BEA WebLogic Integration™

BPM クライアント アプリケーション プログラミング

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

BPM クライアント アプリケーション プログラミング

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

5. 値オブジェクトの使い方

6. JMS コネクタの確立

7. BPM トランザクション モデルの理解

8. プロセス エンジンからの切断

9. セキュリティ レルムのコンフィグレーション

Part III. 設計

13. ワークフロー テンプレートの作成および管理

14. ワークフロー テンプレート定義の作成および管理

15. タスクの管理

16. タスクのルーティングの管理

17. XML リポジトリの管理

18. ワークフロー オブジェクトの発行

23. 実行時の変数のモニタリング

24. ワークフロー例外のモニタリング

A. DTD フォーマット

B. 値オブジェクトのまとめ

C. EJB 記述子および Java クラス記述子

このマニュアルの内容

このマニュアルでは、WebLogic Integration Business Process Management (BPM) API の概要と、コンフィグレーション、設計、実行時管理、またはモニタ用のカスタム クライアントをこの API で作成する方法について説明します。

このマニュアルの内容は以下のとおりです。

- 第 1 章「Business Process Management API による開発」では、WebLogic Integration BPM API を使用したアプリケーション開発の概要を説明します。具体的には、WebLogic Integration プロセス エンジンと公開 API の両方の概要と、アプリケーション開発プロセスにおける主なタスクの要約、およびこのマニュアルに記載のコード例のソースである API サンプルについて説明します。
- Part I「API 開発の基本」には、BPM クライアント アプリケーションの開発に必要な各基本タスクを詳しく説明した以下の章が含まれます。
  - 第 2 章「パッケージおよびインタフェースのインポート」では、BPM クライアント アプリケーションで通常使用され、インポートが必要と思われるパッケージとインタフェースについて説明します。
  - 第 3 章「プロセス エンジンへの接続」では、WebLogic Integration プロセス エンジンへの接続方法を説明します（具体的には、セッション EJB にアクセスすることによる接続方法）。
  - 第 4 章「プロセス エンジン情報へのアクセス」では、`ServerProperties` EJB を使用してサーバに関する情報にアクセスする方法を説明します。
  - 第 5 章「値オブジェクトの使い方」では、BPM 値オブジェクトを使用して状態情報を作成する方法や状態情報にアクセスする方法を説明します。
  - 第 6 章「JMS コネクタの確立」では、クライアント アプリケーションから JMS に接続する方法を説明します。
  - 第 7 章「BPM トランザクション モデルの理解」では、BPM トランザクション モデルについて説明し、そのサンプルも紹介します。
  - 第 8 章「プロセス エンジンからの切断」では、WebLogic Integration プロセス エンジンから切断し、その他のリソースを解放する方法を説明します。

- Part II「コンフィグレーション」には、コンフィグレーションに使用される 4 つの EJB メソッド、`Admin`、`EJBCatalog`、`Permission`、および `WLPIPrincipal` についてそれぞれ詳しく説明した以下の章が含まれます。
  - 第 9 章「セキュリティ レルムのコンフィグレーション」では、セキュリティ レルムをコンフィグレーションする方法を説明します。
  - 第 10 章「ビジネス オペレーションのコンフィグレーション」では、ビジネス オペレーションをコンフィグレーションする方法を説明します。
  - 第 11 章「イベント キーのコンフィグレーション」では、イベント キーをコンフィグレーションする方法を説明します。
  - 第 12 章「ビジネス カレンダーのコンフィグレーション」では、ビジネス カレンダーをコンフィグレーションする方法を説明します。
- Part III「設計」には、ビジネス プロセスの設計に使用される各 `Admin` EJB メソッドについて詳しく説明した以下の章が含まれます。
  - 第 13 章「ワークフロー テンプレートの作成および管理」では、ワークフロー テンプレートの作成および管理方法を説明します。
  - 第 14 章「ワークフロー テンプレート定義の作成および管理」では、ワークフロー テンプレート定義の作成および管理方法を説明します。
  - 第 15 章「タスクの管理」では、テンプレート定義作成プロセスの一部として定義されるタスクの管理方法を説明します。
  - 第 16 章「タスクのルーティングの管理」では、タスク再ルーティングの定義および管理方法と、指定の期間にわたってタスクをあるユーザまたはロールから別のユーザまたはロールに再ルーティングする方法を説明します。
  - 第 17 章「XML リポジトリの管理」では、WebLogic Integration ビジネス プロセス コンポーネント用のデータ ストレージ機能である XML リポジトリの管理方法を説明します。
  - 第 18 章「ワークフロー オブジェクトの発行」では、ワークフロー オブジェクトの作成、エクスポートおよびインポート方法を説明します。
- Part IV「実行時の管理」には、実行時管理に使用される各 `Worklist` EJB メソッドについて詳しく説明した以下の章が含まれます。
  - 第 19 章「アクティブなオーガニゼーションの管理」では、アクティブなオーガニゼーションの管理方法を説明します。

