BEA WebLogic Integration™

B2B Integration
cXML の実装

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

B2B Integration cXML の実装

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

4. cXML でのワークフローの使用

索引

このマニュアルの内容

WebLogic Integration は、XOCP、RosettaNet、および cXML メッセージの管理および解決を可能とするルーティング アーキテクチャをサポートしています。このアーキテクチャにより、WebLogic Integration ではこれらのいずれのプロトコル標準を使用しても、企業間での対話に参加することができます。

このマニュアルでは、WebLogic Integration の cXML 機能について説明します。

注意： XOCP および cXML ビジネス プロトコルは、WebLogic Integration の本リリースより非推奨になりました。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。

WebLogic Integration 上の cXML は、cXML メッセージの送受信機能を提供します。詳細については、「*cXML User 痴 Guide*」（`http://www.cxml.org`）を参照してください。

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、WebLogic Integration 上の cXML についての概要と、WebLogic Integration に cXML を実装するためのアーキテクチャについて説明します。
- 第 2 章「cXML の管理」では、WebLogic Integration における cXML 固有の管理およびセキュリティに関する問題について説明します。
- 第 3 章「cXML API の使用」では、cXML API の概要および使い方について説明します。
- 第 4 章「cXML でのワークフローの使用」では、WebLogic Integration Studio を使用して、cXML を組み込んだワークフローを作成する方法について説明します。

対象読者

このマニュアルは主に、次のユーザを対象としています。

- WebLogic Integration Studio を使用して、WebLogic Integration 環境と統合可能なワークフローを、特に cXML の実装に主力を置いて設計するビジネスプロセス設計者
- cXML API を使用して、WebLogic Integration を使用するバイヤまたはサプライヤ アプリケーションを実装するアプリケーション開発者
- cXML 環境で WebLogic Integration アプリケーションを設定および管理するシステム管理者

WebLogic Integration のアーキテクチャの概要については、『*B2B Integration 入門*』の「概要」を参照してください。

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 マニュアル CD にあります。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。

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

Java 2 Enterprise Edition（J2EE）、eXtensible Markup Language（XML）、および Java プログラミングについての詳細は、次の URL にある Javasoft Web サイトを参照してください。
`http://java.sun.com`

また、次の URL にある BEA e-docs Web サイトでも、役に立つ情報を入手できます。
`http://edocs.beasys.co.jp/e-docs/`

cXML の詳細については、次の URL にある cXML.org の Web サイトを参照してください。
`http://www.cxml.org`

サポート情報

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、BEA WebLogic Integration リリース 7.0 のドキュメントをご使用の旨をお書き添えください。

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

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

- お名前、電子メール アドレス、電話番号、ファクス番号

- 会社の名前と住所
- お使いの機種とコード番号
- 製品の名前とバージョン
- 問題の状況と表示されるエラー メッセージの内容

表記規則

このマニュアルでは、全体を通して以下の表記規則が使用されています。

表記法	適用
太字	用語集で定義されている用語を示す。
〔Ctrl〕+〔Tab〕	複数のキーを同時に押すことを示す。
斜体	強調または書籍のタイトルを示す。

等幅テキスト	コード サンプル、コマンドとそのオプション、データ構造体とそのメンバー、データ型、ディレクトリ、およびファイル名とその拡張子を示す。等幅テキストはキーボードから入力するテキストも示す。
	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

注意： cXML ビジネス プロトコルは、WebLogic Integration の本リリースより非推奨になりました。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。

この章では、電子ビジネス トランザクション用の cXML 標準について説明します。cXML は Ariba によって開発された拡張可能な電子商取引向け XML 標準で、電子商取引の購買トランザクションに広く採用されています。

このセクションでは、cXML 標準および WebLogic Integration で使用する場合について説明します。

- WebLogic Integration アーキテクチャと cXML
- cXML API
- ビジネス ドキュメント
- デジタル署名と共有秘密
- メッセージ検証
- 制限事項

WebLogic Integration アーキテクチャと cXML

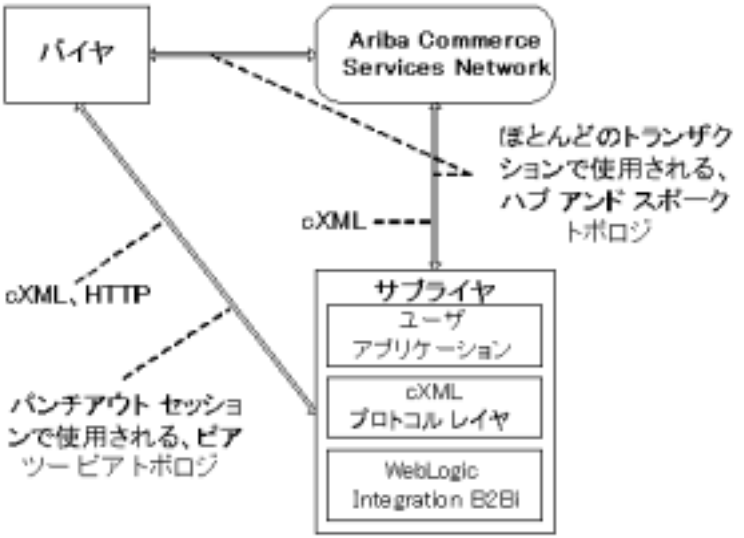
WebLogic Integration が提供する cXML サポートは、以下のコンポーネントから構成されています。

- cXML プロトコル レイヤ
- cXML API サポート

cXML 統合は、Business Process Management（BPM）のビジネス処理と cXML API を使用することによって実現されます。詳細については、第 3 章「cXML API の使用」および第 4 章「cXML でのワークフローの使用」を参照してください。

次の図に、WebLogic Integration が採用している cXML アーキテクチャと、WebLogic Integration が cXML を使用して他のシステムとやり取りする方法を示します。.

図 1-1 WebLogic Integration の cXML アーキテクチャ



WebLogic Integration の cXML サポートにより、cXML と標準 B2B Integration インフラストラクチャのシームレスな統合が可能になります。B2B Integration アーキテクチャの詳細については、『*B2B Integration 入門*』を参照してください。

cXML の設計環境のために、Ariba Commerce Services Network 以外のハブのサポートは提供されていません。

cXML プロトコル レイヤ

cXML プロトコル レイヤは、転送、メッセージ パッケージ化、およびセキュリティの cXML 仕様に従って、インターネットを介してメッセージを送受信する機能を提供します。WebLogic Integration は、個々の cXML セッションを作成します。各セッションは、WebLogic Integration サーバが cXML メッセージを受信する URL を作成および管理します。WebLogic Integration B2B Console またはコンフィグレーション ファイルのいずれかを使用することで、必要に応じて cXML セッションをコンフィグレーションできます。WebLogic Integration コンフィグレーション ファイルを使用するには、`WLC.dtd` ファイルに基づいてコンフィグレーション ファイルを作成して、必要に応じて cXML セッションをコンフィグレーションします。この方法を採用する場合は、Bulk Loader を使用してコンフィグレーション ファイルをリポジトリにロードします。

cXML API

WebLogic Integration には、cXML ユーザ アプリケーションを作成するための包括的な API サポートが用意されています。cXML API の詳細については、第 3 章「cXML API の使用」および『*BEA WebLogic Integration Javadoc*』を参照してください。

