BEA WebLogic Integration™

Application Integration ユーザーズ ガイド

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

Application Integration ユーザーズ ガイド

パート番号	日付	ソフトウェアのバージョン
なし	2002 年 6 月	7.0

目次

このマニュアルの内容

1. Application Integration の概要

2. アプリケーション ビューの定義

3. Studio におけるアプリケーション ビューの使用

4. カスタム コードの作成によるアプリケーション ビューの使用方法

5. Application View Console の使用方法

A. Application Integration データの移行

B. アプリケーション ビューのインポートおよびエクスポート

C. Application Integration のモジュラ デプロイメント

索引

このマニュアルの内容

『*Application Integration ユーザーズ ガイド*』の内容は以下のとおりです。

- 「Application Integration の概要」では、BEA WebLogic Integration Framework の概要と、このフレームワークがどのように WebLogic Server 環境と調和して BEA EAI ソリューションに貢献するのかについて説明します。
- 「アプリケーション ビューの定義」では、Application View Console にログインする方法と、アプリケーション ビューを作成およびコンフィグレーションして企業のビジネス プロセスを表現する方法について説明します。
- 「Studio におけるアプリケーション ビューの使用」では、WebLogic Integration Studio を使用してワークフローを設定し、WebLogic Server 環境でアプリケーション ビューを使用する方法について説明します。
- 「カスタム コードの作成によるアプリケーション ビューの使用方法」では、カスタム Java コードを作成して、WebLogic Server 環境でアプリケーション ビューを使用する方法について説明します。
- 「Application View Console の使用方法」では、ネームスペースを使用して、アダプタではなく場所や部門に基づいてアプリケーション ビューを編成する方法について説明します。
- 付録 A 「Application Integration データの移行」では、WebLogic Server ドメイン間で Application Integration データを移行する方法について説明します。
- 付録 B「アプリケーション ビューのインポートおよびエクスポート」では、インポート / エクスポート ユーティリティを使用して、リポジトリから Application Integration ビュー メタデータ オブジェクトをエクスポートし、またそれらのオブジェクトをリポジトリにインポートする方法を説明します。
- 付録 C「Application Integration のモジュラ デプロイメント」では、WebLogic Integration ドメイン外の Application Integration エンタープライズ アプリケーションをデプロイする方法を説明します。

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*ービジネス アナリストは、テクニカル アナリストと協力して企業内のビジネス インタフェース機能の精度向上を図ります。また、企業内で使用するアプリケーション ビューを作成し、使用します。
- *テクニカル アナリスト*ーテクニカル アナリストは、アダプタのコンフィグレーションや、WebLogic Integration のサービス設定を行い、レガシー システムとの情報転送を実行します。また、アダプタを使用してソリューションをコンフィグレーションし、WebLogic Server 環境の評価、マッピング、デプロイ、および保守を担当します。このユーザーズ ガイドは、テクニカル アナリストがシステム全体に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、e-docs Web サイトにある BEA WebLogic Integration マニュアル ホーム ページで入手できます。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

以下の関連情報があります。

- BEA WebLogic Server ドキュメント（`http://edocs.beasys.co.jp/e-docs/index.html`）
- BEA WebLogic Integration ドキュメント（`http://edocs.beasys.co.jp/e-docs/index.html`）
- XML スキーマ仕様（`http://www.w3c.org/TR/xmlschema-formal/`）
- Sun Microsystems, Inc. の Java 関連サイト（`http://www.javasoft.com/`）
- Sun Microsystems, Inc. の J2EE コネクタ アーキテクチャの仕様（`http://java.sun.com/j2ee/connector/`）

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、BEA WebLogic Application 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 BEA WebLogic Application Integration のリリースをお書き添えください。

本バージョンの BEA WebLogic Integration について不明な点がある場合、または BEA WebLogic Application 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
〔**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:
	`chmod u+w *`
	`c:\startServer`
	`.doc`
	`wls.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** ( )`
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

表記法	適用
. . .	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。 *例*: `import com.sap.rfc.exception.*;`
. . .	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 Application Integration の概要

このマニュアルでは、BEA WebLogic Integration Adapter Development Kit (ADK) を使用して構築するアダプタの使い方を説明します。アプリケーション ビューのサービスとイベントを定義し、それらを WebLogic Integration 環境のビジネス プロセスで使用する方法について解説します。

この章では、以下のトピックを取り上げます。

- 始める前に
- アダプタに対するインタフェースの作成
- アプリケーション ビューの定義
- ワークフローにおけるアプリケーション ビューの使用方法
- Web サービスにおけるアプリケーション ビューの使用方法

注意： アダプタおよびアプリケーションは、それぞれすべて異なっていますので、このマニュアルに記載される説明はあくまで一般的な内容になっており、特定のアダプタないしアプリケーション用に書かれたものではありません。ADK で提供されている DBMS アダプタの詳細については、『*アダプタの開発*』の「DBMS アダプタ」を参照してください。

始める前に

アダプタを使用して企業システムの統合を開始する前に、環境をセットアップし、WebLogic Integration でアダプタおよびアプリケーション ビューを使用して統合を行うしくみを理解する必要があります。

この節では、以下の内容を説明します。

- ソフトウェア要件

- 基本概念の理解

ソフトウェア要件

注意： 前提条件の詳細リストについては、『*BEA WebLogic Platform リリース ノート*』を参照してください。

以下の前提条件が満たされていることを確認してください。

- WebLogic Platform がインストールされていること。
- JDK 1.3.1 がインストールされていること。WebLogic Platform をインストールすると、JDK 1.3 開発キットが自動的にインストールされます。ただし、1.3.1 に準拠してさえいれば、独自のバージョンをインストールしてもかまいません。
- WebLogic Platform のインストール時に、BEA WebLogic Integration もインストール コンポーネントとして組み込んでいること。
- アプリケーション ビューを定義するアダプタをデプロイしていること。

今回のリリースでは、WebLogic Integration のアプリケーション統合機能は、完全に独立した 1 つの J2EE EAR ファイルにパッケージ化されています。このパッケージ化により、有効な WebLogic Server ドメインであればどこにでもアプリケーション統合機能をデプロイできます。たとえば、Web サービス開発者および WebLogic Portal 開発者は、アプリケーション ビューを使って EIS アプリケーションと対話操作ができます。詳細については、付録 C「Application Integration のモジュラ デプロイメント」を参照してください。

基本概念の理解

アプリケーション統合の基本概念をよくご存知でない場合は、この「*Application Integration の概要*」に記載されたアプリケーション統合の概要説明をお読みになることをお勧めします。概要を理解してから、どのような場合にあるアプリケーション統合方式を他の方式に優先して採用するか、選択した方式をどのように実装するか、といった実際的な問題を解決する作業を習得します。

アダプタに対するインタフェースの作成

企業内で使用される各アダプタには、アダプタが提供するサービスおよびイベントに対するインタフェースを設定する必要があります。このインタフェースは、アプリケーション ビューを定義する、またはカスタム コードを作成するかのいずれかによって作成します。

アプリケーション ビューは、アダプタのリソースにアクセスする非常に便利な機能を提供します。各アダプタが提供するアプリケーション機能を公開する場合、通常この方法を選択すれば、おそらく十分満足していただけるでしょう。しかし、アダプタ機能に対してアプリケーション ビューで可能な以上の制御機能が必要な場合は、カスタム コードを作成することもできます。

アプリケーション ビューとカスタム コードでどちらのほうが企業にとってより好ましい結果が得られるかは、よく検討の上判断してください。以下の各節では、この 2 つの方法を選択する上での基本的な指針を説明します。詳細については、第 2 章、「アプリケーション ビューの定義」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

ほとんどの EIS (Enterprise Information System: エンタープライズ情報システム) は、アプリケーション ビューを定義することにより簡単に統合できます。一般に、以下の基準の 1 つ以上に該当する場合は、アプリケーション ビューを定義する方法を選択してください。

- 企業内に複数の EIS があるが、それらのシステムすべてに精通した開発者がいない場合。
- WebLogic Integration Studio を使用して、ビジネス プロセスを構築したい場合。
- アダプタのパラメータまたはそのプロセスを更新する必要がある。

カスタム コードを作成する場合

以下の基準の 1 つ以上に該当する場合に限り、アダプタのインタフェースとしてカスタム コードを作成する方法を選択してください。

- 企業内の EIS が 1 つだけで、開発者がコード作成するビジネス プロセスに関連してその EIS に精通している場合。
- WebLogic Integration で提供されている Business Process Management (BPM) 機能を使う必要がない場合。
- 将来的にコード変更の必要がない場合。

アプリケーション ビューの定義

アダプタのアプリケーション ビューは、WebLogic Server と特定の EIS アプリケーションとの間の XML ベースのインタフェースです。アプリケーション ビューは、企業内で使用されるアダプタごとに 1 つずつ定義する必要があります。

この節では、以下の事項について説明します。

- アプリケーション ビュー定義の内容
- アプリケーション ビューの定義方法

アプリケーション ビュー定義の内容

アプリケーション ビューを定義する場合、専用の通信パラメータをコンフィグレーションして、次にサービスおよびイベントを追加します。アプリケーション ビューのサービスおよびイベントによって、アプリケーションの特定の機能が公開されます。また、アプリケーション ビューの通信パラメータによって、アプリケーション ビューから対象の EIS への接続方法が管理されます。

アプリケーション ビュー定義では、以下の項目を指定します。

- アプリケーション ビューに対する固有名
- アプリケーション ビューのユーザに対するセキュリティ特権
- 以下に対するパラメータ：
  - アプリケーション
  - アプリケーションとアプリケーション ビューとの間のネットワーク接続
  - アプリケーション ビューで使用できる接続プールの管理
  - アプリケーション ビューによって行われるロード バランシング

アプリケーション ビューの定義方法

この節では、アダプタのアプリケーション ビューを定義する時に実行する手順を概説します。詳細な手順については、第 2 章「アプリケーション ビューの定義」を参照してください。

アプリケーション ビューを定義する手順は、次のとおりです。

- 手順 1 : アプリケーション ビューの接続パラメータに名前を付けてコンフィグレーションする
- 手順 2 : アプリケーション ビューにサービスおよびイベントを追加する
- 手順 3 : サービスおよびイベントのテスト

手順 1 : アプリケーション ビューの接続パラメータに名前を付けてコンフィグレーションする

アダプタ用のアプリケーション ビューを定義するときは、まず Application View Console にログオンしてから、アプリケーション ビューの保存先フォルダを選択または作成し、次に EIS の接続パラメータをコンフィグレーションします。

アプリケーション ビューの作成およびコンフィグレーションの詳細については、以下のトピックを参照してください。

- 2-5 ページの「手順 1 : [Application View Console] へのログオン」

- 2-7 ページの「手順 2 および 3 : アプリケーション ビューの定義および接続パラメータのコンフィグレーション」

手順 2 : アプリケーション ビューにサービスおよびイベントを追加する

サービスおよびイベントは、ユーザが指定したアプリケーション機能と WebLogic Server クライアントとが対話できるようにすることにより、アプリケーションのビジネス プロセスのサブセットをサポートします。アプリケーション ビューが提供するサービスおよびイベントによって、WebLogic Server と EIS アプリケーションとの間で特定の種類のトランザクションを実行できます。

アプリケーション ビューにサービスおよびイベントを追加する詳細については、以下のトピックを参照してください。

- 2-11 ページの「手順 4A : アプリケーション ビューへのサービスの追加」
- 2-25 ページの「手順 6B : アプリケーション ビュー イベントのテスト」

手順 3 : サービスおよびイベントのテスト

追加したサービスまたはイベントが EIS アプリケーションと正しく対話できるかどうか検査します。

サービスおよびイベントのテストに関する詳細については、以下のトピックを参照してください。

- 2-22 ページの「手順 6A : アプリケーション ビュー サービスのテスト」
- 2-25 ページの「手順 6B : アプリケーション ビュー イベントのテスト」

ワークフローにおけるアプリケーション ビューの使用方法

WebLogic Integration 環境にアプリケーション ビューを定義したら、WebLogic Server にデプロイし、これを使用してビジネス プロセス ワークフローの企業ビジネス プロセスを実装することができます。

アプリケーション ビューは、次のいずれかの方法によりビジネス プロセスで使用できます。

- WebLogic Integration Studio にビジネス プロセス ワークフローを設計する
- カスタム コードを作成する

ビジネス プロセス ワークフロー内でアプリケーション ビューが使用される場合、最終的に企業のビジネス プロセスの電子表現形式がデプロイされます。ワークフローは、ビジネス プロセスを達成するためにアプリケーションが実行するトランザクションを指定します。アプリケーション ビューは、そのトランザクションを実行します。

WebLogic Integration Studio でアプリケーション ビューを使う

企業のビジネス プロセス内でアプリケーション ビューを使用する場合、WebLogic Integration Studio でワークフローを設計するのが最も一般的な方法です。Studio は、ビジネス プロセス ワークフローを設計するためのグラフィカル ユーザ インタフェース（GUI）です。ワークフローには、アプリケーション ビューのサービスおよびイベントを組み込むことができます。

アプリケーション ビューを使用することにより、以下の 4 通りの任意の方法でサービスおよびイベントをサポートできます。

- タスク 1 : アプリケーション ビュー サービスを呼び出すタスク ノードの設定

- タスク 2 : 非同期アプリケーション ビュー サービスからの応答を待機するイベント ノードの設定
- タスク 3 : アプリケーション ビュー イベントによって開始されるワークフローの作成
- タスク 4 : アプリケーション ビュー イベントを待機するイベント ノードの設定

各タスクの詳細については、第 3 章「Studio におけるアプリケーション ビューの使用」を参照してください。

カスタム コードを作成してアプリケーション ビューを使う

Studio でアプリケーション ビューを使ってビジネス プロセスを実装しない場合、カスタム Java コードを作成する必要があります。手順については、第 4 章「カスタム コードの作成によるアプリケーション ビューの使用方法」を参照してください。

ビジネス プロセスの実装方法の選択

WebLogic Integration では、Studio でワークフローを作成するか、カスタム コードを作成するかのいずれかの方法によりビジネス プロセスを実装できます。Studio ワークフローとしてならば、どのようなビジネス プロセスでも実装できます。

一方、カスタム コード作成は、対象となるビジネス プロセスが非常に単純で特定用途に使用する場合に限って選択するようにしてください。このマニュアルでは、カスタム コードによる方法は、あくまで、それが必要な状況で使用される代替的な手段として取り上げています。そのような状況のリストについては、1-9 ページの「カスタム Java コードを作成する方がよい場合」を参照してください。

WebLogic Integration Studio を使用する方がよい場合

以下の基準の 1 つ以上に該当する場合は、ビジネス プロセスの実装に WebLogic Integration Studio を使用してください。

- ビジネス プロセスで、複雑なエラー管理、永続的プロセス、および高度な条件付き分岐が必要な場合。

  たとえば、使用するビジネス プロセスで、多数のイベントを受信する、イベントのサブセットを選択する、複雑な分岐アクションを実行する、多数の複雑なメッセージを生成する、複数の WebLogic Server クライアントにメッセージを送信する、などといったことが必要な場合は、Studio を使用するようにします。

- ビジネス プロセスを定期的に変更する必要がある場合。

  Studio を使用すれば、必要なコンパイル、テスト、デバッグを繰り返す回数が減ります。

- 開発者が貴重で数が少ない場合（多くのオーガニゼーションがこれに該当すると思われる）。

カスタム Java コードを作成する方がよい場合

カスタム コードを作成してビジネス プロセスを実装するのは、以下の基準の 1 つ以上に該当する場合に限られます。

- ビジネス プロセスが単純である、すなわち、複雑なエラー回復処理、永続的プロセス、条件付き分岐、およびプロセス フローの結合を含まないプロセスの場合。

  たとえば、使用するビジネス プロセスが、着信メッセージに対して一定の限られたアクションしか行わず、そのメッセージを少数のクライアント アプリケーションに転送するような場合は、カスタム コードを作成しても問題ないと言えます。

- ビジネス プロセスを頻繁に更新する必要がないと考えられる場合。

  カスタム コードを更新する場合は、常に完全なコンパイル、テスト、およびデバッグ サイクルが必要になり、これには費用がかさむことがあります。

- オーガニゼーション内で、コードによるビジネス プロセスの実装に専従する開発者を割り当てる余裕がある場合。

Web サービスにおけるアプリケーション ビューの使用方法

Web サービス開発者は、AppView Control を使って、BEA WebLogic Workshop ユーザに、EIS アプリケーションと対話操作ができる Web サービスを提供することができます。この対話機能は、Java API により実装されます。Web サービス開発者は、この機能を使う場合にその EIS の専門家である必要はありません。開発者は、アプリケーション ビュー サービスを同期的または非同期的のいずれでも呼び出すことができ、単純な Java オブジェクトを使ってアプリケーション ビュー イベントをサブスクライブできます。AppView Control の使用法の詳細については、以下の場所にある『BEA WebLogic Workshop オンライン マニュアル』の「アプリケーション ビュー コントロール : Web サービスからエンタープライズ アプリケーションにアクセスする」を参照してください。

```
http://edocs.beasys.co.jp/e-docs/workshop/docs70/help/index.html#
guide/controls/appview/conAppViewCtrlAccessAnEnterpriseAppFromAJW
S.html
```

2　アプリケーション ビューの定義

この章では、以下のトピックを取り上げます。

- 始める前に
- アプリケーション ビュー定義の基本手順
- アプリケーション ビュー定義の手順の具体例
- アプリケーション ビューの編集

始める前に

アプリケーション ビューを定義する場合、WebLogic Server と企業内の特定 EIS アプリケーションの間で使用される XML ベースのインタフェースを作成します。アプリケーション ビューを作成したら、ビジネス アナリストはそれを使用して、ビジネス プロセスを作成できます。任意のアダプタに対して、アプリケーション ビューをいくつでも作成できます。また、各アプリケーション ビューには、サービスとイベントをいくつ定義しても構いません。

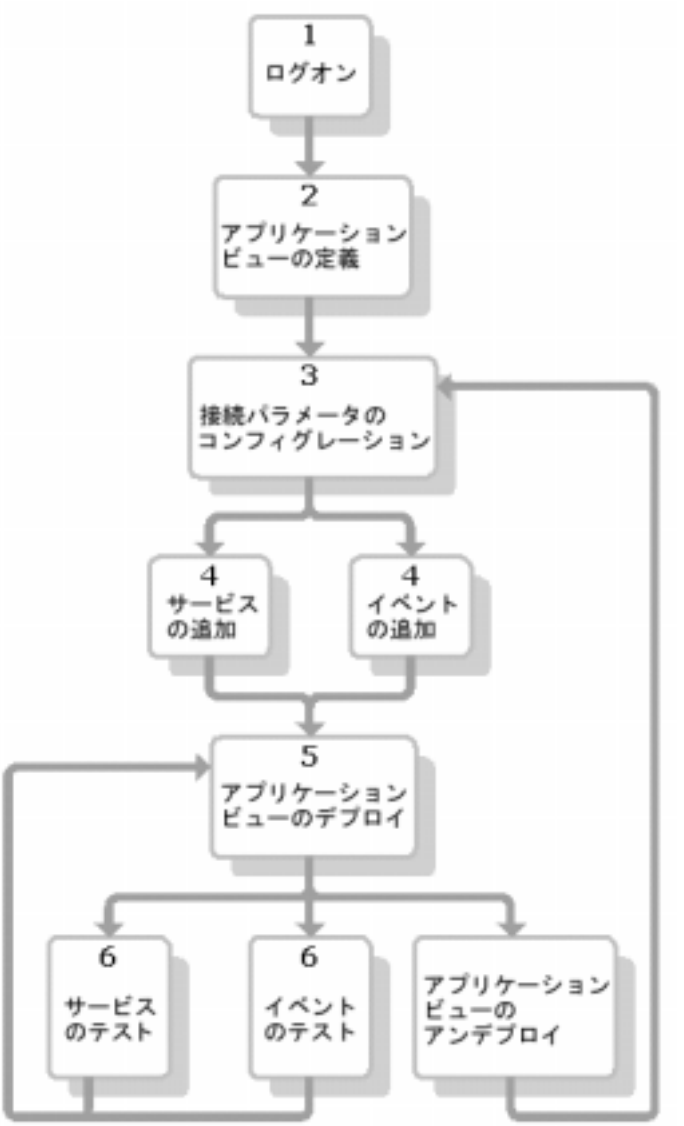
アプリケーション ビューの定義を開始する前に、以下の前提条件が満たされていることを確認してください。