- 第 20 章「手動によるワークフローの開始」では、ワークフローを手動で開始する方法を説明します。
- 第 21 章「実行時タスクの管理」では、タスクの取得、実行、割り当て、およびソート方法、クライアント要求への応答方法、タスクに完了または未完了マークを付ける方法、タスク プロパティの設定方法および例外ハンドラの呼び出し方法を説明します。

■ Part V「モニタ」には、モニタに使用されるメソッド、`Admin` EJB および `com.bea.wlpi.common.WorkflowException` のそれぞれについて詳しく説明した以下の章が含まれます。

- 第 22 章「実行時ワークフロー インスタンスのモニタ」では、実行時ワークフロー インスタンスのモニタ方法を説明します。
- 第 23 章「実行時の変数のモニタリング」では、実行時変数のモニタ方法を説明します。
- 第 24 章「ワークフロー例外のモニタリング」では、エラーの処理機能および監査機能について説明します。

■ 付録 A「DTD フォーマット」では、BPM DTD フォーマットについて詳しく説明します。

■ 付録 B「値オブジェクトのまとめ」では、BPM 値オブジェクトの取得メソッドおよび設定メソッドについて詳しく説明します。

■ 付録 C「EJB 記述子および Java クラス記述子」では、EJB および Java クラス記述子オブジェクトとそのメソッドについて説明します。

■ 付録 D「データベース スキーマ」では、BPM データベース スキーマについて説明します。

■ 付録 E「Studio および Worklist のロゴとテキストのカスタマイズ」では、BEA WebLogic Integration Studio および Worklist クライアントのロゴとテキストのカスタマイズ方法を説明します。

対象読者

このマニュアルは、コンフィグレーション、設計、実行時管理またはモニタ用のカスタム クライアントを作成すること、あるいは単に BPM API の理解を深めることを目的としているアプリケーション開発者を対象としています。このマニュアルでは、読者が WebLogic Integration 製品、Java プログラミングおよび XML に精通していることを前提としています。

e-docs Web サイト

BEA 製品のドキュメントは、BEA 社の 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、または次の URL にある「*e-docs*」という製品ドキュメント ページを直接表示してください。

`http://edocs.beasys.co.jp/e-docs/index.html`

このマニュアルの印刷方法

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

このマニュアルの PDF 版は、Web サイトで入手できます。WebLogic IntegrationPDF 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、次の URL にある Adobe の Web サイトから無料で入手できます。

`http://www.adobe.co.jp/`

関連情報

以下の WebLogic Integration ドキュメントには、BPM クライアント アプリケーション、Studio および Worklist を使用するプログラマに役立つ情報が含まれます。これらのアプリケーションは BPM API を使用して構築されています。

注意： Worklist クライアント アプリケーションは、このリリースの WebLogic Integration からは非推奨になります。Worklist クライアント アプリケーションに代わる機能については、『*WebLogic Integration リリース ノート*』を参照してください。

- *WebLogic Integration Studio ユーザーズ ガイド*
- *WebLogic Integration Worklist ユーザーズ ガイド*
- *WebLogic Integration BPM ユーザーズ ガイド*
- *BEA WebLogic Integration Javadoc*

BPM プラグインのプログラミングに関しては、『*WebLogic Integration BPM プラグイン プログラミング ガイド*』を参照してください。

Java アプリケーションの概説については、次の Sun Microsystems 社の Java Web サイトを参照してください。

`http://java.sun.com/`

XML および XML パーサの概説については、次の O' Reilly & Associates 社の XML.com Web サイトを参照してください。

`http://www.xml.com/`

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、ご使用の WebLogic Integration のリリースをお書き添えください。