ビジネス ドキュメント

ビジネス ドキュメントの処理は、*パブリック プロセス*と*プライベート プロセス*の組み合わせを使用して WebLogic Integration で実行されます。*パブリック プロセス*は、トレーディング パートナ間のトランザクションを統合および管理するために使用されるプロセスです。*プライベート プロセス*は、トレーディング パートナによって内部的に使用されるプロセスです。たとえば、企業の*パブリック プロセス*とその内部の ERP および CRM システム間のやり取りに使用されます。このため、*プライベート プロセス*はトレーディング パートナが直接使用することはありません。詳細については、『B2B Integration 入門』の「*概要*」の「ビジネス プロセスの管理」を参照してください。

cXML ビジネス ドキュメントは、トレーディング パートナが電子ビジネス トランザクションの実行中に参加する*パブリック プロセス*の一部です。たとえば、*パンチアウト* は、*顧客*トレーディング パートナが購入を望み、*製品サプライヤ*トレーディング パートナが販売を望んでいる商品について、その*顧客* がその価格と在庫の情報をリポジトリから取得するために*製品サプライヤ*と一緒に実行するプロセスです。*パンチアウト* を使用するトレーディング パートナは、以下のことを行う必要があります。

- *パンチアウト* に関連付けられるパブリック プロセスを実装する
- 内部システム、およびプライベート プロセスとワークフローをパブリック プロセスに接続する

WebLogic Integration は、cXML で使用可能な以下のすべてのビジネス ドキュメントを実装します。

- *カタログ*
- パンチアウト
- 発注書
- サブスクリプション

cXML ビジネス ドキュメントの詳細については、次の URL の cXML.org Web サイトを参照してください。

`http://www.cxml.org`

デジタル署名と共有秘密

cXML でトランザクションをセキュアにする標準の方法は、*共有秘密* です。cXML 用語では、共有秘密とは通常ユーザ名 / パスワードの組み合わせのことです。これは、ビジネス通信の開始前にセキュアな転送によって交換されます。

WebLogic Integration は、cXML 共有秘密を完全にサポートしています。共有秘密の実装とコンフィグレーションの詳細については、『*B2B Integration Administration Console オンライン ヘルプ*』を参照してください。また、オプションでメッセージ転送に https を使用することもできます。

cXML v1.2 では、Base64-encoded X.509 V3 証明書モデルに基づくオプションのデジタル署名が導入されました。これらのデジタル署名は、WebLogic Integration に実装されている RSA CertJ デジタル署名と同じではありません。現時点では、WebLogic Integration は cXML デジタル署名をサポートしていません。この詳細については、次の URL の cXML.org Web サイトを参照してください。

http://www.cxml.org

メッセージ検証

cXML 標準では、すべての cXML ドキュメントは有効であり、発行された cXML 文書型定義（DTD）を参照する必要があります。cXML 標準では検証は必要とされていませんが、WebLogic Integration では検証機能がサービスとして提供されています。

制限事項

cXML 関連機能のいくつかは、WebLogic Integration のこのリリースではサポートされていません。

- デジタル証明書 : 前述のとおり、cXML 1.2 デジタル署名書はサポートされていません。
- cXML 1.2 添付ファイルは、現時点ではサポートされていません。
- 非 ACSN ハブのサポート : Ariba Commerce Services Network 以外のハブはサポートされていません。すべてのハブベースのトランザクションは、ACSN を介してルーティングされなければなりません。ピア ツー ピアのサポートは提供されています。
- 現時点ではサンプル実装は提供されていません。

2 cXML の管理

注意： cXML ビジネス プロトコルは、WebLogic Integration の本リリースより非推奨になりました。代替機能に関する情報については、『*BEA WebLogic Integration リリース ノート*』を参照してください。

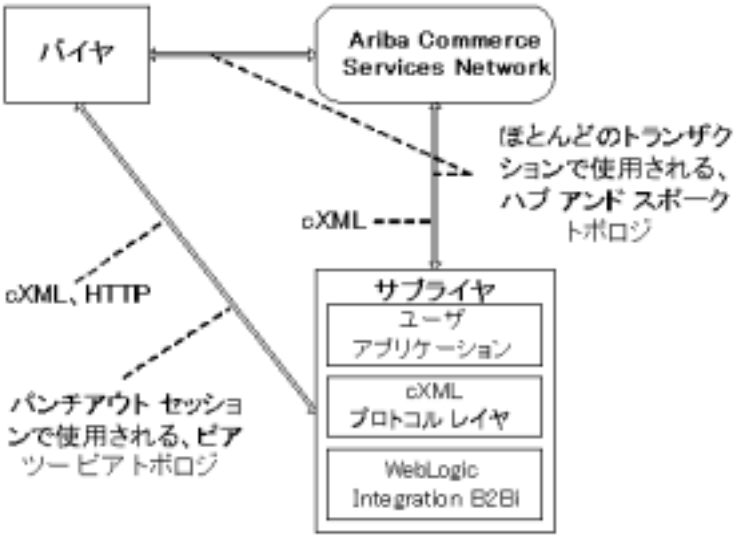
cXML トランザクションの管理は、WebLogic Integration B2B Console を使用して行います。以下の節では、cXML トランザクションのサポートに必要な管理タスクについて説明します。

- 他の cXML トレーディング パートナへの接続
- コラボレーション アグリーメント
- セキュリティ

他の cXML トレーディング パートナへの接続

cXML トレーディング パートナ間の対話では、ピア ツー ピアとハブ アンド スポークの両方のコンフィグレーションが使用されます。これらのコンフィグレーションは、『*B2B Integration 入門*』で、ほとんどの状況に適用されると説明しましたが、cXML でのこれらの使い方は若干異なります。次の図に、通常の cXML トランザクションで両方のトポロジが同時に使用される様子を示します。

図 2-1 cXML デプロイメント コンフィグレーション



この図では、Ariba Commerce Services Network（ACSN）がハブです。ほとんどのトランザクションはハブを介して実行され、個々のトレーディング パートナは関連するスポークとなります。しかし、パートナのカタログを参照するときには、直接リモート システムに接続するパンチアウト トレーディング セッションが作成されます。この場合、ハブ アンド スポーク トポロジはバイパスされ、ピア ツー ピア コンフィグレーションが使用されます。パンチアウト セッションが終了し、バイヤが注文またはサブスクリプションを送信すると、システム トポロジはハブ アンド スポーク モデルに戻り、ACSN は再びハブとして機能します。

WebLogic Integration で cXML を使用する場合、ACSN が唯一の認可されたハブであることに注意してください。WebLogic Integration はこれ以外のハブ アンド スポーク デプロイメントのサポートを提供せず、他のどのようなシステムも cXML ベース トランザクション用のハブとして機能することはできません。

