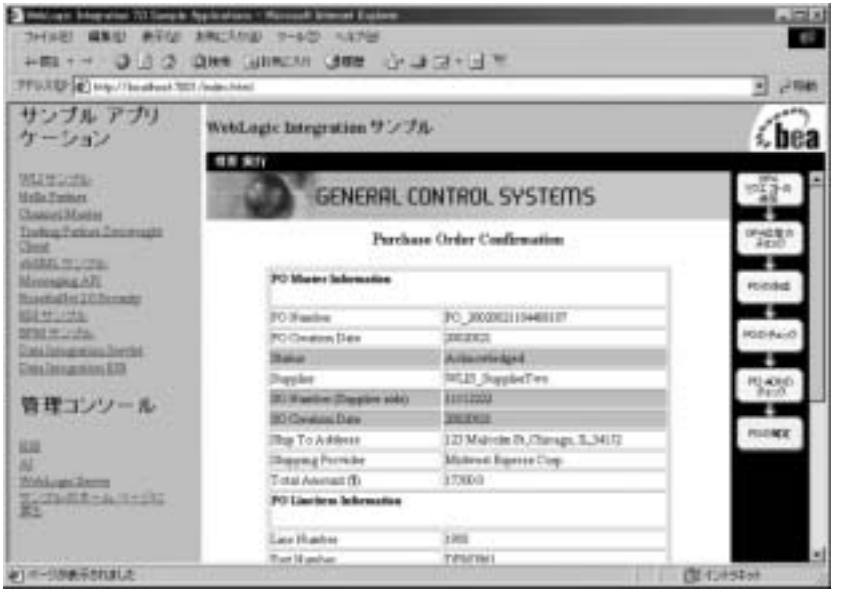


上の図には、選択したサプライヤにユーザ（バイヤ）が送信した PO 情報が表示されています。発注書を受信すると、サプライヤは確認メッセージを返します。このメッセージを表示するには、[発注確認書のチェック] をクリックします。

ステップ 8: 発注確認書を確認する

サプライヤが発注書の受信を確認すると、次の図に示す [PO の確定] ページがバイヤに対して表示されます。

図 2-8 [PO の確定] ウィンドウ



上の図は、サプライヤがバイヤ（GCS）に送信した発注確認書を示しています。バイヤが送信し、サプライヤが確認した元の発注書に、サプライヤが出荷情報を追加していることに注目してください。図 2-7 と 図 2-8 の [PO Master Information] セクションを見比べてください。

WebLogic Integration のサンプル アプリケーションの実行手順は以上です。このアプリケーションのしくみについては、第 3 章「サンプルについて」を参照してください。

3　サンプルについて

注意： この章を読む前に、サンプルを実行しておくことをお勧めします。手順については、第 2 章「サンプルの設定と実行」を参照してください。ここで説明するサンプル コードは、WebLogic Platform のインストール ディレクトリの *SAMPLE_HOME*\integration\samples\wlis ディレクトリにあります。

SAMPLES_HOME は WebLogic Platform のサンプル ディレクトリを示します。

サンプル アプリケーションは、多くのビジネス プロセスを自動化し、バックエンドのエンタープライズ情報システム（EIS）を統合して、ビジネス パートナを結ぶサプライチェーンのハブをデプロイします。この章では、仮想企業 General Control Systems (GCS) が直面しているサプライチェーンの課題を、サンプル統合ソリューションでどのように解決するかについて説明します。具体的には、以下のトピックを取り上げます。

- 概要
- B2B Integration
- ビジネス プロセスおよびワークフローのモデル化
- Application Integration と Data Integration

注意： このマニュアルで説明する WebLogic Integration サンプルは、WebLogic Integration の本リリースから非推奨になった XOCP ビジネス プロトコルに基づいています。XOCP の代替機能に関する詳細については、『*WebLogic Integration リリース ノート*』を参照してください。

概要

サンプル シナリオでは、General Control Systems (GCS) は、サプライチェーンの課題に対して WebLogic Integration ソリューションを実装することを決定します。最初のステップは何でしょう。

GCS では、WebLogic Integration を使用して以下のタスクを実行する方法を分析することから始めました。

- トレーディング パートナ間のトランザクションを推進するビジネス プロセスのモデル化
- シナリオに登場するビジネス パートナどうしの企業間（B2B）統合の定義および管理
- Web ベースの新しいフロントエンド システムと既存のエンタープライズ情報システムとの統合
- 異種データ形式の処理

ここからは、GCS が WebLogic Integration を使用して、これらのタスクを実行する方法について説明します。

ビジネス プロセスのモデル化

WebLogic Integration には、ビジネス プロセスの管理に必要なツールが用意されています。

- WebLogic Integration プロセス エンジン－ワークフロー テンプレートの作成と管理、ワークフローの開始と終了、実行時のワークフロー インスタンスの制御と管理などのサービスを提供する EJB のコレクションです。
- WebLogic Integration Studio －ビジネス プロセスを作成し、実行時にモニタするためのグラフィカル デザイン ツールです。

B2B 統合の管理

WebLogic Integration は、外部のトレーディング パートナとのインターネット経由の通信をサポートしています。具体的には、XML メッセージの送受信用として XOCP、RosettaNet (1.1 と 2.0)、および Ariba cXML ビジネス プロトコルをサポートしています。

B2B 統合 BPM プラグインは WebLogic Integration Studio を拡張します。その結果、Studio を使用して、協調的ワークフロー、つまり外部のトレーディング パートナとメッセージをやり取りするワークフローを定義できるようになります。

新しいシステムと既存システムとの統合

WebLogic Integration は、*アダプタ*を使用することで、Web ベースの新しいフロントエンド システムと既存のエンタープライズ情報システム（EIS）アプリケーションとの統合をサポートしています。Application Integration アダプタは、大きく次のように分類できます。

- サービス アダプター WebLogic Integration から EIS アプリケーションへの同期要求および応答統合を実現します。
- イベント アダプター EIS アプリケーションから WebLogic Integration への一方向の非同期統合を実現します。

Application Integration アダプタのコンフィグレーションには、Web ブラウザのインタフェースを使用します。Application Integration アダプタを経由するビジネス イベントは、WebLogic Integration リポジトリの XML スキーマ定義（XSD）として定義します。

アプリケーション統合 BPM プラグイン WebLogic Integration Studio を拡張します。その結果、Studio では、次の処理を行うプロセスを定義できるようになります。

- アプリケーション統合サービス アダプタへのイベントの送信
- Application Integration アダプタによって生成されたイベントの処理

異種データ形式の処理

XML は、WebLogic Integration システムで使用されるメッセージ形式です。しかし、WebLogic Integration フレームワークでは、バイナリ メッセージ形式を使用する環境に WebLogic Integration アプリケーションを簡単に統合することができます。

WebLogic Integration は、次のデータ統合をサポートしています。

- バイナリ ー XML 間変換 : WebLogic Integration は、XML とバイナリ形式との間で双方向の変換を実行します。この変換では、メッセージのデータ表現（XML またはバイナリ）だけが変更され、メッセージのデータ構造および内容は変更されません。
- XML ー XML 変換 : WebLogic Integration では、XSLT を使用して、XML メッセージのデータ構造または内容（あるいはその両方）をソース メッセージが対象のメッセージにマップできます。

  XSL スタイルシートは、手動で作成することも、WebLogic Integration 製品に同梱されている Contivo Analyst マッピング ツールを使用して作成することもできます。スタイルシートは、WebLogic Integration リポジトリに格納されます。

  変換は実行時に次のように行われます。まず入力メッセージがワークフロー変数に割り当てられ、プロセス エンジンがリポジトリ内の XSL スタイルシートを参照し、出力メッセージが別のワークフロー変数に割り当てられます。

以下の節では、WebLogic Integration のコンポーネントを協調して動作させて、サンプル アプリケーションをビルドおよびデプロイする方法について説明します。

B2B Integration

電子商取引のトランザクションに（このシナリオのバイヤおよびサプライヤのロールで）参加するトレーディング パートナ間の関係を定義して管理することは、この統合ソリューションを開発する上で基本となる作業です。トレーディング パートナを定義および管理するためのデータは WebLogic Integration リポジト

リに格納され、WebLogic Integration に用意されているツールおよびプロセスを使用して、流動的で多様なトレーディング パートナの関係を効率的に管理できます。

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超えます。そこで、この節では、サンプル アプリケーションで使用される WebLogic Integration リポジトリ データ、そのデータをサンプルでロードする方法、およびデータを表示してサンプルにおけるビジネス会話の進捗状況をモニタする方法について簡単に説明します。

B2B Integration の詳細と、それをサポートする WebLogic Integration 環境のコンフィグレーション手順については、それぞれ『*B2B Integration 入門*』と『*B2B Integration 管理ガイド*』を参照してください。

この付録のトピックは以下のとおりです。

- リポジトリ データのロード
- リポジトリ データについて
- WebLogic Integration B2B Console の使用

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で必要なデータは、サンプルの設定（2-2 ページの「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 `\dbscripts` ディレクトリにあります。

  `WLI_HOME``\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` ー次のディレクトリにあります。

`SAMPLE_HOME\integration\samples\wlis\lib`

上の行の *SAMPLES_HOME* は、WebLogic Platform のサンプル ディレクトリです。

この `BulkLoaderData.xml` ファイルには、WebLogic Integration 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

各データ要素の詳細については、次の「リポジトリ データについて」を参照してください。

リポジトリ データについて

この節では、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

注意： 3-5 ページの「リポジトリ データのロード」で説明したとおり、サンプル アプリケーションをサポートするために、2 つの XML ファイルのデータが WebLogic Integration リポジトリにインポートされます。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。詳細については、3-16 ページの「WebLogic Integration B2B Console の使用」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、WebLogic Integration がサポートしているすべてのビジネス プロトコル（XOCP、RosettaNet、および cXML）の定義を含むシステム データが格納されています。WebLogic Integration サンプル アプリケーションでは、XOCP のみを使用します。

SystemRepData.xml ファイルの次の抜粋部分は、XOCP ビジネス プロトコル定義を示しています。

コード リスト 3-1 SystemRepData.xml ファイル内の XOCP ビジネス プロトコル定義

```
<!-- XOCP BUSINESS PROTOCOL DEFINITIONS -->
<business-protocol-definition
       name="XOCP-SPOKE"
       business-protocol-name="XOCP"
       protocol-version="1.1"
       endpoint-type="SPOKE">
       <java-class>com.bea.b2b.protocol.xocp.XOCPSpokeProtocol</java-class>
       <decoder>XOCP-Decoder</decoder>
       <encoder>XOCP-Encoder</encoder>
</business-protocol-definition>

<business-protocol-definition
      name="XOCP-Hub"
      business-protocol-name="XOCP"
      protocol-version="1.1"
      endpoint-type="HUB">
      <java-class>com.bea.b2b.protocol.xocp.XOCPHubProtocol</java-class>
      <decoder>XOCP-Decoder</decoder>
      <system-router>XOCP-System-Router</system-router>
       :
       :
      <system-router>XOCP-Router-Enqueue</system-router>
      <system-filter>XOCP-System-Filter</system-filter>
       :
      <encoder>XOCP-Encoder</encoder>
       :
```

```
        :
</business-protocol-definition>
```

上のリストの次の点に注目してください。

- WebLogic Integration は、XOCP-HUB と XOCP-SPOKE という XOCP ビジネス プロトコルの 2 つの定義をサポートしています。
- endpoint-type は定義ごとに指定されています。
- 各ビジネス プロトコル定義は、指定した Java クラスによって実装されます。

ビジネス プロトコル 定義のコンフィグレーションの詳細については、『*B2B Integration 管理ガイド*』の「コンフィグレーション要件」および『*B2B Integration Administration Console オンライン ヘルプ*』の「トレーディング パートナのコンフィグレーション 」を参照してください。

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。

注意： カスタム ロジック プラグインは本リリースの WebLogic Integration から非推奨となりました。代替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。

`SystemRepData.xml` ファイルには、WebLogic Integration がサポートしているすべてのビジネス プロトコル（XOCP、RosettaNet、および cXML）の定義を含むシステム データが格納されています。このサンプルでは、XOCP ロジック プラグインのみを使用します。

表 3-1 XOCP 固有のロジック プラグイン

ロジック プラグイン	説明
XOCP ルータ	このルータ ロジック プラグインは、着信ビジネス メッセージを処理し、その処理で使用するメッセージコンテキスト ドキュメントを生成する。デフォルトでは、これがルータ チェーンで最初のロジック プラグインとなる。ハブの配信チャネル用の XOCP ルータ ロジック プラグインは、XPath ルータ式に基づいて受信先トレーディング パートナのリストを変更できる。詳細については、『*B2B Integration ロジック プラグイン プログラミングガイド*』の「ビジネス メッセージのルーティングとフィルタ処理」を参照。
XOCP ルータ エンキュー	ルータ エンキュー ロジック プラグインは、ビジネス メッセージをルータ メッセージ キューに追加する。デフォルトでは、これがルータ チェーンで最後のロジック プラグインとなる。
XOCP フィルタ	このフィルタ ロジック プラグインは、送信ビジネス メッセージを処理し、その処理で使用するメッセージコンテキスト ドキュメントを生成する。デフォルトでは、これがフィルタ チェーンで唯一のロジック プラグインとなる。ハブの配信チャネル用の XOCP フィルタ ロジック プラグインは、任意の XPath フィルタ式を使用して、メッセージが送信されたかどうかを調べる。詳細については、『*B2B Integration ロジック プラグイン プログラミングガイド*』の「ビジネス メッセージのルーティングとフィルタ処理」を参照。
XOCP エンコーダ	エンコーダは、メッセージを B2B 転送サービスに転送する。
XOCP デコーダ	デコーダは、XOCP ヘッダを処理し、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。

ロジック プラグインの開発と、WebLogic Integration B2B エンジンを使用したメッセージの処理の詳細については、『*B2B Integration ロジック プラグイン プログラミングガイド*』を参照してください。

トレーディング パートナ

WebLogic Integration サンプル シナリオでは、バイヤ（General Control Systems）と 2 つのサプライヤという 3 つのビジネス パートナが関係します。ビジネス パートナごとに、`BulkLoaderData.xml` ファイルでトレーディング パートナをコンフィグレーションします。サンプル用には、WLIS_Buyer、WLIS_SupplierOne、および WLIS_SupplierTwo というトレーディング パートナが定義されています。

トレーディング パートナは XOCP ビジネス プロトコルを使用して通信するので、General Control Systems では、WebLogic Integration システムを ハブ アンド スポーク コンフィグレーションとして定義する必要があります。B2B 統合のコンフィグレーションに関する詳細については、『*B2B Integration 入門*』の「B2B Integration の基礎」を参照してください。GCS 側には、`BulkLoaderData.xml` ファイルで第 4 のトレーディング パートナの WLIS_Hub を定義します。

WLIS_Hub トレーディング パートナは、仲介役として機能します。このトレーディング パートナは、スポーク トレーディング パートナである WLIS_Buyer、WLIS_SupplierOne、および WLIS_SupplierTwo 間のメッセージを仲介します。WLIS_Hub トレーディング パートナはビジネス メッセージの送信元でも受信先でもありませんが、トランザクションでは場合に応じて代理バイヤおよび代理サプライヤとなります。

3 つのトレーディング パートナ（WLIS_Buyer、WLIS_SupplierOne、および WLIS_SupplierTwo）のそれぞれには、WLIS_Hub トレーディング パートナとのコラボレーション アグリーメントがリンクされています。それぞれのコラボレーション アグリーメントをリンクするのは、WLIS_Hub トレーディング パートナです。たとえば、あるコラボレーション アグリーメントの一部としてメッセージを受信し、別のコラボレーション アグリーメントの一部として別のトレーディング パートナにルーティングする場合には、こうしたリンクが必要不可欠です。同じ配信チャネル（WLIS_HUB トレーディング パートナ用に定義されているチャネル）を使用するコラボレーション アグリーメントは相互にリンクされます。このシナリオのコラボレーション アグリーメントの詳細については、3-13 ページの「コラボレーション アグリーメント」を参照してください。

各トレーディング パートナ要素は、さまざまな属性および下位要素によって特性が記述されます。こうした属性および下位要素の一部は、名前、電子メール アドレス、電話番号、ファックス番号などの単純な ID 情報を格納します。次の表では、その他のトレーディング パートナのコンフィグレーション情報について説明します。

表 3-2 トレーディング パートナのコンフィグレーション

要素	説明
Type	サンプル シナリオのすべてのトレーディング パートナは、すべて同じ WebLogic Server インスタンス上で実行されるので、`Type="LOCAL"` と指定される。
パーティ識別子	トレーディング パートナ名と一緒に使用してコラボレーション アグリーメントのパーティを識別する一意のパーティ ID。パーティ ID には、一意の名前、ビジネス ID、およびビジネス ID タイプを指定する。
Business ID	パーティ ID に関連付けられるビジネス ID の記述名。サンプル シナリオでは、GCS が WLIS_Hub および WLIS_Buyer トレーディング パートナを所有しているので、これらのパートナには、同じビジネス ID（business-id="999999999"）が使われる。 各サプライヤ トレーディング パートナは、一意のビジネス ID を持つ。
配信チャネル	各トレーディング パートナの転送およびドキュメント交換を指す。WLIS_Hub トレーディング パートナの配信チャネルは `routing-proxy="true"` にコンフィグレーションされ、その他のトレーディング パートナ（シナリオではそれぞれスポークとしてコンフィグレーション）の配信チャネルは `routing-proxy="false"` にコンフィグレーションされる。
ドキュメント交換	ビジネス プロトコル（シナリオでは XOCP）と実行時パラメータを定義する。

表 3-2 トレーディング パートナのコンフィグレーション

要素	説明
転送	転送プロトコル（HTTP）とエンドポイントの URI を定義する。

注意： サンプルのトレーディング パートナのコンフィグレーションについては、`BulkLoaderData.xml` ファイルを参照してください。

会話定義

`BulkLoaderData.xml` ファイルには、2 つの XOCP 会話定義が格納されています。一方は Query Price and Availability（QPA）会話用で、もう一方は Purchase Order（PO）会話用です。会話ごとに、バイヤとサプライヤの 2 つのロールがあります。各ロールは次のものを参照します。

- 次のような BPM ワークフロー テンプレート

  `wlpi-template="WLIS_SupplierQPA"`

- 次のような WebLogic Integration BPM オーガニゼーション

  `process-implementation wlpi-org="ORG1"`

注意： WebLogic Integration の BPM コンポーネントは、以前の WebLogic Process Integrator（WLPI）です。ここで示すテンプレートおよびオーガニゼーションの名前として WebLogic Process Integrator または WLPI が使用されている場合があります。

ワークフロー テンプレートは、適用先の会話定義を参照します。たとえば、図 3-12 で取り上げている会話は、このサンプルで使用するパブリック ワークフローのプロパティを示しています。

このサンプルを実装するためのワークフローの詳細については、3-19 ページの「ビジネス プロセスおよびワークフローのモデル化」を参照してください。

オーガニゼーションは、さまざまなビジネス エンティティ、地理的な位置、または会社の特定のビジネスに関連する分類を表します。BPM オーガニゼーションの詳細については、『*BPM クライアント アプリケーション プログラミング*』の「セキュリティ レルムのコンフィグレーション」を参照してください。

次のリストは、BulkloaderData.xml ファイルの抜粋です。WLIS_QPAConversation を定義します。

コード リスト 3-2 BulkLoaderData.xml ファイルの会話定義

```
...
<conversation-definition
        name="WLIS_QPAConversation"
        version="1.1"
        business-protocol-name="XOCP"
        protocol-version="1.1">
        <role
            name="Buyer"
            wlpi-template="WLIS_BuyerQPAPublic">
            <process-implementation wlpi-org="ORG1" />
        </role>
        <role
            name="Supplier"
            wlpi-template="WLIS_SupplierQPAPublic">
            <process-implementation wlpi-org="ORG1" />
            </role>
</conversation-definition>

...
```