- ADK を使用して、該当するアダプタが開発されていること。アプリケーション ビューの作成およびコンフィグレーションを実行できるのは、既存のアダプタに限られます。
- コンフィグレーション対象のアプリケーション ビューを使用するビジネス プロセスが決定されていること。必要なビジネス プロセスによって、アプリケーション ビューに定義するサービスとイベントのタイプが決まります。したがって、ビジネス アナリストからアプリケーションのビジネス要件に関する情報を収集する必要があります。必要なビジネス プロセスが決まったら、適切なサービスおよびイベントを定義してテストできます。

アプリケーション ビュー定義の基本手順

図 2-1 は、アプリケーション ビューを定義、コンフィグレーションする手順の要約図です。

図 2-1　アプリケーション ビューの定義およびコンフィグレーションの手順



1. WebLogic Integration Application View Console にログオンします。詳細については、2-5 ページの「手順 1 : [Application View Console] へのログオン」を参照してください。
2. [Add Application View] をクリックします。該当するアダプタ用に新しいアプリケーション ビューを作成します。アプリケーション ビューを使用して、指定されたアダプタの対象 EIS アプリケーション用に、一連のビジネス プロセスを作成できます。詳細については、2-7 ページの「手順 2 および 3 : アプリケーション ビューの定義および接続パラメータのコンフィグレーション」を参照してください。
3. [Configure Connection Parameters] 画面で、アプリケーションの接続パラメータを入力します。パラメータを入力する代わりに、[Select Existing Connection] 画面で既にデプロイ済みの接続を選択することもできます。詳細については、2-7 ページの「手順 2 および 3 : アプリケーション ビューの定義および接続パラメータのコンフィグレーション」を参照してください。

   入力情報の有効性が検証されると、アプリケーション ビューが指定したシステムと接続するようにコンフィグレーションされます。
4. [Add Event] または [Add Service] をクリックして、このアプリケーション ビューに適切なイベントとサービスを定義します。
5. アプリケーション ビューを WebLogic Server 上にデプロイします。デプロイして、その他のエンティティがセキュリティ設定に従ってこのビューと対話できるようにします。

   注意 : アプリケーション ビューは、デプロイされていなければテストできません。
6. すべてのサービスおよびイベントをテストします。対象の EIS アプリケーションと適切に対話できることを確認します。

   サービスおよびイベントをテストして正常に機能することが分かったら、アプリケーション ビューをワークフロー内で使用できます。詳細については、第 3 章「Studio におけるアプリケーション ビューの使用」を参照してください。
7. 接続パラメータの再コンフィグレーションや、サービスおよびイベントの追加が必要な場合は、アプリケーション ビューをアンデプロイします。

   注意 : アプリケーション ビューをアンデプロイすると、他のエンティティはアンデプロイされたビューとの対話ができなくなります。

アプリケーション ビュー定義の手順の具体例

この節では、*DBMS* という名前の仮想データベース EIS に対する EIS アダプタを使って、アプリケーション ビューを定義、保守する方法を説明します。ここで示す各手順は、図 2-1 のそれぞれの手順に対応しています。

実際に企業用にアプリケーション ビューを作成する場合、このマニュアルで示されたビューとは見かけ上かなり異なることもあると思われます。アプリケーション ビューのアダプタによって、各アプリケーション ビューの画面に必要な情報が決まり、また各企業には独自の専用アダプタがあるため、このような違いが生じるのは当然のことと言えます。企業内で使用するアダプタの詳細については、担当のテクニカル アナリストまたは EIS のスペシャリストに問い合わせてください。

注意： 以下の手順を実行する前に、WebLogic Server がシステムで稼動していることを確認してください。

手順 1 : [Application View Console] へのログオン

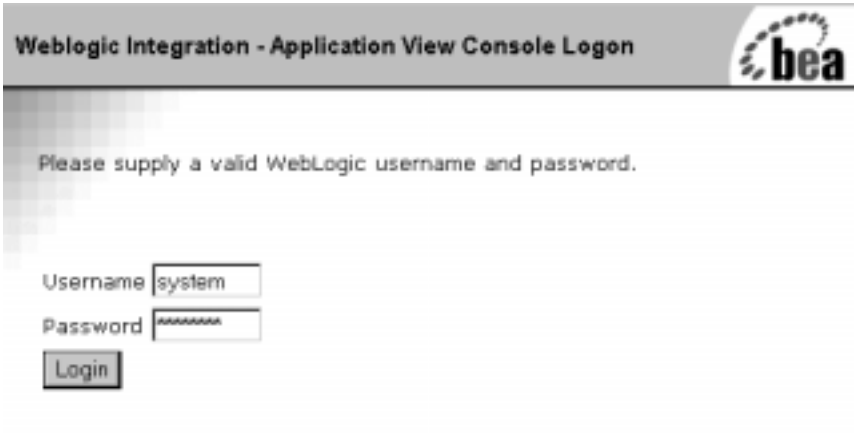
[Application View Console] には、WebLogic Integration 環境内のすべてのアプリケーション ビューがフォルダ編成で表示されます。

Application View Console にログオンする手順は、次のとおりです。

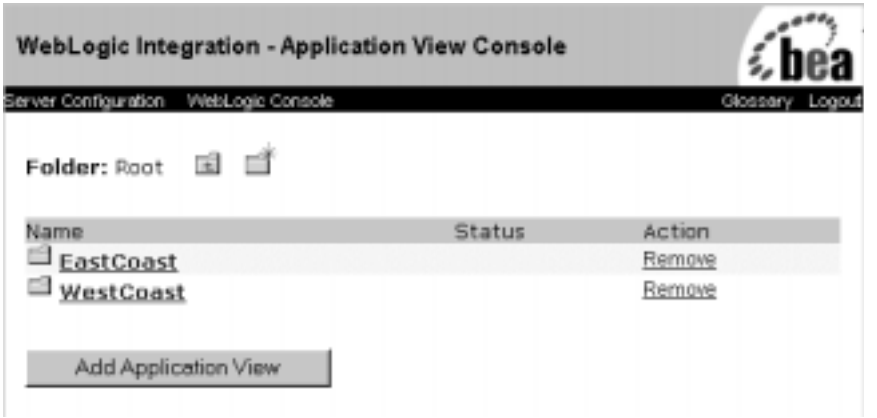
1. 新しいブラウザ ウィンドウを開きます。
2. 該当するシステムの Application View Console の URL を入力します。実際に入力する URL は、システムによって異なります。入力フォーマットは次のとおりです。

   `http://host:port/wlai`

   [Application View Console Logon] 画面が表示されます。



3. WebLogic Server ユーザ名およびパスワードを入力し、[Login] をクリックします。[Application View Console] が表示されます。



注意： この画面が表示されない場合は、WebLogic Server の管理者に連絡してください。

4. フォルダを追加するには、[New Folder] アイコンをクリックします。

詳細については、5-3 ページの「フォルダの作成」を参照してください。

手順 2 および 3 : アプリケーション ビューの定義および接続パラメータのコンフィグレーション

1. 現在のフォルダに新しいアプリケーション ビューを追加するには、[Add Application View] をクリックします。

   注意 : この手順を実行する前に、その時点で作業しているフォルダが、該当するフォルダになっていることを確認してください。アプリケーション ビューを定義した後で、そのビューを別のフォルダに移動することはできません。

   [Define New Application View] 画面が表示されます。

2. [Application View Name] フィールドに、アプリケーション ビュー名を入力します。入力する名前は、このアプリケーションが実行する機能を表すものにしてください。また、アダプタに対してユニークなアプリケーション ビュー名にする必要があります。名前には、任意の有効な Java 識別子が使用できます。

   注意：「Root」という名前は予約語であるため、アプリケーション ビュー名には使用できません。ビュー名に「Root」を指定すると、インポート / エクスポート ユーティリティを使ってアプリケーション ビューをインポートまたはエクスポートできなくなります。

3. [Description] フィールドに、WebLogic Integration Studio に作成したワークフローでこのアプリケーション ビューを使う人にわかりやすいコメントを入力します。

4. [Associated Adapter] リストから、このアプリケーション ビューの作成に使用するアダプタを選択します。

5. [OK] をクリックします。[Select Existing Connection] 画面が表示されます。

[Select Existing Connection] では、アプリケーション ビューに関連付ける接続ファクトリの種類を選択できます。

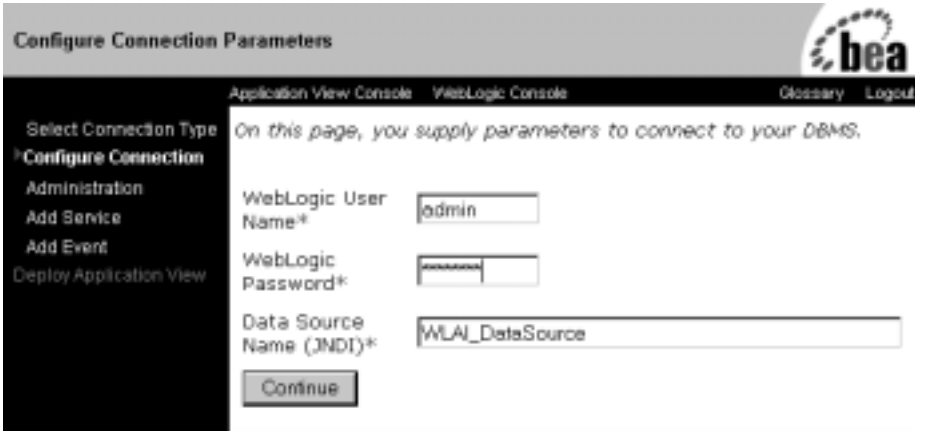
- 新しい接続ファクトリを作成する場合は、[New Connection] オプションを選択します。新しい接続ファクトリの作成後は、他のアプリケーション ビューで同名の新しい接続を作成することはできません。

- 他のアプリケーション ビューと接続ファクトリを共有する場合は、既存の接続ファクトリを使用するオプションを選択します。既存の接続ファクトリに関連付けてデプロイされているアプリケーション ビューの名前を表示するには、既存の接続ファクトリ名の横にある [References] リンクをクリックします。

[Connection Factory] 選択画面からは、いつでも [Select Connection] または [Connection Configuration] 画面を表示できます。新しい接続ファクトリと既存のファクトリとは、アプリケーション ビューのデプロイ前であればいつでも切り替えることができます。

既存の接続ファクトリを使用すれば、特に複数のアダプタが 1 つの EIS と対話する場合のサーバ管理を簡素化できます。また、接続ファクトリを共有して使用することにより、管理者はその接続ファクトリのコンフィグレーション パラメータを設定し、ユーザに既存の接続を選択するように指示できます。その場合、ユーザが接続パラメータのコンフィグレーション方法をあらかじめ知っている必要はありません。

6. [Continue] をクリックします。新しい接続ファクトリを作成する選択をした場合は、[Configure Connection Parameters] 画面が表示されます。



既存の接続ファクトリを使用する選択をした場合は、[Application View Administration] 画面が表示されます（[Application View Administration] 画面の詳細については、手順 9. を参照）。[Configure Connection Parameters] 画面では、アプリケーション ビューが対象の EIS と対話するために必要なネットワーク関連の情報を定義します。この情報は、アプリケーション ビューごとに 1 度だけ入力する必要があります。

7. WebLogic Server の ユーザ名とパスワードを入力します。

   注意： この画面に表示される実際のフィールドは、ここに示すものと異なることがあります。どのようなフィールドが表示されるかは、アダプタによって異なります。

8. 残りのフィールドの入力に関する情報については、担当のテクニカル アナリストまたは EIS のスペシャリストに問い合わせてください。

9. [Continue] をクリックします。[Application View Administration] 画面が表示されます。

手順 4A：アプリケーション ビューへのサービスの追加

1. [Application View Administration] 画面で [Service] 行内の [Add] をクリックします。[Add Service] 画面が表示されます。

注意： この画面に表示される実際のフィールドは、ここに示すものと異なることがあります。どのようなフィールドが表示されるかは、アダプタによって異なります。

2. [Unique Service Name] フィールドに、サービス名を入力します。入力する名前は、このサービスが実行する機能を表すものにしてください。また、アプリケーション ビューに対してユニークなサービス名にする必要があります。名前には、任意の有効な Java 識別子が使用できます。

3. [Description] フィールドに、WebLogic Integration Studio に作成したワークフローでこのアプリケーション ビューを使う人にわかりやすいコメントを入力します。

4. 残りのフィールドの入力に関する情報については、担当のテクニカル アナリストまたは EIS のスペシャリストに問い合わせてください。

   多くの場合、この必須情報は、データベースからの情報検索、またはデータベース情報の更新に使用する SQL 文に組み込まれます。次のサンプル文では、ユーザが指定した国の値に基づき、顧客テーブルから顧客情報を検索します。

   ```
   select * from PBPUBLIC.CUSTOMER_TABLE
   where COUNTRY=[country varchar]
   ```

   WebLogic Integration に付属のサンプル アプリケーション ビューには、SQL 文を使用するサービスが収められています。[Application View Administration] 画面を表示して、サービスに対する [View Summary] リンクをクリックしてください。[Summary] 画面に SQL 文が表示されます。

5. 入力が終了したら、[Add] をクリックします。

手順 4B：アプリケーション ビューへのイベントの追加

1. [Application View Console] で、[Administration] を選択します。[Application View Administration] 画面が表示されます。

2. [Events] 行で [Add] をクリックします。[Add Event] 画面が表示されます。

注意： この画面に表示される実際のフィールドは、ここに示すものと異なることがあります。どのようなフィールドが表示されるかは、アダプタによって異なります。

3. [Unique Event Name] フィールドにイベント名を入力します。アプリケーション ビューに対してユニークなイベント名にする必要があります。名前には、任意の有効な Java 識別子が使用できます。
4. [Description] フィールドに、WebLogic Integration Studio に作成したワークフローでこのアプリケーション ビューを使う人にわかりやすいコメントを入力します。
5. 残りのフィールドの入力に関する情報については、担当のテクニカル アナリストまたは EIS のスペシャリストに問い合わせてください。
6. 入力が終了したら、[Add] をクリックします。[Application View Administration] 画面が表示されます。
7. サービスおよびイベントの追加が終了したら [Continue] をクリックして、アプリケーション ビューをデプロイします。

手順 5 : アプリケーション ビューのデプロイ

イベントまたはサービスを最低 1 つ追加すれば、アプリケーション ビューをデプロイできます。アプリケーション ビューは、そのサービスおよびイベントをテストしたり、WebLogic Server 環境でそのビューを使用する前に、デプロイする必要があります。アプリケーション ビューをデプロイすることによって、サービスおよびイベント関連のメタデータが、実行時メタデータ リポジトリ内に格納されます。また、デプロイにより、他の WebLogic Server クライアントがアプリケーション ビューを使用できるようになります。その結果、ビジネス プロセスとアプリケーション ビューとの対話が可能になり、アプリケーション ビューのサービスおよびイベントをテストすることができるようになります。

アプリケーション ビューをデプロイする手順は、次のとおりです。

1. 2-5 ページの「手順 1 : [Application View Console] へのログオン」の説明に従って、アプリケーション ビューを開きます。[Summary for Application View] 画面が表示されます。

2. [Edit] をクリックします。[Application View Administration] 画面が表示されます。

3. [Continue] をクリックします。[Deploy Application View] 画面が表示されます。

注意： [Deploy Application View] 画面にどのようなフィールドが表示されるかは、使用されているアダプタによって異なります。すべてのフィールドの説明については、担当のテクニカル アナリストまたは EIS のスペシャリストに問い合わせてください。アプリケーション ビューで共有接続ファクトリを使用している場合、その接続ファクトリのプロパティが表示されます。

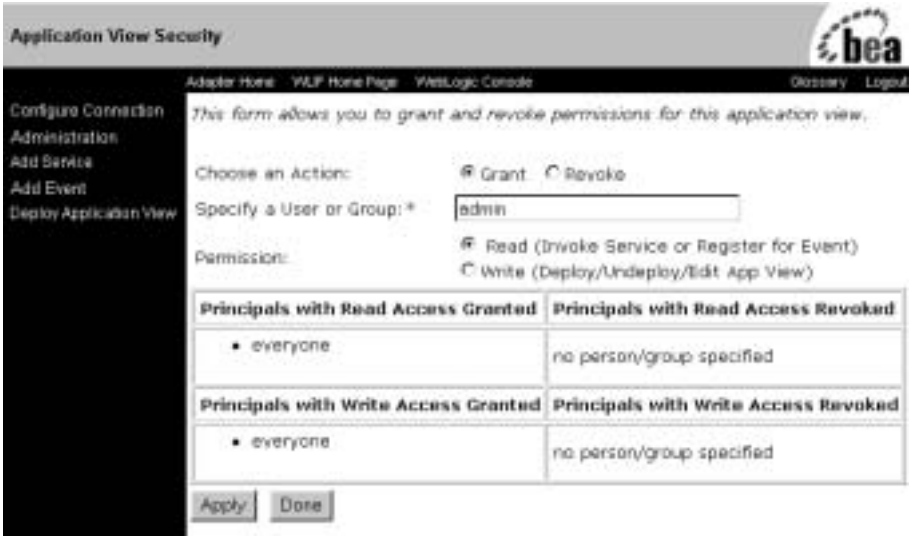
4. WebLogic Integration Studio または権限の与えられた他のクライアントで、このアプリケーション ビューから使用できるサービスを非同期的に呼び出せるようにするには、[Enable Asynchronous Service Invocation] を選択します。

   エンティティがアプリケーション ビューのサービスを非同期的に呼び出す場合、サービスからの応答を待機せずにプロセスを続行します。

5. アプリケーション ビューがイベントをサポートしている場合、アダプタのイベント ルータの URL を入力します。
   例：`http://localhost:7001/YourEIS_EventRouter/EventRouter`