コラボレーション アグリーメント

cXML で使用するコラボレーション アグリーメントの範囲と効果は、他のトレーディング プロトコル用にコンフィグレーションされたコラボレーション アグリーメントとほぼ同じです。コラボレーション アグリーメントの詳細については、『*B2B Integration 管理ガイド*』および『*B2B Integration Administration Console オンライン ヘルプ*』を参照してください。

コラボレーション アグリーメントのコンフィグレーションにおける 1 つの重要な相違点は、資格のコンフィグレーション方法にあります。cXML は Ariba Commerce Services Network を認証ハブとして使用するので、すべての資格は、別のトレーディング パートナではなく ACSN に関連してコンフィグレーションされます。

このため、共有秘密は、自社とトレーディング パートナ間のみで定義されたものではなく、常に ACSN に登録されます。

セキュリティ

WebLogic Integration は、cXML 1.1 で具象化されたセキュリティ モデルをサポートしています。このモデルでは、メッセージの真実性を検証するために共有秘密の概念が使用されます。共有秘密は、特定のパートナのアイデンティティを検証するために使用されるパスワードまたは文字列です。パスワードと同じように、特定のトレーディング パートナ エンティティが特定の共有秘密にリンクされて、1 対 1 のアイデンティティ マッピングが提供されます。ただし、複数のトレーディング パートナが同一の共有秘密を使用することを禁止する規定は存在しません。

cXML 1.2 は、Base64-encoded X.509 V3 証明書モデルに基づくデジタル署名の特定の実装を採用しています。現時点では、WebLogic Integration はこのデジタル署名の実装をサポートしていません。

共有秘密のコンフィグレーション

共有秘密をコンフィグレーションするには、WebLogic Integration B2B Console を使用します。その手順については、『*B2B Integration Administration Console オンライン ヘルプ*』を参照してください。

3 cXML API の使用

注意： cXML ビジネス プロトコルは、WebLogic Integration の本リリースより非推奨になりました。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。

以下の節では、cXML API の主要なプログラミング問題について説明します。

- cXML メソッド
- cXML メッセージの構造
- cXML DTD
- 共有秘密の処理
- 受信メッセージの処理
- 送信メッセージの処理

ビジネス処理のプログラミングの詳細については、『*B2B Integration ワークフローの作成*』を参照してください。

cXML メソッド

次の表に、cXML メッセージを操作するためのメソッドを示します。

表 3-1 パブリック cXML メソッド

メソッド	パッケージ	説明
onMessage	com.bea.b2b.protocol.cxml.CXMLListener	送信された CXMLMessage を受信する。
deregister	com.bea.b2b.protocol.cxml.CXMLManager	この CXMLManager からアプリケーションの登録を解除する。プロパティ セットを使用して登録を選択する。
getInstance	com.bea.b2b.protocol.cxml.CXMLManager	CXMLManager のインスタンスを取得する。
getSharedSecret	com.bea.b2b.protocol.cxml.CXMLManager	このトレーディング パートナの共有秘密を取得する。トレーディング パートナ名を使用して共有秘密を検索する。
register	com.bea.b2b.protocol.cxml.CXMLManager	この CXMLManager にアプリケーションを登録する。プロパティ セットを使用して、このトレーディング パートナのコラボレーション アグリーメントを選択する。 cXML メッセージの送信に使用する。
getHttpStatusCode	com.bea.b2b.protocol.cxml.CXMLHttpStatusException	例外から HTTP ステータス コードを返す。
getAsString	com.bea.b2b.protocol.cxml.messaging.CXMLDocument	cXML パートを文字列として取得する。
getDocument	com.bea.b2b.protocol.cxml.messaging.CXMLDocument	関連付けられている XML ドキュメントを取得する。

表 3-1 パブリック cXML メソッド

メソッド	パッケージ	説明
getFromCredentialDomains	com.bea.b2b.protocol.cxml.messaging.CXMLDocument	ドキュメント ヘッダから From Credential Domain を取得する。
getFromCredentialIdentities	com.bea.b2b.protocol.cxml.messaging.CXMLDocument	ドキュメント ヘッダから From Credential Identity を取得する。
getIdentifier	com.bea.b2b.protocol.cxml.messaging.CXMLDocument	ドメイン識別子またはメッセージ識別子を取得する。
getNodeValue	com.bea.b2b.protocol.cxml.messaging.CXMLDocument	指定された XPath 式を使用してドキュメント ノードの値を取得して、ドキュメント ノードを検索する。複数のノードが XPath 式に一致した場合、最初のノードだけが使用される。
getSenderCredentialDomain	com.bea.b2b.protocol.cxml.messaging.CXMLDocument	ドキュメント ヘッダから Sender Credential Domain を取得する。
getSenderCredentialIdentity	com.bea.b2b.protocol.cxml.messaging.CXMLDocument	ドキュメント ヘッダから Sender Credential Identity を取得する。
getSenderSharedSecret	com.bea.b2b.protocol.cxml.messaging.CXMLDocument	ドキュメント ヘッダから Sender Credential Shared Secret を取得する。
getSenderUserAgent	com.bea.b2b.protocol.cxml.messaging.CXMLDocument	ドキュメント ヘッダから Sender User Agent を取得する。
getTimeStamp	com.bea.b2b.protocol.cxml.messaging.CXMLDocument	ドキュメント タイムスタンプまたはメッセージ タイムスタンプのいずれかを取得する。
getToCredentialDomain	com.bea.b2b.protocol.cxml.messaging.CXMLDocument	ドキュメント ヘッダから To Credential Domain を取得する。
getToCredentialIdentity	com.bea.b2b.protocol.cxml.messaging.CXMLDocument	ドキュメント ヘッダから To Credential Identity を取得する。
getVersion	com.bea.b2b.protocol.cxml.messaging.CXMLDocument	ドキュメント バージョンを取得する。

表 3-1 パブリック cXML メソッド

メソッド	パッケージ	説明
`setDocument`	`com.bea.b2b.protocol.cxml.messaging.CXMLDocument`	関連付けられている XML ドキュメントを設定する。
`setNodeValue`	`com.bea.b2b.protocol.cxml.messaging.CXMLDocument`	指定された XPath 式を使用してドキュメント ノードの値を設定して、ドキュメント ノードを検索する。
`reply`	`com.bea.b2b.protocol.cxml.messaging.CXMLMessage`	要求メッセージに応答する。このメソッドは、このオブジェクトが `CXMLListener.onMessage()` へのパラメータとして使用される場合にのみ有効。応答は、このメソッドの呼び出し後に非同期に送信できる。
`getReplyDocument`	`com.bea.b2b.protocol.cxml.messaging.CXMLMessage`	応答 cXML ドキュメントを取得する。
`getRequestDocument`	`com.bea.b2b.protocol.cxml.messaging.CXMLMessage`	要求 cXML ドキュメントを取得する。
`send`	`com.bea.b2b.protocol.cxml.messaging.CXMLMessage`	要求メッセージを送信する。このメソッドは、応答が受信されるまでブロックされる。応答には、`getReplyDocument()` を介してアクセスできる。
`setCollaborationAgreement`	`com.bea.b2b.protocol.cxml.messaging.CXMLMessage`	このメッセージが属するコラボレーション アグリーメントのコラボレーション アグリーメント ID を設定する。プロパティ セットを使用してコラボレーション アグリーメントを選択する。
`setReplyDocument`	`com.bea.b2b.protocol.cxml.messaging.CXMLMessage`	応答 cXML ドキュメントを設定する。
`setRequestDocument`	`com.bea.b2b.protocol.cxml.messaging.CXMLMessage`	要求 cXML ドキュメントを設定する。