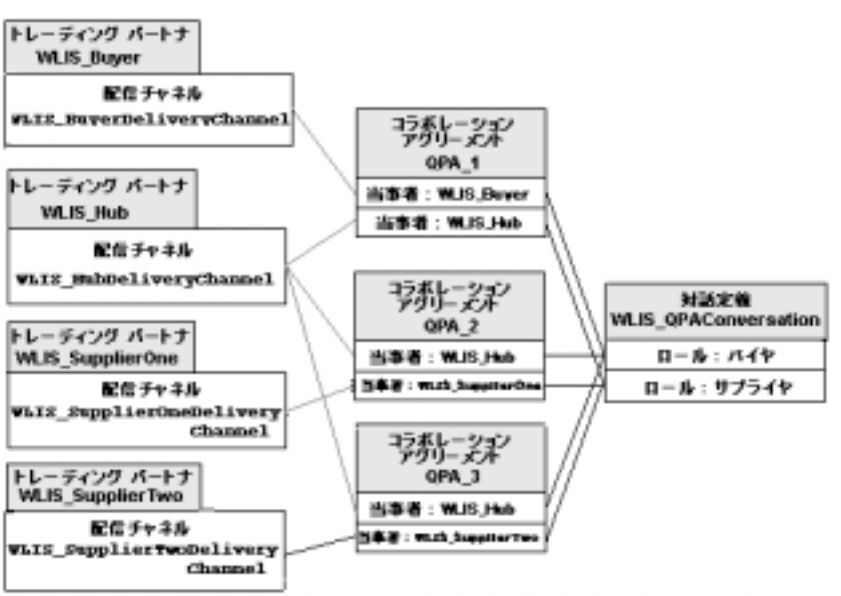
コラボレーション アグリーメント

このサンプルでは、QPA 会話と PO 会話でそれぞれ 3 つずつ、合計 6 つのコラボレーション アグリーメントを使用します。各会話用として、次の組み合わせのビジネス エンティティ間でコラボレーション アグリーメントが定義されています。

- WLIS_Buyer と WLIS_Hub
- WLIS_SupplierOne と WLIS_Hub
- WLIS_SupplierTwo と WLIS_Hub

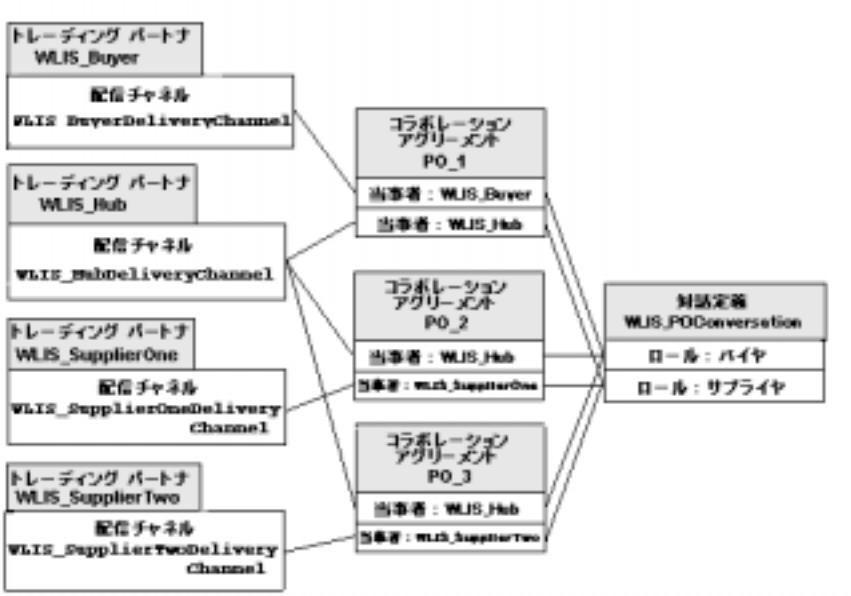
次の図は、QPA 会話（WLIS_QPAConversation）に参加するトレーディング パートナ間のコラボレーション アグリーメントを示しています。

図 3-1　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



次の図は、PO 会話（WLIS_POConversation）に参加するトレーディング パートナ間のコラボレーション アグリーメントを示しています。

図 3-2 PO 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



上の図の次の点に注目してください。

- WLIS_Hub トレーディング パートナは、各コラボレーション アグリーメントのパーティになっています。このトレーディング パートナは、QPA_1 および PO_1 コラボレーション アグリーメントではサプライヤ（代理）のロールが割り当てられ、その他のコラボレーション アグリーメントではバイヤ（代理）のロールが割り当てられています。

  たとえば、WLIS_Hub トレーディング パートナは、WLIS_Buyer から QPA メッセージを受信する場合、QPA_1 コラボレーション アグリーメントの代理サプライヤの役割を果たします。QPA_2 および QPA_3 コラボレーション アグリーメントでは、ロールを変えて、代理バイヤとなります。

- コラボレーション アグリーメントのパーティは、会話定義のロールに関連付けられます。コラボレーション アグリーメントの各パーティにロール名をコンフィグレーションする必要があります。

次のリストは、BulkLoaderData.xml ファイルの抜粋です。この抜粋部分では、WLIS_Hub と WLIS_Buyer との間のコラボレーション アグリーメントが記述されています。

コード リスト 3-3 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="WLIS_QPAConversation|1.1|WLIS_Buyer|WLIS_Hub"
global-identifier="WLIS_QPAConversation|1.1|WLIS_Buyer|WLIS_Hub
version="1.1"
status="ENABLED"
conversation-definition-name="WLIS_QPAConversation"
conversation-definition-version="1.1">
<party
        trading-partner-name="WLIS_Buyer"
        party-identifier-name="WLIS_BuyerPartyId"
        delivery-channel-name="WLIS_BuyerDeliveryChannel"
        role-name="Buyer"/>
<party
        trading-partner-name="WLIS_Hub"
        party-identifier-name="WLIS_HubPartyId"
        delivery-channel-name="WLIS_HubDeliveryChannel"
        role-name="Supplier"/>
</collaboration-agreement>

...
```

WebLogic Integration B2B Console の使用

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。WebLogic Integration 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（3-5 ページの「リポジトリ データのロード」を参照）。また、B2B Console を使用して、サンプルの実行中に進行している会話をモニタすることもできます。

WebLogic Integration B2B Console 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。

注意： 既に実行中の場合は、2-1 ページの「サンプルの設定と実行」の説明に従って WebLogic Integration サンプルを実行します。

- Windows システム上で B2B Console を起動するには、次のいずれかを実行します。
  - サンプルを起動した起動用 Web ページ（`http://localhost:7001`）で、[`Administration Consoles`] の下の [`B2B`] リンクをクリックします。
  - 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 7.01 ｜ B2B Console] を選択します。
  - 次のように、コマンド ラインから `startB2bconsole` 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 をインストールしたディレクトリの `bin` ディレクトリに移動します。たとえば、次を入力します。

    ```
    cd WLI_HOME\bin
    ```

    c. 次のように入力して B2B Console を起動します。

    ```
    startB2bconsole
    ```
- UNIX システム上で B2B Console を起動するには、次のいずれかを実行します。
  - サンプルを起動した起動用 Web ページ（`http://localhost:7001`）で、[`Administration Consoles`] の下の [`B2B`] リンクをクリックします。
  - Web ブラウザを起動して次の URL を入力します。

    ```
    http://localhost:7001/b2bconsole
    ```

次の図は、WebLogic Integration サンプルのデータがロードされた状態の WebLogic Integration B2B Console を示しています。

図 3-3　サンプル データを表示した状態の WebLogic Integration B2B Console

WebLogic Integration B2B Console を使用して B2B 統合をコンフィグレーションする方法については、『*B2B Integration Administration Console オンライン ヘルプ*』と『*B2B Integration 管理ガイド*』を参照してください。Bulk Loader の詳細については、『*B2B Integration 管理ガイド*』の「Bulk Loader の操作」を参照してください。

ビジネス プロセスおよびワークフローのモデル化

この節では、WebLogic Integration の Business Process Management (BPM) の機能を簡単に紹介してから、Studio の使い方と、WebLogic Integration サンプルで実装されている 2 つのビジネス プロセスの Query Price and Availability (QPA) および Purchase Order (PO) について説明します。この章の内容は以下のとおりです。

- BPM の概要
- WebLogic IntegrationStudio の使い方
- QPA ビジネス プロセス
- PO ビジネス プロセス

BPM の概要

会話定義（3-12 ページの「会話定義」を参照）でトレーディング パートナに割り当てられるロールを実装するワークフローは、*協調的ワークフロー*と呼ばれます。

ワークフロー テンプレートはワークフローの見本で、さまざまなワークフロー テンプレート定義（バージョン）の実装を組み合わせたものです。ワークフロー テンプレートは、WebLogic Integration Studio を使用して設計および編集します。複数の BPM プラグインを使用して、Studio の機能を拡張できます。

- B2B Integration プラグイン－ B2B 統合、つまり協調的ワークフローの設計および管理をサポートします。Studio を使用すると、ワークフローにプロパティを割り当てることができます。それらのプロパティを割り当てることで、ワークフローが B2B integration 環境で使用できるようになります。
- Application Integration プラグイン－バックエンドのシステムおよび従来のエンタープライズ情報システム（EIS）を統合可能なワークフローを設計できるようにします。

- Data Integration プラグイン－異種 EIS アプリケーション間でのデータ共有を可能にすることで、異種データ形式を統合するワークフローを設計できるようにします。

サンプル シナリオでは、トレーディング パートナは、プライベート ワークフローと協調的ワークフローの両方を実装しています。プライベート ワークフローは協調的ワークフローと連携して動作し、トレーディング パートナのローカル プロセスを実装します。会話定義では、ローカルおよびプライベート プロセスを指定する必要はありません。たとえば、トレーディング パートナが会話を開始する場合、そのトレーディング パートナのプライベート ワークフローが、その会話を開始する協調的ワークフローを開始します。

以下の節では、サンプルのビジネス トランザクションのバイヤサイドおよびサプライヤサイドにおけるビジネス プロセスの実装について説明します。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、特に強調されています。

WebLogic IntegrationStudio 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。WebLogic Integration 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グレーションを調べたりする場合には Studio が役立ちます。また、Studio を使用して、サンプルの実行中にワークフローをモニタすることもできます。

この節では、Studio の起動手順および使い方と、ビジネス プロセスの管理で使用するサンプルのコンポーネント リストについて説明します。

Studio の起動

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 : admin、パスワード : security）。

- 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 をインストールしたディレクトリの bin ディレクトリに移動します。たとえば、次を入力します。

```
cd WLI_HOME\bin
```

c. 次のように入力して studio コマンドを実行します。

```
studio
```

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

- 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の bin ディレクトリに移動します。たとえば、次を入力します。

```
cd WLI_HOME/bin
```

b. 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
```

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

Studio でのワークフロー テンプレートの表示

Studio でワークフロー テンプレートとそのプロパティを表示するには、次の手順を実行します。

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。このサンプルに関与するテンプレート定義は、表 3-3 の「WebLogic Integration リポジトリにインポートされるコンポーネント」に示

すとおりです。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。2-3 ページの「ステップ 1: 起動用 Web ページをコンフィグレーションして呼び出す」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。

サンプル内の BPM コンポーネント

このサンプル アプリケーション用として Studio およびプロセス エンジンで必要なワークフロー テンプレートおよびその他のデータは、サンプルをコンフィグレーションするときに、`workflow.jar` ファイルによって WebLogic Integration リポジトリにロードされます。インポートされるコンポーネントは、次の表のとおりです。

表 3-3 WebLogic Integration リポジトリにインポートされるコンポーネント

コンポーネント	名前

ビジネス オペレーション	`binary to file`
	`file to binary`
	`xmlToFile`
	`create POAck from PO`

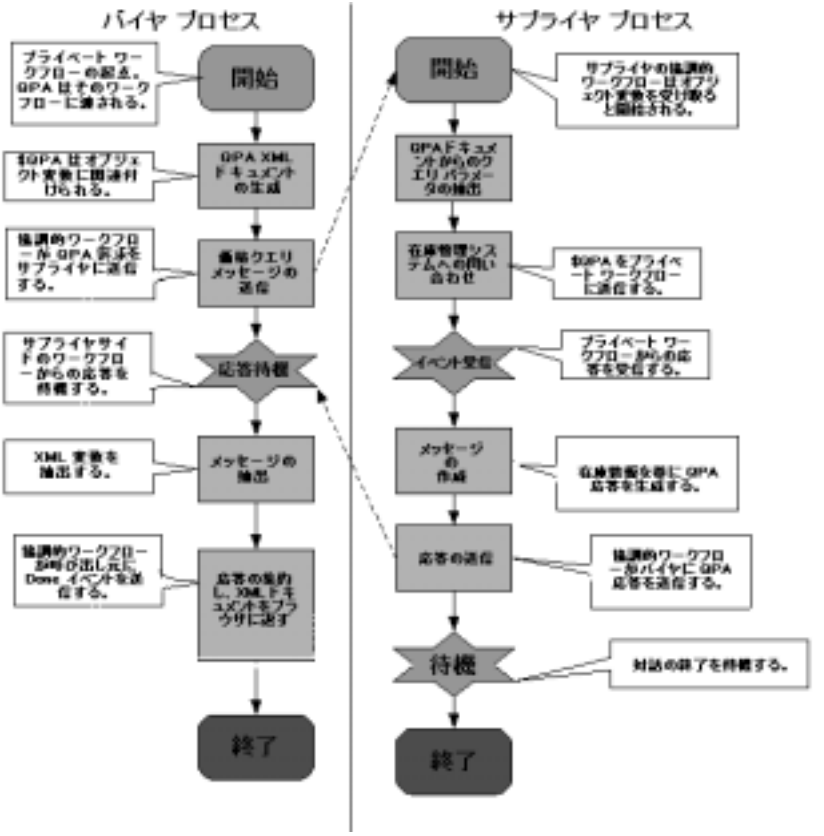
表 3-3 WebLogic Integration リポジトリにインポートされるコンポーネント

コンポーネント	名前
テンプレート定義	WLIS_BuyerPOPrivate WLIS_BuyerPOPublic WLIS_BuyerQPAPrivate WLIS_BuyerQPAPublic WLIS_SupplierOnePOPrivate WLIS_SupplierOneQPAPrivate WLIS_SupplierPOPublic WLIS_SupplierQPAPublic WLIS_SupplierTwoPOPrivate WLIS_SupplierTwoQPAPrivate
XML リポジトリ フォルダ	com.bea.wlxt.MFL wlis
XML リポジトリ エンティティ	PO.mfl PO.dtd POAck.mfl POAck.dtd
イベント キー	PORequest AggregatedQPAResponse PurchaseOrderAcknowledgement

QPA ビジネス プロセス

金属ボックスの供給不足のため、GCS (バイヤ トレーディング パートナ) は、選択したサプライヤにそのボックスの QPA メッセージを送信します。次の図は、QPA ビジネス プロセスのイベント フローを示しています。

図 3-4　QPA ビジネス プロセスのプロセス フロー



上の図のイベントをまとめると、次のようになります。

1. バイヤ トレーディング パートナが QPA を作成します。
2. QPA がサプライヤに送信されます。
3. サプライヤが QPA を処理して、応答を生成します。
4. バイヤ トレーディング パートナがサプライヤからの応答を集約します。

注意： 上の図は、QPA ビジネス プロセスの高レベルのビューを示しています。各サイドのプロセスは、パブリック（協調的）ワークフローとプライベート ワークフローによって実装されます。以下の節では、これらのワークフローについて説明します。

- QPA 実装の概要
- バイヤサイドの実装
- サプライヤサイドの実装

QPA 実装の概要

このシナリオでは、各トレーディング パートナは、QPA プロセスのプライベート ワークフローおよびパブリック ワークフローを実装しています。サンプルの QPA プロセスでは、以下の 5 つのワークフロー テンプレートが使用されます。

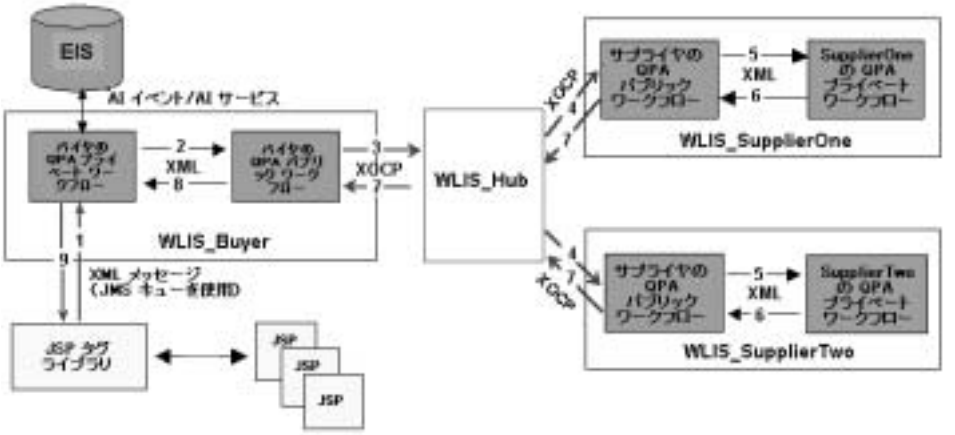
表 3-4 サンプル QPA プロセスのワークフロー

ロール	パブリック / プライベート	ワークフロー名
バイヤ	プライベート	`WLIS_BuyerQPAPrivate`
バイヤ	パブリック	`WLIS_BuyerQPAPublic`
サプライヤ	パブリック	`WLIS_SupplierQPAPublic` **注意：** シナリオのバイヤはどちらも同じパブリック ワークフローを使用します。
サプライヤ	プライベート	`WLIS_SupplierOneQPAPrivate`
サプライヤ	プライベート	`WLIS_SupplierTwoQPAPrivate`

WebLogic Integration は、ビジネス パートナ間のビジネス会話およびコラボレーション アグリーメントを管理し、バイヤとサプライヤ間のビジネス メッセージの交換を自動化します。コラボレーション アグリーメントおよび会話で取り上げるワークフローについては、3-6 ページの「リポジトリ データについて」で説明します。

このサンプルでは、JSP と JSP タグ ライブラリを使用して、QPA プロセスを開始し、QPA 要求と応答データを表示します。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 3-5　QPA ビジネス プロセスのデータ フロー



バイヤサイドおよびサプライヤサイドの実装については、3-28 ページの「バイヤサイドの実装」と 3-50 ページの「サプライヤサイドの実装」でさらに詳しく説明しています。

トレーディング パートナおよびワークフロー間のデータ フローをまとめると、次のイベント シーケンスになります。

1. QPA フォームを含む JSP (図 2-2 を参照) から、QPA 要求が JMS キューに送信され、`WLIS_BuyerQPAPrivate` ワークフローが開始されます。
2. `WLIS_BuyerQPAPrivate` ワークフローは、QPA 要求 XML ドキュメントを渡して、`WLIS_BuyerQPAPublic` ワークフローを呼び出します。このワークフローは、QPA 会話を開始します。
3. `WLIS_BuyerQPAPublic` ワークフローは、トレーディング パートナ WLIS_Buyer と WLIS_Hub 間のコラボレーション アグリーメントに基づいて、QPA 要求 XML を XOCP メッセージにパックし、WLIS_Hub トレーディング パートナに送信します。

注意： WLIS_Hub は、メッセージを受信する場合には、そのコラボレーション アグリーメントの代理サプライヤの役割を果たします。

4. WLIS_Hub トレーディング パートナは、自身と各サプライヤ間で登録されているコラボレーション アグリーメントに基づいて、送り先のトレーディング パートナ（WLIS_SupplierOne および WLIS_SupplierTwo）にメッセージをルーティングします。