注意： このアプリケーション ビューにイベントが定義されていない場合は、このフィールドは表示されません。

6. [Minimum Pool Size] フィールドに、このアプリケーション ビューが使用する接続プールの最小数を入力します。たとえば、`1` と入力します。
7. [Maximum Pool Size] フィールドに、このアプリケーション ビューが使用する接続プールの最大数を入力します。たとえば、`10` と入力します。
8. [Target Fraction of Maximum Pool Size] フィールドに、0 から 1.0 までの値を使用して、理想的なプール サイズを入力します。たとえば、このフィールドに `0.7` を入力した場合、[Maximum Pool Size] が 10、[Target Fraction] は 0.7 と定義されます。この結果、アダプタでは、最大サイズの 70% の接続プール、すなわちこの場合は、接続数 7 を維持することを目標としたロード バランシングが行われます。
9. 未使用の接続プールが自動的に削除されるようにするには、[Allow Pool to Shrink] を選択します。
10. [Log Configuration] 領域で、ロギング要件に従って以下のオプションのいずれかを選択します。
    - [Log errors and audit messages]
    - [Log warnings, errors, and audit messages]
    - [Log informational messages, warnings, errors, and audit messages]
    - [Log all messages]
11. 必要に応じて、[Restrict Access to your_application using J2EE Security] をクリックします。[Application View Security] 画面が表示されます。



この画面では、WebLogic Server のユーザまたはグループに対して、このアプリケーション ビューの読み込みおよび書き込みアクセスを許可または制限できます。

12. パーミッションの設定が終了したら、[Apply] をクリックして変更内容を保存します。
13. このアプリケーション ビューをすぐにデプロイするか、後からデプロイするかを決めます。アプリケーション ビューを後からデプロイする場合は、手順 14 に進み、すぐにデプロイする場合は手順 15 に進みます。
14. アプリケーション ビューを後からデプロイする場合は、[Done] をクリックして [Deploy Application View] 画面に戻ります。

    デプロイせずにアプリケーション ビューを保存するには、[Save] をクリックします。

    注意： すぐにデプロイせずにアプリケーション ビューを途中で保存する場合は、任意の時点で [Save] をクリックできます。

15. すぐにアプリケーション ビューをデプロイする場合は、[Deploy Persistently] を選択して、WebLogic Server の再起動のたびにそのアプリケーション ビューが自動的に再デプロイされるようにしてから、[Deploy Application View] をクリックします。[Summary for Application View] 画面が表示されます。

アプリケーション ビューで共有接続ファクトリを使用している場合、その接続ファクトリのプロパティが [Deploy] および [Connection] タブに表示されます。既存の接続ファクトリ名の横にある [References] リンクをクリックすると、その接続ファクトリを使用してデプロイされるアプリケーション ビューの名前が表示されます。

任意手順 : アプリケーション ビューのアンデプロイ

アプリケーション ビューの接続パラメータの編集、サービスやイベントの追加、またはクライアントによるアプリケーション ビューの使用を不可能にする場合、アプリケーション ビューをアンデプロイする必要があります。接続パラメータの編集については、2-7 ページの「手順 2 および 3 : アプリケーション ビューの定義および接続パラメータのコンフィグレーション」を参照してください。アプ

リケーション ビューをアンデプロイすると、他の WebLogic Server クライアントはそのビューと対話できなくなり、そのビューのサービスまたはイベントのテストもできなくなります。

アプリケーション ビューをアンデプロイする手順は、次のとおりです。

1. [Application View Console] で [Summary] をクリックします。[Summary for Application View] 画面が表示されます。

2. WebLogic Server からアプリケーション ビューをアンデプロイするには、[Undepoy] をクリックします。[Undepoy Application View] ウィンドウが表示されます。

3. [Confirm] をクリックします。[Summary for Application View] 画面が表示され、ここで再度アプリケーション ビューをデプロイすることができます。

手順 6A：アプリケーション ビュー サービスのテスト

アプリケーション ビュー サービスのテストは、そのサービスが対象 EIS と適切に対話できるかどうかを評価することがその目的です。テストできるのは、デプロイされ、少なくとも 1 つのイベントまたはサービスが定義されているアプリケーション ビューに限られます。アプリケーション ビューのサービスをテストする手順は、次のとおりです。

1. ウィンドウの左側のナビゲーション領域で、[Summary] を選択します。[Summary for Application View] 画面が表示されます。

2. [Events and Services] タブの [Services] 領域で、該当するサービスを見つけ、[Test] をクリックします。[Test Service] 画面が表示されます。

3. 必要に応じて、該当するフィールドに必要なデータを入力します。

注意： [Test Service] 画面に表示される実際のフィールドは、ここに示すものと異なることがあります。どのようなフィールドが表示されるかは、アプリケーション ビュー サービスによって異なります。すべての

フィールドの説明については、担当のテクニカル アナリストまたは EIS のスペシャリストに問い合わせてください。

4. [Test] をクリックします。アプリケーション ビュー サービスで手順 3 で指定した入力データが正しく処理されれば、テストは成功ということになります。[Test Result] 画面が表示され、ここにすべての入力ドキュメントおよび出力ドキュメントがリストされます。

5. テスト対象の各サービスに対してこの手順（手順 1 ～ 4）を繰り返します。
6. アプリケーション ビュー サービスのテスト終了後は、当該アプリケーション ビューはデプロイしたままの状態にすることも、編集するためアンデプロイすることもできます。

手順 6B：アプリケーション ビュー イベントのテスト

アプリケーション ビュー イベントのテストは、そのアプリケーション ビューが EIS アプリケーションに対して適切に応答できるかどうかを評価することがその目的です。テストできるのは、デプロイされ、少なくとも 1 つのイベントまたはサービスが定義されているアプリケーション ビューに限られます。アプリケーション ビューのイベントをテストする手順は、次のとおりです。

1. ウィンドウの左側のナビゲーション領域で、[Summary] をクリックします。[Summary for Application View] 画面が表示されます。

2. [Events and Services] タブの [Events] 領域で、該当するサービスを確認し、[Test] をクリックします。[Test Event] 画面が表示されます。

注意： [Test Event] 画面に表示される実際のフィールドは、ここに示すものと異なることがあります。どのようなフィールドが表示されるかは、アプリケーション ビュー イベントによって異なります。すべてのフィールドの説明については、担当のテクニカル アナリストまたは EIS のスペシャリストに問い合わせてください。

3. テスト イベントを生成する方法を選択します。

   - [Service] ー アプリケーション ビュー固有のサービスを使用して、*定義済み*のイベントを生成する場合は、[Service] を選択します。続いて、2-27 ページの「[Service] を選択した場合」に示された手順を実行します。
   - [Manual] ー EIS アプリケーションにログオンし、該当するイベント生成機能を実行してイベントを生成する場合は、[Manual] を選択します。続いて、2-29 ページの「[Manual] を選択した場合」に示された手順を実行します。

   指定した時間までにアプリケーション ビューのイベントが適切に応答すれば、テストは成功ということになります。

[Service] を選択した場合

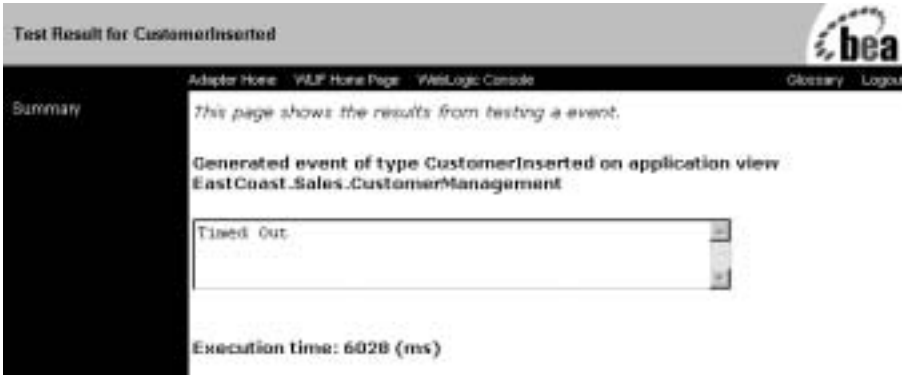
1. [Service] メニューからテスト対象のイベントを呼び出すサービスを選択します。たとえば、[NewCustomer] イベントをテストする場合、[Insert Customer] など、そのイベントを呼び出すサービスを選択します。
2. [Time] フィールドに適当と思われる待機時間をミリ秒単位で入力します（1 分 = 60,000 ミリ秒）。指定された時間が経過してもイベントが成功しないと、テストはタイムアウトとなり、エラー メッセージが表示されます。
3. [Test] をクリックします。イベントを呼び出すサービスが実行されます。

   サービスが入力データを必要とするときは、入力画面が表示されます。

4. サービスの入力データが必要な場合、該当するフィールドにデータを入力し、[Test] をクリックします。

   サービスが実行されます。テストが成功した場合、[Test Result] 画面に、イベント ドキュメント、サービス入力ドキュメント、およびサービス出力ドキュメントが表示されます。

テストが失敗すると、[Test Result] 画面にはタイム アウト メッセージだけが表示されます。



5. テストが失敗したときは、イベント定義を編集するか、システム管理者またはアプリケーション マネージャに連絡してください。
6. テストが成功したときは、残りのテスト対象イベントについてもこのテスト手順を繰り返します。
7. テストが終了したらアプリケーション ビューを保存します。

[Manual] を選択した場合

1. [Time] フィールドに適当と思われる待機時間をミリ秒単位で入力します（1 分 = 60,000 ミリ秒）。この時間が経過してもイベントが成功しないと、テストはタイムアウトとなり、エラー メッセージが表示されます。
2. イベントの呼び出しに使用されるアプリケーションがまだ開かれていない場合は、ここでそのアプリケーションを開きます。
3. [Test] をクリックします。テストはイベントが呼び出されるまで待機します。
4. イベントを呼び出すアプリケーションを使用して、次にアプリケーション ビューのイベントをテストするサービスを実行します。

   テストが成功すると [Test Result] 画面が表示されます。この画面にアプリケーションからのイベント ドキュメント、サービス入力ドキュメント、およびサービス出力ドキュメントが表示されます。

テストが失敗するかタイム アウトが発生した場合は、[Test Result] 画面にタイム アウト メッセージが表示されます。

5. テストが失敗したときは、イベント定義を編集するか、システム管理者またはアプリケーション マネージャに連絡してください。
6. テストが成功したときは、残りのテスト対象イベントについてもこのテスト手順を繰り返します。
7. テストが終了したらアプリケーション ビューを保存します。

アプリケーション ビューの編集

アプリケーション ビューを定義するときは、接続パラメータをコンフィグレーションする必要があります。サービスおよびイベントを追加してテストした後でも、接続パラメータを再コンフィグレーションしたり、サービスとイベントを削除したりできます。

既存のアプリケーション ビューを編集する手順は、次のとおりです。

1. アプリケーション ビューを開きます。[Summary for Application View] 画面が表示されます。

2. [Edit] をクリックします。[Application View Administration] 画面が表示されます。

3. アプリケーション ビューの接続パラメータを再コンフィグレーションするには、左側ペインの [Configure Connection] をクリックします。[Configure Connection Parameter] 画面が表示されます。2-7 ページの「手順 2 および 3 : アプリケーション ビューの定義および接続パラメータのコンフィグレーション」の手順 6 以降を実行します。

4. サービスおよびイベントを追加するには、[Add Service] または [Add Event] をクリックします。2-11 ページの「手順 4A : アプリケーション ビューへのサービスの追加」または 2-13 ページの「手順 4B : アプリケーション ビューへのイベントの追加」の手順に従います。

3 Studio におけるアプリケーション ビューの使用

この章では、以下のトピックを取り上げます。

- 始める前に
- タスク 1 : アプリケーション ビュー サービスを呼び出すタスク ノードの設定
- タスク 2 : 非同期アプリケーション ビュー サービスからの応答を待機するイベント ノードの設定
- タスク 3 : アプリケーション ビュー イベントによって開始されるワークフローの作成
- タスク 4 : アプリケーション ビュー イベントを待機するイベント ノードの設定
- ワークフロー内におけるアプリケーション ビューのローカル トランザクションの処理

始める前に

企業に必要なアプリケーション ビューのサービスとイベントをすべて作成すれば、そのアプリケーション ビューを使用してビジネス プロセスを実行できます。ビジネス プロセスを実行する場合、WebLogic Integration Studio でアプリケーション ビューのサービスとイベントを使用するビジネス プロセス ワークフローを設計するのが、最も簡単な方法です。

WebLogic Integration Studio は、ビジネス プロセス ワークフローを設計するためのグラフィカル ユーザ インタフェース（GUI）です。ワークフローには、WebLogic Integration によって定義したアプリケーション ビューのサービスおよびイベントを組み込むことができます。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。

WebLogic Integration Studio でアプリケーション ビュー サービスを呼び出したり、アプリケーション ビュー イベントを受信するには、以下の前提条件を満たしていることを確認する必要があります。

- アプリケーション ビューの作成と、アプリケーション ビューに対するサービスとイベントの定義が済んでいること。
- アプリケーション ビューとそのアダプタが正常に機能し、保存されていること。実行中のワークフローからアプリケーション ビューのサービスおよびイベントを呼び出す場合には、アプリケーション ビューのデプロイも必要です。
- BPM および Application Integration 機能が使用可能であること。
- Application Integration のプラグインがロードされていること。
- 定義するワークフローに必要なビジネス ロジックに関する情報を適切なビジネス アナリストから入手していること。
- ワークフロー テンプレート定義が開いていること。

ワークフローのセットアップ タスク

以下の節では、アプリケーション ビュー サービスおよびイベントを使用するワークフローをセットアップするときに実行する 4 つのタスクについて説明します。

- タスク 1 : アプリケーション ビュー サービスを呼び出すタスク ノードの設定
- タスク 2 : 非同期アプリケーション ビュー サービスからの応答を待機するイベント ノードの設定
- タスク 3 : アプリケーション ビュー イベントによって開始されるワークフローの作成
- タスク 4 : アプリケーション ビュー イベントを待機するイベント ノードの設定

これらのタスクは自由に組み合わせて実行し、独自のワークフローを作成できます。

このマニュアルでは、WebLogic Integration で提供されている Business Process Management (BPM) 機能の使い方を完全に説明しているわけではありません。詳細については、『*WebLogic Integration チュートリアル*』を参照してください。

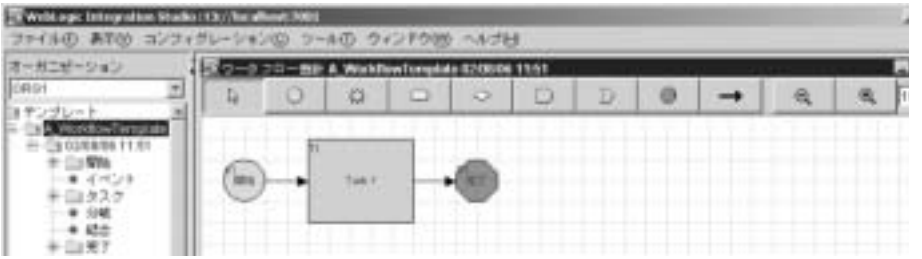
タスク 1 : アプリケーション ビュー サービスを呼び出すタスク ノードの設定

オーガニゼーションにおいて、ワークフローの中からアプリケーション ビューのサービスを呼び出すことが必要になる場合があります。そのような呼び出しができるようにするには、ワークフローにタスク ノードを追加し、以下に該当するアプリケーション ビュー サービスのアクションをそのタスク ノードに追加します。ワークフローを保存してアクティブ化すれば、タスク ノードの実行時に常にこのアプリケーション ビュー サービスが呼び出されるようになります。

アプリケーション ビューのサービスを呼び出すタスク ノードの設定手順

アプリケーション ビュー サービスを呼び出すタスク ノードを作成する手順は、次のとおりです。

1. WebLogic Integration Studio で、テンプレート定義を開きます。[ワークフロー設計] ウィンドウが表示されます。



2. タスク ノードがない場合は作成します。
3. アプリケーション ビューのサービスを呼び出す [タスク] ノードをダブルクリックします。[タスクのプロパティ] ダイアログ ボックスが表示されます。

4. [アクション] 領域で、どのタブからサービスを呼び出すのかを選択します。選択肢は、ビジネス プロセスによって異なります。
5. [追加] をクリックします。[アクションを追加] ダイアログ ボックスが表示されます。

6. ナビゲーション ツリーから、[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] の順に選択して、[OK] をクリックします。[サービスの呼び出し] ダイアログ ボックスが表示されます。

7. ナビゲーション ツリーで、呼び出しの対象となるサービスを探し、これを選択します。

   ナビゲーション ツリーでは、各アプリケーション ビュー サービスがフォルダ（例 : Test2）別、およびアプリケーション ビュー（例 : Test）別に編成されています。すべてのアプリケーション ビュー サービスは、ナビゲーション ツリーの最下位レベルに表示されます。

   注意 : 任意の時点で、保存されたアプリケーション ビューとイベントの最新状況をチェックするには、[ツリーの更新] をクリックします。ナビゲーション ツリーが表示されない、または表示が狭すぎる場合は、長すぎる XML 変数名または文字列変数名が含まれていることが考えられます。その場合、XML 変数または文字列変数の名前を短くしてください。

8. [要求ドキュメント変数] リストから、アプリケーション ビュー サービスの入力データを含む既存の XML 変数を選択します。

9. 該当する XML 変数が存在しない場合は、<new> を選択して [変数プロパティ] ダイアログ ボックスを開き、新しい XML 変数を作成します。

10. [名前] フィールドに、変数名を入力します。
11. [型] メニューから [XML] を選択します（このメニューの選択肢は、[XML] のみ）。

    新しい変数の定義方法については、『*WebLogic Integration Studio ユーザーズ ガイド*』を参照してください。
12. [OK] をクリックします。[サービスの呼び出し] ダイアログ ボックスに戻ります。
13. 指定されたサービスのサービス要求テンプレートを作成する、または既存のサービス要求テンプレートを修正するには、次のいずれかを選択します。
    - [Set] ーテンプレートを作成する
    - [Edit] ー既存のテンプレートを編集する

    どちらの項目を選択しても、[サービス要求テンプレート] ダイアログ ボックスが表示されます。

[サービス要求テンプレート] ダイアログ ボックスには、手順 11 で指定されたタイプのすべてのサービス要求に使用されるテンプレートが表示されます。このテンプレートは、サービスに関する入力スキーマが基準になっています。

このアクションが実行されると、指定された要求ドキュメント変数にテンプレート データが割り当てられ、サービス用の入力ドキュメントとして使用されます。このテンプレートは、その変数に対する既存のすべての設定をオーバーライドします（テンプレートを設定し、かつ新しい入力変数ではなく既存の入力変数を使用したい場合、[アプリケーション ビュー サービスの呼び出し] を削除し、再作成するのが最も簡単な方法）。

[Service Request Template] ダイアログ ボックスの使用方法については、『*WebLogic Integration Studio ユーザーズ ガイド*』を参照してください。

14. [OK] をクリックします。[サービスの呼び出し] ダイアログ ボックスに戻ります。
15. 入力ドキュメントの XML スキーマを検証する必要がある場合は、[要求定義の表示] を選択します。[View Definition] ダイアログ ボックスが表示されます。

16. 終了したら [閉じる] をクリックします。

17. [同期] または [非同期] を選択して、アプリケーション ビューを同期的または非同期的に呼び出します。

    注意： ノードがサービスを同期的に呼び出す場合、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処理を続行します。ノードが非同期的にサービスを呼び出す場合、ワークフローは、応答を待機せずに処理を続行します。

18. 同期サービスで応答データを格納する必要のある場合、[応答ドキュメント変数] リストから事前定義された XML 変数を選択します。Studio がアプリケーション ビュー サービスから応答を受信すると、その応答データは応答ドキュメント変数に格納されます。応答データが必要ない場合は、このフィールドを空白のままにします。