表 3-1 パブリック cXML メソッド

メソッド	パッケージ	説明
getHttpStatusCode	com.bea.b2b.protocol.cxml.messaging.CXMLMessageToken	HTTP ステータス コードを取得する。

個々のメソッドの詳細については、『*BEA WebLogic Integration Javadoc*』を参照してください。

コラボレーション アグリーメントを検索するためのプロパティ

cXML は、定義済みのプロパティ セットを使用して一意なコラボレーション アグリーメントを特定します。特定のコラボレーション アグリーメントを検索する場合は、以下のすべてのプロパティの値を指定する必要があります。

- BusinessProcessName
- BusinessProcessVersion
- DeliveryChannel
- toRole
- fromTradingPartner
- toTradingPartner

cXML メッセージの構造

cXML メッセージは、メッセージ エンベロープをベースとしています。メッセージ エンベロープには、以下のデータ構造が含まれています。

図 3-1 cXML メッセージ アーキテクチャ

メッセージのデータ構造は次のとおりです。

- ヘッダ ー アドレス指定および検証情報（From、To、および Sender データ配列）が含まれます。これらのデータ配列は、トランザクションのさまざまなパーティに関する情報、および各参加者の検証情報を提供します。
- ペイロード ー メッセージ本文を構成するすべてのビジネス ドキュメントと添付ファイルです。

アプリケーションは、ペイロードに含まれるすべてのメッセージング オブジェクトを処理する必要があります。

- ビジネス ドキュメント ー 1 つまたは複数の cXML ドキュメントです。各ドキュメントには cXML データが含まれ、その各パートは cXML DTD を使用して検証できます。
- cXML ドキュメント ー データを含む cXML 標準ドキュメントです。cXML ドキュメントは、cXML DTD を使用して個々に検証できます。
- 添付ファイル ー オプションの MIME エンコードのバイナリ フォーマットです。WebLogic Integration は cXML 添付ファイルをサポートしていません。これらを利用する場合は、プライベート プロセスをコンフィグレーションして MIME エンコードの添付ファイルを解決します。

cXML DTD

必要な DTD は、以下の場所で入手できます。

- cXML DTD は、次の URL の cXML.org Web サイトから入手できます。

  `http://xml.cxml.org/schemas/cXML/`*`version`*`/cXML.dtd`

  ここで、*`version`* は完全な cXML バージョン番号（1.1、1.2、など）です。

- 確認および出荷通知トランザクションは、次の URL にある別の DTD に含まれています。

  `http://xml.cxml.org/schemas/cXML/`*`version`*`/Fulfill.dtd`

  ここで、*`version`* は完全な cXML バージョン番号（1.1、1.2、など）です。

cXML メッセージを送信するときには、これらの DTD を使用した検証は必須ではありません。しかし、cXML メッセージング構造の前提の 1 つは、送信するすべてのメッセージが検証済みであるということです。このため、少なくとも新しいトレーディング パートナとの相互運用性をテストするときには、メッセージを DTD と照らし合わせて検証するようにしてください。トレーディング パートナとの相互運用性に問題がなければ、オプションでメッセージ検証を解除してパフォーマンスを向上させても構いません。

共有秘密の処理

cXML API は、リポジトリに格納される共有秘密の値へのアクセスを提供します。`GetSharedSecret` メソッドを使用すると、リポジトリから共有秘密を取得して、受信ドキュメントに格納されている共有秘密と比較したり、送信 cXML ドキュメントで使用したりできます。

受信ドキュメントの場合、ビジネス処理コードは、受信メッセージの共有秘密の検証を実行する必要があります。そのためには、その共有秘密の値と、リポジトリ内のコンフィグレーションに指定されている値を比較します。

送信ドキュメントの場合、ビジネス処理コードは、各送信 cXML ドキュメントの Credential ノードに共有秘密を挿入する必要があります。

受信メッセージの処理

受信メッセージを処理するには、まずそのメッセージを初期化する必要があります。登録機能は、コラボレーション アグリーメントをリスナまたは送信アプリケーションに関連付けます。初期化プロセスによって返されるトークンは、メッセージの送受信時に cXML メッセージで使用されます。

初期化

受信メッセージを初期化するには、次の手順を行います。

1. リスナ オブジェクトのコピーを取得します。
2. リターン トークンを定義します。
   ```
   private static CXMLToken token;
   ```
3. cXML Manager クラス `com.bea.b2b.protocol.cxml.CXMLManager` のインスタンスを検索します。
   ```
   private static CXMLManager cxmlm = CXMLManager.getInstance();
   ```
4. プロパティ セットを定義して、このアプリケーションとリスナを登録します。これらのプロパティは、一意のコラボレーション アグリーメントの検索およびマップに使用されます。一意のコラボレーション アグリーメントのマップに必要なプロパティについては、3-5 ページの「コラボレーション アグリーメントを検索するためのプロパティ」を参照してください。
   ```
   prop.setProperty("BusinessProcess", businessProcess);
   prop.setProperty("BusinessProcessVersion",
   businessProcessVersion);
   prop.setProperty("DeliveryChannel", deliveryChannel);
   prop.setProperty("thisTradingPartner", myTradingPartnerName);
   prop.setProperty("otherTradingPartner",
   otherTradingPartnerName);
   prop.setProperty("toRole", toRole);
   prop.setProperty("Party", "duns4");
   ```
5. `CXMLManager` クラスから登録メソッドを呼び出します。
   ```
   token = cxmlm.register(prop);
   ```

メッセージの処理

前節の「初期化」で説明したとおり受信メッセージを初期化したら、そのメッセージを処理できます。そのためには、アプリケーションは以下のことを行う必要があります。

1. `onMessage()` コールバック メソッドを使用して、受信した cXML メッセージから要求 cXML ドキュメントを取得します。このメソッドは、受信した cXML メッセージを WebLogic Integration ランタイムからアプリケーション コードに渡します。
2. cXML ドキュメントから XML DOM ドキュメントを取得します。

   ```
   // cXML ドキュメントを取得
     CXMLDocument reqMsgDoc = cmsg.getRequestDocument();

     // XML DOM ドキュメントを取得
     Document reqXMLDoc = reqMsgDoc.getDocument();
   ```

3. ペイロードに基づいて要求ドキュメントを処理します。
4. 受信メッセージから共有秘密を取得します。トレーディング パートナの共有秘密は、メッセージの `//cXML/Header/From/Credential` に定義されています。

   ```
   String otherSharedSecret =
   cxmlm.getSharedSecret(otherTradingPartnerName);
   ```