   注意： このステップでは、WLIS_Hub トレーディング パートナはロールを変えて、自身とサプライヤ間のコラボレーション アグリーメントの代理バイヤとなります。

   各サプライヤのパブリック ワークフローは、XOCP メッセージを受信することで開始されます。このシナリオでは、WLIS_SupplierOne と WLIS_SupplierTwo は、パブリック ワークフロー（`WLIS_SupplierQPAPublic`）を共有しています。パブリック ワークフローは、メッセージから QPA 要求 XML ドキュメントを抽出します。

5. `WLIS_SupplierQPAPublic` ワークフローは、各サプライヤのプライベート ワークフローを呼び出し、QPA 要求 XML ドキュメントを渡します。

6. 各サプライヤのプライベート ワークフローは、独自の QPA 応答（XML ドキュメント）を作成し、パブリック ワークフローの戻り変数にアタッチします。

7. `WLIS_SupplierQPAPublic` ワークフローは、QPA 応答 XML ドキュメントを抽出し、XOCP メッセージにパックして、バイヤに送信します。

   WLIS_Hub トレーディング パートナは、WLIS_Buyer のルーティング プロキシの役割を果たします。サプライヤ トレーディング パートナが（WLIS_Hub と各サプライヤ トレーディング パートナ間のコラボレーション アグリーメントに基づいて）WLIS_Hub に応答メッセージを送信する場合、WLIS_Hub は代理バイヤの役割を果たします。

   次に、WLIS_Hub は代理サプライヤのロールに変わり、WLIS_Hub と WLIS_Buyer 間のコラボレーション アグリーメントに基づいて、応答メッセージをバイヤ（WLIS_Buyer）にルーティングします。

8. バイヤのワークフロー（`WLIS_BuyerQPAPublic`）は以下の処理を行います。

   a. 受信した XOCP メッセージから QPA 応答 XML ドキュメントを抽出します。

b. 両方のサプライヤの応答ドキュメントを 1 つの XML ドキュメントとして集約し、JMS キューを使用してバイヤのプライベート ワークフロー（WLIS_BuyerQPAPrivate）にポストします。

c. QPA 会話を終了し、サプライヤのパブリック ワークフロー（WLIS_SupplierQPAPublic）に通知します。

9. バイヤのプライベート ワークフロー（WLIS_BuyerQPAPrivate）は、集約された QPA 応答 XML ドキュメントを受信し、XML ファイルに書き込みます。JSP は XML を解析し、集約された QPA 応答を 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。

バイヤサイドの実装

このソリューションを実装するため、バイヤ（GCS）は、サンプルの操作、メッセージの処理、および JMS を使用した WebLogic Integration プロセス エンジンとの XML メッセージの交換を行うカスタム クライアント（Web ユーザ インタフェース）を実装しています。また、GCS では、バイヤのバックエンド プロセスを管理するプライベート ワークフローと、QPA 会話でメッセージの交換を管理するパブリック ワークフローも実装しています。この節では、以下のコンポーネントについて説明します。

- バイヤサイドの Web ユーザ インタフェース
- バイヤの QPA プライベート ワークフロー
- バイヤの QPA パブリック ワークフロー

バイヤサイドの Web ユーザ インタフェース

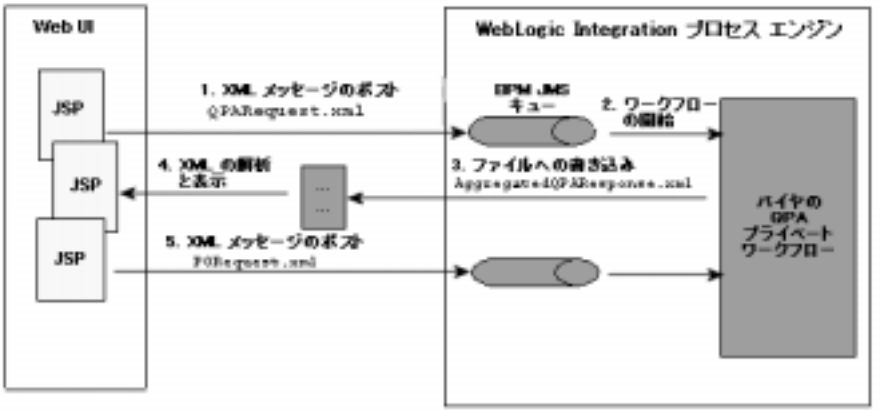
QPA プロセスを開始し、要求および応答データを Web ブラウザに表示するには、Java Server Page (JSP) と JSP タグ ライブラリが使用されます。QPA プロセスに関連するソース ファイルは、次の表に示したディレクトリにあります。*SAMPLES_HOME* は WebLogic Platform のサンプル ディレクトリを示します。

表 3-5 QPA プロセスのソース ファイル

WebLogic Integration のインストール ディレクトリ内の場所	ソース ファイル
SAMPLE_HOME\integration\samples\wlis\src\examples\wlis\tags	■ SendQPARequestTag.java ■ CheckQPAResponseTag.java
SAMPLE_HOME\integration\samples\wlis\web	■ SendQPARequest.jsp ■ CheckQPAResponse.jsp ■ QPAform.htm ■ WaitQPAResponse.htm
SAMPLE_HOME\integration\samples\wlis\lib\xsl	ProcessQPAResponse.xsl

次の図に示すように、バイヤのプライベート ワークフロー（WLIS_BuyerQPAPrivate）との会話には Web ユーザ インタフェースを使用します。

図 3-6 Web ユーザ インタフェースとバイヤのプライベート ワークフロー間の会話



上の図の会話をまとめると、次のイベント シーケンスになります。

1. QPA 要求が、HTML Web フォームの入力（2-10 ページの「ステップ 4: QPA 要求を送信する」を参照）に基づいて、XML として作成されます。XML が、JSP (SendQPARequest.jsp) を使用して、JMS キューに送信されます。

   次のリストは、JMS キューの名前と XML メッセージのポスト先のキュー接続ファクトリを定義する SendQPARequestTag.java ファイルのコードです。

コード リスト 3-4 SendQPARequestTag.java

```
// JMS 接続ファクトリの定義
final String JMS_FACTORY = "com.bea.wlpi.QueueConnectionFactory";

// JMS キューの定義
final String QUEUE = "com.bea.wlpi.EventQueue";
...
```

2. バイヤのプライベート ワークフロー（WLIS_BuyerQPAPrivate）が開始されます。
3. WLIS_BuyerQPAPrivate ワークフローは、両方のサプライヤからの応答を受信して集約し、ローカル ファイルの AggregatedQPAResponse.xml に QPA 応答を書き込みます。
4. JSP (CheckQPAResponse.jsp) は JSP タグを使用してファイルを読み出し、XSL（*SAMPLE_HOME*\integration\samples\wlis\lib\xsl\processQPAResponse.xsl を参照）を使用して XML データを解析して、結果を Web ブラウザに表示します。図 2-5、2-12 ページの「[PO の作成] ウィンドウ」を参照してください。

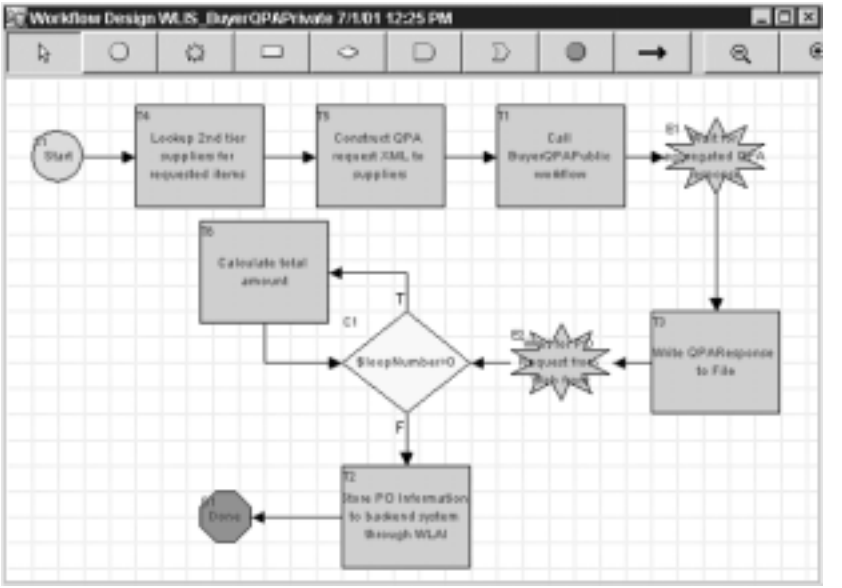
   注意： *SAMPLES_HOME* は WebLogic Platform のサンプル ディレクトリを示します。
5. サンプルを実行する場合、バイヤはサプライヤを選択します。2-12 ページの「ステップ 6: 発注書を作成する」を参照してください。JSP は、選択したサプライヤの QPA 応答データを格納した XML メッセージを JMS トピックにポストします。バイヤのプライベート ワークフロー（WLIS_BuyerQPAPrivate）は、そのメッセージを受信して処理します。

バイヤの QPA プライベート ワークフロー

WLIS_BuyerQPAPrivate ワークフローは、QPA 応答データを格納する JMS メッセージを JSP から受信すると開始されます。ワークフローと Web ユーザ インタフェースについては、3-28 ページの「バイヤサイドの Web ユーザ インタフェース」を参照してください。

次の図は、WLIS_BuyerQPAPrivate ワークフロー テンプレートを示しています。

図 3-7 WLIS_BuyerQPAPrivate 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 WLIS_BuyerQPAPrivate ワークフロー テンプレート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
- Lookup 2nd Tier suppliers for requested items
- Construct QPA Request XML to Suppliers
- Call BuyerQPAPublic workflow
- Wait for aggregated QPA Response

- Write QPAResponse to File
- Wait for PO Request from Web Front
- 分岐
- Calculate total amount
- Store PO Information to backend system through WLAI

開始

`SendQPARequest.jsp` JSP から XML イベントを受信すると、ワークフローが開始されます。

XML は、`QPARequest.dtd` ファイルで定義されます。付録 A「DTD」には、サンプルの DTD ファイルが記載されています。`QPARequest.dtd` の最初の要素に注目してください。

`QPARequestId: (<!ELEMENT QPARequestId (#PCDATA)>)`

サンプル シナリオの場合、`QPARequestId` 要素の一意な値は、バイヤの [QPA リクエストの送信] Web ページ（2-9 ページの「ステップ 3: サンプルを起動する」を参照）から QPA 要求が生成されるたびに、JSP によって作成されます。サンプルを実行するたびに `QPARequestId` に対して作成される値を確認するには、バイヤに返された QPA 応答を格納するフォームを参照します（2-11 ページの「ステップ 5: QPA 要求を確認する」を参照）。

開始ノードでは、XPath 式を使用して XML メッセージを抽出し、ワークフロー変数（`qpaRequestXML`）に格納します。また、開始では、`QPARequestId` 要素の値を抽出し、ワークフロー変数の `QPARequestId` に割り当てます。2 番目の変数は、*イベント キーの キー値表現*としてこのサンプルの他のワークフローで使用されます。

イベント キーとは

WebLogic Integration プロセス エンジンは、[開始のプロパティ] または [イベントのプロパティ] ダイアログ ボックスの DOCTYPE またはルート要素が着信 XML メッセージのものと一致しない限り、ワークフローまたはイベント ノードを開始しません。DOCTYPE またはルート要素を使用するだけでなく、イベント キーを使用することで、ワークフローまたはイベント ノードの開始を修飾できます。イベント キーを使用すると、開始またはイベント ノードを開始する XML メッセージの内容を指定できます。

イベント キーは、*キー値表現*と*イベント キー表現*という 2 つの部分で構成されます。キー値表現は、ノードを呼び出すために受け取ったドキュメントが含まなければならないデータを正確に指定するワークフロー式です。イベント キー表現は、対象のデータを見つける XML メッセージの要素を正確に指定する式です。

イベント キー表現をコンフィグレーションしたり、このサンプルのコンフィグレーションを確認したりするには、Studio のタスク メニューから [コンフィグレーション | イベント] を選択します。

このサンプル シナリオでは、次のイベント キー表現がコンフィグレーションされています。

イベント キー表現

```
PORequest.QPARequestId

AggregatedQPAResponse.QPARequestId

PurchaseOrderAcknowledgement.PONumber
```

イベント キー表現をコンフィグレーションしたら、オーガニゼーション内のすべてのワークフローで利用可能になります。イベント キー表現は、[開始のプロパティ] または [イベントのプロパティ] ダイアログ ボックスで定義したキー値表現に指定した値と対応している必要があります。その結果、プロセス エンジンは実行時に 2 つの値を比較して、一致しているかどうかを判定できます。

このマニュアルの他の節では、ワークフローのノードに対して `QPARequestId` キー値表現が使用されています。Studio ツールのイベント キーを定義および使用する方法については、『*WebLogic Integration Studio ユーザーズ ガイド*』の「ワークフロー リソースのコンフィグレーション」を参照してください。

Lookup 2nd Tier suppliers for requested items

このサンプル シナリオで第 2 層（またはカテゴリ 2）と呼ぶサプライヤ群は、バイヤのエンタープライズ情報システム（EIS）で定義されます。サンプルの EIS は RDBMS です。これらのサプライヤのデータは、`RunSamples` スクリプトを実行してサンプルのデータベースを設定（2-2 ページの「サンプルの実行」を参照）するときに、WebLogic Integration リポジトリにロードされます。これらのサプライヤには、サンプル シナリオのケースのように、ある品物の一次サプライヤが需要を満たせない場合にアクセスします。

アプリケーション ビュー（WLISAppView.sav）もこのサンプル用にデプロイされます。アプリケーション ビューの詳細については、3-84 ページの「Application Integration」を参照してください。Application Integration プラグインを使用すると、Studio の機能が拡張され、アプリケーション ビュー サービスとイベントをワークフロー テンプレートに統合できるようになります。Studio からは、呼び出すサービスを選択して、サービス要求および応答変数（XML 形式）を指定することができます。

このタスク ノードでは、以下のアクションを実行します。

1. ワークフロー変数（supplierLookupInputXML）を作成します。

2. WLISAppView.sav アプリケーション ビューで getSupplier を呼び出し、第 2 層サプライヤに関する情報を EIS から取り出します。

呼び出すサービスをワークフローから選択する方法は、次のとおりです。

1. タスク ノードをダブルクリックして、[タスク プロパティ] ダイアログ ボックスを表示します。

2. [追加] をクリックして、[アクションを追加] ダイアログ ボックスを表示します。

3. [AI アクション｜アプリケーション ビュー サービスの呼び出し] を選択して、[サービスの呼び出し] ダイアログ ボックスを表示します。このダイアログ ボックスには、次の図に示すように、WLISAppView.sav アプリケーション ビューのサービスのリストが表示されます。

図 3-8 [アプリケーション ビュー サービスの呼び出し] ダイアログ ボックス

上の図の [要求ドキュメント変数] フィールドの値がこのノードで作成された XML 変数（`supplierLookupInputXML`）となっていることに注目してください。

このノードでは、アプリケーション ビュー サービスが同時に呼び出されます。したがって、ワークフローはサービスが応答ドキュメントを返すのを待ってから続行します。

[応答ドキュメント変数] フィールドの値は、`supplierLookupOutputXML` です。この変数は、アプリケーション ビュー サービスから応答を受け取ります。`supplierLookupOutputXML` 変数は、要求 XML メッセージを作成するためにワークフローの次のノードで使用されます。

注意： 入力または応答時の XML スキーマを調べるには、それぞれ [要求定義の表示] または [応答定義の表示] をクリックします。[定義を表示] ダイアログ ボックスが表示されます。必要でなくなったら [閉じる] をクリックします。

WebLogic Integration で提供される Application Integration プラグインの使い方については、『*Application Integration ユーザーズ ガイド*』を参照してください。

Construct QPA Request XML to Suppliers

このタスク ノードでは、サプライヤの QPA 要求 XML メッセージを保持するワークフロー変数（`qpaRequestXMLOne` と `qpaRequestXMLTwo`）を作成します。これらの XML メッセージは、XOCP ビジネス メッセージの一部としてバイヤのパブリック ワークフロー（`WLIS_BuyerQPAPublic`）からサプライヤのパブリック ワークフロー（`WLIS_SupplierQPAPublic`）に送信されます。

サプライヤ トレーディング パートナは WebLogic Integration の 1 つのインスタンスに配置され、このサンプル用に同じパブリック ワークフローを共有するので、着信 XOCP メッセージの宛先を正しく識別しなければなりません。

そのため、一意の属性値（***`SupplierName`***）が QPA 要求 XML のルート要素に追加されます。`SupplierName` の値は、ワークフローで前にあるノード（`supplierLookupOutputXML`）の `CallApplicationViewService:sav.getSupplier` イベントの結果として返される変数から作成されます。たとえば、`qpaRequestXMLOne` ワークフロー変数の `SupplierName` 属性の値は、XPath 式に基づいて生成されます。

```
ToString(XPath("//Row[1]/supplierName/text()",
$supplierLookupOuputXML))
```

`qpaRequestXMLOne` と `qpaRequestXMLTwo` の属性および要素を確認するには、次の手順に従います。

1. このノードをダブルクリックして、[タスク 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 | アクティブ化] を選択します。

3. 次のいずれかのアクションをダブルクリックします。

`Set workflow variable qpaRequestXMLOne structure`

`Set workflow variable qpaRequestXMLOne structure`

次の図に示す [XML 構造] ダイアログ ボックスが表示されます。

図 3-9 [ワークフロー変数を設定] ダイアログ ボックス

Call BuyerQPAPublic workflow

このタスク ノードでは、WLIS_BuyerQPAPublic ワークフローを開始して、QPA 要求 XML ドキュメントを格納したワークフロー変数をパブリック ワークフローに渡します。WLIS_BuyerQPAPrivate ワークフローは WebLogic Integration B2B 会話（WLIS_QPAConversation）のイニシエータなので、このタスク ノードでは、デフォルトの ワークフローを開始 の代わりに、B2B Integration プラグインによって提供される特殊なパブリック ワークフローを開始 アクションを使用します。

パブリック ワークフローを開始 アクションと ワークフローを開始 アクションとの主要な違いは、パブリック ワークフローを開始 アクションを使用する場合にワークフロー テンプレートが動的にバインドされることです。WLIS_BuyerQPAPrivate ワークフローでは、パブリック ワークフローを開始 アクションで定義した会話プロパティに基づいて、実行時に呼び出すワークフロー テンプレートが決定されます。

また、B2B エンジンは、パブリック ワークフローを開始 呼び出しの一部として渡された会話情報に基づいて、WebLogic Integration リポジトリに格納されているコラボレーション アグリーメント情報を使用して、呼び出されたワークフローを同じコラボレーション アグリーメントの他のパブリック ワークフローに関連付けます。

WebLogic Integration Studio でパブリック ワークフローを開始 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 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 | 統合アクション | B2B Integration | パブリック ワークフローを開始] を選択して、[パブリック ワークフローを開始] ダイアログ ボックスを表示します。

このノードに対して既に指定されている パブリック ワークフローを開始 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 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 | アクティブ化] を選択します。

3. [パブリック ワークフローを開始] をダブルクリックして、次の図に示す [パブリック ワークフローを開始] ダイアログ ボックスを表示します。

図 3-10　[パブリック ワークフローを開始] ダイアログ ボックス

上の図で、WebLogic Integration サンプルの以下のプロパティに注目してください。

- [会話] タブ－ WebLogic Integration では、このタブで指定した情報を使用して、指定された会話の指定されたパーティ間のアグリーメントを定義するアクティブなコラボレーション アグリーメントをリポジトリ内で見つけます。
  - 会話名、バージョン、およびロールは、それぞれ WLIS_QPAConversation、1.1、および buyer として定義されています。
  - 会話のパーティは、トレーディング パートナの名前（WLIS_Buyer と WLIS_Hub）で指定されています。関係のあるコラボレーション アグリーメントの検索で使用する値を、[名前] フィールド以外にも、[ロール] および [配信チャネル] フィールドに任意で追加することもできます。