19. 該当する XML 変数が存在しない場合は、<new> を選択して [変数プロパティ] ダイアログ ボックスを開き、新しい XML 変数を作成します。詳細については、「アプリケーション ビューのサービスを呼び出すタスク ノードの設定手順」の手順 9. を参照してください。

    新しい変数の定義方法については、『*WebLogic Integration Studio ユーザーズ ガイド*』を参照してください。

20. 応答ドキュメントの XML スキーマを検証する必要がある場合は、[応答定義の表示] をクリックします。[View Definition] ダイアログ ボックスが表示されます。

21. 終了したら [閉じる] をクリックします。

22. 非同期サービスで要求 ID を格納する必要がある場合、[要求 ID 変数] リストから事前定義され文字列変数を選択します。

23. 該当する文字列変数が存在しない場合は、<new> を選択して [変数プロパティ] ダイアログ ボックスを開き、新しい文字列変数を作成します。

24. [名前] フィールドに、変数名を入力します。

25. [型] メニューから [String] を選択します（このメニューの選択肢は、[String] のみ）。

    新しい変数の定義方法については、『*WebLogic Integration Studio ユーザーズ ガイド*』を参照してください。

注意 : タスク ノードを設定して非同期アプリケーション ビュー サービスを呼び出すと、結果が Studio に返されます。ワークフローは、選択された要求 ID 変数によってこの応答を識別します。イベント ノードを設定して応答を受信するには、イベント ノードに対して、必ず同一の要求 ID 変数を使用する必要があります。このようなイベント ノードの作成方法については、3-11 ページの「タスク 2 : 非同期アプリケーション ビュー サービスからの応答を待機するイベント ノードの設定」を参照してください。

26. [OK] をクリックしてアクションを保存します。

27. [タスクのプロパティ] ダイアログ ボックスで、[OK] をクリックしてノードを保存します。

タスク 2 : 非同期アプリケーション ビュー サービスからの応答を待機するイベント ノードの設定

この節では、非同期アプリケーション ビュー サービスの応答を受信する方法と、その際に発生する可能性のあるエラーの処理方法について説明します。

ワークフロー内で、アクションの実行時にアプリケーション ビュー サービスが非同期的に呼び出されると、常にアプリケーション ビュー サービスが応答を返します。応答が必要な場合、その応答を待機する、対応する非同期イベント ノードを設定する必要があります。この節では、イベント ノードが、エラー チェックを行わずにアプリケーション ビュー サービスの応答を受信するといった非常に単純なシナリオについて説明します。

非同期イベント ノードを設定して非同期アプリケーション ビュー サービスからの応答を待機するには、イベント ノードを作成し、次にタイプが AI Async Response のイベントを待機するようにイベント ノードをコンフィグレーションします。

応答受信のコンフィグレーション

非同期サービス応答を受信するイベント ノードを設定するには、以下のいずれかの方法を使用できます。

- [応答ドキュメント（推奨）] タブを選択する（推奨方法）。次に要求 ID 変数（文字列）と応答ドキュメント変数（XML 型）を選択します。この方法の詳細については、3-13 ページの「非同期サービス応答受信のコンフィグレーション手順（推奨方法）」を参照してください。
- [非同期変数（レガシー）] タブを使用する（従来の方法）。次に、要求 ID 変数、非同期サービス応答変数（文字列）、および非同期サービス応答変数（AsyncServiceResponse タイプ）を選択します。この方法の詳細については、3-17 ページの「非同期サービス応答受信のコンフィグレーション手順（従来の方法）」を参照してください。

  注意： [応答ドキュメント] タブを使用する方法をお勧めします。この方法は、非同期応答と同期応答の両方の受信に対応しているからです。この方法を使用する場合は、応答が非同期的か同期的かということには無関係に XML ドキュメントが受信されます。したがって、非同期サービスの応答変数の値をクエリする必要はありません。

できる限り、応答ドキュメント変数を使用して、非同期サービスの応答を受信してください。AI Async Response タイプのイベントを待機するサービスをコンフィグレーションするには、[イベントのプロパティ] ダイアログ ボックスを使用します。このダイアログ ボックスで応答を受信する非同期変数を使用するオプションが提供される場合があります（提供されない場合もある）。このオプションが使用可能かどうかは、以下の条件によって決まります。

- 応答を受信する非同期サービス応答変数を使用するように設定された既存の AI Async Response イベント ノードを編集する場合、[イベントのプロパティ] ダイアログ ボックスに 2 つのタブが表示されます。[非同期変数（レガシー] タブ（従来の方法）と [応答ドキュメント] タブ（推奨方法）の 2 つです。したがって、サービス応答受信のコンフィグレーションに使用できる方法は、この場合 2 つあります。
- 非同期サービス応答変数を使用していない既存の AI Async Response イベント ノードを編集する、または新しい AI Async Response イベント ノードを作成する場合、[イベントのプロパティ] ダイアログ ボックス内に、タブのないダイアログ ボックスが表示されます。この場合、サービス応答を受信する応答ドキュメントを設定する必要があります（推奨方法）。

非同期アプリケーション ビュー サービス応答でのエラー処理

このタスクでは、アプリケーション ビュー サービス応答に対するエラー処理のコンフィグレーションは含まれませんが、通常、自分のワークフロー内でエラーを処理したい、と考えるでしょう。AsyncServiceResponse 変数を使用するワークフロー内で、非同期サービス応答のエラーを処理するには、Application Integration のプラグインに実装されている機能を使用できます。

Application Integration プラグインには、変数タイプ AsyncServiceResponse と次の 3 つの関数が実装されています。

- `AIHasError()`
- `AIGetErrorMsg()`
- `AIGetResponseDocument()`

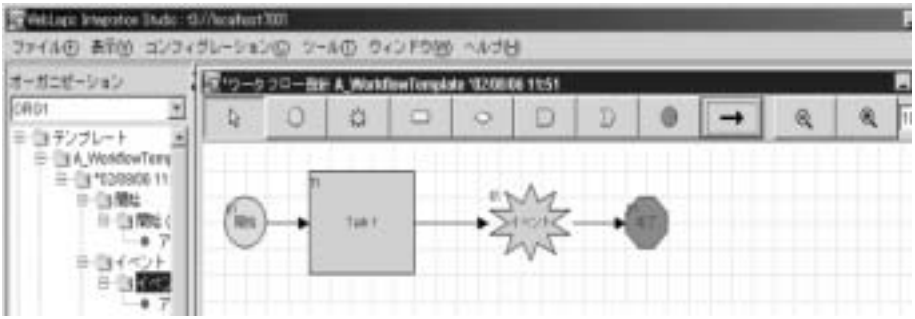
これらの関数の詳細については、3-19 ページの「Application Integration プラグインに実装される関数」を参照してください。

非同期サービス応答受信のコンフィグレーション手順（推奨方法）

非同期イベント ノードを設定して非同期アプリケーション ビュー サービスからの応答を待機するには、イベント ノードを作成し、次にタイプが AI Async Response のイベントを待機するようにイベント ノードをコンフィグレーションします。

非同期サービス応答の受信に XML 変数を使用するイベント ノードを設定する手順は、次のとおりです。

1. WebLogic Integration Studio で、ワークフロー テンプレート定義を開きます。[ワークフロー設計] ウィンドウが表示されます。



2. イベント ノードがない場合はここで作成します。このイベント ノードは、指定されたアプリケーション ビュー サービスからの非同期応答を待機します。
3. イベント ノードをダブルクリックします。[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。

4. （省略可能）[説明] フィールドに名前を入力します。
5. [型] リストから、[Al 非同期応答] を選択します。
6. [応答ドキュメント（推奨）] タブを選択します。

   注意 : ワークフローに AsyncServiceResponse 変数が使用されていない場合、または新しい [Al 非同期応答] イベント ノードを作成する場合は、[イベントのプロパティ] ダイアログ ボックスの中に、タブのないダイアログ ボックスが表示されます。このダイアログ ボックスを使用して、サービス応答を受信する応答ドキュメントを設定します（この方法を推奨する）。

7. [要求 ID 変数] リストから、定義済みの文字列変数を選択します。
   WebLogic Integration のプロセス エンジンでは、この変数に格納された ID に一致する ID を持つ非同期応答をリスンします。

8. 該当する文字列変数が存在しない場合は、<new> を選択して [変数プロパティ] ダイアログ ボックスを開き、新しい文字列変数を作成します。詳細については、「アプリケーション ビューのサービスを呼び出すタスク ノードの設定手順」の手順 23. を参照してください。

   新しい変数の定義方法については、『*WebLogic Integration Studio ユーザーズ ガイド*』を参照してください。

   注意 : このイベント ノードの目的は、ワークフロー内で以前に非同期的に呼び出されたアプリケーション ビュー サービスの呼び出しアクションに対する応答を待機することです。アプリケーション ビュー サービスの呼び出しアクションにより、要求 ID 変数が設定されます。このアクションとイベント ノードは、両方で同じリクエスト ID 変数を使用している場合に限って連動できます。アプリケーション ビュー サービスの呼び出しアクションの設定方法については、3-3 ページの「タスク 1 : アプリケーション ビュー サービスを呼び出すタスク ノードの設定」を参照してください。

9. 非同期サービスで応答データを格納する必要のある場合、[応答ドキュメント変数] リストから事前定義した XML 変数を選択します。これで WebLogic Integration がアプリケーション ビュー サービスから応答を受信すると、その応答データは応答ドキュメント変数に格納されます。応答データが必要でない場合は、この手順は省略してください。

10. 該当する XML 変数が存在しない場合は、<new> を選択して [変数プロパティ] ダイアログ ボックスを開き、新しい XML 変数を作成します。詳細については、「アプリケーション ビューのサービスを呼び出すタスク ノードの設定手順」の手順 9. を参照してください。

    新しい変数の定義方法については、『*WebLogic Integration Studio ユーザーズ ガイド*』を参照してください。

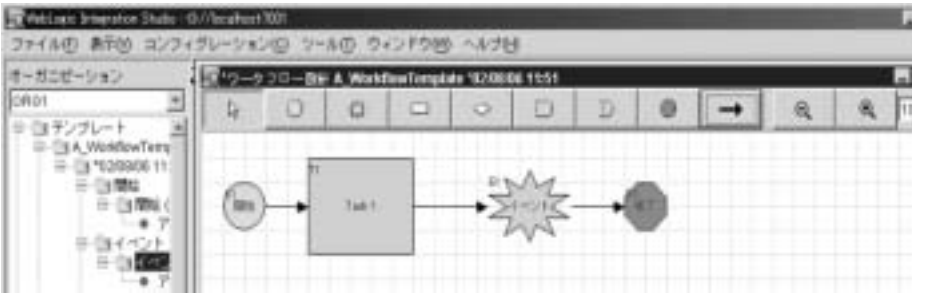
11. [OK] をクリックして、イベント ノードを保存します。

非同期サービス応答受信のコンフィグレーション手順（従来の方法）

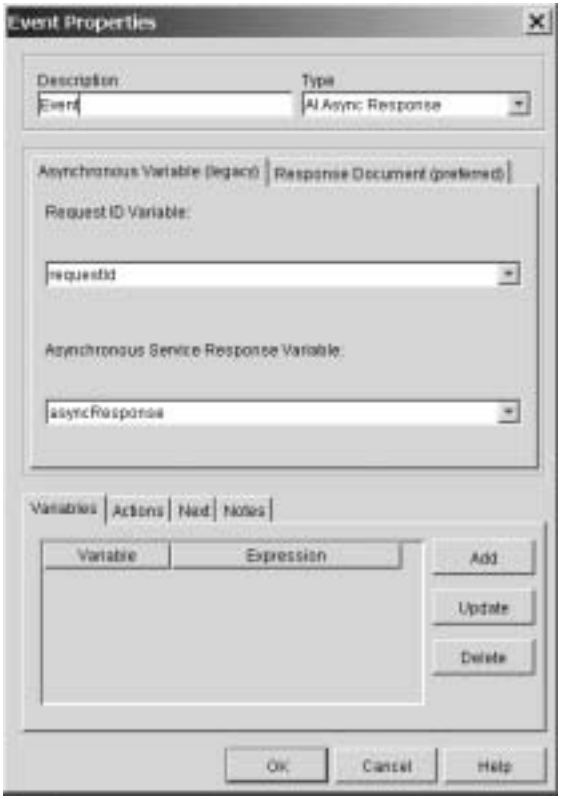
非同期サービスの応答を受信する場合、XML タイプの応答ドキュメント変数を使用することをお勧めします。ただし、既存のワークフロー内に非同期イベント ノードがあり、AsynchServiceResponse 変数を使用して、非同期アプリケーション ビュー サービスからの応答を待機するように設定されている場合、そのイベント ノードを修正して使用することができます。

非同期サービス応答の受信に AsyncServiceResponse 変数を使用するイベント ノードを修正する手順は、次のとおりです。

1. WebLogic Integration Studio で、ワークフロー テンプレート定義を開きます。[ワークフロー設計] ウィンドウが表示されます。



2. 非同期イベント ノードをダブルクリックします。[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。



3. [非同期変数（レガシー）] タブを選択します。
4. [要求 ID 変数] リストから、定義済みの文字列変数を選択します。WebLogic Integration では、この変数に格納された ID に一致する ID を持つ非同期応答をリスンします。

   注意 : このイベント ノードの目的は、ワークフロー内で以前に非同期的に呼び出されたアプリケーション ビュー サービスの呼び出しアクションに対する応答を待機することです。アプリケーション ビュー サービスの呼び出しアクションにより、要求 ID 変数が設定されます。このアクションとイベント ノードは、両方で同じリクエスト ID 変数を使用している場合に限って連動できます。アプリケーション ビュー サービスの呼び出しアクションの設定方法については、3-3 ページの

「タスク 1 : アプリケーション ビュー サービスを呼び出すタスク ノードの設定」を参照してください。

5. [非同期サービス応答変数] リストから、応答データを格納する AsyncServiceResponse 変数を選択します。

   注意 : 既存の非同期イベント ノードを修正する場合、[非同期サービス応答変数] フィールドは既にデータが表示されているはずです。応答データが必要ない場合は、[応答ドキュメント（推奨）] タブを選択します。推奨方法の詳細については、3-13 ページの「非同期サービス応答受信のコンフィグレーション手順（推奨方法）」を参照してください。

6. [OK] をクリックして、イベント ノードを保存します。

Application Integration プラグインに実装される関数

企業で AI Async Response 変数を使用しており、Application Integration プラグインの使用時にそれらの変数を取り出したい場合は、次の関数を使用します。

- `AIHasError()`
- `AIGetErrorMsg()`
- `AIGetResponseDocument()`

上記の関数を使用して、正常条件またはエラー条件を処理するための分岐ノードを設定できます。

注意 : これらの関数が使用できるのは、Application Integration プラグインがインストールされている場合に限り、対象となるのは非同期サービス応答を受信するのに非同期変数を使用する方法だけです。詳細については、3-17 ページの「非同期サービス応答受信のコンフィグレーション手順（従来の方法）」を参照してください。

AIHasError()

非同期サービス応答ステータスを判断するには、`AIHasError()` を使用します。次の表には、この関数の詳細が示されています。

オペランド	AsyncServiceResponse 変数
前提条件	■ タイプが AsyncServiceResponse の変数が作成されていること。 ■ 非同期アプリケーション ビュー サービスが呼び出されていること。 ■ アプリケーション ビュー サービスによって返された応答が、AsyncServiceResponse 変数に格納されていること。
戻り値	Boolean
出力データの説明	False －非同期アプリケーション ビュー サービスの呼び出しに成功。 True －非同期アプリケーション ビュー サービスの呼び出しに失敗。

AIGetErrorMsg()

非同期アプリケーション ビュー サービスから返されたエラー メッセージ文字列を読み込むには、`AIGetErrorMsg()` を使用します。次の表には、この関数の詳細が示されています。

オペランド	AsyncServiceResponse 変数
前提条件	■ タイプが AsyncServiceResponse の変数が作成されていること。 ■ 非同期アプリケーション ビュー サービスが呼び出されていること。 ■ アプリケーション ビュー サービスによって返された応答が、AsyncServiceResponse 変数に格納されていること。
戻り値	文字列

出力データの説明	エラー文字列 －非同期アプリケーション ビューの応答に失敗した理由を説明するエラー文字列が返される。 空の文字列 －エラーはなし。

AIGetResponseDocument()

非同期アプリケーション ビュー サービスから返された XML 応答ドキュメントを読み込むには、`AIGetResponseDocument()` を使用します。次の表には、この関数の詳細が示されています。

オペランド	AsyncServiceResponse 変数
前提条件	■ タイプが AsyncServiceResponse の変数が作成されていること。 ■ 非同期アプリケーション ビュー サービスが呼び出されていること。 ■ アプリケーション ビュー サービスによって返された応答が、AsyncServiceResponse 変数に格納されていること。
戻り値	XML
出力データの説明	XML ドキュメント －非同期サービスの応答を表す XML ドキュメントが返される。 NULL －エラーが発生したため、応答ドキュメントは返されていない。

タスク 3 : アプリケーション ビュー イベントによって開始されるワークフローの作成

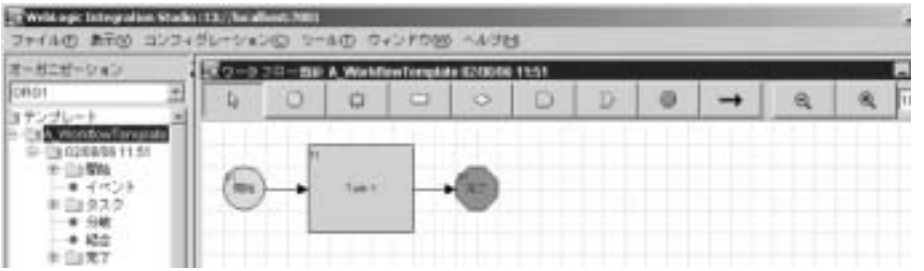
時として、指定されたアプリケーション ビュー イベントが発生するたびに一定のワークフローが開始するように設定したい場合があります。そのようなワークフローを作成するには、そのワークフローがタイプ AI Start のイベントに応答す

るようにワークフローの開始ノードを編集し、該当するアプリケーション ビュー イベントを選択します。必要に応じて、イベントのフィルタ処理を実行するための条件を設定できます。このワークフローを保存してアクティブ化すると、そのアプリケーション ビュー イベントが発生するたびに開始ノードが実行されます。

アプリケーション ビュー イベントによって開始されるワークフローの作成手順

アプリケーション ビュー イベントによって起動される開始ノードを持つワークフローを設定する手順は、次のとおりです。

1. WebLogic Integration Studio で、テンプレート定義を開きます。[ワークフロー設計] ウィンドウが表示されます。



2. 開始ノードがない場合はここで作成します。この開始ノードは、ユーザが指定したアプリケーション ビュー イベントに応答します。
3. 開始ノードをダブルクリックします。[開始のプロパティ] ダイアログ ボックスが表示されます。

4. （省略可能）[説明] フィールドに名前を入力します。
5. [イベント] をクリックします。
6. [イベント] リストから、[AI の起動] を選択します。
7. ナビゲーション ツリーで、アプリケーション ビュー イベントを選択します。

   ナビゲーション ツリーでは、各アプリケーション ビュー イベントがフォルダ（この前の [開始のプロパティ] ダイアログ ボックスに表示されている [EastCoast] フォルダおよび [Sales] フォルダなど）別、およびアプリケーション ビュー（CustomerManagement など）別に編成されています。すべて