5. メッセージの共有秘密が、トレーディング パートナのコンフィグレーションに定義されている共有秘密に一致するかどうかを検証します。トランザクションがピア ツー ピアの場合、トレーディング パートナはバイヤまたはサプライヤです。トランザクションがハブを介して発生する場合、トレーディング パートナはハブです。

   ```
   debug("Stored Shared Secret for " + otherTradingPartnerName + ":
   " + otherSharedSecret);
   ```

   以下の比較失敗オプションが発生する場合があります。

表 3-2 比較失敗オプション

結果	理由
比較が実行されなかった	トレーディング パートナの共有秘密がコンフィグレーションされていない。
メッセージが拒否され、http ステータス コード *400*（不正な要求）が表示された	要求メッセージを解析できなかった。この問題は、WebLogic Integration ランタイムで解決される。
メッセージが拒否され、http ステータス コード *401*（不正なアクセス）が表示された	共有秘密が一致しない。
メッセージが拒否され、http ステータス コード *500*（要求の転送不能）が表示された	リスナが適切にコンフィグレーションされていなかった。この問題は、WebLogic Integration ランタイムで解決される。

6. 応答 XML DOM 実装ドキュメントを作成します。

```
DOMImplementationImpl domi = new DOMImplementationImpl();

DocumentType dType =
domi.createDocumentType("request", null, "cXML.dtd");

org.w3c.dom.Document punchoutDoc = new DocumentImpl(dType);
CxmlElementFactory cf = new CxmlElementFactory(punchoutDoc);
```

7. 応答 cXML ドキュメントを作成します。

```
Element request = punchoutDoc.createElement("Request");
```

8. ドキュメントにヘッダ要素を作成します。

```
// ヘッダ
cf.createHeaderElement(
// 送信元
cf.createFromElement(
cf.createCredentialElement(
"DUNS",
myTradingPartnerName,
null)),
```

```
// 送信先
cf.createToElement(
cf.createCredentialElement(
"DUNS",
otherTradingPartnerName,
null)),
// 送信者
cf.createSenderElement(
cf.createCredentialElement(
"AribaNetworkUserId",
"admin@acme.com",
otherSharedSecret),
"Ariba ORMS 5.1P4")),
```

9. cXML ドキュメントに XML ドキュメントを設定します。

```
CXMLDocument replyMsgDoc = new CXMLDocument();
replyMsgDoc.setDocument(replyXMLDoc);
```

10. 応答 cXML メッセージに cXML ドキュメントを設定します。

```
cmsg.setReplyDocument(replyMsgDoc);
```

11. cXML メッセージにコラボレーション アグリーメントを設定します。

```
cmsg.setCollaborationAgreement(prop);
```

12. 応答メッセージを送信し、送信 cXML メッセージをアプリケーションから WebLogic Integration ランタイムにディスパッチします。

```
cmsg.reply();
```

送信メッセージの処理

送信メッセージを送信するには、そのメッセージを初期化しておく必要があります。そのための手順は次のとおりです。

1. リターン トークンを定義します。

```
private static CXMLToken token;
```

2. cXML Manager クラスのインスタンスを取得します。
`com.bea.b2b.protocol.cxml.CXMLManager.`

```
private static CXMLManager cxmlm = CXMLManager.getInstance();
```

3. プロパティ セットを定義して、このアプリケーションを登録します。これらのプロパティは、一意のコラボレーション アグリーメントの検索およびマップに使用されます。一意のコラボレーション アグリーメントのマップに必要なプロパティについては、3-5 ページの「コラボレーション アグリーメントを検索するためのプロパティ」を参照してください。

```
prop.setProperty("BusinessProcess", businessProcess);
prop.setProperty("BusinessProcessVersion",
businessProcessVersion);
prop.setProperty("DeliveryChannel", deliveryChannel);
prop.setProperty("thisTradingPartner", myTradingPartnerName);
prop.setProperty("otherTradingPartner",
otherTradingPartnerName);
prop.setProperty("toRole", toRole);
prop.setProperty("Party", "duns4");
```

4. register メソッドを呼び出します。

```
token = cxmlm.register(prop);
```

メッセージの送信

メッセージを送信するには、アプリケーションは以下のアクションを実行する必要があります。

1. cXML メッセージを作成します。

```
DOMImplementationImpl domi = new DOMImplementationImpl();

DocumentType dType =
domi.createDocumentType("request", null, "cXML.dtd");

org.w3c.dom.Document punchoutDoc = new DocumentImpl(dType);
CxmlElementFactory cf = new CxmlElementFactory(punchoutDoc);
```

2. XML DOM 要求ドキュメントを作成します。

```
Element request = punchoutDoc.createElement("Request");
Element trans =
punchoutDoc.createElement(&dlq;PunchoutSetupRequest&drq;);
request.appendchild(trans);
```

3. 要求ドキュメントにヘッダ要素を作成します。

```
punchoutDoc.appendChild(
cf.createCxmlElement(
// ヘッダ
```

```
cf.createHeaderElement(
// 送信元
cf.createFromElement(
cf.createCredentialElement(
"DUNS",
myTradingPartnerName,
null)),
// 送信先
cf.createToElement(
cf.createCredentialElement(
"DUNS",
otherTradingPartnerName,
null)),
// 送信者
cf.createSenderElement(
cf.createCredentialElement(
"AribaNetworkUserId",
"admin@acme.com",
otherSharedSecret),
"Ariba ORMS 5.1P4")),
```

4. 該当するトレーディング パートナ プロファイルから、受信側トレーディング パートナの共有秘密を取得します。ピア ツー ピア メッセージの場合、これは実際の受信側トレーディング パートナの共有秘密です。ハブを介してルーティングされるメッセージの場合、これはハブの共有秘密です。

   受信側トレーディング パートナの共有秘密の値（`//cXML/Header/To/Credential` に定義される）は、送信者の共有秘密要素（`//cXML/Header/Sender/Credential` に定義される）に更新されます。

   ```
   String otherSharedSecret =
   cxmlm.getSharedSecret(otherTradingPartnerName);
   debug("Stored Shared Secret for " + otherTradingPartnerName + ":
   " + otherSharedSecret);
   ```

5. cXML ドキュメントを作成します。

   ```
   CXMLDocument reqMsgDoc = new CXMLDocument();
   ```

6. cXML メッセージに cXML ドキュメントを設定します。

   ```
   reqMsgDoc.setDocument(reqXMLDoc);
   cmsg.setRequestDocument(reqMsgDoc);
   ```

7. cXML メッセージにコラボレーション アグリーメントを設定します。

   ```
   cmsg.setCollaborationAgreement(prop);
   ```

8. メッセージを送信します。

```
CXMLMessageToken sendToken = (CXMLMessageToken) cmsg.send();
```