  BulkLoaderData.xml ファイルの抜粋については、コード リスト 3-3 の「BulkLoaderData.xml ファイルのコラボレーション アグリーメント」を参照してください。この抜粋部分は、プロセス エンジンがリポジトリ内で検索し、[会話] タブの指定内容を満たすコラボレーション アグリーメントを示しています。

- [ワークフロー] タブ－呼び出されるワークフローに渡す変数は、[ワークフロー] タブで定義します。ここで指定する変数名は、呼び出されるワークフロー（WLIS_BuyerQPAPublic）に対する入力変数としても指定されます。変数は以下のとおりです。
  - QPARequestXMLOne －ワークフローで前にあるノードで定義された qpaRequestXMLOne 変数の値を格納します。
  - QPARequestXMLTwo －ワークフローで前にあるノードで定義された qpaRequestXMLTwo 変数の値を格納します。
  - QPAPrivateFlowId －パブリック ワークフローを呼び出す WLIS_BuyerQPAPrivate ワークフローのインスタンスを識別するワークフロー インスタンス ID です。ワークフロー テンプレートの複数のインスタンスが実行されている場合があります。このインスタンス ID をパブリック ワークフローで使用する場合については、3-49 ページの「Publish Aggregated QPA Response」を参照してください。

このノードでは、サブワークフロー（WLIS_BuyerQPAPublic）も非同期で呼び出されます。[タスク プロパティ] ダイアログ ボックスで [アクション｜アクティブ化] を選択すると、以下のアクションがこの順序で指定されます。

1. Start Public Workflow
2. Mark task ìCallBuyerPublic workflowì done

B2B 会話のワークフローの詳細については、『*B2B Integration ワークフローの作成*』を参照してください。

Wait for aggregated QPA Response

このイベント ノードのワークフローでは、`WLIS_BuyerQPAPublic` ワークフローからの指定した XML イベントを待機します。XML イベントを受け取ると、XPath 式を使用して XML を抽出し、ワークフロー変数に格納します。XML は、`AggregatedQPAResponse.dtd` ファイルで定義されます。付録 A「DTD」には、サンプル アプリケーションの DTD が記載されています。

このノードでは、このノード用に定義されたキー値表現の `QPARequestId` を使用します。この表現を使用して、ノードを呼び出すために受け取ったドキュメントが含まなければならないデータを正確に指定します。

プライベート ワークフロー（`WLIS_BuyerQPAPrivate`）のインスタンスが多数ある場合、適切なワークフロー テンプレート インスタンスの適切な Wait for aggregated QPA Response ノードを、QPA プロセスのこの時点で呼び出す必要があります。呼び出すノードを格納するワークフロー インスタンスは、`QPARequestId` の値で指定されます (3-32 ページの「開始」のイベント キーの説明を参照)。

Write QPAResponse to File

このアクション ノードでは、`xmlToFile` ビジネス オペレーションを使用して、両方のサプライヤからの QPA 応答を、WebLogic Platform のインストール ディレクトリの次のローカル ファイルに書き込みます。

`SAMPLE_HOME``\integration\samples\data\AggregatedQPAResponse.xml`

上の行の *`SAMPLES_HOME`* は、WebLogic Platform のサンプル ディレクトリです。

このローカル ファイルは JSP によって処理されます。

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。WebLogic Platform の

インストール ディレクトリの次の場所には、xmlToFile ビジネス オペレーションに関連する Java クラスがあります。

SAMPLE_HOME\integration\samples\wlis\src\examples\wlis\common\util\Utils.java

SAMPLES_HOME は WebLogic Platform のサンプル ディレクトリを示します。

Wait for PO Request from Web Front

[PO Request] Web ページから PO 要求を受け取るまでワークフローを待機させるイベント ノードです。PO 要求 XML メッセージは、このノードで poRequestXML 変数に割り当てられます。このノードでは、その他に 3 つの変数が割り当てられます。これらの変数（loopNumber、counter、および totalAmount）は、ワークフローの次のノードで使用されます。

分岐

このノードでは、評価対象となる条件を指定します。ワークフローがたどる経路は、評価結果によって決まります。条件の評価が true の場合、つまり loopNumber の値がゼロより大きい場合、ワークフローは Calculate total amount ノードに進みます。評価結果が false の場合、ワークフローは Store PO information to the back-end system through WLAI ノードに進みます。

Calculate total amount

このタスク ノードでは、[ワークフロー変数を設定] ダイアログ ボックスで定義した式の結果に基づいて、3 つのワークフローに値を割り当てます。最初に計算される値は、要求された項目の数量および単価です。注文の総額は、数量と単価の積となります。

Store PO Information to backend system through WLAI

このタスク ノードでは、WebLogic Integration の Application Integration の機能を使用して、PO 情報をエンタープライズ情報システム（このケースでは RDBMS）に入力します。Studio からは、このサンプルでもデプロイされている WLISAppView.sav アプリケーション ビューで呼び出すサービスを選択したり、サービス要求および応答変数を指定したりすることができます。なお、変数は XML 形式で指定する必要があります。

このノードから呼び出されるアプリケーション ビュー サービスの insertPOData および insertLine は、同時に呼び出されます。した

がって、ワークフローはサービスが応答ドキュメントをこのノードに返すのを待ってから完了ノードに進みます。

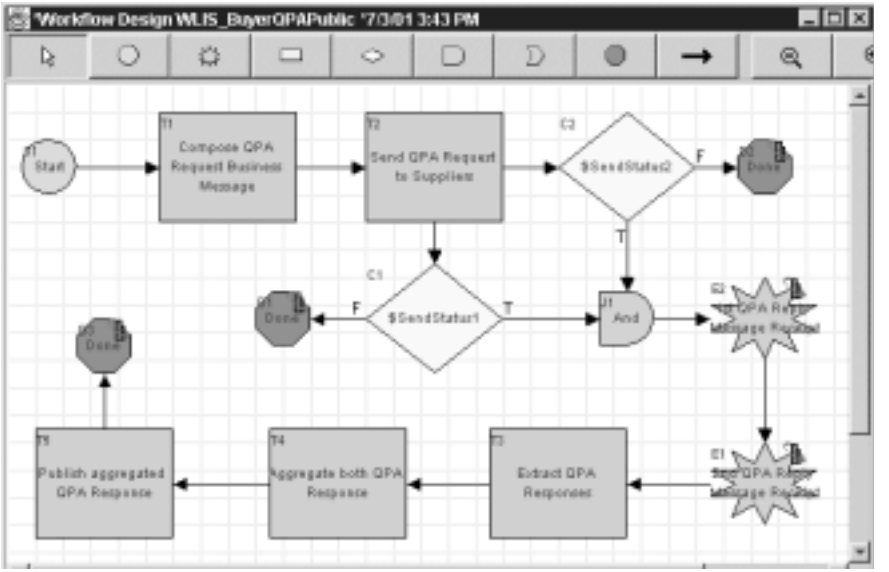
詳細については、3-66 ページの「バイヤの PO プライベート ワークフロー」を参照してください。

バイヤの QPA パブリック ワークフロー

バイヤのパブリック ワークフロー（WLIS_BuyerQPAPublic）は、バイヤのプライベート ワークフロー（WLIS_BuyerQPAPrivate）によって QPA ビジネスプロセス向けに開始されます。

次の図は、WLIS_BuyerQPAPublic ワークフロー テンプレートを示しています。

図 3-11　WLIS_BuyerQPAPublic 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 WLIS_BuyerQPAPublic ワークフロー テンプレート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
- Compose QPA Request Business Message

- Send QPA Request to Suppliers
- 1st/2nd QPA Reply Message Receipt
- Extract QPA Responses
- Aggregate both QPA Responses
- Publish Aggregated QPA Response
- 完了

注意： B2B 会話で呼び出されるワークフロー、つまり *パブリック* ワークフローに対して会話のプロパティを定義する必要があります。パブリック ワークフローは、WebLogic Integration 環境では *協調的*ワークフローとも呼ばれます。`WLIS_BuyerQPAPublic` ワークフローは、協調的ワークフローの一例です。

`WLIS_BuyerQPAPublic` ワークフロー テンプレートの会話のプロパティを確認するには、Studio の左ペインでテンプレート名を右クリックし、ドロップダウン メニューから [プロパティ] を選択します。次の図に示す [テンプレート定義] ダイアログ ボックスが表示されます。

図 3-12 WLIS_BuyerQPAPublic ワークフローの [テンプレート定義] ダイアロ

グ ボックス

会話の名前、バージョン、ロール、およびビジネス プロトコルのバインディングが [B2B Integration] タブで定義されています。このタブでは、`WLIS_BuyerQPAPublic` ワークフローが [会話開始者] として定義されています。

開始

QPA 会話を開始します。`WLIS_BuyerQPAPublic` ワークフローは、前の図で [テンプレート定義] ダイアログ ボックスで [会話開始者] として指定されています。このワークフローは、バイヤのプライベート ワークフロー（`WLIS_BuyerQPAPrivate`）によって開始されるので、呼び出されるワークフローとして定義されます。

このノードで割り当てるワークフロー変数には、QPA ビジネス メッセージの送信先のサプライヤを識別する変数（`FirstSupplierName` と `SecondSupplierName`）も含まれます。このワークフローの Send QPA Request to Suppliers ノードを参照してください。

これらの変数には、このワークフローに渡された XML メッセージから情報を取り出す XPath 式を使用して値（`QPARequestXMLOne` または

QPARequestXMLTwo）が割り当てられます。たとえば、次の XPath 式では、値として QPARequestXMLOne が割り当てられます。

```
XPath("/QPARequest/@SupplierName/text()",
$QPARequestXMLOne)
```

Compose QPA Request Business Message

このタスク ノードでは、XML ドキュメントを XOCP メッセージ ペイロードにパックすることで、XOCP ビジネス メッセージを作成します。XML はバイヤのプライベート ワークフロー（WLIS_BuyerQPAPrivate）によって作成され、QPARequestXMLOne または QPARequestXMLTwo 変数としてこのワークフローに渡されます。

こうしたメッセージを Studio で作成するには、B2B Integration プラグインで提供される B2B アクションである ビジネス メッセージの作成アクションを選択する必要があります。[ビジネス メッセージの作成] ダイアログ ボックスには次のようにしてアクセスします。

1. このノードをダブルクリックして、[タスク 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 | アクティブ化] を選択します。

3. [追加 | 統合アクション | B2B Integration | ビジネス メッセージの作成] を選択して、[ビジネス メッセージの作成] ダイアログ ボックスを表示します。

このノードでは、2 つのビジネス メッセージ、QPARequestMessageOne および QPARequestMessageTwo が作成されます。これらは、Java オブジェクト型の変数として格納されます。

Send QPA Request to Suppliers

このタスク ノードで定義したアクションでは、WLIS_Hub トレーディング パートナを経由して、XOCP メッセージを 2 つのサプライヤそれぞれに送信します。

サンプル アプリケーションでは、このタスク ノードでルータ式を使用します。WLIS_Hub は、[ルータ式の型]（このケースでは [トレーディング パートナ名]）に基づいて、QPA 要求 XOCP メッセージをサプライヤにルーティングします。

このノードで定義されている [ルータ式] を確認するには、次のように、[ビジネス メッセージの送信] ダイアログ ボックスを呼び出します。

1. このノードをダブルクリックして、[タスク 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 | アクティブ化] を選択し、[ビジネス メッセージの送信] をダブルクリックします。

次の図に示すダイアログ ボックスが表示されます。

図 3-13　[ビジネス メッセージの送信] ダイアログ ボックス

上の図で、[ルータ式の型] で選択されている値が [トレーディング パートナ名] となっており、[ルータ式] の値が `FirstSupplierName` 変数に格納されることに注目してください。`FirstSupplierName` および `SecondSupplierName` 変数は、このワークフローの 開始ノードで割り当てられます。

1st/2nd QPA Reply Message Receipt

これらのイベント ノードでは、ワークフローがサプライヤからの応答を待機します。サプライヤは 2 つあるので、2 つのイベント ノードが定義されています。応答メッセージを受け取る順序は定義されません。つまり、着信ビジネス メッセージは対象の変数に割り当てられます。サプライヤの名前は、受信後、ビジネス メッセージ XML から派生します。

これらのイベント ノードはどちらも *Conversation Event* 型です。いずれかのノードをダブルクリックすると、次の図に示す [イベントのプロパティ] ダイアログ ボックスが表示されます。

図 3-14 B2B 会話のイベント ノードのプロパティ

ビジネス メッセージを待機しているイベント ノードを、B2B 会話の他のトレーディング パートナや他のイベント ノード（内部 XML イベントを待つノードなど）から区別するには、待機しているイベント ノードを Conversation Event 型として指定します。上の図の [イベントのプロパティ] ダイアログ ボックスでは、最初に受け取る QPA 応答メッセージを待機するノードのプロパティを定義しています。

Extract QPA Responses

両方のサプライヤから応答を受け取ると、このタスク ノードでは XOCP メッセージ（`FirstQPAResponseMessage` および `FirstQPAResponseMessage` 変数内）をアンパックします。アンパックしたメッセージから QPA 応答 XML ドキュメントを抽出して、`FirstQPAResponseXML` および `SecondQPAResponseXML` 変数に格納します。

Aggregate both QPA Responses

このタスク ノードでは、XML 構造体を作成することで、ワークフロー変数を設定 アクションを定義します。このアクションは、XML 応答ドキュメントを 1 つの QPA 応答 XML ドキュメントに集約します。集約された XML ドキュメントは、`AggregatedQPAReply` 変数に格納されます。

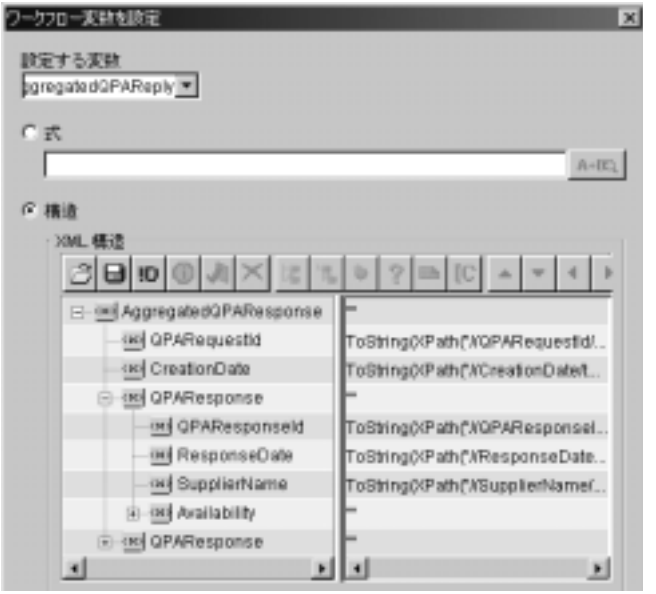
このサンプルで作成される XML ワークフロー変数を [ワークフロー変数を設定] ダイアログ ボックスで表示するには、次の手順に従います。

1. ノードをダブルクリックします。

2. [アクション | アクティブ化] を選択します。

3. このノードで定義されている ワークフロー変数を設定 アクション（`Set workflow variable "AggregatedQPAReply" XML structure`）をダブルクリックして、次の図に示す [ワークフロー変数を設定] ダイアログ ボックスを呼び出します。

図 3-15 [ワークフロー変数を設定] ダイアログ ボックス



上の図で、AggregatedQPAResponse XML が各サプライヤの QPARequestId、Creation Date、および QPA Response で構成されていることに注目してください。

Publish Aggregated QPA Response

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 プロパティ] ダイアログ ボックスを表示してから、[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、集約された QPA 応答（前のノードで定義された AggregatedQPAReply 変数）を*内部* XML イベントとしてポストします。

内部 XML イベントとは、BPM に対して内部のイベントということです。このイベントは、現在のワークフローまたは別のワークフローのイベントを呼び出すか、またはイベントによって呼び出された開始ノードに対応する別のワークフローを開始します。送り先は内部にあるので、XML メッセージは BPM が管理している内部 JMS キューに送信されます。内部 JMS キューの JNDI 名は、com.bea.wlpi.EventQueue です。

集約された QPA 応答メッセージは*アドレス指定されたメッセージ*と呼ばれ、特定のワークフロー インスタンスに合わせて永続することを示します。このケースでは、アドレス指定にはインスタンス ID の `QPAPrivateFlowId` を使用します。このインスタンス ID は、このパブリック ワークフローを呼び出すプライベート ワークフローのインスタンスを指定します。このケースでは、まず `QPAPrivateFlowId` インスタンス ID が、このパブリック ワークフロー（`WLIS_BuyerQPAPrivate`）を呼び出すプライベート ワークフローで指定されます。3-37 ページの「Call BuyerQPAPublic workflow」を参照してください。このようにして、呼び出し元および呼び出し先のワークフローの各インスタンスを同期します。

完了

QPA 会話を正常に終了します。ワークフローの他の完了ノードでは、ビジネス メッセージが Send QPA Request to Suppliers ノードから両方のサプライヤに送信されなかった場合、会話は正常に終了しません。

サプライヤサイドの実装

このソリューションを実装するために、サプライヤは、バックエンドプロセスを管理するプライベート ワークフローと、QPA 会話でメッセージの交換を管理するパブリック ワークフローを実装します。この節では、以下のワークフローについて説明します。

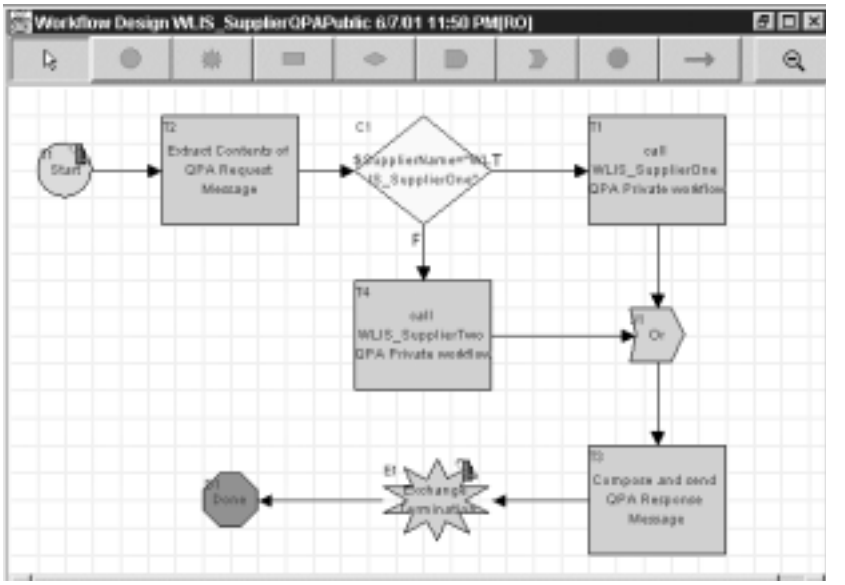
- サプライヤの QPA パブリック ワークフロー
- サプライヤの QPA プライベート ワークフロー

サプライヤの QPA パブリック ワークフロー

サプライヤの QPA パブリック ワークフロー（`WLIS_SupplierQPAPublic`）は、WLIS_Hub トレーディング パートナからのメッセージの受信によって開始されます。

次の図は、WLIS_SupplierQPAPublic ワークフロー テンプレートを示しています。

図 3-16 WLIS_SupplierQPAPublic 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 WLIS_SupplierQPAPublic ワークフロー テンプレート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
- Extract Contents of QPA Request Message
- Call WLIS_SupplierOneQPA Private Workflow
- Or 結合
- Call WLIS_SupplierTwoQPA Private Workflow
- Compose and Send QPA Response Message
- Exchange Termination