のアプリケーション ビュー イベントは、ナビゲーション ツリーの最下位レベルに表示されます。

注意： 任意の時点で、保存されたアプリケーション ビューとイベントの最新状況をチェックするには、[ツリーの更新] をクリックします。

ナビゲーション ツリーが表示されない、または表示が狭すぎる場合は、長すぎる XML 変数名または文字列変数名が含まれていることが考えられます。その場合、XML 変数または文字列変数の名前を短くしてください。

8. [キー値表現] フィールドで、イベントのキー値を定義します。キー値を表す文字列を入力するか、実行時に評価され、キー値を生成する式を作成します ([A + B] オプションを選択すれば [Expression Builder] ダイアログ ボックスが表示されるので、ここで適切な式を作成します)。WebLogic Integration がイベントを起動するためには、ここに指定するキー値が、着信イベントの XML 内の要素と一致する必要があります。

   また、WebLogic Integration が、指定したキー値と着信イベントに対する XML メッセージのキー値を比較できるように、着信キー値を検索する式を定義する必要があります。キー値式がない場合、式を作成するように求められます。作成しなければイベントは保存できません。

   [Event Descriptor for AI Events] は、次の形式によるアプリケーション ビュー イベントの完全修飾名です。

   `namespace.application view name.event`

   `Root` は、この完全修飾イベント名には含まれません。[Event Descriptor] フィールドは、キー値式が直接 [イベント] ダイアログ ボックスから作成された場合、自動的に入力されます。内容とは無関係に、指定したアプリケーション ビュー イベントのインスタンスごとに受け入れる場合は、このフィールドを空白にしておきます。

9. 必要に応じてイベントにフィルタ処理を設定します。フィルタ処理する方法には、[条件] フィールドに条件を入力する方法と、[A + B] オプションを選択して [Expression Builder] ダイアログ ボックスを表示し、該当する式を作成する方法とがあります。

   条件の設定と XPath 式の詳細については、『*WebLogic Integration Studio ユーザーズ ガイド*』を参照してください。

10. [イベント ドキュメント変数] リストから、XML 変数を選択します。開始ノードが受信したアプリケーション ビュー イベントのデータが、この変数に格納されます。イベント データが必要ない場合は、この手順を省略してください。

11. 該当する XML 変数が存在しない場合は、<new> を選択して [変数プロパティ] ダイアログ ボックスを開き、新しい XML 変数を作成します。詳細については、「アプリケーション ビューのサービスを呼び出すタスク ノードの設定手順」の手順 9. を参照してください。

    新しい変数の定義方法については、『*WebLogic Integration Studio ユーザーズ ガイド*』を参照してください。

12. イベント ドキュメントの XML スキーマを検証する必要がある場合は、[ビュー定義] をクリックします。[ビュー定義] ダイアログ ボックスが表示されます。

13. [閉じる] をクリックすると、[開始のプロパティ] ダイアログ ボックスに戻ります。

14. [開始のプロパティ] ダイアログ ボックスで、[OK] をクリックします。新規または修正した開始ノードが保存されます。

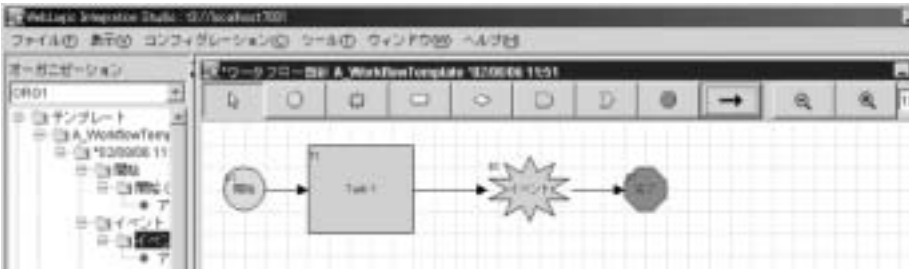
タスク 4: アプリケーション ビュー イベントを待機するイベント ノードの設定

ワークフローでは、時としてアプリケーション ビュー イベントによって起動されるイベント ノードを設定したい場合があります。そのようなノードを作成するには、AI Event タイプのイベントに応答するようにイベント ノードを編集し、それから該当するアプリケーション ビュー イベントを選択します。必要に応じて、目的のアプリケーション ビュー イベントにフィルタ処理をするための条件を設定できます。ワークフローを保存してアクティブ化すると、ワークフローはこのイベント ノードまで進み、指定されたアプリケーション ビュー イベントを待機してから処理を続行します。

アプリケーション ビュー イベントを待機するノードの設定手順

アプリケーション ビュー イベントによって起動されるイベント ノードを設定する手順は、次のとおりです。

1. WebLogic Integration Studio で、テンプレート定義を開きます。[ワークフロー設計] ウィンドウが表示されます。



2. イベント ノードがない場合はここで作成します。このイベント ノードは、指定されたアプリケーション ビュー イベントによって起動されます。
3. イベント ノードをダブルクリックします。[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。

4. （省略可能）[説明] フィールドに名前を入力します。
5. [型] リストから、[AI イベント] を選択します。
6. ナビゲーション ツリーで、アプリケーション ビュー イベントを選択します。

   ナビゲーション ツリーでは、各アプリケーション ビュー イベントがフォルダ（この前の [開始のプロパティ] ダイアログ ボックスに表示されている [EastCoast] フォルダおよび [Sales] フォルダなど）別、およびアプリケーション ビュー（CustomerManagement など）別に編成されています。すべて

のアプリケーション ビュー イベントは、ナビゲーション ツリーの最下位レベルに表示されます。

注意： 任意の時点で、保存されたアプリケーション ビューとイベントの最新状況をチェックするには、[ツリーの更新] をクリックします。

ナビゲーション ツリーが表示されない、または表示が狭すぎる場合は、長すぎる XML 変数名または文字列変数名が含まれていることが考えられます。その場合、XML 変数または文字列変数の名前を短くしてください。

7. [キー値表現] フィールドで、イベントのキー値を定義します。キー値を表す変数または文字列を入力するか、実行時に評価され、キー値を生成する式を作成します（[A + B] オプションを選択すれば [Expression Builder] ダイアログ ボックスが表示されるので、ここで適切な式を作成する）。WebLogic Integration がイベントを起動するためには、ここに指定するキー値が、着信イベントの XML 内の要素と一致する必要があります。

   また、WebLogic Integration が、指定したキー値と着信イベントに対する XML メッセージのキー値を比較できるように、着信キー値を検索する式を定義する必要があります。キー値式がない場合、式を作成するように求められます。作成しなければイベントは保存できません。

   [Event Descriptor for AI Events] は、次の形式によるアプリケーション ビュー イベントの完全修飾名です。

   ```
   namespace.application view name.event
   ```

   `Root` は、この完全修飾イベント名には含まれません。[Event Descriptor] フィールドは、キー値式が直接 [イベント] ダイアログ ボックスから作成された場合、自動的に入力されます。内容とは無関係に、指定したアプリケーション ビュー イベントのインスタンスごとに受け入れる場合は、このフィールドを空白にしておきます。

8. 必要に応じてイベントにフィルタ処理を設定します。フィルタ処理する方法には、[条件] フィールドに条件を入力する方法と、[A + B] オプションを選択して [Expression Builder] ダイアログ ボックスを表示し、該当する式を作成する方法とがあります。

   条件の設定と XPath 式の詳細については、『*WebLogic Integration Studio ユーザーズ ガイド*』を参照してください。

9. [イベントのプロパティ] ダイアログ ボックスで、[イベント ドキュメント変数] リストから XML 変数を選択します。開始ノードが受信したアプリケーション ビュー イベントのデータが、この変数に格納されます。イベント データを保存する必要がない場合は、この手順を省略してください。

10. 該当する XML 変数が存在しない場合は、<new> を選択して [変数プロパティ] ダイアログ ボックスを開き、新しい XML 変数を作成します。詳細については、「アプリケーション ビューのサービスを呼び出すタスク ノードの設定手順」の手順 9. を参照してください。

    新しい変数の定義方法については、『*WebLogic Integration Studio ユーザーズ ガイド*』を参照してください。

11. イベント ドキュメントの XML スキーマを検証する必要がある場合は、[ビュー定義] をクリックします。[ビュー定義] ダイアログ ボックスが表示されます。

12. 終了したら [閉じる] をクリックします。

13. [イベントのプロパティ] ダイアログ ボックスで、[OK] をクリックします。

ワークフロー内におけるアプリケーション ビューのローカル トランザクションの処理

LocalTransaction インタフェースは、CCI (Common Client Interface) 接続クラスによってアダプタ クライアントに公開されます。現在、アプリケーション ビュー インタフェースでは、CCI LocalTransaction インタフェースは使用しません。ローカル トランザクションを管理するには、ユーザはまず接続オブジェクトから LocalTransaction を入手する必要があります。

ローカル トランザクション管理規約

ローカル トランザクション管理規約は、アダプタが、そのリソース マネージャで実行されるローカル トランザクションをサポートするために `javax.resource.spi.LocalTransaction` を実装する場合に作成されます。このタイプの規約によって、アプリケーション サーバがトランザクション管理のインフラストラクチャと実行時環境を整えられます。アプリケーション コンポーネントはこのトランザクション インフラストラクチャを利用して、コンポーネントレベルのトランザクション モデルをサポートします。

トランザクション境界設定の詳細については、以下を参照してください。

```
http://java.sun.com/blueprints/guidelines/designing_enterprise_applications/transaction_management/platform/index.html
```

ユーザ定義のトランザクション境界設定がないローカル トランザクションのコネクタ サポート

以下に示すのは、Application Integration プラグイン内でアプリケーション ビューのローカル トランザクションをサポートする場合のシナリオです。このシナリオは、コネクタがローカル トランザクションだけをサポートするため、EJB トランザクションの `TX_REQUIRES_NEW` に類似しています。

このシナリオでは、コネクタはローカル トランザクションだけをサポートし、BPM 設計者はローカル トランザクションの開始と終了の境界を明示的に設定しません。WebLogic Integration では、LocalTransaction オブジェクトの周囲に XA ラッパーを配置することにより、コネクタがグローバル トランザクションに参加できます。XA ラッパーは、LocalTransaction インスタンスに委託できない XAResource インタフェース上のすべてのメソッドに無演算命令を出します。WebLogic Integration では、トランザクション チェーンに組み込める非 XA リソースは 1 つだけです。したがって、ユーザがワークフロー内に設定できるアプリケーション ビュー LocalTransaction は 1 つだけです。

XA トランザクションのコネクタ サポート

このシナリオでは、アプリケーション ビュー サービスは、ローカル トランザクション内では呼び出されません。各サービス呼び出しは、リソース アダプタが XA をサポートしているため、自動的にグローバル XA トランザクションに追加されます。

4　カスタム コードの作成によるアプリケーション ビューの使用方法

開発者は、カスタム コードを作成してアプリケーション ビューを修正できます。Application View Console でほとんどのアプリケーション ビュー機能を使用できますが、一部にはカスタム コードを作成しなければ使用できない機能もあります。

この章ではカスタム コードが使用される例として 2 つのシナリオを提示します。

- シナリオ 1: 特定の資格に基づいた接続の作成
- シナリオ 2: ビジネス プロセスのカスタム コードの作成

シナリオ 1: 特定の資格に基づいた接続の作成

アプリケーション ビューのサービスを呼び出す前に、そのビューに対して一定のセキュリティ レベルを設定する必要がある場合、該当するに資格を設定することにより、セキュリティを設定できます。資格設定には、`ApplicationView` メソッドの `setConnectionSpec()` および `getConnectionSpec()` を使用します。どちらのメソッドでも `ConnectionSpec` オブジェクトが使用されます。

`ConnectionSpec` オブジェクトをインスタンス化するには、BEA WebLogic Integration Adapter Development Kit (ADK) に用意されている `ConnectionRequestInfoMap` クラスを使用するか、または、ユーザ独自のクラスを実装するか、のいずれかの方法によります。独自のクラスを実装する場合は、`ConnectionSpec`、`ConnectionRequestInfo`、`Map`、および `Serializable` の 4 つのインタフェースを組み込む必要があります。

ConnectionSpec の実装

setConnectionSpec() または getConnectionSpec() を使用する前に、ConnectionSpec オブジェクトをインスタンス化する必要があります。ADK で用意されている ConnectionRequestInfoMap クラスを使用するか、またはユーザ独自のクラスを作成します。

ConnectionSpec を実装する手順は次のとおりです。

1. ADK で用意されている ConnectionRequestInfoMap クラスを使用するか、独自のクラスを実装するかを決定します。
2. 独自の ConnectionSpec クラスを実装する場合は、そのクラスに以下のインタフェースをインクルードしてください。
   - ConnectionSpec (JCA クラス)
   - ConnectionRequestInfo (JCA クラス)
   - Map (SDK クラス)
   - Serializable (SDK クラス)

setConnectionSpec() と getConnectionSpec() の呼び出し

ConnectionSpec クラスを実装して ConnectionSpec オブジェクトをインスタンス化したら、それらと次の ApplicationView メソッドと組み合わせて使用できます。

- setConnectionSpec()
- getConnectionSpec()

以下に示すのは setConnectionSpec() のコードです。

コード リスト 4-1 setConnectionSpec() のコード全文

```
/**
* EIS との接続に対して connectionSpec を設定する。ConnectionSpec
* の設定により、サービスの呼び出し時に、このクラスを使用して
* との接続が確立される。接続スペックをクリアしてデフォルトの
* 接続パラメータを使用するには、NULL 値を使用してこのメソッドを呼び出す。
*
* EIS の @params connectionCriteria 接続基準。
*/
public void setConnectionSpec(ConnectionSpec connectionCriteria)
{
m_connCriteria = connectionCriteria;
}
```

以下に示すのは getConnectionSpec() のコードです。

コード リスト 4-2 getConnectionSpec() のコード全文

```
/**
* setConnectionSpec によって設定された ConnectionSpec が返される。
* ConnectionSpec が設定されていなければ、NULL 値が返される。
*
* @returns ConnectionSpec
*/
public ConnectionSpec getConnectionSpec()
{
return m_connCriteria;
}
```

ConnectionSpec クラスの使い方

ConnectionSpec を設定するには、適切に初期化した ConnectionSpec オブジェクトを ConnectionSpec に渡します。ConnectionSpec を消去するには、NULL 値を持つ ConnectionSpec オブジェクトを ConnectionSpec に渡します。

コード リスト 4-3 は、ConnectionSpec の使用例を示しています。

コード リスト 4-3 ConnectionSpec クラスの使用例

```
Properties props = new Properties();
ApplicationView applicationView = new
ApplicationView(getInitialContext(props),"appViewTestSend");

ConnectionRequestInfoMap map = new ConnectionRequestInfoMap();
// プロパティをここにマップする
map.put("PropertyOne","valueOne");
map.put("PropertyTwo","valueTwo");
.
.
.
// 新しい接続スペックを設定する
applicationView.setConnectionSpec(map);

IDocumentDefinition requestDocumentDef =
applicationView.getRequestDocumentDefinition("serviceName");

SOMSchema requestSchema = requestDocumentDef.getDocumentSchema();

DefaultDocumentOptions options = new DefaultDocumentOptions();
options.setForceMinOccurs(1);
options.setRootName("ROOTNAME");
options.setTargetDocument(DocumentFactory.createDocument());
IDocument requestDocument = requestSchema.createDefaultDocument(options);

requestDocument.setStringInFirst("//ROOT/ElementOne","value");
requestDocument.setStringInFirst("//ROOT/ElementTwo","value");
.
.
.
// サービス呼び出しで、この接続スペック セットを使って EIS に接続する
IDocument result = applicationView.invokeService("serviceName",
requestDocument);
System.out.println(result.toXML());
```

シナリオ 2: ビジネス プロセスのカスタム コードの作成

ビジネス プロセスでアプリケーション ビューを使用する最も簡単なのは、WebLogic Integration Studio を使用する方法ですが、どのような場合でも代替方法としてビジネス プロセスを表すカスタム Java コードを作成できます。カスタム コードを作成する開発者は、この節で提示される、カスタム プロセスの作成方法を示す簡単な例を参照、検討してください。

アプリケーション ビューを使用するための 2 つの方法の詳細な比較については、1-8 ページの「ビジネス プロセスの実装方法の選択」を参照してください。

このシナリオについて

CRM (Customer Relationship Management: 顧客関係管理) システムと OP (Order Processing: 発注処理) システムを所有する企業があると想定します。経営陣は、CRM システムで顧客が登録されると、OP システムの対応する顧客レコードの登録が連動して確実に起動されるようにしたいと考えます。したがって、経営陣は開発者に対して、この 2 つのシステムで管理される情報が常に同期を保つようなビジネス プロセスを作成するように指示します。付属の Java クラス `SyncCustomerInformation` により、このビジネス ロジックを実装します。

この例で、カスタム コードを使用して実行できるすべての処理が説明できるわけではありません。この例は、あくまでオーガニゼーション独自のビジネス プロセスを実装する際の基本的な手順を紹介し、実際のビジネス プロセスのカスタム コードを作成するにあたってのテンプレートとして提示するためのものです。

このシナリオでは、`SyncCustomerInformation` という特定のクラス例を使用してカスタム コードの作成方法を説明します。通常、ビジネス プロセス内でアプリケーション ビューを使用するためのカスタム コードを作成するには、次の 2 つの手順に従う必要があります。

1. ビジネス プロセスを実装するアプリケーションを表現する Java クラスがあることを確認します。
2. その Java クラスの中に、ビジネス ロジックを実装するコードを定義します。

始める前に

カスタム コードを作成してビジネス プロセスを実装する前に、以下の前提条件を満たす必要があります。

- アプリケーション ビューを作成し、そのアプリケーション ビューの中に 1 つまたは複数のイベントまたはサービスを定義すること。
- 定義するビジネス プロセス ワークフローに必要なビジネス ロジックに関する情報を、適切なビジネス アナリストから入手すること。ホスト サーバ名とポート番号、ユーザ ID とパスワードなど、WebLogic Server との接続に必要な情報をすべて把握していること。

さらに、このシナリオでは以下の前提条件を満たしていることを仮定しています。

- ソース CRM システムおよび対象 OP システム用のアプリケーション ビューが定義されていて、正常に機能すること。アプリケーション ビューの定義方法については、2-1 ページの「アプリケーション ビューの定義」を参照してください。
- どちらのアプリケーション ビューも East Coast フォルダに配置されていること。ソース アプリケーション ビューの名前は「East Coast.Customer Mgmt」で、対象アプリケーション ビューの名前は「East Coast.Order Processing」です。

  注意： 実際のオーガニゼーションでは、独自のフォルダとアプリケーション ビューが使用されます。
- 操作者が、Application Integration API をよく理解しているか、またはこの API を理解している Java プログラマと共に作業すること。
- アプリケーション ビューが実装された Application Integration サーバと接続するために必要な情報が、すべて用意されていること。

  注意： 実際のオーガニゼーションの固有情報は、システム管理者から入手してください。

SyncCustomerInformation クラスの作成

カスタム コードの作成を始める前に、ビジネス クラスに必要な各アプリケーションを表現する Java クラスを準備する必要があります。必要な Java クラスがない場合は、ここでそれらを作成します。この例では、`SyncCustomerInformation` というアプリケーション クラスを呼び出します。もちろん、実際の独自コードでは異なる変数名を使用します。`SyncCustomerInformation` Java クラスを作成する手順は次のとおりです。