9. 応答ドキュメントを取得します。

```
CXMLDocument replyMsgDoc = cmsg.getReplyDocument();
```

10. XML ドキュメントを抽出します。

```
org.w3c.dom.Document replyXMLDoc = replyMsgDoc.getDocument();
```

11. 応答を検証します。

コード サンプル

この節では、バイヤとサプライヤがメッセージを処理するために使用するコードの例を示します。これらの例は cXML クラスの処理を示すためのものに過ぎず、実行を意図したものではありません。以下に示す例は、ピア ツー ピア処理用にコンフィグレーションされています。

cXML クラスの詳細については、『*BEA WebLogic Integration Javadoc*』を参照してください。

サンプル バイヤ

コード リスト 3-1 サンプル バイヤのコード例

```
/*
 *        Copyright (c) 2001 BEA
 *        All rights reserved
*/
package examples.ibcxmlverifier;

import java.io.*;
import java.util.*;
import javax.servlet.*;
import javax.servlet.http.*;

import org.w3c.dom.*;
import org.apache.html.dom.*;
import org.apache.xml.serialize.*;
import org.apache.xerces.dom.*;

import com.bea.b2b.protocol.cxml.messaging.*;
```

```
import com.bea.b2b.protocol.cxml.*;

import com.bea.eci.logging.*;

/**
 * This example provides a simple test that will verify message flow of cXML
 * peer-to-peer sending and receiving a cXML document.
 * The two peers (Partner1 and Partner2) are running on a single WLS.
 * Partner1 sends a PunchoutRequest to Partner2.  Partner2 generates a
 * PunchoutSetupResponse and returns it to Partner1.  Shared Secrets are verified
 * at both ends.
 */
public class Partner1Servlet extends HttpServlet
{
  static final boolean DEBUG = true;

  private final static String businessProcess = "PunchoutSetup";
  private final static String businessProcessVersion = "1.1.009";
  private final static String deliveryChannel = "CXMLPartnerVerifier1";
  private final static String myTradingPartnerName = "CXMLPartnerVerifier1";
  private final static String otherTradingPartnerName = "CXMLPartnerVerifier2";
  private final static String toRole = "Supplier";
  private final static String expectedURL = "http://xyz/abc?from=" +
myTradingPartnerName;
  private DocSerializer ds;

  // このアプリケーションのトークンを作成
  private static CXMLToken token;

  // マネージャ インスタンスを取得
  private static CXMLManager cxmlm = CXMLManager.getInstance();

  private static Properties prop = new Properties();

  public void init(ServletConfig sc) {
    try {
      debug("Initializing servlet for Partner1");


      // コラボレーション アグリーメントを検索するためのプロパティを設定
      prop.setProperty("BusinessProcess", businessProcess);
      prop.setProperty("BusinessProcessVersion", businessProcessVersion);
      prop.setProperty("DeliveryChannel", deliveryChannel);
      prop.setProperty("thisTradingPartner", myTradingPartnerName);
      prop.setProperty("otherTradingPartner", otherTradingPartnerName);
      prop.setProperty("toRole", toRole);
      prop.setProperty("Party", "duns4");
```

```
      // プロパティを使用してバイヤをマネージャに登録
      token = cxmlm.register(prop);

    } catch (Exception e) {
      debug("CXMLPartnerVerifier1 init exception: " + e);
      e.printStackTrace();
    }
  }

  private org.w3c.dom.Document getBusinessDocument() {
    DOMImplementationImpl domi = new DOMImplementationImpl();

    DocumentType dType =
      domi.createDocumentType("request", null, "cXML.dtd");

    org.w3c.dom.Document punchoutDoc = new DocumentImpl(dType);
    CxmlElementFactory cf = new CxmlElementFactory(punchoutDoc);

    try {
      String otherSharedSecret = cxmlm.getSharedSecret(otherTradingPartnerName);
      debug("Stored Shared Secret for " + otherTradingPartnerName + ": " +
otherSharedSecret);

      // ヘッダ
      Element request = punchoutDoc.createElement("Request");
      Element trans = punchoutDoc.createElement("PunchoutSetupRequest");
      request.appendChild(trans);

      punchoutDoc.appendChild(
        cf.createCxmlElement(
      // ペイロード
      "1233444-200@ariba.acme.com",

      // ヘッダ
      cf.createHeaderElement(
      // 送信元
      cf.createFromElement(
      cf.createCredentialElement(
        "DUNS",

        myTradingPartnerName,
        null)),
      // 送信先
      cf.createToElement(
      cf.createCredentialElement(
        "DUNS",
        otherTradingPartnerName,
        null)),
```

```
    // 送信者
    cf.createSenderElement(
    cf.createCredentialElement(
      "AribaNetworkUserId",
      "admin@acme.com",
      otherSharedSecret),
      "Ariba ORMS 5.1P4")),
    // 要求
    request));

  }
  catch( Exception e )  {
    debug("MessageDeliveryException: " + e.toString());
    e.printStackTrace();
  }
  return punchoutDoc;
}

/**
 * The actual work is done in this routine.  Construct a message document,
 * publish the message, wait for a reply, terminate and report back.
 */
public void service(HttpServletRequest req, HttpServletResponse res)
  throws ServletException, IOException
{
  try {

    // クライアントへの応答表示用のセットアップ
    res.setContentType("text/html");
    PrintWriter pw = res.getWriter();
    pw.println("<HTML><BODY BGCOLOR=#ff0000>");
    pw.println("<P><IMG SRC=logo.jpg WIDTH=185 HEIGHT=156"+
      " ALIGN=TOP BORDER=0 NATURALSIZEFLAG=3></P>");
    pw.println("<P><FONT SIZE=-1>Partner1 process flow:<BR>");
    pw.println("Starting Partner1...");

    debug("Starting Partner1: get Document...");

    CXMLMessage cmsg = new CXMLMessage();

    org.w3c.dom.Document reqXMLDoc = getBusinessDocument();


    CXMLDocument reqMsgDoc = new CXMLDocument();
    reqMsgDoc.setDocument(reqXMLDoc);
    cmsg.setRequestDocument(reqMsgDoc);

    DocSerializer ds = new DocSerializer();
```

```
    debug("buyer: request document:\n" +
      ds.docToString(reqXMLDoc, true) + "\n");

    // プロパティで CA を設定
    cmsg.setCollaborationAgreement(prop);

    // メッセージを送信して応答を取得
    CXMLMessageToken sendToken = (CXMLMessageToken) cmsg.send();
    CXMLDocument replyMsgDoc = cmsg.getReplyDocument();

    debug("Got document");
    if (replyMsgDoc == null) {
      debug("replyMsgDoc bad");
    }
    org.w3c.dom.Document replyXMLDoc = replyMsgDoc.getDocument();

    debug("buyer: reply document:\n" +
      ds.docToString(replyXMLDoc, true) + "\n");

    // 適切な応答の取得を検証
    String punchoutURL = replyMsgDoc.getNodeValue(
      "//cXML/Response/PunchoutSetupResponse/StartPage/URL");
    if (punchoutURL.equals(expectedURL)) {
      debug("Correct response received");
      pw.println("<P>Correct response received");
    }
    else  {
      debug("Unexpected response received");
      pw.println("<P>Unexpected response received");
    }

    // 共有秘密が自分のものであることを検証
    String dss = replyMsgDoc.getSenderSharedSecret();
    debug("Document Shared Secret for " + myTradingPartnerName + ": " + dss);

    String sss = cxmlm.getSharedSecret(myTradingPartnerName);
    debug("Stored Shared Secret for " + myTradingPartnerName + ": " + sss);

    if (dss.equals(sss)) {
      debug("Shared Secret match");
      pw.println("<P>Shared Secret match");
    } else {

      debug("Shared Secret mismatch");
      pw.println("<P>Shared Secret mismatch");
    }
  }
```

```
    catch( Exception e )  {
      debug("MessageDeliveryException: " + e.toString());
      e.printStackTrace();
    }
  }

  /**
   * A simple routine that writes to the wlc log
   */
  private static void debug(String msg){
    if (DEBUG)
      UserLog.log("***Partner1Servlet: " + msg);
  }
}
```