開始

WLIS_SupplierQPAPublic ワークフローは、QPA 要求ビジネス メッセージの受信というイベントによって開始されます。

Extract Contents of QPA Request Message

最初の ビジネス メッセージの要素を抽出 アクションは、受け取ったビジネス メッセージ（QPARequestMessage 変数）から XML メッセージを抽出し、XML を QPARequestXML 変数に割り当てます。

このノードで定義されているもう 1 つのアクションは、QPA 要求のメッセージに対してサプライヤの名前を定義するワークフロー変数を設定します。このサンプルのサプライヤは、WebLogic Integration の同じインスタンス上に配置され、QPA 会話で同じロールを持ち、同じパブリック ワークフロー テンプレートを共有しています。WLIS_Buyer トレーディング パートナは、2 つの XOCP メッセージ（各サプライヤ用）を WLIS_Hub トレーディング パートナ経由で送信します。このアクションは、次の XPath 式を使用して、受け取った QPA 要求 XML ドキュメントから一意な属性値を抽出することで、変数（SupplierName）を設定します。

```
ToString(XPath("/QPARequest/@SupplierName",
$QPARequestXML))
```

Call WLIS_SupplierOneQPA Private Workflow

このワークフローで前にあるノードで設定された SupplierName 変数の値に基づいて、このパブリック ワークフローのインスタンスは該当するサプライヤのプライベート ワークフロー（このケースでは WLIS_SupplierOneQPAPrivate）を開始します。このアクションは、プライベート ワークフローを開始します。WLIS_BuyerQPAPrivate ワークフローの一部として説明した パブリック ワークフローを開始 アクション（3-37 ページの「Call BuyerQPAPublic workflow」）とは異なります。

ワークフローを開始 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 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 | アクティブ化 | 追加 | ワークフロー アクション | ワークフローを開始] を選択して、[ワークフローを開始] ダイアログ ボックスを表示します。

3. ワークフローの名前を選択して、他のパラメータを呼び出して定義します。

このワークフローのこのノードで定義されている ワークフローを開始アクションを表示するには、次の手順に従います。

1. ノードをダブルクリックします。

2. [アクション | アクティブ化] を選択します。

3. このノードで定義されている ワークフローを開始 アクション（Start workflow "WLIS_SupplierOneQPAPrivate"）をダブルクリックして、[ワークフロー変数を設定] ダイアログ ボックスを呼び出します。

このノードの場合、[ワークフローを開始] ダイアログ ボックスでは、WLIS_SupplierOneQPAPrivate ワークフローを [開始するワークフロー] フィールドに定義できます。[ワークフローを開始] ダイアログ ボックスには [結果] タブもあり、このタブには [変数] フィールドおよび [結果] フィールドがあります。このケースでは、[変数] フィールドには QPAResponseXML が指定され、[結果] フィールドには QPAReplyXML が指定されています。[結果] タブには、呼び出されるワークフローの出力パラメータとして定義されている変数が表示されます。呼び出し元ワークフローは、出力パラメータを使用して、呼び出されるワークフローからの値を受け取ります。

このケースでは、[結果] タブで QPAResponseXML 変数を QPAReplyXML Result (結果) に割り当てます。Result 値は、WLIS_SupplierOneQPAPrivate ワークフローの Create QPA Reply XML ノードで QPAReplyXML 出力パラメータに割り当てられます（3-55 ページの「サプライヤの QPA プライベート ワークフロー」を参照）。

呼び出し元ワークフロー（このケースでは WLIS_SupplierQPAPublic）は、値を割り当ててから次のタスクに進みます。要約すると、このノードからプライベート ワークフローへの呼び出しは同期呼び出しです。

Or 結合

結合ノードです。呼び出されるワークフロー（WLIS_SupplierOneQPAPrivate と WLIS_SupplierTwoQPAPrivate）は、同時に呼び出されます。その結果、呼び出し元ワークフロー（WLIS_SupplierQPAPublic）は、いずれかのプライベート ワークフローの結果を取得するまで待機します。

Call WLIS_SupplierTwoQPA Private Workflow

このワークフローの Extract Contents of QPA Request Message ノードで設定された SupplierName 変数の値に基づいて、このパブリック ワークフ

ローのこのインスタンスは該当するサプライヤのプライベート ワークフロー（このケースでは WLIS_SupplierTwoQPAPrivate）を開始します。

このノードで実行されるタスクは、既に説明した Call WLIS_SupplierOneQPA Private Workflow でのタスクと同じですが、このノードでは WLIS_SupplierTwoQPAPrivate ワークフローが呼び出される点が異なります。

Compose and Send QPA Response Message

このノードの最初のアクションは、サプライヤからの応答メッセージを作成します。このアクションは、プライベート ワークフローから QPAResponseXML 変数を受け取り、応答メッセージを QPAReplyMessage 出力変数として構築します。

ビジネス メッセージの送信 アクションは、QPAReplyMessage 変数に割り当てられるビジネス メッセージを、WLIS_Hub 経由で WLIS_Buyer トレーディング パートナに送信します。WLIS_Hub は、[ルータ式の型] (このケースでは [トレーディング パートナ名]) に基づいて、この QPA 要求 XOCP メッセージを WLIS_Buyer トレーディング パートナにルーティングします。このノードで定義されているルータ式を確認するには、次のように、[ビジネス メッセージの送信] ダイアログ ボックスを呼び出します。

1. このノードをダブルクリックして、[タスク 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 | アクティブ化] を選択し、[ビジネス メッセージの送信] をダブルクリックして [ビジネス メッセージの送信] ダイアログ ボックスを表示します。

ダイアログ ボックスには、3 つのタブ（[メッセージ]、[トークン]、および [QoS]）があります。[メッセージ] タブの以下のパラメータに注目してください。

フィールド	値	説明
[入力メッセージ変数]	QPAReplyMessage	WLIS_Buyer トレーディング パートナに送信する XOCP メッセージを格納する Java オブジェクト変数。

[ルータ式の型]	`Trading Partner Name`	WLIS_Hub は、トレーディング パートナ名に基づいて、この QPA 応答メッセージを WLIS_Buyer トレーディング パートナにルーティングする。
[ルータ式]	`$QPASenderName`	この QPA 応答メッセージを受け取るトレーディング パートナは、`QPASenderName` 変数の値によって識別される。
[対象ロール]	バイヤ	会話でのロールを、メッセージを受け取るトレーディング パートナに対して定義する。
[会話 QoS を使用]	未選択	ワークフロー テンプレートが XOCP プロトコルでコンフィグレーションされているので、このオプションが用意されている。このオプションでは、会話レベルまたはこの ビジネス メッセージの送信 アクション レベルで定義されているサービス品質（Quality of Service）を使用するかどうかを指定する。サンプルでは、プロセス エンジンは、この ビジネス メッセージの送信 アクション レベルで定義されている QoS 情報を使用する。

Exchange Termination

Conversation Event 型のイベント ノードです。このノードでは、QPA 会話を終了します。

サプライヤの QPA プライベート ワークフロー

このカテゴリには、各サプライヤに 1 つずつ、合わせて 2 つのワークフロー テンプレートがあります。

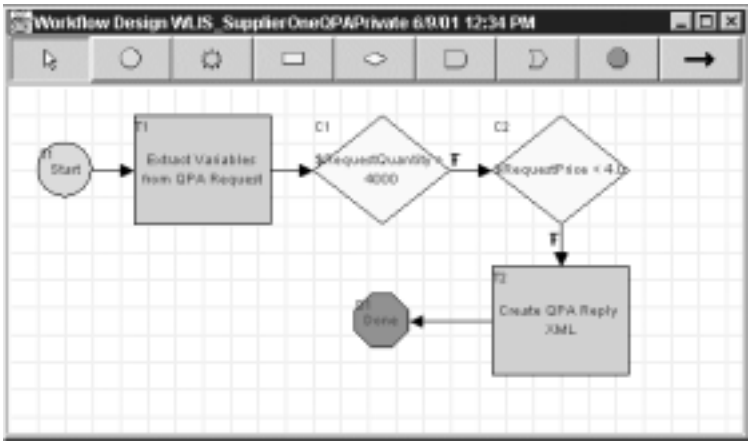
- `WLIS_SupplierOneQPAPrivate`
- `WLIS_SupplierTwoQPAPrivate`

2 つのテンプレートには若干の違いがあります。詳細については説明しないので、各自で確認してください。

サプライヤの QPA プライベート ワークフローは、サプライヤの QPA パブリック ワークフロー（WLIS_SupplierQPAPublic）から呼び出されます。

次の図は、WLIS_SupplierOneQPAPrivate ワークフロー テンプレートを示しています。

図 3-17　WLIS_SupplierOneQPAPrivate 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 WLIS_SupplierOneQPAPrivate ワークフロー テンプレート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
- Extract Variables from QPA Request
- 分岐ノード
- Create QPA Reply XML

開始

このワークフローは呼び出されるワークフローです。つまり、サプライヤのパブリック ワークフロー（WLIS_SupplierQPAPublic）によって開始されます。selfName という変数は、このノードで設定されます。このケースでは、selfName の値は WLIS_SupplierOne です。

Extract Variables from QPA Request

このノードでは、複数の ワークフロー変数を設定 アクションを定義します。これらのアクションは XPath 式を使用して、受け取った QPA 要求 XML ドキュメントから数量、単価、および納品日を抽出し、データをワークフローに割り当てます。

たとえば単価は、次の XPath 式を使用して、受け取った XML ドキュメントから抽出され、RequestPrice 変数に割り当てられます。

```
ToString(XPath("//Availability/UnitPrice/text()",
$QPARequestXML))
```

分岐ノード

ワークフローには、数量および価格条件を評価する 2 つの分岐ノードが含まれています。

RequestQuantity

RequestQuantity 変数の内容が評価され、その結果に従って、戻り値（ReplyQuantity）が設定されます。

RequestPrice

RequestPrice 変数の内容が評価され、その結果に従って、戻り値（ReplyPrice）が設定されます。

図 2-2 の 「[QPA リクエストの送信] ウィンドウ」と 図 2-5 の 「[PO の作成] ウィンドウ」を見て、サンプル シナリオのバイヤが要求した数量

および単価と、バイヤがサプライヤから受け取った見積もりを比較してください。

Create QPA Reply XML

このタスク ノードでは、XML エディタを使用して、QPA 応答 XML ドキュメントを作成します。ドキュメントは、このプライベート ワークフローを呼び出したパブリック ワークフロー（`WLIS_SupplierQPAPublic`）の戻りワークフロー変数（`QPAReplyXML`）に割り当てられます。

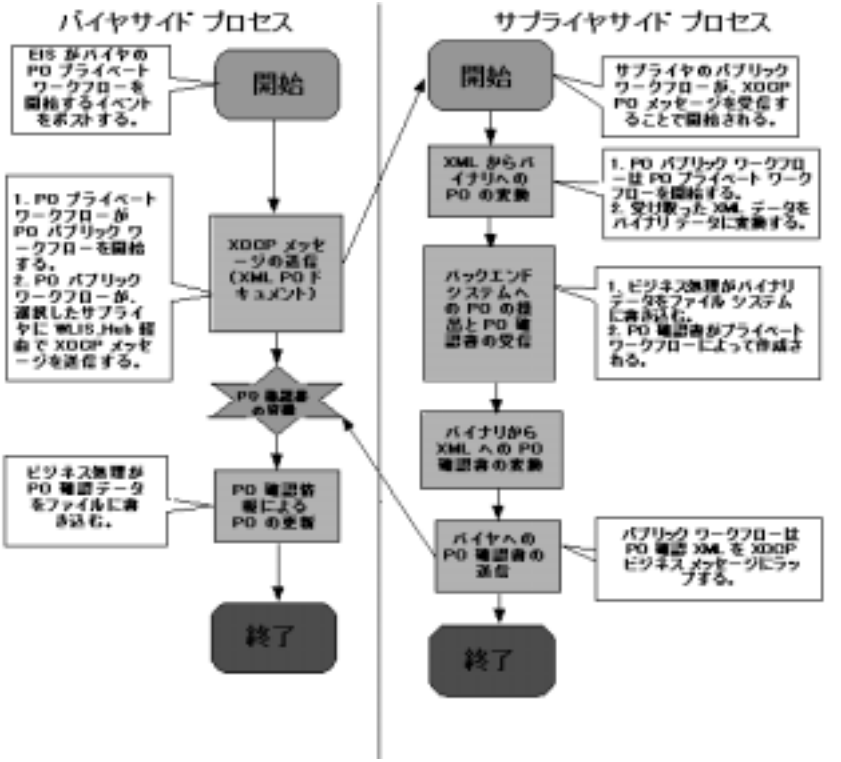
QPA 応答 XML ドキュメントは、`QPAResponse.dtd` DTD で定義します。QPA 応答 DTD については、付録 A「DTD」を参照してください。

`QPAReplyXML` 変数がこのワークフローで設定されると、そのワークフローを呼び出したパブリック ワークフローは次のタスクに進みます。`WLIS_SupplierQPAPublic` パブリック ワークフローのこのタスク順序については、3-52 ページの「Call WLIS_SupplierOneQPA Private Workflow」で説明しています。

PO ビジネス プロセス

このサンプル シナリオのバイヤになったつもりでお読みください。サプライヤからの QPA 応答を確認したら、いずれかのサプライヤを選択し、PO ビジネス プロセスを開始します。2-12 ページの「ステップ 6: 発注書を作成する」を参照してください。次の図は、PO ビジネス プロセスのイベント フローを示しています。

図 3-18　PO ビジネス プロセスのプロセス フロー



上の図のイベントをまとめると、次のようになります。

1. バイヤが発注（PO）ドキュメントを作成します。
2. バイヤが選択したサプライヤに PO ドキュメントを送信します。
3. サプライヤが PO を受信し、XML ベースの PO ドキュメントをバイナリ データ ファイルに変換します。このサンプルでは、サプライヤの注文管理システムがバイナリ ファイルを受け取って注文を処理することを前提にしています。
4. サプライヤが、別のバイナリ データ ファイルに基づいて、XML ベースの PO 確認書を作成します。
5. サプライヤがバイヤに PO 確認書を送信します。

6. バイヤが、PO 確認書に基づいて、PO 情報を更新します。

注意： 上の図は、PO ビジネス プロセスの高レベルのビューを示しています。各サイドのプロセスは、パブリック ワークフローとプライベート ワークフローによって実装されます。以下の節では、これらのワークフローについて説明します。

- PO 実装の概要
- バイヤサイドの実装
- サプライヤサイドの実装

PO 実装の概要

このシナリオでは、各トレーディング パートナは、PO プロセスのプライベート ワークフローおよびパブリック ワークフローを実装しています。サンプルでは、以下のワークフロー テンプレートが使用されています。

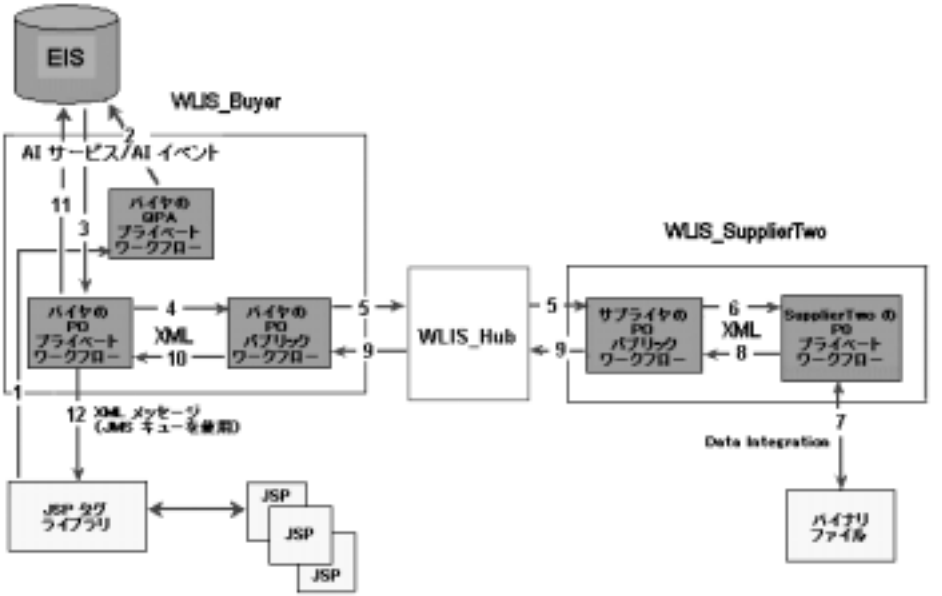
表 3-6 サンプル PO プロセスのワークフロー

ロール	パブリック / プライベート	ワークフロー名
バイヤ	プライベート	`WLIS_BuyerQPAPrivate`
バイヤ	プライベート	`WLIS_BuyerPOPrivate`
バイヤ	パブリック	`WLIS_BuyerPOPublic`
サプライヤ	パブリック	`WLIS_SupplierPOPublic` **注意：** シナリオのバイヤはどちらも同じパブリック ワークフローを使用します。
サプライヤ	プライベート	`WLIS_SupplierOnePOPrivate`
サプライヤ	プライベート	`WLIS_SupplierTwoPOPrivate`

このサンプルの PO 実装では、アプリケーション統合、データ統合、およびビジネス プロセス管理に関して WebLogic Integration のサポートを必要とします。この節では、PO ワークフローについて説明します。バックエンド アプリケーションおよび異種データ形式との統合についても説明します。このシナリオの PO ビジネス プロセスにあてはめた場合のアプリケーション統合およびデータ統合機能の詳細については、3-83 ページの「Application Integration と Data Integration」を参照してください。

次の図は、PO ビジネス プロセス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 3-19 PO ビジネス プロセスのデータ フロー



バイヤサイトおよびサプライヤサイドの実装の詳細については、3-64 ページの「バイヤサイドの実装」と 3-50 ページの「サプライヤサイドの実装」を参照してください。

トレーディング パートナ、ワークフロー、およびバックエンド システムの間のデータ フローをまとめると、次のイベント シーケンスになります。

1. PO ビジネス プロセスは、バイヤがサプライヤを選択し、発注書を作成すると開始されます。2-12 ページの「ステップ 6: 発注書を作成する」を参照してください。

   JSP は、パーティ ID、単価、数量、出荷日などの PO 要求情報を抽出し、XML メッセージに入れます。XML メッセージは、BPM JMS キューにポストされます。

2. XML イベントのサブスクライバであるバイヤのプライベート ワークフロー（`WLIS_BuyerQPAPrivate`）は、イベントがポストされると呼び出されます。

   `WLIS_BuyerQPAPrivate` ワークフローは、`WLISAppView.sav` アプリケーション ビューの `insertPO` サービスを呼び出して、PO 情報をエンタープライズ情報システム（EIS）に挿入します。このサンプルでは、RDBMS が EIS に該当します。`WLISAppView.sav` アプリケーション ビューは、サンプルの設定時に、サービスおよびイベントと一緒に WebLogic Integration でコンフィグレーションされてデプロイされます。

3. EIS は、PO 情報を含むイベントを WebLogic Integration プロセス エンジンに送信します。

   アプリケーション ビュー イベントは、バイヤの PO プロセス用プライベート ワークフロー（`WLIS_BuyerPOPrivate`）を開始します。

4. `WLIS_BuyerPOPrivate` ワークフローは、バイヤのパブリック ワークフロー（`WLIS_BuyerPOPublic`）を開始します。

5. `WLIS_BuyerPOPublic` は、XOCP メッセージをサプライヤのパブリック ワークフロー（`WLIS_SupplierPOPublic`）に送信します。このステップは PO 会話を開始します。