1. この Java アプリケーション クラスのソース コード全文については、4-9 ページの「サンプル Java クラスのコード」を参照してください。

   注意： 独自のプロジェクトを開始する場合、`SyncCustomerInformation` コードをテンプレートまたは参考ガイドとして使用してください。コード例の `SyncCustomerInformation` には詳細なコメントが付いています。

2. East Coast.New Customer をリスンするコードを作成します。

3. WebLogic Server 内で `NamespaceManager` (変数名 `m_namespaceMgr`) および `ApplicationViewManager` (変数名 `m_appViewMgr`) への参照を取得します。この手順を実行するには、WebLogic Server から JNDI ルックアップを使用します。

4. `NamespaceManager` を使って `nm.getRootNamespace()` を呼び出し、ルート ネームスペースの参照を取得します。この参照は、`root` という変数に格納されます。

5. `root` 変数を使用して、`root.getNamespaceObject( 摘 ast Coastî)` を呼び出し、East Coast ネームスペースへの参照を取得します。この参照は、`eastCoast` という変数に格納されます。

6. `eastCoast` 変数を使用して、顧客管理の `ApplicationView` への一時参照を取得し、それを `custMgmtHandle` という変数に格納します。

7. この `custMgmtHandle` 一時参照を使用して、顧客管理用の `ApplicationView` インスタンスへの参照を取得します。具体的には、`avm.getApplicationViewInstance (custMgmtHandle.getQualifiedName())` として、`ApplicationViewManager` を呼び出します。返された参照を `custMgmt` という変数に格納します。

8. `custMgmt.addEventListener(` 哲 `ew Customerî, listener)` を呼び出し、*`listener`* を New Customer イベントに応答できるオブジェクトで置換して New Customer のリスンを開始します（イベント リスナと `EventListener` インタフェースに関する詳細は、Application Integration API を参照）。

9. リスナ クラスの `onEvent` メソッドを実装します。

   New Customer イベントを受信すると、リスナの `onEvent` メソッドが呼び出されます。

   `onEvent` メソッドは、イベントに応答するメソッドを呼び出します。この例では、`onEvent` メソッドを使用して、そのイベントに関連するデータを格納するイベント オブジェクトを提供します。このイベントに応答するメソッドは、`handleNewCustomer` という名称になっています。

10. New Customer イベントに応答する `handleNewCustomer` メソッドを実装します。具体的には、次のアクション シーケンスを実装するコードを作成します。

    a. `handleNewCustomer` メソッドによって、イベント内で参照される XML ドキュメントを East Coast.Order Processing.Create Customer サービスから要求される形式に変換する。この変換処理は、XSLT を使用して実行するか、カスタム変換コードを使用して手動で実行します。変換後は、East Coast.Order Processing.Create Customer サービスのリクエスト ドキュメントに関するスキーマに準拠した XML ドキュメントになります。このドキュメントを `createCustomerRequest` という変数に格納します。

    b. `handleNewCustomer` は、East Coast.Order Processing アプリケーション ビューへの参照を、前述の East Coast.Customer Management アプリケーション ビューの場合と同様に取得する。この参照は、`orderProc` という変数に格納されます。

    c. `handleNewCustomer` が `orderProc.invokeService(` 鼎 `reate Customerî, createCustomerRequest)` を呼び出して、East Coast.Order Processing アプリケーション ビューの Create Customer サービスを呼び出す。`createCustomerRequest` は Create Customer サービスに対する要求ドキュメントを格納する変数です。このサービスに関する応答ドキュメントは、`createCustomerResponse` という変数に格納されます。

    d. `handleNewCustomer` は、実行を終了し、次の着信 New Customer イベントで使用可能になる。

この最後の手順が終了すると、SyncCustomerInformation という新しい Java クラスが作成されたことになります。このクラスにより Sync Customer Information ビジネス ロジックが実装されます。この SyncCustomerInformation クラスは、Application Integration API を使用して CRM システムからイベントを取得し、OP システム上でサービスを呼び出します。

サンプル Java クラスのコード

次のコード リストは、SyncCustomerInformation Java クラスのソース コード全文です。このコードにより、この節の前半で説明したシナリオに関するビジネス ロジックが実装されます。このサンプルを、実際の企業のビジネス プロセスを実装するコードを作成するときのテンプレートとして活用してください。

コード リスト 4-4 SyncCustomerInformation のクラス ソース コード全文

```
import java.util.Hashtable;
import javax.naming.*;
import java.rmi.RemoteException;
import com.bea.wlai.client.*;
import com.bea.wlai.common.*;
import com.bea.document.*;

/**
 * このクラスにより Sync Customer Information ビジネス プロセスのビジネス ロジックが
 * 実装される。このクラスは WLAI API を使用して CRM システムのイベントをリスンし、
 * OP システムのサービスを呼び出す。ここでは ApplicationViews が 2 つ定義されており、
 * それらが「EastCoast」ネームスペースにデプロイされていることを仮定している。
 * 2 つのアプリケーション ビュー、および必要なイベント、サービスは次のとおり。
 *
 *
 * CustomerManagement
 *   イベント（NewCustomer）
 *   サービス（なし）
 *
 * OrderProcessing
 *   イベント（なし）
 *   サービス（CreateCustomer）
 */
```

```
public class SyncCustomerInformation
  implements EventListener
{
  /**
   * このアプリケーションを起動する主要メソッド。引数は必要なし。
   */
  public static void
  main(String[ ] args)
  {
    // サーバ接続に必要な情報の有無をチェック。

    if (args.length != 3)
    {
      System.out.println("Usage: SyncCustomerInformation ");
      System.out.println("       <server url> <user id> <password>");
      return;
    }

    try
    {
      // 連携して動作する SyncCustomerInformation のインスタンスを作成

      SyncCustomerInformation syncCustInfo =
        new SyncCustomerInformation(args[0], args[1], args[2]);

      // WLAI への接続を取得

      InitialContext initialContext = syncCustInfo.getInitialContext();

      // 「EastCoast.CustomerManagement」のインスタンスへの参照を取得
      // アプリケーション ビュー

      ApplicationView custMgmt =
        new ApplicationView(initialContext, "EastCoast.CustomerManagement");

      // 「New Customer」イベントのリスナを追加する。この場合、
      // アプリケーション クラスに EventListener を実装し、直接イベントをリスン
      // できるようにする。

      custMgmt.addEventListener("NewCustomer", syncCustInfo);

      // 10 個までのイベントを処理してから終了する。

      syncCustInfo.setMaxEventCount(10);
      syncCustInfo.processEvents();
    }
    catch (Exception e)
    {
```

```
    e.printStackTrace();
  }

  return;


}

/**
 * 「New Customer」イベントに応答する EventListener メソッド
 */
public void
onEvent(IEvent newCustomerEvent)
{
  try
  {
    // 着信した「New Customer」イベントの内容を印刷する。

    System.out.println("Handling new customer: ");
    System.out.println(newCustomerEvent.toXML());

    // 処理する。

    IDocument response = handleNewCustomer(newCustomerEvent.getPayload());

    // 応答を印刷する。

    System.out.println("Response: ");
    System.out.println(response.toXML());

    // 処理対象のすべてのイベントを処理したら終了する。

    m_eventCount++;
    if (m_eventCount >= m_maxEventCount)
    {
      quit();
    }
  }
  catch (Exception e)
  {
    e.printStackTrace();
    System.out.println("Quitting...");
    quit();
  }
}

/**
 * 'Order Processing' ApplicationView で 「Create Customer」サービスを
```

```
 * 呼び出し、「New Customer」イベントを処理する。サービスから
 * 戻された応答ドキュメントが、このメソッドの戻り値として
 * 返される。
 */
public IDocument
handleNewCustomer(IDocument newCustomerData)
  throws Exception


{
  // 'OrderProcessing' ApplicationView のインスタンスを取得。
  if (m_orderProc == null)
  {
    m_orderProc =
      new ApplicationView(m_initialContext, "EastCoast.OrderProcessing");
  }

  // newCustomerData のデータを変換し、'Order Processing' ApplicationView
  // の「Create Customer」に対する要求ドキュメントに適する形式に
  // する。

  IDocument createCustomerRequest =
    transformNewCustomerToCreateCustomerRequest(newCustomerData);

  // サービスを呼び出す。

  IDocument createCustomerResponse =
    m_orderProc.invokeService("CreateCustomer", createCustomerRequest);

  // 応答を返す。

  return createCustomerResponse;
}

// ---------------------------------------------
// メンバ変数
// ---------------------------------------------

/**
 * WLAI サーバの URL （例 : t3://localhost:7001）
 */
private String m_url;

/**
 * WLAI にログインするときに使用されるユーザ ID。
 */
private String m_userID;
```

```
/**
 * m_userID で指定されたユーザが WLAI にログインするときに使用される
 * パスワード。
 */
private String m_password;


/**
 * WLAI との通信に使用される初期コンテキスト。
 */


private InitialContext m_initialContext;

/**
 * handleNewCustomer に使用される 'East Coast.Order Processing'
 * ApplicationView のインスタンス。
 */
private ApplicationView m_orderProc;

/**
 * handleNewCustomer で処理される最大イベント数が入る
 */
private int m_maxEventCount;

/**
 * handleNewCustomer で処理されるイベント数のカウント
 */
private int m_eventCount;

/**
 * 終了要求されるまで待機できるようにするモニタ変数。
 */
private String m_doneMonitor = new String("Done Monitor");

/**
 * 処理が終了したかどうかを示すフラグ。
 */
private boolean m_done = false;

// ------------------------------------------------
// ユーティリティ メソッド
// ------------------------------------------------

/**
 * コンストラクタ
 */
public SyncCustomerInformation(String url, String userID, String password)
```

```
{
  m_url = url;
  m_userID = userID;
  m_password = password;
}

/**
 * WLAI に対して初期コンテキストを設定する。
 */
public InitialContext
getInitialContext()
  throws NamingException
{


  // WLAI サーバの InitialContext を入手するためのプロパティを設定する。

  Hashtable props = new Hashtable();

  // WLAI ホスト、ポート、ユーザ ID、およびパスワードをプロパティに入力する。

  props.put(Context.INITIAL_CONTEXT_FACTORY,
            "weblogic.jndi.WLInitialContextFactory");
  props.put(Context.PROVIDER_URL, m_url);
  props.put(Context.SECURITY_PRINCIPAL, m_userID);
  props.put(Context.SECURITY_CREDENTIALS, m_password);

  // WLAI サーバに接続する。

  InitialContext initialContext = new InitialContext(props);

  // 後で使用できるようにこれを保存する。

  m_initialContext = initialContext;

  return initialContext;
}


/**
 * 「New Customer」イベントのドキュメントを「Create Customer」サービスで必要
 * とされるドキュメントに変換する。
 */
public IDocument
transformNewCustomerToCreateCustomerRequest(IDocument newCustomerData)
  throws Exception
{
  // ここでは XSLT 変換を実行しても、ソースから対象ドキュメントに手動で
```

```
  // データを移動してもよい。この変換の詳細は、このサンプル
  // の範囲外となっている。XSLT については、http://www.w3.org/TR/xslt
  // を参照のこと。ドキュメント間の手動によるデータ移動
  // についての詳細は、JavaDoc マニュアルの com.bea.document.IDocument
  // インタフェースを参照。

  return newCustomerData;
}

/**
 * イベント処理 / 待機のループ
 */
public void
processEvents()
{
  synchronized(m_doneMonitor)


  {
    while (!m_done)
    {
      try
      {
        m_doneMonitor.wait();
      }
      catch (Exception e)
      {
        // 無視する
      }
    }
  }
}

/**
 * 処理したい最大のイベント数を設定する。
 */
public void
setMaxEventCount(int maxEventCount)
{
  m_maxEventCount = maxEventCount;
}

/**
 * このアプリケーションを（完全に）強制終了するメソッド
 */
public void
quit()
{
```

```
    synchronized(m_doneMonitor)
    {
      m_done = true;
      m_doneMonitor.notifyAll();
    }
  }
}
```

5 Application View Console の使用方法

Application View Console は、企業内のあらゆるアプリケーション ビューへのアクセス、ビューの編成および編集が簡単に実行できるグラフィカル ユーザ インタフェース（GUI）です。Application View Console を使用すれば、フォルダを新しく作成し、それらのフォルダに新しいアプリケーション ビューを追加することができます。これらのフォルダにアプリケーション ビューを格納すれば、アプリケーション ビューが属するアダプタとは無関係に、ユーザ独自のナビゲーション方式に従ってアプリケーション ビューを編成できます。

この章では、以下のトピックを取り上げます。

- Application View Console へのログオン
- フォルダの作成
- アプリケーション ビューの削除
- フォルダの削除

Application View Console へのログオン

Application View Console にログオンする手順は次のとおりです。

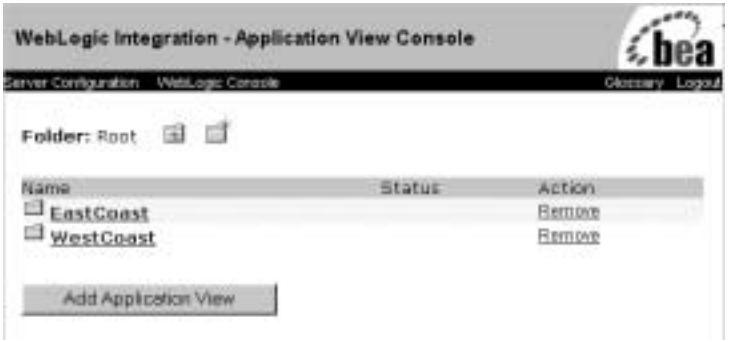
1. ブラウザ ウィンドウを開きます。
2. 該当するシステムの Application View Console の URL を次のフォーマットで入力します。

   `http://your_server:your_port/wlai`

   例：`http://wli1:7001/wlai`

   ログオン 画面が表示されます。

3. WebLogic Server ユーザ名およびパスワードを入力し、[OK] をクリックします。[Application View Console] が表示されます。



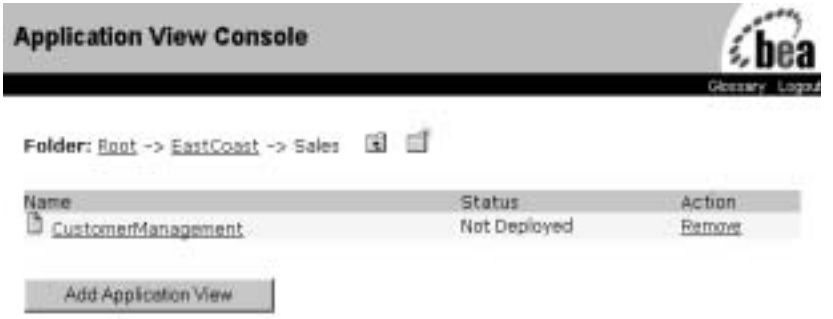
フォルダの作成

企業内のアプリケーション ビューは、フォルダに編成され、フォルダは、さらにアプリケーション ビューおよび従属フォルダで構成されます。一度フォルダを作成すると、そのフォルダを別のフォルダに移動することはできません。フォルダを削除するには、先にそのフォルダ内のすべてのアプリケーション ビューおよびサブフォルダを削除する必要があります。

フォルダ内にアプリケーション ビューを作成した後でそのアプリケーション ビューを削除できますが、別のフォルダには移動できません。

フォルダを作成する手順は次のとおりです。

1. [Application View Console] にログオンした状態で、新しいフォルダの作成先となるフォルダに移動します。



2. [New Folder] アイコンをクリックします。

[Add Folder] 画面が表示されます。

3. [New Folder] フィールドに、フォルダ名を入力します。名前には、任意の有効な Java 識別子が使用できます。

   注意：「Root」という名前は予約語であるため、フォルダ名には使用できません。フォルダ名に「Root」を指定すると、インポート / エクスポート ユーティリティを使ってフォルダをインポートまたはエクスポートできなくなります。

4. [Save] をクリックします。

アプリケーション ビューの削除

アプリケーション ビューを削除するのは、そのビューが使われなくなったか、またはビューが属するアプリケーションを破棄するときです。

アプリケーション ビューを削除できるのは、以下の条件が 2 つとも満たされる場合に限られます。

- アプリケーション ビューがアンデプロイされていること（2-20 ページの「任意手順 : アプリケーション ビューのアンデプロイ」を参照）。アプリケーション ビューのステータスが Not Deployed となっていることを確認してください。
- 適切な書き込み特権を持つユーザ アカウントを使用して WebLogic Server にログオンしていること。

アプリケーション ビューを削除する手順は次のとおりです。

1. [Application View Console] にログオンした状態で対象のアプリケーション ビューが格納されているフォルダに移動します。

2. [Remove] をクリックします。確認画面が表示されます。[Confirm] をクリックすると、アプリケーション ビューが削除されます。

フォルダの削除

不要なフォルダは削除します。フォルダを削除する手順は次のとおりです。

1. 対象フォルダからすべてのアプリケーション ビューとサブフォルダを削除します。
2. [Application View Console] にログオンし、対象フォルダがあるフォルダに移動します。

3. [Remove] をクリックします。確認画面が表示されます。

4. [Confirm] をクリックします。フォルダが削除されます。

A Application Integration データの移行

この章では、以下のトピックを取り上げます。

- データ移行の概要
- 単独の EIS インスタンス内でのデータの移行
- 複数の EIS インスタンス内でのデータの移行
- 推奨事項

データ移行の概要

Application Integration 機能のコンフィグレーション データは、Business Process Management (BPM) 機能のコンフィグレーション データと同じリポジトリに格納されます。したがって、どちらの機能のデータ移行にも同じツールを使用できます。ただし、Application Integration データを移行し、およびそのデータを対象環境にデプロイする場合、特に考慮すべき点がいくつかあります。

注意： 現行の Application Integration エンジンでは、モジュラ デプロイメントをサポートしています。この方式では、EAR (Enterprise Application Archive) ファイルにまとめられ、有効なすべての WebLogic ドメインで使用できます。BPM 機能は必ずインストールされているとは限りませんので、アプリケーション ビューに対してはインポート / エクスポート ツールのコマンドライン インタフェースが用意されています。詳細については、付録 B「アプリケーション ビューのインポートおよびエクスポート」を参照してください。

Application Integration データの移行は、単独の EIS (Enterprise Information System : エンタープライズ情報システム) インスタンス内の WebLogic Server ドメイン間で行う場合は簡単にできます。異なる EIS インスタンス内の WebLogic

Server ドメイン間でアプリケーション データを移行する場合、移行対象の環境でソリューションが正常に機能するためには特別な手順を実行する必要があります。

この節では、以下のシナリオを使用して WebLogic Server ドメイン間で Application Integration データを移行する方法について説明します。

- 単独の EIS インスタンス内でのデータの移行
- 複数の EIS インスタンス内でのデータの移行

単独の EIS インスタンス内でのデータの移行

この節では、単独の EIS インスタンス内にある複数のドメイン間で Application Integration データを移行する方法について説明します。たとえば、WebLogic Integration の異なるドメインに対するリポジトリ間でアプリケーション ビュー定義を移動するとします。この場合は WebLogic Integration ドメインだけが変更され、アプリケーション ビュー内で参照される対象の EIS インスタンスは同じです。