サンプル サプライヤ

コード リスト 3-2　サンプル サプライヤのコード例

```
/*
 *        Copyright (c) 20001 BEA
 *        All rights reserved
*/
package examples.ibcxmlverifier;

import java.io.*;
import java.util.*;

import javax.servlet.*;
import javax.servlet.http.*;

import org.w3c.dom.*;
import org.apache.html.dom.*;
import org.apache.xml.serialize.*;
import org.apache.xerces.dom.*;

import com.bea.b2b.protocol.messaging.*;
import com.bea.b2b.protocol.cxml.messaging.*;
import com.bea.b2b.protocol.cxml.CXMLListener;
import com.bea.b2b.protocol.cxml.*;

import com.bea.eci.logging.*;

/**
```

```
 * This example provides a simple test that will verify message flow of cXML
 * peer-to-peer sending and receiving a cXML document.
 * The two peers (Partner1 and Partner2) are running on a single WLS.
 * Partner1 sends a PunchoutRequest to Partner2.  Partner2 generates a
 * PunchoutSetupResponse and returns it to Partner1.  Shared Secrets are verified
 * at both ends.
 */
public class Partner2Servlet extends HttpServlet {

  static final boolean DEBUG = true;

  private final static String businessProcess = "PunchoutSetup";
  private final static String businessProcessVersion = "1.1.009";
  private final static String deliveryChannel = "CXMLPartnerVerifier2";
  private final static String myTradingPartnerName = "CXMLPartnerVerifier2";
  private final static String otherTradingPartnerName = "CXMLPartnerVerifier1";
  private final static String toRole = "Buyer";

  // このアプリケーションのトークンを作成
  private static CXMLToken token;

  // マネージャ インスタンスを取得
  private static CXMLManager cxmlm = CXMLManager.getInstance();

  private static Properties prop = new Properties();

  public void init(ServletConfig sc) {
    try {
      debug("Initializing servlet for Partner2");

      // コラボレーション アグリーメントを検索するためのプロパティを設定
      prop.setProperty("BusinessProcess", businessProcess);
      prop.setProperty("BusinessProcessVersion", businessProcessVersion);
      prop.setProperty("DeliveryChannel", deliveryChannel);
      prop.setProperty("thisTradingPartner", myTradingPartnerName);
      prop.setProperty("otherTradingPartner", otherTradingPartnerName);
      prop.setProperty("toRole", toRole);
      prop.setProperty("Party", "duns5");

      // プロパティを使用してサプライヤ リスナをマネージャに登録
      token = cxmlm.register(new Partner2MessageListener(), prop);

      debug("Partner2 waiting for message...");
    } catch (Exception e) {
      debug("CXMLPartnerVerifier2 init exception: " + e);
      e.printStackTrace();
    }
  }
```

```
/**
 * This routine starts the peer
 */
public void service(HttpServletRequest req, HttpServletResponse res)
  throws ServletException, IOException{
  debug("Starting Partner2");
}

/**
 * A simple routine that writes to the wls log
 */
private static void debug(String msg){
  if (DEBUG)
    UserLog.log("***Partner2Servlet: " + msg);
}

public class Partner2MessageListener
  implements CXMLListener

{
  public void onMessage(CXMLMessage cmsg) {
      XPathHelper xp = new XPathHelper();

    try {
      debug("Partner2 received message");
      // QualityOfService qos = cmsg.getQoS();

      CXMLDocument reqMsgDoc = cmsg.getRequestDocument();
      if (reqMsgDoc == null){
        throw new Exception("Did not get a request payload");
      }
      Document reqXMLDoc = reqMsgDoc.getDocument();
      if (reqXMLDoc == null){
        throw new Exception("Did not get a request document");
      }
      String from = reqMsgDoc.getNodeValue(
        "//cXML/Header/From/Credential/Identity" );
      if (from == null) {
    from = "nobody";
      }
      debug("Received request from " + from );

      DocSerializer ds = new DocSerializer();

      debug("supplier: request document:\n" +
        ds.docToString(reqXMLDoc, true) + "\n");

      debug("Building reply document");
```

```
        DOMImplementationImpl domi = new DOMImplementationImpl();
        DocumentType dType =
      domi.createDocumentType("response", null, "cXML.dtd");

        org.w3c.dom.Document replyXMLDoc = new DocumentImpl(dType);
        CxmlElementFactory cf = new CxmlElementFactory( replyXMLDoc );

       String otherSharedSecret = cxmlm.getSharedSecret(otherTradingPartnerName);
        debug("Stored Shared Secret for " + otherTradingPartnerName  + ": " +
otherSharedSecret);

        replyXMLDoc.appendChild(
          cf.createCxmlElement(
        // ペイロード
          "1233444-200@ariba.acme.com",

        // ヘッダ
      cf.createHeaderElement(
        // 送信元

      cf.createFromElement(
      cf.createCredentialElement(
        "DUNS",
        myTradingPartnerName,
        null)),
        // 送信先
      cf.createToElement(
      cf.createCredentialElement(
        "DUNS",
        otherTradingPartnerName,
        null)),
      // 送信者
      cf.createSenderElement(
      cf.createCredentialElement(
        "AribaNetworkUserId",
        "admin@acme.com",
        otherSharedSecret),
        "Ariba ORMS 5.1P4")),
      // 本文
      cf.createResponseElement(
        "200",
        "ok",
      cf.createPunchoutSetupResponseElement(
        "http://xyz/abc?from=" + from ))));

        CXMLDocument replyMsgDoc = new CXMLDocument();
        replyMsgDoc.setDocument(replyXMLDoc);
```

```
        cmsg.setReplyDocument(replyMsgDoc);

        debug("supplier: reply document:\n" +
        ds.docToString(replyXMLDoc, true) + "\n");

        // 共有秘密が自分のものであることを検証
        String dss = reqMsgDoc.getSenderSharedSecret();
       debug("Document Shared Secret for " + myTradingPartnerName + ": " + dss);

        String sss = cxmlm.getSharedSecret(myTradingPartnerName);
        debug("Stored Shared Secret for " + myTradingPartnerName  + ": " + sss);

        if (dss.equals(sss)) {
          debug("Shared Secret match");
        } else {
          debug("Shared Secret mismatch");
        }

        // プロパティで CA を設定
        cmsg.setCollaborationAgreement(prop);
        cmsg.reply();


        debug("Partner2 sent reply");
      } catch(Exception e) {
        debug("Exception errors" + e);
        e.printStackTrace();
      }
    }

    public void onTerminate(Message msg) throws Exception {
      debug(" received terminate notification for " + msg.getConversationId());

      // マネージャから登録を解除
      cxmlm.deregister(prop);
    }
  }
}
```