   WLIS_Hub トレーディング パートナは、自身と WLIS_Buyer 間で登録されているコラボレーション アグリーメントに基づいて、送り先のトレーディング パートナ（WLIS_Buyer）にメッセージをルーティングします。

   注意： このステップでは、WLIS_Hub トレーディング パートナはロールを変えて、自身とバイヤ間のコラボレーション アグリーメントの代理サプライヤとなります。

6. サプライヤサイドでは、サプライヤが XOCP メッセージを受信すると、サプライヤのパブリック フロー（`WLIS_SupplierPOPublic`）インスタンスが開始されます。

WLIS_SupplierPOPublic ワークフローは、サプライヤのプライベート ワークフロー（WLIS_SupplierOnePOPrivate または WLIS_SupplierTwoPOPrivate）を開始します。

7. 選択したサプライヤのプライベート ワークフローは、以下の処理を実行します。
   a. WebLogic Integration の Data Integration の機能を使用して、受け取った XML PO データをバイナリ データに変換します。
   b. PO 確認書をバイナリ形式で作成します。
   c. PO 確認書をバイナリ形式から XML に変換します。
8. サプライヤのプライベート ワークフローは、PO 確認 XML データを WLIS_SupplierPOPublic ワークフローに返します。
9. WLIS_SupplierPOPublic ワークフローは、PO 確認書の情報を XOCP メッセージにラップして、バイヤのパブリック ワークフロー（WLIS_BuyerPOPublic）に送信します。
10. WLIS_BuyerPOPublic ワークフローは XOCP メッセージを受信し、XML コンテンツを抽出し、XML イベントをバイヤのプライベート ワークフロー（WLIS_BuyerPOPrivate）に送信します。

    このステップは、PO 会話が終了したことを示します。
11. WLIS_BuyerPOPrivate ワークフローは、WebLogic Integration で提供されるアプリケーション統合フレームワークを使用して、PO 確認書として提供されたデータに基づいて EIS 内の PO 情報を更新します。また、このワークフローは、PO 確認書の情報を POAcknowledgement.xml ファイルに書き込みます。
12. JSP は POAcknowledgement.xml ファイルを読み出し、その内容を [PO の確定] ウィンドウに表示します。2-14 ページの「ステップ 8: 発注確認書を確認する」を参照してください。

    このステップは、PO ビジネス プロセスが終了したことを示します。

バイヤサイドの実装

サンプル シナリオのバイヤ（GCS）は、ビジネス トランザクションを操作するための Web ユーザ インタフェース、バイヤのバックエンド プロセスを管理するためのプライベート ワークフロー、および PO 会話でのメッセージ交換を管理するパブリック ワークフローを実装しています。WLIS_Buyer トレーディング パートナは、PO 情報を RDBMS に格納します。

WebLogic Integration で提供されるアプリケーション統合フレームワークを使用することで、PO ビジネス プロセスのワークフローと RDBMS との統合が可能になります。アプリケーションの統合をサポートするため、サンプル ドメインをセットアップおよびコンフィグレーションするときに、アプリケーション ビュー（`WLISAppView.sav`）がデプロイされます。サンプル ドメインのセットアップについては、2-4 ページの「ステップ 1A : RunSamples スクリプトを起動する」を参照してください。

この節では、バイヤサイドの PO ビジネス プロセスの以下のコンポーネントについて説明します。

- バイヤサイドの Web ユーザ インタフェース
- バイヤの PO プライベート ワークフロー
- バイヤの PO パブリック ワークフロー

バイヤサイドの Web ユーザ インタフェース

バイヤサイドの実装用の Web ブラウザとバイヤサイド ワークフローとの会話は、QPA ビジネス プロセスの場合と似ています。Java Server Pages (JSP) と JSP タグ ライブラリを使用して、PO プロセスを開始し、要求および応答データを Web ブラウザに表示します。PO ビジネス プロセスに関連するソース ファイルは、次の表に示したディレクトリにあります。*`SAMPLES_HOME`* は WebLogic Platform のサンプル ディレクトリを示します。

表 3-7 PO プロセスのソース ファイル

WebLogic Integration のインストール ディレクトリ内の場所	ソース ファイル
SAMPLE_HOME\integration\samples\wlis\src\examples\wlis\tags	■ CreatePOTag.java ■ CheckPOTag.java ■ CheckPOAckTag.java
SAMPLE_HOME\integration\samples\wlis\web	■ CreatePO.jsp ■ CheckPO.jsp ■ CheckPOAck.jsp ■ WaitPOCreated.htm ■ WaitPOAck.htm
SAMPLE_HOME\integration\samples\wlis\lib\xsl	■ ProcessPO.xsl ■ ProcessPOAck.xsl

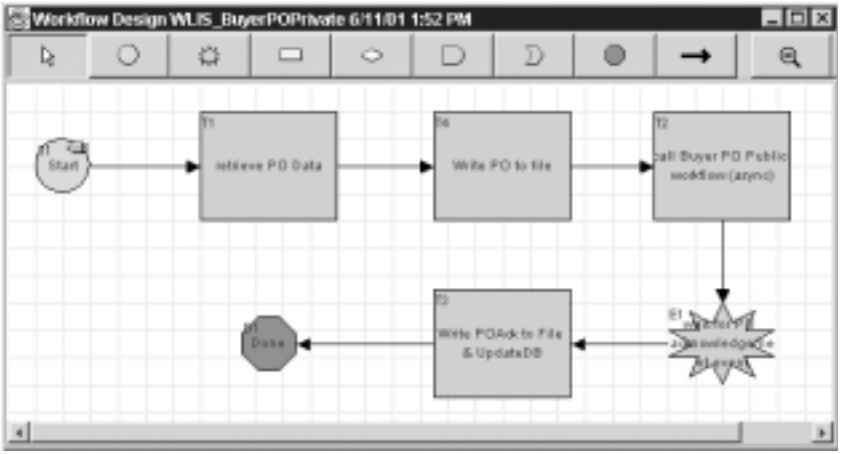
バイヤの PO プライベート ワークフロー

`WLIS_BuyerPOPrivate` ワークフローは、以下の主要なタスクを実行します。

- プロセス エンジンから、バイヤの EIS からの PO 情報を格納した `insertPO` イベントを受け取ります。`insertPO` イベントは、`WLISAppView.sav` アプリケーション ビュー用に定義されているイベントです。
- EIS から PO 情報を取り出します。
- PO 情報を XOCP ビジネス メッセージにラップします。
- バイヤのパブリック ワークフロー（`WLIS_BuyerPOPublic`）を呼び出します。このワークフローが PO 情報をサプライヤのパブリック ワークフロー（`WLIS_SupplierPOPublic`）に送信することで、PO 会話が開始されます。
- サプライヤからの PO 確認情報を待機します。
- WebLogic Integration で提供されるアプリケーション統合フレームワークを使用し、PO 確認書に基づいて EIS 内の PO 情報を更新します。
- PO 確認情報を `POAcknowledgement.xml` ファイルに書き込みます。

次の図は、`WLIS_BuyerPOPrivate` ワークフロー テンプレートを示しています。

図 3-20　WLIS_BuyerPOPrivate 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 WLIS_BuyerPOPrivate ワークフロー テンプレート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
- Retrieve PO Data
- Write PO to file
- バイヤの PO パブリック ワークフロー
- Wait for PO acknowledgement
- Write POAcktoFile & UpdateDB

開始

バイヤの QPA プライベート ワークフロー（WLIS_BuyerQPAPrivate）の最後のタスクは、3-31 ページの「バイヤの QPA プライベート ワークフロー」で説明したように、このサンプル アプリケーション用にデプロイされた WLISAppView.sav アプリケーション ビューで定義されている insertPO サービスを使用して、PO 情報を EIS システムに入力することです。アプリケーション ビューの詳細については、3-84 ページの「Application Integration」を参照してください。

続いて、EIS はアプリケーション ビュー イベントを使用して、WebLogic Integration プロセス エンジンにイベントをポストします。イベント（WLISAppView.sav.insertPOEvent）によって、WLIS_BuyerPOPrivate ワークフローがこの開始ノードから開始されます。

このノードの [開始のプロパティ] ダイアログ ボックスでは、このワークフローをアプリケーション統合イベント、つまり WLISAppView.sav アプリケーション ビューで定義されている insertPO というイベントで開始することが指定されています。

この開始ノードをダブルクリックして、次の図に示す [開始のプロパティ] ダイアログ ボックスを表示します。

図 3-21　[開始のプロパティ] ダイアログ ボックス

上の図の [開始のプロパティ] ダイアログ ボックスにある [] フィールドに `aiEventXML` 変数が指定されていることに注目してください。このイベントのデータ形式は XML で、イベント スキーマは XML スキーマ言語によって定義されます。開始ノードがアプリケーション ビューからデータを受け取ると、データは `aiEventXML` 変数に格納されます。

注意： イベント ドキュメントの XML スキーマを調べるには、[開始のプロパティ] ダイアログ ボックスの [定義を表示] をクリックします。

ワークフロー変数（PONumber）は、PO データから PO 番号を取得する次の XPath 式を使用して、このノードで設定されます。

```
XPath("/PURCHASEORDER.insert/PONUMBER/text()",$aiEventXML)
```

Retrieve PO Data

このタスク ノードでは、以下のアクションを実行します。

1. ワークフロー変数を作成します。

2. アプリケーション ビュー サービスを呼び出して、EIS から PO 情報を取り出します。

このタスク ノードは、Lookup 2nd Tier suppliers for requested items というノードに用意されているものと同じ手順を使用して、アプリケーション ビューのサービスを呼び出すように設定されています。3-31 ページの「バイヤの QPA プライベート ワークフロー」を参照してください。

WebLogic Integration で提供される Application Integration プラグインの使い方については、『*Application Integration ユーザーズ ガイド*』を参照してください。

Write PO to file

このアクション ノードでは、xmlToFile ビジネス オペレーションを使用して、PO をローカル ファイル（*SAMPLE_HOME*\integration\samples/config/samples/data/PO.xml）に書き込みます。このローカル ファイルは JSP によって処理されます。

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。[] ダイアログ ボックスが表示されます。ビジネス オペレーションをダブルクリックすると、詳細を確認できます。WebLogic Platform のインストール ディレクトリの次の場所には、xmlToFile ビジネス オペレーションに関連する Java クラスがあります。

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

上の行の *SAMPLES_HOME* は、WebLogic Platform のサンプル ディレクトリです。

バイヤの PO パブリック ワークフロー

このタスク ノードでは、バイヤのパブリック ワークフロー（WLIS_BuyerPOPublic）を非同期で開始して、QPA 要求 XML ドキュメントを格納したワークフロー変数をパブリック ワークフローに渡します。WLIS_BuyerPOPrivate ワークフローは WebLogic Integration B2B 会話（WLIS_POConversation）の開始者なので、このタスク ノードでは、デフォルトの ワークフローを開始 の代わりに、B2B Integration プラグ インによって提供される特殊な パブリック ワークフローを開始 アクションを使用します。

パブリック ワークフローを開始 アクションと ワークフローを開始 アクションとの主要な違いは、パブリック ワークフローを開始 アクションを使用する場合にワークフロー テンプレートが動的にバインドされることです。WLIS_BuyerPOPrivate ワークフローでは、パブリック ワークフローを開始 アクションで定義した会話プロパティに基づいて、実行時に呼び出すワークフロー テンプレートが決定されます。

また、B2B エンジンは、パブリック ワークフローを開始 呼び出しの一部として渡された会話情報に基づいて、WebLogic Integration リポジトリに格納されているコラボレーション アグリーメント情報を使用して、呼び出されたワークフローを同じコラボレーション アグリーメントの他のパブリック ワークフローに関連付けます。

WebLogic Integration Studio でパブリック ワークフローを開始 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 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 | 統合アクション | B2B Integration | パブリック ワークフローを開始] を選択して、[パブリック ワークフローを開始] ダイアログ ボックスを表示します。

このノードに対して既に指定されている パブリック ワークフローを開始 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 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 | アクティブ化] を選択します。

3. [パブリック ワークフローを開始] をダブルクリックして、次の図に示す [パブリック ワークフローを開始] ダイアログ ボックスを表示します。

図 3-22 [パブリック ワークフローを開始] ダイアログ ボックス

上の図で、パブリック ワークフローを開始 アクション用の以下のパラメータに注目してください。

- [会話] タブ ー WebLogic Integration では、このタブで指定した情報を使用して、指定された会話の指定されたパーティ間のアグリーメントを定義するアクティブなコラボレーション アグリーメントをリポジトリ内で見つけます。
  - 会話名、バージョン、およびロールは、それぞれ `WLIS_POConversation`、`1.1`、および `buyer` として定義されています。
  - 会話のパーティは、トレーディング パートナの名前（`WLIS_Buyer` と `WLIS_Hub`）で指定されています。関係のあるコラボレーション アグリーメントの検索で使用する値を、[名前] フィールド以外にも、[ロール] および [配信チャネル] フィールドに任意で追加することもできます。
- [ワークフロー] タブー呼び出されるワークフローに渡す変数は、[ワークフロー] タブで定義します。ここで指定する変数名は、呼び出されるワークフ

ロー（WLIS_BuyerPOPublic）に対する入力変数としても指定されます。変数は以下のとおりです。

- POxml ー Retrieve PO Data ノード（3-69 ページの「Retrieve PO Data」を参照）で定義された purchaseOrderXML 変数の値を格納します。
- POPrivateFlowId ーパブリック ワークフローを呼び出す WLIS_BuyerPOPrivate ワークフローのインスタンスを指定するワークフロー インスタンス ID です。ワークフロー テンプレートの複数のインスタンスが実行されている場合があります。このインスタンス ID をパブリック ワークフローで使用する場合については、3-76 ページの「Send PO Acknowledgement to PO Private Workflow」を参照してください。

Wait for PO acknowledgement

このイベント ノードのワークフローでは、WLIS_BuyerPOPublic ワークフローからの指定した XML イベントを待機します。このノードでは、キー値表現を使用して、ノードを呼び出すために受け取ったドキュメントが含まなければならないデータを正確に指定します。このノードに定義されているキー値表現は、PONumber です。PONumber キー値は、WLIS_BuyerPOPrivate ワークフローの開始ノードで設定されます。3-67 ページの「開始」を参照。

バイヤのプライベート ワークフロー（WLIS_BuyerPOPrivate）のインスタンスが多数ある場合、適切なワークフロー テンプレート インスタンスの適切な Wait for PO Acknowledgement ノードを、PO プロセスのこの時点で呼び出す必要があります。呼び出すノードを格納するワークフロー インスタンスは、PONumber の値で指定されます。3-32 ページの「開始」のイベント キーの説明を参照してください。

Write POAcktoFile & UpdateDB

このタスク ノードで実行されるアクションは、以下のとおりです。

xmlToFile ビジネス オペレーションを実行する

このビジネス オペレーションでは、PO 確認 XML データが 次のファイルに書き込まれます。

```
SAMPLE_HOME\integration\samples\data\
POAcknowledgement.xml
```

上の行の *SAMPLES_HOME* は、WebLogic Platform のサンプル ディレクトリです。

ワークフロー変数「updatePOXML」を XML 構造に設定する

XML ドキュメントを作成し、その内容を XML 型の `updatePOXML` 変数に格納します。

アプリケーション ビュー サービスを呼び出す

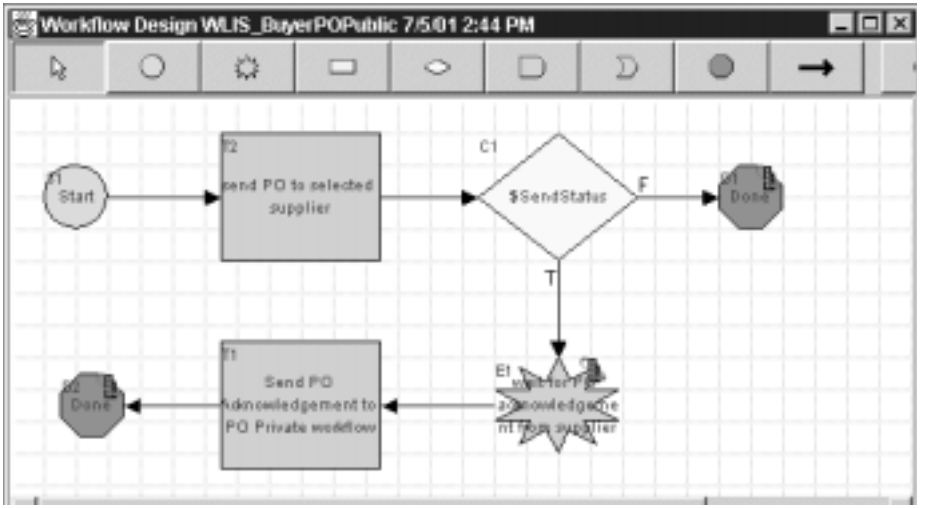
`WLISAppView.sav` アプリケーション ビューの `updatePOData` サービスを呼び出し、PO データを使用して RDBMS を更新します。

バイヤの PO パブリック ワークフロー

`WLIS_BuyerPOPublic` ワークフローの主なタスクは、発注会話（`WLIS_POConversation`）で XOCP ビジネス メッセージを送受信することです。

次の図は、`WLIS_BuyerPOPublic` ワークフロー テンプレートを示しています。

図 3-23 WLIS_BuyerPOPublic 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 `WLIS_BuyerPOPublic` ワークフロー テンプレート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
- Send PO to Selected Supplier

- Wait for PO Acknowledgement from Supplier
- Send PO Acknowledgement to PO Private Workflow
- 完了

開始

このワークフローは、3-70 ページの「バイヤの PO パブリック ワークフロー」で説明したように、バイヤのプライベート ワークフロー（WLIS_BuyerPOPrivate）によって開始されます。バイヤの PO パブリック ワークフロー ノードの [ワークフロー] タブで指定された POxml 変数は、このワークフローの入力変数です。サプライヤ名を格納するワークフロー変数（SupplierName）は、次の XPath 式を使用して、このノードで設定されます。

```
ToString(XPath("//SupplierName/text()", $POxml))
```

Send PO to Selected Supplier

このノードで定義されている最初のアクションは、ビジネス メッセージの作成です。このアクションは、バイヤから選択したサプライヤに送信する PO メッセージを作成します。このアクションは、プライベート ワークフローから POxml 変数を受け取り、応答メッセージを POXOCPMessage 出力変数として構築します。

このノードには、ビジネス メッセージの送信 アクションも定義されています。PO メッセージ（POXOCPMessage）は、WLIS_Hub トレーディング パートナを介してルーティングされます。WLIS_Hub トレーディング パートナは、このノードで定義される [ルータ式の型] (このケースでは [トレーディング パートナ名]) に基づいて、メッセージをルーティングします。このノードで定義されているルータ式を確認するには、次のように、[ビジネス メッセージの送信] ダイアログ ボックスを呼び出します。

1. このノードをダブルクリックして、[タスク 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 | アクティブ化] を選択し、[ビジネス メッセージの送信] をダブルクリックして [ビジネス メッセージの送信] ダイアログ ボックスを表示します。

ダイアログ ボックスには、3 つのタブ（[メッセージ]、[トークン]、および [QoS]）があります。[メッセージ] タブの以下のパラメータに注目してください。

フィールド	値	説明
[入力メッセージ変数]	POXOCPMessage	選択したサプライヤに送信する XOCP メッセージを格納する Java オブジェクト変数。
[ルータ式の型]	Trading Partner Name	WLIS_Hub は、トレーディング パートナ名に基づいて、この PO メッセージを選択したサプライヤ トレーディング パートナにルーティングする。
[ルータ式]	$SupplierName	PO メッセージを受け取るトレーディング パートナは、SupplierName 変数の値によって識別される。SupplierName 変数は、このワークフローの開始ノードで設定される。
[対象ロール]	サプライヤ	会話でのロールを、メッセージを受け取るトレーディング パートナに対して定義する。
[会話 QoS を使用]	未選択	ワークフロー テンプレートが XOCP プロトコルでコンフィグレーションされているので、このオプションが用意されている。このオプションでは、会話レベルまたはこの ビジネス メッセージの送信 アクション レベルで定義されているサービス品質（Quality of Service）を使用するかどうかを指定する。このケースでは、プロセス エンジンは、この ビジネス メッセージの送信 アクション レベルで定義されている QoS 情報を使用する。