本バージョンの 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 (**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。

表記法	適用

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 ( )`**
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。

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
	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。

Part I　API 開発の基本

1 Business Process Management API による開発

この章では、Business Process Management (BPM) API を使用したアプリケーション開発の概要について説明します。この章の内容は以下のとおりです。

- はじめに
- WebLogic Integration プロセス エンジン
- BPM API
- BPM アプリケーション開発タスク
- BPM API のサンプル

はじめに

Business Process Management (BPM) をサポートするために、WebLogic Integration には以下のコンポーネントが搭載されています。

- プロセス エンジン ― BPM サーバ
- WebLogic Integration Studio ―設計クライアント
- Worklist ―実行時管理クライアント

注意： Worklist クライアント アプリケーションは、このリリースの WebLogic Integration からは非推奨になります。Worklist クライアント アプリケーションに代わる各機能については、『*WebLogic Integration リリース ノート*』を参照してください。

上記のコンポーネントにより、ビジネス プロセスの設計、実行およびモニタを行い、WebLogic Integration に関連するデータを管理するための基本 BPM フレームワークが構成されます。

これらの設計クライアントおよび実行時管理クライアントを使用することに加えて、ビジネス プロセスの管理や既にあるユーザ インタフェース機能セットの拡張を行うためのカスタム クライアントも BPM Application Programming Interface (API) を使用して作成できます。

このマニュアルでは、BPM API を使用してコンフィグレーション、設計、実行時管理およびモニタ用のカスタム クライアントを作成する方法を説明します。API の詳細については、『*BEA WebLogic Integration Javadoc*』を参照してください。

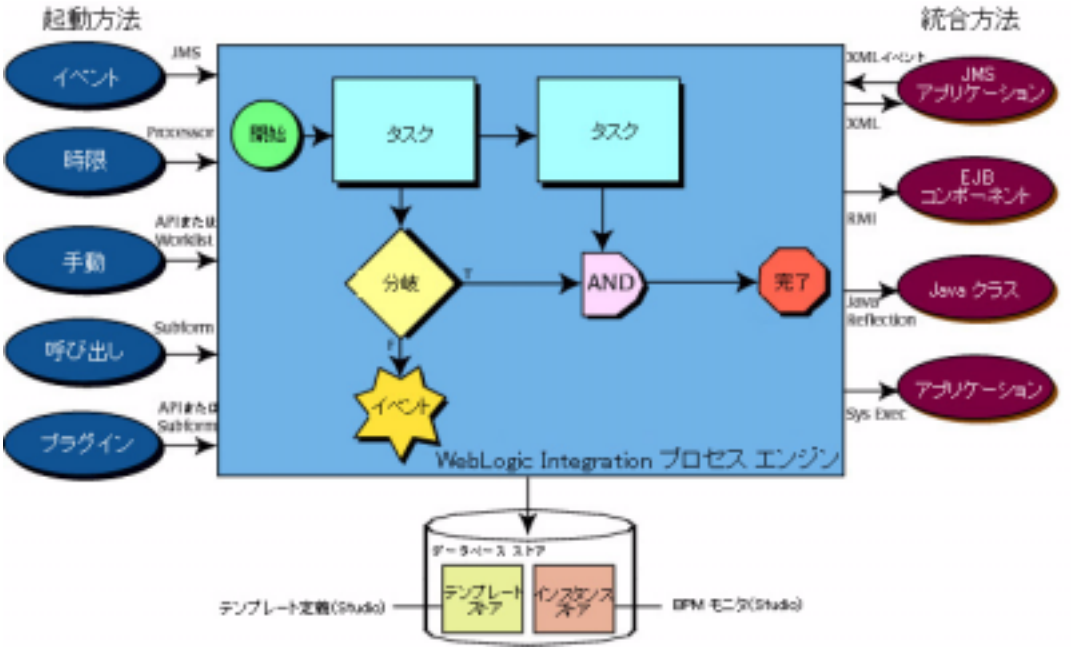
注意： ビジネス プロセス モデルの概要や Studio クライアント インタフェースを使用したビジネス プロセスの管理、設計、およびモニタの詳細については、『*WebLogic Integration Studio ユーザーズ ガイド*』を参照してください。