4　cXML でのワークフローの使用

注意： cXML ビジネス プロトコルは、WebLogic Integration の本リリースより非推奨になりました。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。

WebLogic Integration では、Business Process Management（BPM）ワークフローを使用して通常のビジネス メッセージを交換できます。WebLogic Integration 用の cXML プラグインは存在しませんが、ビジネス オペレーションを使用することによって cXML ビジネス ドキュメントを統合できます。

以下の節では、ワークフローと cXML API 対応インタフェースを使用して、WebLogic Integration で cXML ビジネス メッセージを交換する方法について説明します。

- ワークフローへの cXML の組み込み
- ビジネス メッセージを交換するためのワークフローの設計
- ビジネス メッセージの使い方

WebLogic Integration を使用したワークフローの開発については、『*B2B Integration ワークフローの作成*』を参照してください。

ワークフローへの cXML の組み込み

cXML を使用するワークフローは、外部で作成したビジネス オペレーション クラスを利用して、WebLogic Integration で使用される cXML API をカプセル化する必要があります。

この開発プロセスの結果が、ラッパー クラスに定義されたメソッドを実行時に呼び出すワークフローです。これらのメソッドは、定義済みの cXML ビジネス オペレーションを実行します。

ワークフロー統合タスク

cXML と BPM ワークフローを使用する場合、特定の管理タスク、設計タスク、およびプログラミング タスクの組み合わせが必要となります。

プログラミング タスク

外部で作成されたビジネス オペレーション クラスは、cXML API を使用して特定のビジネス オペレーションを実行します。たとえば、ワークフロー用に `PunchoutSetupRequest` 機能を実装するクラスを作成します。詳細については、次の URL の「*cXML User's Guide*」を参照してください。

`http://www.cxml.org`

ワークフローを使用してパラメータを渡す場合、これらのパラメータを受け付けることができるクラスを作成する必要があります。それが済んだら、ワークフロー変数を使用してパラメータをクラスに渡すことができます。これらのパラメータは、cXML 出力の設定に使用できます。

クラス、そのメソッド、および定義したパラメータをコンフィグレーションするには、WebLogic Integration Studio を開き、[*コンフィグレーション*] メニューから [*ビジネス オペレーション*] を選択します。詳細については、『*WebLogic Integration Studio ユーザーズ ガイド*』を参照してください。

WebLogic Integration Studio では、cXML プロセス処理を呼び出すために使用するビジネス オペレーションをワークフロー アクションとして呼び出すことができます。アクションを追加する場合、[*アクションを追加*] ダイアログ ボックスの [*統合アクション*] フォルダから [*ビジネス オペレーションを実行*] を選択します。このオプションを選択すると、ワークフロー変数を cXML ラッパー クラスによって使用されるメソッド パラメータにマップできます。詳細については、『*WebLogic Integration Studio ユーザーズ ガイド*』を参照してください。

管理タスク

ワークフローで cXML を使用する前に、以下の管理タスクを完了しておく必要があります。これらのタスクは、WebLogic Integration Studio を使用して WebLogic Integration で使用するためのワークフローを生成するときに通常実行するタスクに追加されるものです。

- WebLogic Integration B2B Console を使用して、WebLogic Integration リポジトリの cXML トランザクションに含まれるエンティティ（トレーディング パートナ、コラボレーション アグリーメントなど）を作成およびコンフィグレーションします。詳細については、『*B2B Integration 管理ガイド*』を参照してください。
- ビジネス オペレーションクラスを作成したら、WebLogic Integration Studio にビジネス オペレーションを作成してビジネス オペレーションクラスを利用します。ビジネス オペレーションの作成については、第 3 章「cXML API の使用」を参照してください。

設計タスク

ワークフローで cXML を使用する場合、WebLogic Integration で使用するためのワークフローを作成するために必要な設計タスクに加え、別の設計作業を行う必要があります。特に、ビジネス オペレーションを使用してすべての cXML 機能を実行するようにワークフローを設計する必要があります。実行する必要がある cXML 機能ごとに、別個のビジネス オペレーションを作成する必要があります。

ビジネス メッセージを交換するためのワークフローの設計

WebLogic Integration でビジネス メッセージを交換するためのワークフローを使用するには、WebLogic Integration Studio を使用してワークフロー テンプレート定義を設計します。ワークフローの作成の詳細については、『*WebLogic Integration Studio ユーザーズ ガイド*』および『*B2B Integration ワークフローの作成*』を参照してください。

前述のとおり、ワークフローで cXML を使用する場合は、ビジネス オペレーションクラスを作成して cXML API を実装する必要があります。前節では、これらのビジネス オペレーションクラスの作成について説明しました。この節では、ビジネス オペレーションクラスを使用して WebLogic Integration の BPM コンポーネントで cXML を操作する方法について説明します。

ビジネス メッセージの使い方

WebLogic Integration Studio を使用すると、トレーディング パートナがビジネス メッセージを交換できるようになります。cXML は、このタスクを実行する 1 つの方法です。

以下の節では、ワークフローを使用して交換される cXML ビジネス メッセージの使い方について説明します。

- cXML ビジネス メッセージについて
- ビジネス メッセージを交換するために必要なタスク

cXML ビジネス メッセージについて

cXML ビジネス メッセージは、会話内でトレーディング パートナ間でやり取りされる通信の基本単位です。cXML ビジネス メッセージは、以下の要素で構成されるマルチパート MIME メッセージです。

- *ビジネス ドキュメント*。ビジネス メッセージ中の XML ベースのペイロード部分です。ペイロードは、ビジネス メッセージのビジネス コンテンツです。
- *添付ファイル*。ビジネス メッセージ中の XML ペイロード以外の部分です。添付ファイルは cXML1.2 標準のオプション エンティティであり、cXML 1.1 実装では使用できません。

『*B2B Integration ワークフローの作成*』に説明されているように、他の形式のビジネス メッセージと同様、そのコンテンツにはプログラム的にアクセスできます。ただし、XOCP および RosettaNet ビジネス メッセージとは異なり、cXML の WebLogic Integration 実装では、cXML を使用するときに他のメソッドを使用してビジネス メッセージのコンテンツにアクセスできません。

ビジネス メッセージを交換するために必要なタスク

ビジネス メッセージを送受信するには、あらかじめ WebLogic Integration Studio を使用してワークフロー テンプレートに以下のアクションを定義しておく必要があります。

- ビジネス メッセージの送信を定義するには、Manipulate Business Message アクションを定義してビジネス メッセージを作成し、Send Business Message アクションを定義してメッセージを送信します。
- ビジネス メッセージの受信を定義するには、Manipulate Business Message アクションを定義して受信ビジネス メッセージを処理します。

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。

索引