Wait for PO Acknowledgement from Supplier

このノードは会話イベント ノードです。パブリック ワークフローは、サプライヤから PO 確認 XOCP メッセージが返されるのをこのノードで待機します。確認メッセージは、POXOCPMsg という Java オブジェクト変数に割り当てられます。

Send PO Acknowledgement to PO Private Workflow

このノードで定義されている最初のアクション（ビジネス メッセージの要素を抽出）は、選択したサプライヤから受け取った PO 応答 XOCP メッセージからビジネス メッセージ部分を抽出します。次に、XML コンテンツを `POReplyXML` 変数に割り当てます。

このノードの別のアクションは、PO 確認書（`POReplyXML` 変数）を*内部*の XML イベントとして PO プライベート ワークフローにポストする XML イベントをポスト アクションを定義します。サンプル シナリオでは、この内部 XML イベントによって、別のワークフローが呼び出されます。送り先は BPM の内部にあるので、XML メッセージは BPM が管理している内部 JMS キューに送信されます。

ワークフロー ノードに対して XML イベント アクションをポストするには、次の手順に従います。

1. ノードをダブルクリックして、[タスク プロパティ] ダイアログ ボックスを表示します。

2. [追加 | 統合アクション | XML イベントをポスト] を選択して、[XML イベントをポスト] ダイアログ ボックスを表示します。

このノードに対して [XML イベントをポスト] ダイアログ ボックスで既に定義されているパラメータを表示するには、次の手順に従います。

1. ノードをダブルクリックして、[タスク 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 | アクティブ化] を選択し、[内部 XML イベントをポスト] をダブルクリックして [XML イベントをポスト] ダイアログ ボックスを表示します。

このケースでは、XML メッセージは `POReplyXML` 変数に格納されます。この PO 確認メッセージは *アドレス指定されたメッセージ*と呼ばれ、特定のワークフロー インスタンスに合わせて永続することを示します。サンプル シナリオでは、アドレス指定にはインスタンス ID の `POPrivateFlowId` を使用します。このインスタンス ID は、このパブリック ワークフローを呼び出したプライベート ワークフローのインスタンスを指定します。`POPrivateFlowId` インスタンス ID は、`WLIS_BuyerPOPrivate` ワークフロー、つまりこのパブリック ワークフローを呼び出したプライベート ワークフローによって指定されます（3-70 ページの「バイヤの PO パブリック ワークフロー」を参照）。この

ようにして、呼び出し元および呼び出し先のワークフローの各インスタンスを同期します。

完了

PO 会話を正常に終了します。ワークフローが発注書をサプライヤに送信できなかった場合、ワークフローの別の完了ノードでは、会話は正常に終了しません。

サプライヤサイドの実装

このシナリオでは、各サプライヤは、自身のバックエンド プロセスを統合するためのプライベート ワークフローと、PO 会話でのメッセージ交換を管理するためのパブリック ワークフローを実装しています。この節では、以下のワークフローについて説明します。

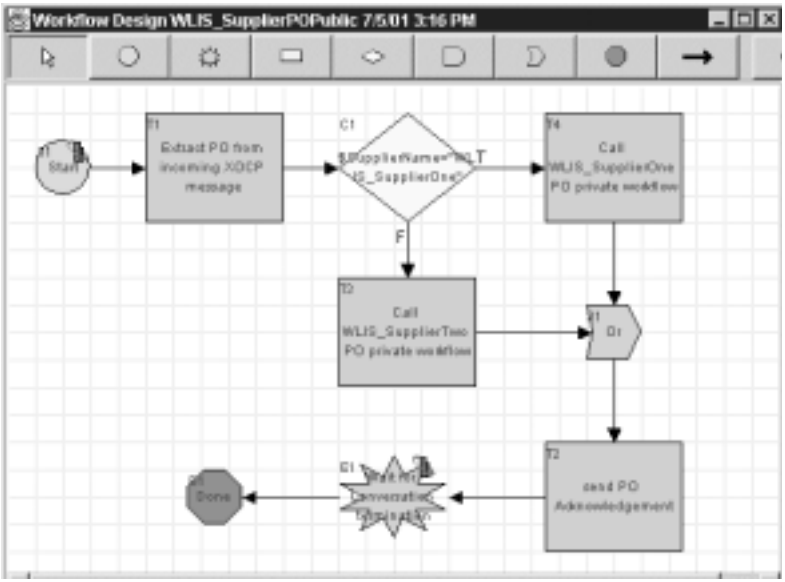
- サプライヤの PO パブリック ワークフロー
- サプライヤの PO プライベート ワークフロー

サプライヤの PO パブリック ワークフロー

PO パブリック ワークフロー（`WLIS_SupplierPOPublic`）は、バイヤの PO パブリック ワークフロー（`WLIS_BuyerPOPrivate`）から XOCP ビジネス メッセージを受信した時点で開始されます。

次の図は、WLIS_SupplierPOPublic ワークフロー テンプレートを示しています。

図 3-24　WLIS_SupplierPOPublic 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 WLIS_SupplierPOPublic ワークフロー テンプレート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
- Extract PO from incoming XOCP message
- Call WLIS_SupplierOne PO private workflow
- Send PO Acknowledgement
- Wait for Conversation Termination
- 完了

開始

このワークフローは、サプライヤがバイヤの PO パブリック ワークフロー（WLIS_BuyerPOPublic）から XOCP ビジネス メッセージを受信したときに開始されます。このメッセージは、PO 会話

（WLIS_POConversation）で交換される複数のメッセージのうちの 1 つです。

Extract PO from incoming XOCP message

このノードの最初のアクションでは、WLIS_BuyerPOPublic ワークフローから受け取った XOCP メッセージからメッセージ部分を抽出します。XML は、PODataXML 変数に割り当てられます。

次いで、SupplierName のワークフロー変数式が設定されます。サプライヤ名は受け取った PO XML から派生し、次の XPath 式を使用して SupplierName に割り当てられます。

```
ToString(XPath("/PurchaseOrder/SupplierInformation/Supplier
Name/text()", $PODataXML))
```

Call WLIS_SupplierOne PO private workflow

受け取った PO XML ドキュメントから抽出され、サプライヤ名を指定する一意の属性値（SupplierName）に基づいて、ワークフローは適切なタスク ノードに進み、WLIS_SupplierOne または WLIS_SupplierTwo の PO プライベート ワークフローを開始します。

[タスク プロパティ] ダイアログ ボックスのアクションは、いずれのノードでも ワークフローを開始 アクションです。[タスク プロパティ] ダイアログ ボックスで、[追加 | ワークフロー アクション | ワークフローを開始] を選択します。ワークフローを開始 アクションを追加する場合、呼び出すワークフローの名前を選択します。

たとえばこのノードでは、WLIS_SupplierOnePOPrivate ワークフローが [ワークフローを開始] ダイアログ ボックスで選択されています。[ワークフローを開始] ダイアログ ボックスでは、このアクションに関して、この他に [結果] タブで変数が指定されています。このタブには、呼び出されるワークフローの出力パラメータとして定義されている変数が表示されます。呼び出し元ワークフローは、出力パラメータを使用して、呼び出されるワークフローからの値を受け取ります。

このケースでは、結果は、WLIS_SupplierOnePOPrivate ワークフローの最後から 2 番目のノード（3-83 ページの「Get PO Ack and transform into XML」）の POAck_XML 出力パラメータに割り当てられます。呼び出し元ワークフローは、値を割り当ててから次のタスクに進みます。したがって、WLIS_SupplierPOPublic ワークフローからプライベート ワークフロー（WLIS_SupplierOnePOPrivate または

WLIS_SupplierTwoPOPrivate）への呼び出しは、同期呼び出しとなります。

Send PO Acknowledgement

このノードは、PO 確認書をサプライヤのプライベート ワークフローから受け取り、XOCP メッセージにラップして、PO 確認ビジネス メッセージを WLIS_Buyer トレーディング パートナに送信します。ビジネス メッセージは、このノードで定義されたルータ式の入力値（このケースではトレーディング パートナの名前）に基づいて、WLIS_Hub 経由でルーティングされます。Send PO Acknowledgement ノードで定義されているルータ式を確認するには、次のように、[ビジネス メッセージの送信] ダイアログ ボックスを呼び出します。

1. このノードをダブルクリックして、[タスク 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 | アクティブ化] を選択し、[ビジネス メッセージの送信] をダブルクリックして [ビジネス メッセージの送信] ダイアログ ボックスを表示します。

ダイアログ ボックスには、3 つのタブ（[メッセージ]、[トークン]、および [QoS]）があります。[メッセージ] タブの以下のパラメータに注目してください。

フィールド	値	説明
[入力メッセージ変数]	POreplyXOCPMsg	バイヤ トレーディング パートナに送信する XOCP メッセージを格納する Java オブジェクト変数。
[ルータ式の型]	Trading Partner Name	WLIS_Hub は、トレーディング パートナ名に基づいて、この PO メッセージをバイヤ トレーディング パートナにルーティングする。
[ルータ式]	$POBuyerName	PO 応答メッセージを受け取るトレーディング パートナは、POBuyerName 変数の値によって識別される。POBuyerName 変数は、このワークフローの開始ノードで設定される。

[対象ロール]	バイヤ	会話でのロールを、メッセージを受け取るトレーディング パートナに対して定義する。

Wait for Conversation Termination

システムは、会話が終了するまで待機します。PO 会話を終了するのは、会話を開始するワークフロー（WLIS_BuyerPOPublic）です。「3-73 ページの「バイヤの PO パブリック ワークフロー」」を参照。

完了

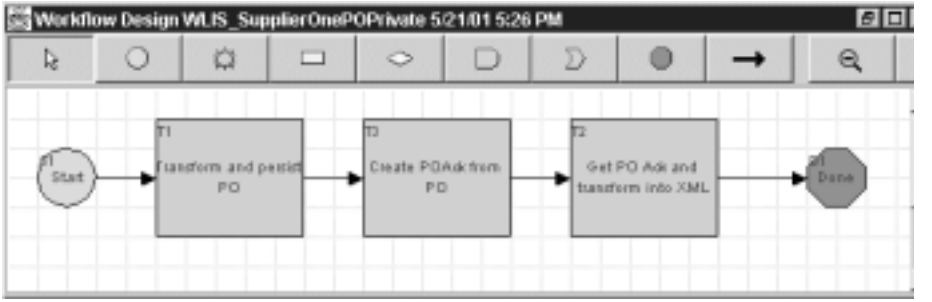
このノードでは、ワークフローは会話を終了します。

サプライヤの PO プライベート ワークフロー

サンプル アプリケーションでは、PO プロセスのサプライヤ用に、WLIS_SupplierOnePOPrivate と WLIS_SupplierTwoPOPrivate というプライベート ワークフロー テンプレートが定義されています。これらのテンプレートはほぼ同じです。

次の図は、WLIS_SupplierOnePOPrivate ワークフロー テンプレートを示しています。

図 3-25 WLIS_SupplierOnePOPrivate 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 WLIS_SupplierOnePOPrivate ワークフロー テンプレート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
- Transform and persist PO

- Create POAck from PO
- Get PO Ack and transform into XML

開始

このプライベート ワークフローは、サプライヤの PO パブリック ワークフロー（`WLIS_SupplierPOPublic`）から呼び出されたときにこのノードで開始されます。

Transform and persist PO

WLIS_SupplierOne は発注情報をバイナリ データとして保持しますが、PO は XML 形式でバイヤから送信されます。このノードでは、ワークフローは、WebLogic Integration の Data Integration の機能を使用して、受け取った XML データをバイナリ データに変換し、サプライヤのシステムで認識できるようにします。この変換には、Data Integration プラグインによって提供される Studio のアクションを使用します。

Studio でデータ統合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 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 | 統合アクション | データ統合] を選択します。

3. [Translate Binary to XML] または [Translate XML to Binary] オプションを選択します。

このケースでは、データ統合アクションは [Translate XML to Binary] に該当します。

このノードでは、変換後のバイナリ データをファイル システムに書き込むビジネス オペレーション（`binary to file`）も定義されています。このシナリオの場合、バックエンド ERP システムがファイル システムの役割を果たします。このビジネス オペレーション用の Java クラスは `examples.wlis.common.util.Utils` です。

Create POAck from PO

このノードでは、バイナリ PO データに基づいて PO 確認メッセージをバイナリ形式で作成するビジネス オペレーション（`create POAck from PO`）が定義されています。このビジネス オペレーション用の Java クラスは `examples.wlis.common.util.Utils` です。

Get PO Ack and transform into XML

このノードで定義されている最初のアクションは、Perform ビジネス オペレーションです。ビジネス オペレーション（`file to binary`）は、ファイル システムから PO 確認バイナリ データを読み出します。読み出されたデータは、`POAckbinary` 変数に割り当てられます。

次いで、このノードでは、WebLogic Integration の Data Integration の機能を使用して、PO 確認バイナリ データを XML に変換します。この変換には、Data Integration プラグインによって提供される Translate XML to Binary アクションを使用します。

変換アクションは、`POAckbinary` 変数を入力として受け取り、変換結果を `POAck_XML` 変数に割り当てます。`POAck_XML` 変数をこのように割り当てたら、このプライベート ワークフローを呼び出したパブリック ワークフロー（`WLIS_SupplierPOPublic`）は、次のタスクに進みます。3-79 ページの「Call WLIS_SupplierOne PO private workflow」を参照してください。

Studio で実行可能なデータ統合アクションの詳細については、3-88 ページの「Data Integration」を参照してください。

Application Integration と Data Integration

この節では、以下のトピックを取り上げます。

- はじめに
- Application Integration
- Data Integration

はじめに

PO ビジネス プロセスのワークフローと WLIS_BuyerQPAPrivate ワークフローは、WebLogic Integration の BPM 機能を、WebLogic Integration の Application Integration と Data Integration の機能と統合します。

バイヤサイドのプロセスでは、アプリケーション統合フレームワークを使用し、PO 確認書に基づいてエンタープライズ情報システム（EIS）を更新します。次に、このワークフローは、PO 確認情報を POAcknowledgement.xml ファイルに書き込みます。

サプライヤサイドのプロセスでは、データ統合フレームワークを使用し、XML データをバイナリ データに、またはバイナリ データを XML データに変換します。

Application Integration

WLIS_BuyerQPAPrivate および WLIS_BuyerPOPrivate ワークフローは、WebLogic Integration で提供される Application Integration の機能と、ワークフローとアプリケーション統合サービス間の会話の最も重要な部分を示します。3-31 ページの「バイヤの QPA プライベート ワークフロー」と 3-66 ページの「バイヤの PO プライベート ワークフロー」を参照してください。

企業は、元のアプリケーション コードやデータ構造を変更することなく、アプリケーション間でデータとビジネス プロセスを共有できるようにするエンタープライズ アプリケーション統合（EAI）ソリューションを必要としています。WebLogic Integration は、*アダプタ*を使用することで、エンタープライズ間の統合をサポートするアプリケーション統合フレームワークを提供します。

アダプタは、アプリケーションがエンタープライズ データにプログラム的にアクセスするために使用できるインタフェースを提供します。たとえば、アダプタは Java クラスを使用してエンタープライズ データを表したり、アプリケーションがデータにアクセスするために呼び出せるメソッドを提供したりできます。アプリケーションがアクセス メソッドを呼び出すと、アダプタはそのメソッドを実行してエンタープライズ データを検索します。WebLogic Integration で提供さ

れる Application Integration の機能は、J2EE Connector Architecture (JCA) に基づいています。WebLogic Integration で利用可能なアプリケーション統合ツールは、J2EE 標準に完全準拠している以外にも、以下の重要な機能を提供します。

- WebLogic Integration は双方向の通信をサポートしているので、Java プログラムは EIS で提供されるサービスを呼び出したり、EIS によって生成されたイベントに応答したりすることができます。
- WebLogic Integration は、XML ベースの高レベル *アプリケーション ビュー* を提供します。アプリケーション ビューは、JCA 準拠の EIS アダプタに介したビジネス サービスの抽象です。アプリケーション ビューは、Java プログラムと EIS 間のイベント / サービス インタフェースの定義で使用します。WebLogic Integration Studio では、EIS をワークフローに加える場合に使用します。

WebLogic Integration 環境の Application Integration の詳細については、『*Application Integration 入門*』を参照してください。

アプリケーション ビューを定義する場合、WebLogic Integration と EIS アプリケーション間の XML ベースのインタフェースを作成します。このサンプルでは、EIS システムは RDBMS です。サンプルで使用する Application Integration アダプタは、WebLogic Integration 製品と一緒にパッケージ化されている DBMS アダプタです。このサンプルのアプリケーション ビュー（`WLISAppView.sav`）は、DBMS アダプタに基づいて、WebLogic Integration で定義およびデプロイされています。WebLogic Integration のインストール ディレクトリの `\samples\wlis\src\examples\wlis\wlai\WLISAppViewDeployer.java` を確認してください。

アプリケーション ビューを定義、テスト、およびデプロイするには、Web ベースのユーザ インタフェースである WebLogic Integration Application View Console を使用します。アプリケーション ビューの定義方法については、『*Application Integration ユーザーズ ガイド*』を参照してください。

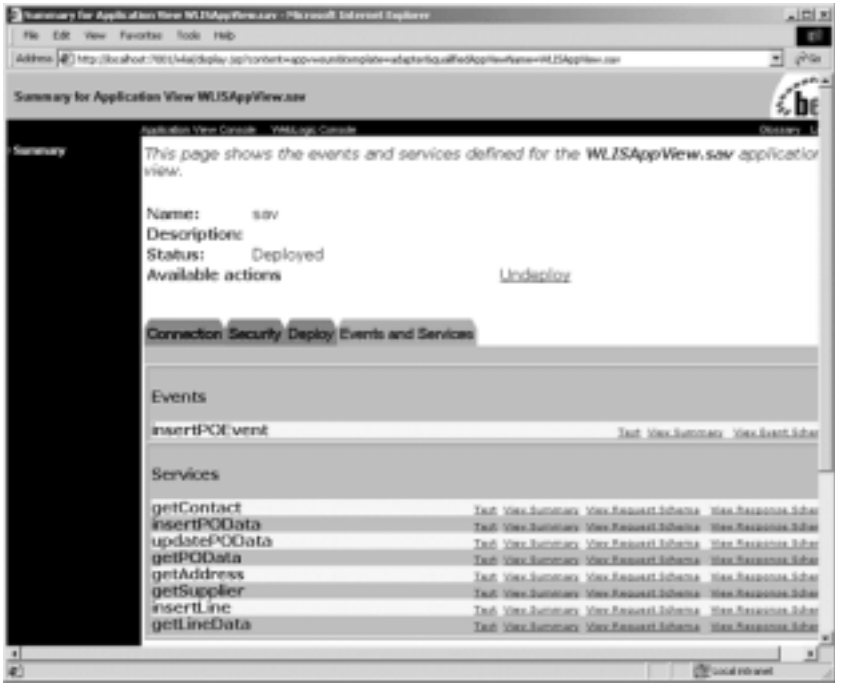
また、Application View Console を使用すると、このサンプル用に `WLISAppviewDeployer.java` Java プログラムによって作成されたアプリケーション ビューの詳細を表示できます。手順は次のとおりです。