この場合、ワークフロー パッケージのインポート / エクスポート ユーティリティ（WebLogic Integration Studio から利用できる）を使えば、データの移行は簡単にできます。ソース ドメイン内の Studio からパッケージをエクスポートし、そのパッケージを対象ドメイン内の Studio にインポートします。

インポート / エクスポート ユーティリティの詳細は、『*WebLogic Integration Studio ユーザーズ ガイド*』の「ワークフロー パッケージのインポートとエクスポート」を参照してください。

アプリケーション ビューのエクスポート

Application Integration 機能を含むワークフローをエクスポートするとき、そのワークフローで使用されるアプリケーション ビューおよびその他のリソースはエクスポート ツールによって自動的に識別されます。コード リスト A-1 およびコード リスト A-2 では、エクスポート ツール内で識別されるアプリケーション ビューと、そのビューで使用されるリソースの値を示します。通常、アプリケーション ビューの値は、コード リスト A-1 で示された位置に表示（エクスポート ツール内）されます。

リスト A-1　BPM エクスポート ツール内でのアプリケーション ビューの位置

```
All Workflow Objects
  |--  XML Repository
    |-- Folder: WLAI.Namespace.Root
      |-- Folder: WLAI.Namespace.Root.firstfolder
        |-- Folder: WLAI.Namespace.Root.firstfolder.nthfolder
          |-- Entity: WLAI.ApplicationView.Root.firstfolder.
                nthfolder.appviewname
```

通常、アプリケーション ビューに関連するエンティティは *nth_folder* の下に表示され、コード リスト A-2 に示す規約に従って名前が付けられます。ただし、すべてのアプリケーション ビューがこの規約に従うわけではありません。

リスト A-2　BPM エクスポート ツール内でのアプリケーション ビュー リソースの位置

```
Entity: WLAI.entity_type.Root.firstfolder.nthfolder.appviewname_
event/servicename_adapterspecific
```

アプリケーション ビューを完全にエクスポートするには、タイプが Schema および ConnectionFactory のエンティティも含め、そのビューに関連するすべてのエンティティを*選択しなければなりません*。

アプリケーション ビューのエクスポート例

EastCoast.Sales フォルダに CustomerManagement というアプリケーション ビューがあり、BPM エクスポート ツールの中では、コード リスト A-3 に示す位置に表示されています。

リスト A-3 BPM エクスポート ツール内でのアプリケーション ビュー

```
All Workflow Objects
  |--  XML Repository
    |-- Folder: WLAI.Namespace.Root
      |-- Folder: WLAI.Namespace.Root.EastCoast
        |-- Folder: WLAI.Namespace.Root.EastCoast.Sales
          |-- Entity:WLAI.ApplicationView.Root.EastCoast.
                   Sales.CustomerManagement
```

CustomerManagement アプリケーション ビューを完全にエクスポートするため、エクスポート ツールでは次のパターンに一致するすべてのエンティティが自動的に選択されます。

```
Entity: WLAI.entitytype.Root.EastCoast.Sales.CustomerManagement
```

CustomerManagement アプリケーション ビューには、いくつかのイベントおよびサービスが含まれています。エクスポート ユーティリティによれば、イベントごとに 1 つの Schema タイプのエンティティと、サービスごとに 2 つの Schema タイプのエンティティが示されています。また、CustomerManagement アプリケーション ビューは、DBMS アダプタも使用しており、CustomerCreated という名前のイベントと CreateCustomer という名前のサービスが 1 つずつあります。したがって、コード リスト A-4 に示すエンティティが、エクスポート ユーティリティによってリストされています。

リスト A-4 アプリケーション ビューによって使用されるエンティティ

```
Entity: WLAI.Schema.Root.EastCoast.Sales.CustomerManagement_CustomerCreated_
CUSTOMER_TABLE_insert

Entity: WLAI.Schema.Root.EastCoast.Sales.CustomerManagement_CreateCustomer_input
```

```
Entity: WLAI.Schema.Root.EastCoast.Sales.CustomerManagement_
CreateCustomer_output
```

CustomerManagement アプリケーション ビューには、接続ファクトリも 1 つ定義されています。この接続ファクトリのエンティティ名は次のとおりです。

```
Entity: WLAI.ConnectionFactory.Root.EastCoast.Sales.CustomerManagement.
ConnectionFactory
```

CustomerManagement アプリケーション ビューが正しくエクスポートされるためには、コード リスト A-4 に示されたすべてのエンティティを選択する必要があります。

アプリケーション ビューのインポート

アプリケーション ビューをインポートする手順は、次のとおりです。

1. ワークフロー パッケージのインポート ユーティリティ（WebLogic Integration Studio から利用できる）を使用して、Application Integration データを含むパッケージをインポートします。このユーティリティにより、あらかじめパッケージ内にエクスポートしたエンティティのすべてが自動的にインポートされます。
2. Application Integration Application View Console (通常は `http://server:port/wlai` にある) を使用して、インポートしたアプリケーション ビューをデプロイします。
3. インポートしたフォルダ内を移動し、インポートしたアプリケーション ビューを探します。
4. 適切なアプリケーション ビューを選択します。次に、[Application View Summary] 画面の [Deploy] オプションを選択します。

これで、アプリケーション ビューが対象環境で使用できるようになります。

複数の EIS インスタンス内でのデータの移行

複数インスタンスの EIS 間の WebLogic Server ドメインどうしでデータを移行する場合、特定の EIS インスタンスに定義されたアプリケーション ビューには、そのインスタンス固有の識別子その他のデータが含まれているため注意が必要です。これは、アプリケーション ビューで使用される接続ファクトリについても当てはまります。

起こり得るエラーを防ぐには、アプリケーション ビューまたは接続ファクトリ内にある EIS インスタンス固有のデータを、次の手順に従って手動で変更する必要があります。

1. Application View Console を開きます。
2. 該当するアプリケーション ビューに移動して、必要に応じて編集します。
   a. アプリケーション ビューとそのイベント、サービス、および接続ファクトリ内にある EIS 固有のデータをすべて見つけ出し、更新する。
   b. EIS インスタンス固有の参照を検索し、移行対象の環境の新しい EIS インスタンスへの参照に置き換える。

アプリケーション ビュー定義と接続ファクトリ定義を確実に編集してください。アプリケーション ビュー定義の以下のパラメータが変更対象です。

- `ApplicationView` デプロイ画面の `EventRouterURL` パラメータは対象環境のイベント ルータを参照する必要があります。
- サービス定義内のパラメータ。これはアダプタ固有のデータで、EIS インスタンス固有のデータを参照している可能性があります。サービスに関する EIS インスタンス固有のパラメータを必要に応じて変更します。
- イベント定義内のパラメータ。これはアダプタ固有のデータで、EIS インスタンス固有のデータを参照している可能性があります。サービスに関する EIS インスタンス固有のパラメータがある場合は、[Application View Summary] 画面の [Edit] 機能を使用して変更します。

アプリケーション ビューのインポート例

CustomerManagement の例では、ソース環境に CUST というデータベースがあり、対象環境には CustDB というデータベースがあります。コード リスト A-5 は、アプリケーション ビューと接続ファクトリを表現する XML テキストを示します。実際に示している内容は、CustomerManagement アプリケーション ビューに関するアプリケーション ビュー記述子です。Application View Console を使用する場合は、設計時 UI フォーム内の該当するフィールドを使用して、この情報を表示し、変更する必要があります。

リスト A-5　アプリケーション ビューと接続ファクトリに関する XML テキストの例

```
<?xml version="1.0"?>
<!DOCTYPE applicationView>
<applicationView asyncEnabled="true"
connectionFactory="com.bea.wlai.connectionFactories.EastCoast.Sales.
CustomerManagement_connectionFactoryInstance"
connectionFactoryName="EastCoast.Sales.CustomerManagement_connectionFactory"
eventRouterURL="http://localhost:7001/DbmsEventRouter/EventRouter"
      name="CustomerManagement"
      ownsConnectionFactory="true">
  <description>Manages customers in the east coast sales database</description>

  Ö

  <service interactionSpecClass="com.bea.adapter.dbms.cci.InteractionSpecImpl"
      name="CreateCustomer"
      ownsRequestSchema="true"
      ownsResponseSchema="true"
requestDocumentType="EastCoast.Sales.CustomerManagement_CreateCustomer_input/In
put"
responseDocumentType="EastCoast.Sales.CustomerManagement_CreateCustomer_output/
RowsAffected">
   <description>create a new customer in database</description>
   <interactionSpecProperty
name="functionName">executeUpdate</interactionSpecProperty>
   <interactionSpecProperty name="sql">insert into CUST.dbo.CUSTOMER_TABLE
(FirstName, LastName, DOB) values ([FirstName varchar], [LastName varchar], [DOB
timestamp])</interactionSpecProperty>
   </service>

   <event name="CustomerCreated"
```

```
      ownsSchema="true"
      rootElementName="CUSTOMER_TABLE.insert"

schemaName="EastCoast.Sales.CustomerManagement_CustomerCreated_CUSTOMER_TABLE_i
nsert">
   <description>New customer created in database</description>
   <eventProperty name="tableName">CUSTOMER_TABLE</eventProperty>
   <eventProperty name="triggerType">insert</eventProperty>
   <eventProperty name="catalogName">CUST</eventProperty>
   <eventProperty name="schemaName">dbo</eventProperty>
  </event>

</applicationView>
```

このアプリケーション ビューには、以下の要素が定義されています。

- イベント ルータ URL に対する明示的参照（この URL は、EIS インスタンスを変更すると対象ドメインでは元のドメインの URL とは異なると思われる）。
- CUST データベース、dbo スキーマ、および CUSTOMER_TABLE テーブルを参照する明示的な SQL 文が指定された interactionSpecProperty 要素（<service> 要素として表記）。
- catalogName を CUST として参照し、schemaName を dbo として参照する eventProperty 要素。この例では、CUST に対するすべての参照（上記のテキストで強調表示されている部分）を CustDB に変更する必要があります。異なるスキーマが使用されている場合は、そのスキーマ参照も変更する必要があります。

各アダプタに作成されるアプリケーション ビュー記述子のサービスおよびイベント記述子には、異なるプロパティが格納されます。新しい EIS インスタンスで正常に機能するためにはどのプロパティを変更する必要があるかについては、アダプタのマニュアルを参照してください。

接続ファクトリ記述子も、新しいインスタンスを参照するように変更する必要があります。コード リスト A-6 は、接続ファクトリの例を示しています。

リスト A-6　接続ファクトリの例

```
<?xml version="1.0"?>
<!DOCTYPE connection-factory-dd>
<connection-factory-dd name="CustomerManagement_connectionFactory">
  <jndi-name>com.bea.wlai.connectionFactories.EastCoast.Sales.CustomerManagemen
t_connectionFactoryInstance</jndi-name>
  <pool-parms allowPoolToShrink="true"
      maxPoolSize="10"
      minPoolSize="0"
      targetFractionOfMaxPoolSize="0.1"/>
  <mcf-parm name="MessageBundleBase">
    <mcf-parm-value>BEA_WLS_DBMS_ADK</mcf-parm-value>
  </mcf-parm>
  <mcf-parm name="DataSourceName">
    <mcf-parm-value>eventSource</mcf-parm-value>
  </mcf-parm>
  <mcf-parm name="AdditionalLogContext">
    <mcf-parm-value>CustomerManagement</mcf-parm-value>
  </mcf-parm>
  <mcf-parm name="UserName">
    <mcf-parm-value>system</mcf-parm-value>
  </mcf-parm>
  <mcf-parm name="Password">
    <mcf-parm-value>security</mcf-parm-value>
  </mcf-parm>
  <mcf-parm name="RootLogContext">
    <mcf-parm-value>BEA_WLS_DBMS_ADK</mcf-parm-value>
  </mcf-parm>
  <mcf-parm name="PingTable">
    <mcf-parm-value>CUST.dbo.CUSTOMER_TABLE</mcf-parm-value>
  </mcf-parm>
  <mcf-parm name="LogLevel">
    <mcf-parm-value>WARN</mcf-parm-value>
  </mcf-parm>
  <mcf-parm name="LogConfigFile">
    <mcf-parm-value>BEA_WLS_DBMS_ADK.xml</mcf-parm-value>
  </mcf-parm>
  <adapter-logical-name>BEA_WLS_DBMS_ADK</adapter-logical-name>
</connection-factory-dd>
```

上記のコードに示されているように、この接続ファクトリ記述子は直接 CUST データベースと eventSource という JDBC データ ソースを参照しています。対象環境でこの接続ファクトリが確実に正しく機能するためには、以下の変更を行う必要があります。

- CUST への参照を CustDB への参照に変更する。
- eventSource JDBC データ ソース参照を変更して、対象ドメインの有効な JDBC データ ソース（CustDB のホストとなっている新しい DBMS）を参照するようにする。

この時点で、必要なすべての参照の修正が完了し、アプリケーション ビューと接続ファクトリで必要とされるすべてのリソースが対象ドメイン内に定義されたことになります。これで、Application View Console (通常は http://*host*:*port*/wlai にあります) を使用して、インポートしたアプリケーション ビューをすべてデプロイできます。

推奨事項

ここでは、複数の環境間で Application Integration データを移行する際に必要となる作業を軽減するためのアドバイスを紹介します。

- ソースドメインと対象ドメインでは、できる限り同じ EIS インスタンスを設定すること。たとえば、DBMS アダプタが使用されるアプリケーション ビューの場合、同じ MS SQL Server の 2 つのインスタンス（同じ名前、ユーザ アカウント、およびデータベース オブジェクト）を、ソース データベースと対象データベースの両方に使用します。このように設定すると、アプリケーション ビューおよび接続ファクトリの記述子を手動で編集する必要がなくなります。
- イベント ルータ URL を変更して、対象環境でイベント ルータの位置を反映させること。そのためには、Application View Console でアプリケーション ビューを編集します。
- アプリケーション ビューをインポートしたら、それらのビューを Application View Console を使ってデプロイします。

B　アプリケーション ビューのインポートおよびエクスポート

この章では、以下のトピックを取り上げます。

- ユーティリティのインポート / エクスポート
- インポート / エクスポート メソッドおよびコマンド ライン

ユーティリティのインポート / エクスポート

WebLogic Integration では、アプリケーション ビューの簡易インポート / エクスポート ユーティリティを提供しています。このユーティリティは、コマンド ラインから実行でき、アプリケーション ビューのインポート / エクスポート API によってコードに組み込むことができます。このユーティリティの出力は、当該アプリケーション ビューが所有するすべてのファイルを含む JAR ファイルになります。使用されるが、アプリケーション ビューが所有していないファイルをインポートまたはエクスポートする場合は、手動操作になります。

注意： WebLogic Integration の Business Process Management (BPM) 機能がインストールされている場合、ワークフロー パッケージのインポート / エクスポート ユーティリティ（WebLogic Integration Studio から利用できる）を使用して Application Integration データを移行することができます。詳細については、付録 A「Application Integration データの移行」を参照してください。

インポート / エクスポート ユーティリティを使用すると、リポジトリから Application Integration メタデータ オブジェクトをエクスポートし、またそれらのオブジェクトをリポジトリにインポートすることができます。このユーティリティでは、次のオブジェクトをインポート / エクスポートできます。

- アプリケーション ビュー
- 接続ファクトリ（およびリポジトリに格納された記述子。WebLogic Server からデプロイされたものは除く）
- スキーマ
- ネームスペース

ユーティリティの呼び出しでエクスポートされたオブジェクトは、JAR アーカイブに格納されます。従前にエクスポートされた JAR がインポートされる場合、その JAR 内のオブジェクトもすべてインポートされます。また、既存のエクスポートされた JAR にオブジェクトを追加すること、インポート時にアプリケーション ビューおよび接続ファクトリをデプロイすることもできます。

インポート / エクスポート メソッドおよびコマンド ライン

ユーティリティは、API およびコマンドライン ツールとして使用できます。どちらの場合も、サーバが稼働している必要があります。以下の節では、コマンドライン パラメータおよびインポート / エクスポート ユーティリティのメソッドについて説明します。

コマンド ラインからのインポート / エクスポート ユーティリティの呼び出し

以下に示すのは、インポート / エクスポート ユーティリティのコマンドライン構文です。

```
使い方 : ImportExport <server_URL> <username> <password> <file>
[-codepage=Cp<codepage_number>] [-dump=< <namespace> | <'Root'> >]
[-append] [-overwrite] [-deploy]
< [-export [object_name]*] | [-import [edit-on-import_filename]] >
```

次の表では、インポート / エクスポート ユーティリティのコマンドライン パラメータを示しています。

パラメータ	説明
server_URL	WebLogic Server の URL。これが必要なのは、リモート クライアントから接続する場合に限る。
username	指定された WebLogic Server のユーザ名。これが必要なのは、リモート クライアントから接続する場合に限る。
password	指定された WebLogic Server のパスワード。これが必要なのは、リモート クライアントから接続する場合に限る。
file	エクスポート時に作成されるファイル、またはリポジトリにインポートされるファイルの名前。
-codepage	コンソールに書き込むときに使用されるコード ページを設定する。この設定により、文字が正しく表示されることが保証される。デフォルト値は Cp437 で、これは米国で使用されるコード ページになっている。他の有効な値は、以下のとおり。 Cp850 マルチリンガル (ラテン I) Cp852 スラブ語系（ラテン II） Cp855 キリル文字（ロシア語） Cp857 トルコ語 Cp860 ポルトガル語 Cp861 アイスランド語 Cp863 カナダ フランス語 Cp865 北欧語 Cp866 ロシア語 Cp869 現代ギリシャ語 MS932 日本語
-dump	指定されたネームスペース、およびそのネームスペース内にネストされた他のネームスペース内のすべてのオブジェクトのリストを印刷する。フォルダ構成全体のオブジェクトのリストを印刷する場合は、Root と指定する。
-append	*file* で指定されたファイルにエクスポートされたオブジェクトを追加する。

パラメータ	説明
-overwrite	インポートの実行時に、既にリポジトリにあるオブジェクトを上書きする。
-deploy	インポート時にアプリケーション ビューまたは接続ファクトリをデプロイする。
-export	エクスポート操作を指示し、エクスポート対象のオブジェクト名を指定する。オブジェクト名にはワイルドカードも使用できる。フォルダ構成全体をエクスポートする場合は、オブジェクト名のリストに Root を入れる。
-import	*file* で指定されたオブジェクトをリポジトリにインポートすることを指示する。edit-on-import_filename が指定されると、オブジェクトはファイル内の命令に従って編集される。編集は、オブジェクトがリポジトリに追加される前、かつデプロイ（要求された場合）の前に実行される。

インポート時の編集

インポート時編集ファイルが指定されると、インポートされるオブジェクトのテキストに対して編集コマンドを実行できます。編集処理は、オブジェクトがリポジトリに格納される、またはデプロイされる前に実行されます。このオプションが指定されない場合、このメソッドは「オブジェクトのインポート」の説明にあるように使用されます。

指定されるドキュメントは、次の DTD に準拠する必要があります。

```
<!ELEMENT ApplicationView (replace*)>
<!ATTLIST ApplicationView name NMTOKEN #REQUIRED
                          newName NMTOKEN #IMPLIED>

<!ELEMENT ConnectionFactory (replace*)>
<!ATTLIST ConnectionFactory name NMTOKEN #REQUIRED
                            newName NMTOKEN #IMPLIED>

<!ELEMENT Schema (replace*)>
<!ATTLIST Schema name NMTOKEN #REQUIRED
                 newName NMTOKEN #IMPLIED>

<!ELEMENT replace EMPTY>
<!ATTLIST replace xpath CDATA #REQUIRED
```

```
old CDATA #REQUIRED
new CDATA #REQUIRED
```