1. 既に実行中の場合は、2-1 ページの「サンプルの設定と実行」の説明に従って WebLogic Integration サンプルを実行します。

2. 以下のいずれか 1 つを実行します。
   - サンプルを起動した Web ページ（`http://localhost:7001`）上の [AI] リンクをクリックします。
   - [スタート | プログラム | BEA WebLogic Platform 7.0 | WebLogic Integration 7..0 | Application View Console] を選択します。

   Application View Console が表示されます。画面には [Root] フォルダが表示されます。このフォルダには、エンタープライズ用のアプリケーション ビューを整理したフォルダ リストが含まれています。このウィンドウのフォルダの 1 つが `WLISAppView` で、このサンプルのアプリケーション ビューを格納しています。
3. `WLISAppView` をクリックして、アプリケーション ビューのリストをウィンドウに表示します。このリストには、サンプル アプリケーション用に定義されているアプリケーション ビュー（`sav`）が含まれています。ステータスが Deployed になっていることに注目してください。
4. `sav` をダブルクリックして、次の図に示す `WLISAppView.sav` アプリケーション ビューの詳細を表示します。

図 3-26　WebLogic IntegrationApplication View Console



WLISAppView.sav アプリケーション ビューには、以下のサービスおよびイベントが含まれています。

サービス	getContact updatePOData insertPOData getPOData getAddress getSupplier insertLine getLineData
イベント	insertPOEvent

Application View Console の以下のリンクをクリックすると、サービスまたはイベントの詳細が表示されます。

- View Summary － SQL 式が表示されます。
- View Request Schema －要求データの XML スキーマが表示されます。
- View Response Schema －応答データの XML スキーマが表示されます。

Data Integration

`WLIS_SupplierOnePOPrivate` および `WLIS_SupplierTwoPOPrivate` ワークフローは、WebLogic Integration で提供される Data Integration の機能の最も重要な部分を示します。この機能は、XML からバイナリ形式への変換、またはバイナリ形式から XML への変換で使用されます。このサンプル アプリケーションの場合、以下の変換が実行されます。

- PO XML データがバイナリ データに変換されます。
- PO 確認バイナリ データが XML に変換されます。

これらの変換については、3-81 ページの「サプライヤの PO プライベート ワークフロー」を参照してください。

データ統合をサポートするために、Studio では以下の Data Integration プラグインのアクションを利用できます。

- Translate Binary to XML
- Translate XML to Binary

Studio を使用してタスク ノードでデータ変換を実行するには、次の手順に従います。

1. ノードをダブルクリックして、[タスク 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] ペインで、[追加] をクリックして [アクションを追加] ダイアログ ボックスを表示します。
3. [統合アクション｜データ統合] を選択します。

4. 以下のアクションからいずれか 1 つを選択します。
   - Translate Binary to XML

     [Binary to XML] ダイアログ ボックスが表示されます。
   - Translate XML to Binary

     [XML to Binary] ダイアログ ボックスが表示されます。
5. [Binary to XML]（または [XML to Binary]）ダイアログ ボックスで、WebLogic Integration リポジトリを参照して利用可能なマップを選択します。

   次の図は、3-81 ページの「サプライヤの PO プライベート ワークフロー」で説明した `WLIS_SupplierTwoPOPrivate` ワークフロー テンプレートで最後のタスク ノード（Get PO Ack and transform into XML）に対応する [Binary to XML] ダイアログ ボックスを示しています。

図 3-27 [Binary to XML] ダイアログ ボックス

上の図の以下の点に注目してください。

- `POAck` ([メッセージ形式] 領域の [名前] フィールドの値) は、`POAck.mfl` 変換マップを表します。

- 入力および出力変数の設定では、POAckBinary 変数のバイナリ データが変換されて、POAck_XML 変数に書き込まれることを指定しています。

このサンプル アプリケーションの場合、2 つの変換マップ（PO.mfl と POAck.mfl）が Format Builder 設計ツールで作成され、WebLogic Integration リポジトリに保存されています。workflow.jar パッケージに格納されているマップは、サンプルのセットアップおよびコンフィグレーション時にインポートされます。

このサンプルの変換マップを表示するには、Format Builder ツールを起動します。次の節を参照してください。

Format Builder の起動

Format Builder 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:
  - メニューから Format Builder を起動する場合

    [スタート | プログラム | BEA WebLogic Platform 7.0 | WebLogic Integration 7.0 | Format Builder] を選択します。
  - コマンド ラインから Format Builder を起動する場合

  a. コマンド ウィンドウを開きます。

  b. setenv スクリプトを実行して、WebLogic Integration 環境変数を設定します。

     ```
     cd WLI_HOME
     setenv
     ```

     WLI_HOME は WebLogic Integration がインストールされているディレクトリを表します。

  c. WebLogic Integration のインストール ディレクトリの bin ディレクトリに移動して、fb コマンドを実行します。たとえば、WebLogic Integration をデフォルトのディレクトリにインストールした場合、次のコマンド シーケンスを入力します。

     ```
     cd WLI_HOME\bin
     fb.cmd
     ```

- UNIX:

  a. setenv スクリプトを実行して、WebLogic Integration 環境変数を設定します。

  ```
  cd WLI_HOME
  setenv
  ```

  WLI_HOME は WebLogic Integration がインストールされているディレクトリを表します。

  b. サンプル ドメインの bin ディレクトリに移動します。

  たとえば、次を入力します。

  ```
  cd WLI_HOME\bin
  ```

  BEA_Home は、BEA 製品のデフォルトのインストール先ディレクトリを表します。

  c. 次のように入力して Format Builder を起動します。

  ```
  . ./fb
  ```

サンプルの変換マップの表示

このサンプルの変換マップを表示するには、次の手順に従います。

- Format Builder のメニュー オプションから、[File | Open] を選択します。

  システム上のファイルを参照するためのダイアログ ボックスが表示されます。

- WebLogic Platform のインストール ディレクトリの次のディレクトリを選択します。

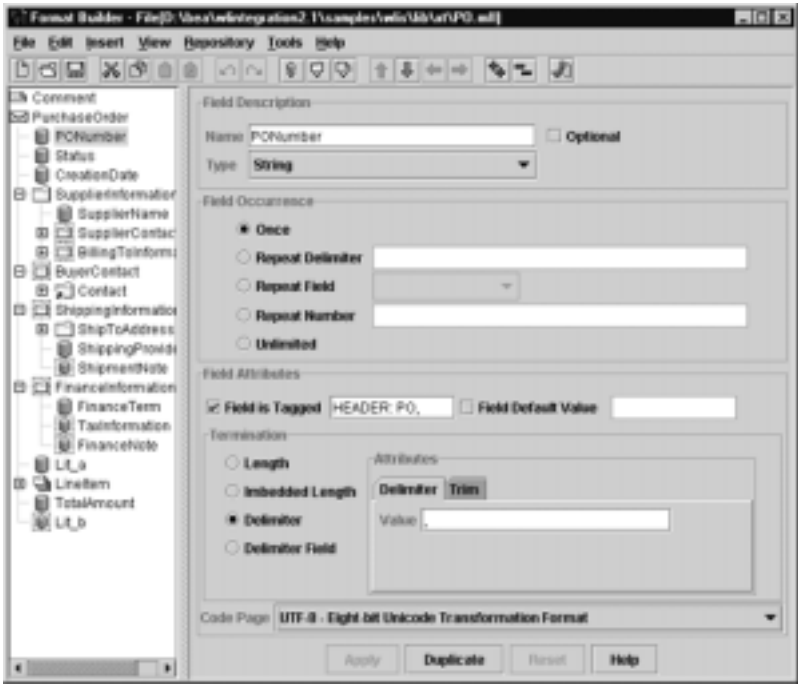
  SAMPLE_HOME\integration\samples\wlis\lib\xt

  上の行の *SAMPLES_HOME* は、WebLogic Platform のサンプル ディレクトリです。

- 以下のマップ ファイルを選択します。
  - PO.mfl
  - POAck.mfl

Format Builder 設計ツールを使用して変換マップを作成およびテストする方法については、『*WebLogic Integration データ変換*』を参照してください。次の図は、Format Builder で定義中の `PO.mfl` 変換マップを示しています。

図 3-28　Format Builder 内の PO マップ



A DTD

WebLogic Integration サンプルで使用する XML データの DTD は、WebLogic Integration のインストール ディレクトリの次のディレクトリにあります。

```
\samples\wlis\lib\dtds
```

比較しやすいように、この付録にまとめて記載してあります。内容は以下のとおりです。

- 共通の DTD
- QPA の DTD
- PO の DTD

共通の DTD

リスト A-1 common.dtd

```
<!-- Some common definition used by the DTD -->

<!ENTITY % ADDRESS "Address">
<!ELEMENT Address (Street1, Street2?, Street3?, City, State,
ZipCode)>
<!ELEMENT Street1 (#PCDATA)>    <!-- String -->
<!ELEMENT Street2 (#PCDATA)>    <!-- String -->
<!ELEMENT Street3 (#PCDATA)>    <!-- String -->
<!ELEMENT City    (#PCDATA)>    <!-- String -->
<!ELEMENT State   (#PCDATA)>    <!-- String -->
<!ELEMENT ZipCode (#PCDATA)>    <!-- String -->

<!ENTITY % CONTACT "Contact">
<!ELEMENT Contact (ContactName, ContactPhone, ContactEmail?,
                   ContactFax?, ContactAddress)>
<!ELEMENT ContactName    (#PCDATA)>    <!-- String -->
<!ELEMENT ContactPhone   (#PCDATA)>    <!-- String -->
<!ELEMENT ContactPhone   (#PCDATA)>    <!-- String -->
<!ELEMENT ContactEmail   (#PCDATA)>    <!-- String -->
```

```
<!ELEMENT ContactFax     (#PCDATA)>    <!-- String -->
<!ELEMENT ContactAddress (%ADDRESS;)>
```

QPA の DTD

リスト A-2 QPARequest.dtd

```
<!ELEMENT QPARequest (QPARequestId, CreationDate, Availability+)>
    <!ELEMENT QPARequestId (#PCDATA)>  <!-- String -->
   <!ELEMENT CreationDate (#PCDATA)>  <!-- DateStamp (CCYYMMDD) -->
    <!ELEMENT Availability (PartId, Quantity, Price, RequiredDate,
     Location*, Note?)>

        <!ELEMENT PartId       (#PCDATA)>    <!-- String -->
        <!ELEMENT Quantity     (#PCDATA)>    <!-- Long  -->
        <!ELEMENT UnitPrice        (#PCDATA)>    <!-- Float  -->
        <!ELEMENT RequiredDate (#PCDATA)>    <!-- DateStamp
(CCYYMMDD) -->
        <!ELEMENT Location     (#PCDATA)>    <!-- String -->
        <!ELEMENT Note         (#PCDATA)>    <!-- String -->
```

リスト A-3 QPAResponse.dtd

```
<!ELEMENT QPAResponse (QPAResponseId, ResponseDate, QPARequestId,
CreationDate, SupplierName, Availability+)>
<!ELEMENT QPAResponseId (#PCDATA)>    <!-- String -->
<!ELEMENT ResponseDate (#PCDATA)>    <!-- DateStamp (CCYYMMDD) -->
<!ELEMENT QPARequestId (#PCDATA)>    <!-- String -->
<!ELEMENT CreationDate (#PCDATA)>    <!-- DateStamp (CCYYMMDD) -->
<!ELEMENT SupplierName (#PCDATA)>    <!-- String -->
<!ELEMENT Availability (PartId, Quantity, Price, AvailableDate,
                        Location*, Note?)>

        <!ELEMENT PartId        (#PCDATA)>    <!-- String -->
        <!ELEMENT Quantity      (#PCDATA)>    <!-- Long  -->
        <!ELEMENT UnitPrice         (#PCDATA)>    <!-- Float  -->
        <!ELEMENT AvailableDate (#PCDATA)>
                                <!-- DateStamp(CCYYMMDD) -->
        <!ELEMENT Location      (#PCDATA)>    <!-- String -->
        <!ELEMENT Note          (#PCDATA)>    <!-- String -->
```

リスト A-4 Aggregated QPAResponse.dtd

```
<!ELEMENT AggregatedQPAResponse (QPARequestId, CreationDate,
QPAResponse+>
<!ELEMENT QPAResponse (QPAResponseId, ResponseDate, SupplierName,
Availability+)>
<!ELEMENT QPARequestId (#PCDATA)>    <!-- String -->
<!ELEMENT CreationDate (#PCDATA)>    <!-- DateStamp (CCYYMMDD) -->
<!ELEMENT QPAResponseId (#PCDATA)>    <!-- String -->
<!ELEMENT ResponseDate (#PCDATA)>    <!-- DateStamp (CCYYMMDD) -->
<!ELEMENT SupplierName (#PCDATA)>    <!-- String -->

<!ELEMENT Availability (PartId, Quantity, UnitPrice, AvailableDate,
Location*, Note?)>
<!ELEMENT PartId        (#PCDATA)>    <!-- String -->
<!ELEMENT Quantity      (#PCDATA)>    <!-- Long  -->
<!ELEMENT UnitPrice         (#PCDATA)>    <!-- Float  -->
<!ELEMENT AvailableDate (#PCDATA)>    <!-- DateStamp (CCYYMMDD) -->
<!ELEMENT Location      (#PCDATA)>    <!-- String -->
<!ELEMENT Note          (#PCDATA)>    <!-- String -->
```

PO の DTD

リスト A-5 PORequest.dtd

```
<!ELEMENT PORequest (SupplierName, QPAResponseId, QPARequestId,
POItems+)>
<!ELEMENT PORequest    (#PCDATA)>    <!-- String -->
<!ELEMENT SupplierName (#PCDATA)>    <!-- String -->
<!ELEMENT QPAResponseId (#PCDATA)>   <!-- String -->
<!ELEMENT QPARequestId (#PCDATA)>    <!-- String -->
<!ELEMENT POItems (PartId, Quantity, UnitPrice, DeliveryDate,
Location*, Note?)>
<!ELEMENT PartId       (#PCDATA)>    <!-- String -->
<!ELEMENT Quantity     (#PCDATA)>    <!-- Long  -->
<!ELEMENT UnitPrice    (#PCDATA)>    <!-- Float  -->
<!ELEMENT DeliveryDate (#PCDATA)>    <!-- DateStamp (CCYYMMDD) -->
<!ELEMENT Location     (#PCDATA)>    <!-- String -->
<!ELEMENT Note         (#PCDATA)>    <!-- String -->
```

リスト A-6 PO.dtd

```
<!ELEMENT PurchaseOrder (PONumber, Status, CreationDate,
SupplierInformation, BuyerContact?, ShippingInformation?,
FinanceInformation?, LineItem+, TotalAmount?)>

    <!ELEMENT PONumber     (#PCDATA)>       <!-- String -->
    <!ELEMENT Status       (#PCDATA)>       <!-- String -->
   <!ELEMENT CreationDate (#PCDATA)>  <!-- DateStamp (CCYYMMDD) -->
   <!ELEMENT BuyerContact (%CONTACT;)>    <!-- Entity: CONTACT -->

<!ELEMENT ShippingInformation (ShipToAddress?,
                               ShippingProvider, ShipmentNote?)>
    <!ELEMENT ShipToAddress   (%ADDRESS;)>  <!-- Entity:ADDRESS -->
    <!ELEMENT ShippingProvider (#PCDATA)>       <!-- String -->
    <!ELEMENT ShipmentNote     (#PCDATA)>       <!-- String -->

<!ELEMENT FinanceInformation (FinanceTerm, TaxInformation?,
FinanceNote?)>

     <!ELEMENT FinanceTerm    (#PCDATA)>    <!-- String -->
     <!ELEMENT TaxInformation (#PCDATA)>    <!-- String -->
     <!ELEMENT FinanceNote    (#PCDATA)>    <!-- String -->

<!ELEMENT SupplierInformation(SupplierName, SupplierContact?,
                                   BillingToInformation?)>

     <!ELEMENT SupplierName    (#PCDATA)>
     <!ELEMENT SupplierContact (%CONTACT)>
```

```
     <!ELEMENT BillingToInformation (BillToAddress, AccountId)>
      <!ELEMENT BillToAddress (%ADDRESS;)> <!-- Entity:ADDRESS -->
      <!ELEMENT AccountID     (#PCDATA)>     <!-- String -->

<!ELEMENT LineItem (LineNumber, PartId, PartDescription?, Quantity,
UnitPrice, DeliveryDate,  Note?)>

        <!ELEMENT LineNumber      (#PCDATA)>    <!-- String -->
        <!ELEMENT PartId          (#PCDATA)>    <!-- String -->
        <!ELEMENT PartDescription (#PCDATA)>    <!-- String -->
        <!ELEMENT Quantity        (#PCDATA)>    <!-- Long  -->
       <!ELEMENT UnitPrice          (#PCDATA)>    <!-- Float  -->
        <!ELEMENT DeliveryDate    (#PCDATA)>    <!-- DateStamp
(CCYYMMDD) -->
        <!ELEMENT Note            (#PCDATA)>    <!-- String -->
<!ELEMENT TotalAmount           (#PCDATA)>    <!-- Float -->
```

リスト A-7　POAcknowledgement.dtd

```
<!ELEMENT PurchaseOrderAcknowledgement (PONumber, POCreationDate,
SONumber, SOCreationDate, SupplierInformation,
ShippingInformation?, FinanceInformation?, LineItem+,
TotalAmount?)>

  <!ELEMENT PONumber     (#PCDATA)>    <!-- String -->
  <!ELEMENT POCreationDate (#PCDATA)>    <!-- DateStamp (CCYYMMDD)
-->
  <!ELEMENT SONumber     (#PCDATA)>    <!-- String -->
  <!ELEMENT SOCreationDate (#PCDATA)>    <!-- DateStamp (CCYYMMDD)
-->
<!ELEMENT SupplierInformation(SupplierName, SupplierContact?,
BillingToInformation?)>
    <!ELEMENT SupplierName    (#PCDATA)>
    <!ELEMENT SupplierContact (%CONTACT)>
    <!ELEMENT BillingToInformation (BillToAddress, AccountId)>
    <!ELEMENT BillToAddress (%ADDRESS;)> <!-- Entity: ADDRESS -->
    <!ELEMENT AccountID     (#PCDATA)> <!-- String -->

<!ELEMENT ShippingInformation (ShipToAddress?, ShippingProvider,
TrackingId, ShipmentNote?)>
   <!ELEMENT ShipToAddress   (%ADDRESS;)> <!-- Entity: ADDRESS -->
    <!ELEMENT ShippingProvider (#PCDATA)>       <!-- String -->
    <!ELEMENT TrackingId       (#PCDATA)>       <!-- String -->
    <!ELEMENT ShipmentNote     (#PCDATA)>       <!-- String -->
<!ELEMENT FinanceInformation (FinanceTerm,
TaxInformation?,FinanceNote?)>
    <!ELEMENT FinanceTerm    (#PCDATA)>    <!-- String -->
    <!ELEMENT TaxInformation (#PCDATA)>    <!-- String -->
    <!ELEMENT FinanceNote    (#PCDATA)>    <!-- String -->
<!ELEMENT LineItem (LineNumber, PartId, PartDescription?, Quantity,
UnitPrice, DeliveryDate,  Note?)>
    <!ELEMENT LineNumber      (#PCDATA)>    <!-- String -->
    <!ELEMENT PartId          (#PCDATA)>    <!-- String -->
```

```
    <!ELEMENT PartDescription (#PCDATA)>    <!-- String -->
    <!ELEMENT Quantity        (#PCDATA)>    <!-- Long  -->
    <!ELEMENT UnitPrice       (#PCDATA)>    <!-- Float  -->
   <!ELEMENT DeliveryDate  (#PCDATA)> <!-- DateStamp (CCYYMMDD) -->
    <!ELEMENT Note            (#PCDATA)>    <!-- String -->
<!ELEMENT TotalAmount           (#PCDATA)>    <!-- Float -->
```