インポート時編集ドキュメントには、編集対象のオブジェクトごとに <ELEMENT> タグで示される複数のセクションが組み込まれています。編集できるオブジェクトは、ApplicationView、ConnectionFactory および Schema です。各セクションでは、名前によってオブジェクトを識別し、置換される要素を指定します。オプションとして、各セクションでオブジェクトに新しい名前を設定するときに使用される newName 属性を指定できます。

replace 要素では、以下を指定します。

- オブジェクトの XML 記述子にあるすべてのノードの中から編集対象のノードを識別するために使用される XPath 式。
- 選択されたノード内で検索する元の値。元の値は Perl5 の正規表現にすることができます。元の値として空の文字列（""）を指定すると、選択されたノード内のすべてのテキストが一致となります。
- 元の値を置換する新しい値。新しい値は単純文字列でなければなりません。

インポート時編集ドキュメントの例

次のインポート時編集記述子ドキュメントは、DBMS.DBMS1 という名前のアプリケーション ビューを編集して、DBMS.DBMS1a という名前に変更する方法を示しています。この XML ドキュメントでは、アプリケーション ビューに対して 3 つの置換、接続ファクトリに対して 1 つの置換を設定しています。

```
<?xml version="1.0"?>
<!DOCTYPE edit SYSTEM "ImportExportEditOnImport.dtd">
<edit>
   <ApplicationView name="DBMS.DBMS1" newName="DBMS.DBMS1a">
      <replace xpath="/applicationView/@connectionFactory"
         old=""
         new="com.bea.wlai.connectionFactories.DBMS.
              DBMS1a_connectionFactoryInstance"/>
      <replace xpath="/applicationView/@connectionFactoryName"
         old=""
         new="DBMS.DBMS1a_connectionFactory"/>
      <replace        xpath="/applicationView/service[@name='Service1']/
       interactionSpecProperty[@name='sql']"
         old="CAJUN."
         new="PBPUBLIC."/>
   </ApplicationView>
```

```
   <ConnectionFactory name="DBMS.DBMS1_connectionFactory"
    newName="DBMS.DBMS1a_connectionFactory">
      <replace xpath="/connection-factory-dd/jndi-name"
         old=""
         new="com.bea.wlai.connectionFactories.DBMS.
              DBMS1a_connectionFactoryInstance"/>
   </ConnectionFactory>
</edit>
```

最初の置換では、アプリケーション ビュー記述子の connectionFactory 属性を編集して、この属性のテキストを新しい値に変更します。ここでは、元の値として空の文字列を使用して、ノード内のすべてのテキストを一致させています。

2 番目の置換では、アプリケーション ビュー記述子の connectionFactoryName 属性を編集して、この属性のテキストを新しい値に変更します。

3 番目の置換では、Service1 という名前のサービスのサービス記述子を編集して、sql という名前の interactionSpecProperty 要素について、CAJUN. をすべて PBPUBLIC. に置換します。

4 番目の置換では、接続ファクトリ記述子の jndi-name 属性を編集して、この属性のテキストを新しい値に変更します。

インポート / エクスポート API の使用方法

以下の節では、インポート / エクスポート API に組み込まれているメソッドを説明します。インポート / エクスポート API のクラス名は、com.bea.wlai.client.ImportExport です。

サーバ インスタンスへの接続

connect(*<multiple signatures>*)

connect() メソッドは、サーバ インスタンスとの接続方法を設定します。接続を開始した場所によって、異なる引数を connect() に指定する必要があります。

接続の開始場所	指定する引数
同じサーバ内	引数なし

接続の開始場所	指定する引数
リモート クライアント（InitialContext なし）	URL (文字列)、ユーザ名、パスワード
リモート クライアント（InitialContext あり）	InitialContext

ネームスペース内のオブジェクトの印刷

```
dumpNamespace(String namespaceName)
```

dumpNamespace() メソッドは、ネームスペースの修飾名を表す文字列を取り、そのネームスペース内のすべてのオブジェクト、およびそのネームスペースに組み込まれた他のネームスペースを印刷します。

オブジェクトのエクスポート

```
exportNamespaceObjects(Set objectNames, boolean append, List errors)
```

exportNamespaceObjects() メソッドは、オブジェクト名のリスト（文字列による修飾名）を取り、指定された出力ファイルにエクスポートします（「インポート / エクスポートするファイルの指定」を参照）。エクスポート対象としてリストとされたすべてのオブジェクトは、依存関係が検査されます。使用されているがアプリケーション ビューの所有になっていないオブジェクトもエクスポートされます。

アプリケーション ビューがエクスポートされる場合、ConnectionFactory およびそのアプリケーション ビューが依存するすべての Schema オブジェクトもエクスポートされます。ネームスペースがエクスポートされる場合は、ネームスペース内のオブジェクト（他のネームスペースも含む）もすべてエクスポートされます。

エクスポートされたファイルを既存のアーカイブに追加する、またはそのファイルを上書きないし作成する場合は、append を true に設定します。

エクスポート中に検出された致命的でないエラーは、すべて指定された errors List オブジェクトに Exception オブジェクトとして格納されます。

オブジェクトのインポート

```
importNamespaceObjects(boolean overwrite, boolean deploy, List errors)
```

importNamespaceObjects() メソッドは、指定された入力ファイル（「インポート / エクスポートするファイルの指定」を参照）のすべてのエントリを取り、それらをリポジトリにインポートします。overwrite を true に設定すると、リポジトリ内の既存のメタデータが上書きされます。deploy を true に設定すると接続ファクトリおよびアプリケーション ビューがインポート時にデプロイされます。

インポート中に検出された致命的でないエラーは、すべて指定された errors List オブジェクトに Exception オブジェクトとして格納されます。

オブジェクトをインポートして編集する

```
importNamespaceObjects(IDocument editOnImportDoc boolean overwrite,
boolean deploy, List errors)
```

インポート時編集ドキュメントが指定されると、importNamespaceObjects() で、インポートされるオブジェクトのテキストに対して編集コマンドを実行できます。編集処理は、オブジェクトがリポジトリに格納される、またはデプロイされる前に実行されます。このオプションが指定されない場合、このメソッドは「オブジェクトのインポート」の説明にあるように使用されます。

editOnImportDoc 引数で指定されたドキュメントは、「インポート時の編集」に示された DTD に準拠する必要があります。

インポート / エクスポートするファイルの指定

```
setFile(File filename)
```

setFile() メソッドは、エクスポート先またはインポート元として使用されるファイルを指定します。

メッセージ印刷場所の選択

```
setPrintWriter(PrintWriter out)
```

`setPrintWriter()` メソッドは、メッセージ（ステータス、診断、およびエラーメッセージなど）が生成されたときに使用される PrintWriter オブジェクトを指定します。

メッセージ印刷の有無の選択

```
setQuiet(boolean quiet)
```

`setQuiet()` メソッドは、進行状況メッセージおよび情報メッセージを印刷するかどうかを指定します。メッセージを印刷する場合は、`quiet` を `false` に設定します。メッセージ印刷を無効にする場合は、`quiet` を `true` に設定します。

C Application Integration のモジュラ デプロイメント

この章では、以下のトピックを取り上げます。

- 概要
- デプロイメント コンポーネント
- WebLogic Integration 環境外のドメインに対するデプロイメント コンフィグレーション
- JMS リソース

概要

WebLogic Integration 7.0 以前の環境では、アプリケーション ビューは、通常、Business Process Management (BPM) ワークフローによって作成、管理されます。しかし、リリース 7.0 環境では、WebLogic Workshop を使用する Web サービス開発者は、エンタープライズ情報システムにアクセスするアプリケーション ビューを使用する必要があります。WebLogic Integration の Application Integration 機能は、EAR (Enterprise Application Archive) ファイルにパッケージ化され、有効なすべての WebLogic ドメインで使用できます。以下の節では、モジュラ デプロイメントに必要なデプロイメント プロセスの変更点を説明します。デプロイメント概念の詳細については、『*WebLogic Integration ソリューションのデプロイメント*』を参照してください。

クラスパスの変更とサーバの再起動

wlai-core.jar ファイルを、WebLogic Integration ベースでないドメインのクラスパスに追加する必要があります。クラスパスの変更を実装するには、追加後にサーバを再起動する必要があります。

リポジトリ

Application Integration エンジンでは、メタデータの永続性を保つためにリポジトリを使用しています。リポジトリに関して Application Integration エンジンでは、あらかじめコンフィグレーションされたリポジトリおよび関連する JDBC 接続プールとデータ ソースに依存しています。モジュラ デプロイメントでは、デプロイ時に JDBC データ ソース名と資格を指定する必要があります。Application Integration エンジンでは、リポジトリは既にデータ ソースにインストールされていることを想定しています。

JMS リソース

アプリケーション ビューでは JMS リソースを使って、イベントおよび非同期サービス呼び出しを処理しています。こうした機能をサポートするため、アプリケーション ビューでは、リソース JMSConnectionFactory、JMSTemplate、JMSJDBCStore、および JMSServer を使用します。さらに、Application Integration エンジンでは、非同期サービス呼び出し処理用の要求および応答キューを定義しています。使用する JMS リソースの決定には、2 つの処理方式が可能です。

- 以下のすべてのリソースに対して、既存の JMS リソースの名前を指定する。
  - JMSConnectionFactory
  - JMSServer

  注意： JMSServer に名前を指定した場合、同時に JMSJDBCStore をコンフィグレーションする必要があります。
- 既存の JMS リソースの名前を指定しない。この場合、始動プロセスで JMS MBeans を使って必要な JMS リソースを定義します。Application Integration

エンジンで使用されるリソースの全リストは、「JMS リソース」に記載されています。

コンフィグレーション

Application Integration の起動クラスおよびシャットダウン クラスは、`StartupBean` という名前の EJB (Enterprise JavaBean) に変わっています。この EJB は `wlai-server-ejb.jar` からデプロイされ、起動時にロードされます。起動時、この EJB によって Application Integration エンジンの初期化シーケンスが開始されます。`StartupBean` EJB の初期パラメータは、Application Integration エンジンのコンフィグレーション パラメータとして機能します。これらのパラメータは、WebLogic Administration Console に付属の標準 EJB 記述子編集ツールを使用して設定できます。次の表は、コンフィグレーション パラメータを一覧にしたものです。

プロパティ名	デフォルト値	説明
`wlai.logLevel`	`warning`	Application Integration ログの冗長性レベル。
`wlai.deploymentRepositoryRootPath`	*`$PWD`*`/wlai/deploy`。`$PWD` は、WebLogic Server の現在の作業ディレクトリ	Application Integration エンジンが接続ファクトリ デプロイメント記述子を保存する場所。
`wlai.hostUserID`	`system`	ユーザ識別子。Application Integration では、リモート イベント ルータ（Web アプリケーションからデプロイされたもの）に対して、イベントのポストを行えるように、WebLogic Server に対する自己の認証を許可している。

wlai.hostPassword	security	ユーザのパスワード。Application Integration では、リモート イベント ルータ（Web アプリケーションからデプロイされたもの）に対して、イベントのポストを行えるように、WebLogic Server に対する自己の認証を許可している。
wlai.jms.autogen	true	Application Integration の起動プロセスで JMS リソースの自動生成を許可するフラグ。
wlai.jms.serverName	WLIJMSServer	ローカル WebLogic Server の JMSServer 名。
wlai.jms.connectionFactoryJNDIName	com.bea.wlai.JMSConnectionFactory	JMS 接続ファクトリの JNDI コンテキスト。
wlai.repositoryDatasourceName	WLAI_DataSource	JDBC データ ソース名。

Application View Console で、[Server Configuration] をクリックすると、Application Integration エンジンのコンフィグレーション パラメータが表示されます。wlai.logLevel プロパティは、この [Server Configuration] 画面で編集できます。その他のパラメータは、すべて WebLogic Administration Console を使用して編集する必要があります。

Application Integration コンポーネントを含む WebLogic Integration アプリケーションをデプロイし、アンデプロイすることにより、Application Integration エンジンを WebLogic Administration Console から起動し、停止します。

インポート / エクスポート ユーティリティ

従来のリリースでは、Application Integration データは、WebLogic Integration Studio から利用可能な、ワークフロー パッケージのインポート / エクスポート ユーティリティを使用して移行されていました。モジュラ デプロイメントをサポートするため、現行の Application Integration では、アプリケーション ビュー

に対してコマンドラインのインポート / エクスポート ツールを提供しています。詳細については、付録 B「アプリケーション ビューのインポートおよびエクスポート」を参照してください。

デプロイメント コンポーネント

以下に示すのは、Application Integration のデプロイメント コンポーネントです。

- `wlai-core.jar`

  この jar ファイルを、起動時に WebLogic Server のクラスパスに追加する必要があります。ファイル内容は以下のとおりです。

  ```
  wlai-core.jar
  |__log4j.jar contents
  |__logtoolkit.jar contents
  |__xmltoolkit.jar contents
  |__bea.jar contents
  |__com/bea/wlai/*.dtd
  |__com/bea/wlai/common/*.class
  |__com/bea/wlai/message/*.class
  |__xcci.jar contents
  ```

  また、この jar ファイルは、Application Integration のすべてのクライアントでも必要とされます。

- `wlai-client.jar`

  この jar ファイルには、アプリケーション ビュー クライアントおよびリソース アダプタの設計時 Web アプリケーションなど、Application Integration エンジンのクライアントで必要なすべてのクラスが組み込まれています。

- `wlai-server.jar`

  この jar ファイルには、サーバ側のコンポーネントだけで必要となるすべてのクラスが組み込まれいるため、アダプタ EAR ファイルまたはクライアントには組み込まないでください。

- `wlai-mbean.jar`

  この jar ファイルには、アプリケーション ビュー MBeans が組み込まれており、WebLogic Server のシステム クラスパスに組み込む必要があります。

- `wlai-server-ejb.jar`

  この jar ファイルには、基本サーバ クラスおよび管理 EJB が組み込まれています。この jar ファイルは、AI エンジンの他のあらゆるコンポーネントに優先してデプロイする必要があります。この jar ファイルには、Application Integration を初期化する起動 EJB が組み込まれています。

- `wlai.war`

  この jar ファイルには、Application View Console の Web アプリケーション、および Application Integration エンジンで使用する `LifeCycleServlet` が組み込まれています。

- `wlai-eventprocessor-ejb.jar`

  この jar ファイルには、Application Integration イベントを処理する、イベント処理メッセージ駆動型 EJB が組み込まれています。

- `wlai-asyncprocessor-ejb.jar`

  この jar ファイルには、非同期サービス呼び出しを処理する、非同期サービス処理メッセージ駆動型 EJB が組み込まれています。

- `wlai-plugin-ejb.jar`

  この jar ファイルには、BPM で使用する Application Integration プラグインのすべてのクラスが組み込まれています。

- `wlai-plugin.war`

  この jar ファイルには、BPM で使用する Application Integration プラグインのオンライン ヘルプ Web アプリケーションが組み込まれています。

WebLogic Integration 環境外のドメインに対するデプロイメント コンフィグレーション

WebLogic Integration ドメイン外の Application Integration エンタープライズ アプリケーションをデプロイするには、`wlai.ear` を使用します。この EAR ファイルには、次の図に示すコンポーネントが組み込まれています。

```
wlai.ear
|__META-INF
|     |__application.xml (EAR ファイル デプロイメント記述子 )
|__wlai.war (Application View Console Web アプリケーション )
|     |__WEB-INF
|         |__lib
|             |__webtoolkit.jar
|__wlai-server-ejb.jar (Application Integration 管理 EJB)
|     |__Startup EJB
|     |__ApplicationView EJB
|     |__SchemaManager EJB
|     |__DeployManager EJB
|     |__ApplicationViewManager EJB
|     |__NamespaceManager EJB
|__wlai-eventprocessor-ejb.jar
|__wlai-asyncprocessor-ejb.jar
|__ecibase.jar (ECI リポジトリ基本クラス )
|__ecirepository.jar (ECI リポジトリ クラス )
```

システム クラスパスに `wlai-core.jar`、`wlai-server.jar`、および `wlai-mbean.jar` を追加し、WebLogic Server を再起動する必要があります。また、`wlai-mbean.jar` ファイルを *`WL_HOME`*`\lib\mbeantypes` ディレクトリにをコピーする必要があります。*`WL_HOME`* は Web Logic Server がインストールされている場所です。

`wlai-core.jar` がクラスパスが追加されたら、以下のアプリケーション コンポーネントを `config.xml` ファイルに追加するか、WebLogic Server Administration Console からこのファイルをアップロードします。

```
<Application Deployed="true" Name="WebLogic Application Integration"
Path="PATH_TO_EAR/wlai.ear">

<EJBComponent Name="WLI&ends;AI Server" Targets="myserver"
URI="wlai-server-ejb.jar"/>

<WebAppComponent Name="wlai"
Targets="myserver" URI="wlai.war"/>

<EJBComponent Name="WLI&ends;AI Async Processor" Targets="myserver"
URI="wlai-asyncprocessor-ejb.jar"/>

<EJBComponent Name="WLI&ends;AI Event Processor" Targets="myserver"
URI="wlai-eventprocessor-ejb.jar"/>

</Application>
```

注意： デプロイメント順序は、`wlai.ear` の `application.xml` ファイルに指定されます。

注意： BPM 用 Application Integration プラグインは、WebLogic Integration 環境外のドメインに対しては、`wlai.ear` ファイルからデプロイされません。

JMS リソース

Application Integration エンジンでは、次の JMS リソースを使用します。

- `JMSServer`
- `JMSConnectionFactory`

  ユーザ設定の `JMSConnectionFactory` が JNDI の場所、`com.bea.wlai.JMSConnectionFactory` にバインドされていない場合、このファクトリは、複製されて、`com.bea.wlai.JMSConnectionFactory` にバインドされます。そのようにすることによって、内部 Application Integration コンポーネントが `JMSConnectionFactory` にアクセスできることが保証されます。

- キュー：
  - `WLAI_EVENT_QUEUE` ― `com.bea.wlai.EVENT_QUEUE` でバインドされている必要がある。
  - `WLAI_ASYNC_REQUEST_QUEUE` ― `com.bea.wlai.ASYNC_REQUEST_QUEUE` でバインドされている必要がある。
  - `WLAI_ASYNC_RESPONSE_QUEUE` ― `com.bea.wlai.ASYNC_RESPONSE_QUEUE` でバインドされている必要がある。
- トピック：
  - `WLAI_EVENT_TOPIC` ― `com.bea.wlai.EVENT_TOPIC` でバインドされている必要がある。

JMS リソース コンフィグレーション

Application Integration エンジンでは、スタンドアロン サーバおよびクラスタに対して JMS リソースを手動により明示的に定義していない場合、必要なすべての JMS リソースが自動的に定義されます。システム管理者は、次のコンフィグレーション パラメータを指定できます。

```
wlai.jms.serverName
wlai.jms.connectionFactoryJNDIName
```

さらに、管理者は、次のコンフィグレーション パラメータを指定することにより、JMS リソースの自動生成を無効にすることもできます。

```
wlai.jms.autogen=false
```

このパラメータにより、Application Integration エンジンによる必須 JMS リソースの自動生成が行われなくなります。このパラメータを指定した場合、すべての必須 JMS リソースを手動で定義する必要があります。

索引