Worklist クライアント インタフェースを使用したビジネス プロセスの実行の詳細については、『*WebLogic Integration Worklist ユーザーズ ガイド*』を参照してください。

WebLogic Integration プロセス エンジン

BPM 処理モデルを次の図に示します。

図 1-1 BPM 処理モデル



注意： BPM 処理モデルの詳細については、『*WebLogic Integration Studio ユーザーズ ガイド*』を参照してください。

BPM 処理モデルは、WebLogic Integration *プロセス エンジン*によって異なります（プロセス エンジンは、実行時インスタンス用のコントローラとして機能し、実行の管理やその進行状況のモニタを行う）。

上の図の左側には、ビジネス プロセスの開始（または*インスタンス化*）や、既にあるビジネス プロセスとの交信に使用できるメソッドが記載されています（XML イベントへの応答であるメソッドや、時間ベースのイベントとして機能するメソッド、あるいは手動呼び出し、別のビジネス プロセスからの呼び出し、またはプラグイン インタフェースを介したメソッドなど）。

図の右側には、ビジネス プロセスに統合できるエンティティ、すなわち JMS (Java Message Service: Java メッセージ サービス) アプリケーション、EJB コンポーネント、Java クラス、またはその他のアプリケーションが記載されています。

図の下部に記載されているデータベース ストアには、テンプレート、テンプレート定義、および実行時インスタンスが格納されます。Studio またはカスタム設計クライアントを使用することで、テンプレートやテンプレート定義の作成、および実行時インスタンスのモニタが可能です。

WebLogic Server インフラストラクチャ

WebLogic Integration プロセス エンジンは BEA WebLogic Server 上で実行され、次の表に記載の J2EE (Java 2 Enterprise Edition: Java 2 エンタープライズ エディション) プラットフォーム サービスを利用します。WebLogic Server サービスの詳細については、次の URL から BEA WebLogic Server ドキュメント内の『*BEA WebLogic Server の紹介*』を参照してください。

http://edocs.beasys.co.jp/e-docs/wls/docs70/intro/index.html

表 1-1 BPM で使用される WebLogic Server サービス

サービス	説明
EJB (Enterprise JavaBean: エンタープライズ JavaBean)	キャッシュ、永続化、トランザクション管理などのライフサイクル管理およびサービスを提供する。プロセス エンジンは、セッション EJB とエンティティ EJB の組み合わせから構成されている。詳細については、1-6 ページの「プロセス エンジンのコンポーネント アーキテクチャ」を参照してください。また、メッセージ駆動型 Bean は、非同期メッセージ サービスの提供に使用される。

表 1-1 BPM で使用される WebLogic Server サービス

サービス	説明
セキュリティ レルム	セキュリティ インタフェース（プリンシパル、グループ、ACL、およびパーミッションを含む）をサポートする。プロセス エンジンは、WebLogic Server セキュリティ レルム上に構築される。Studio で定義されているロールは WebLogic Server セキュリティ グループにマッピングされ、ユーザは WebLogic Server ユーザにマッピングされる。
Web サーバ	Web ブラウザに加えて、HTTP、サーブレットおよび JSP を使用するその他のカスタム設計クライアントと実行時管理クライアントをサポートする。
JNDI (Java Naming and Directory Service: Java ネーミングおよびディレクトリ サービス)	ディレクトリ サービス機能をサポートする。
JDBC (Java Database Connectivity: Java データベース接続性)	Java データベース接続性および永続化をサポートする。プロセス エンジンは JDBC を使用してデータを永続化する。
メッセージング	JMS を実装する（XML コンテンツの伝送を含む）。プロセス エンジンは JMS を使用して通知（ワークリスト、時間、イベント）およびメッセージ（エラー、監査）の通信を行う。詳細については、6-1 ページの「JMS コネクタの確立」を参照してください。
クラスタ化	アプリケーション開発者とエンド ユーザが理解できる方法で複数のサーバをグループ化することで、スケーラビリティと高可用性をサポートする。プロセス エンジンはクラスタ化を利用する。データの整合性と、べき等を保つためには、ユニークなトランザクション ID を指定する（該当のメソッドについて説明したこのマニュアルに後述の節を参照）。
Time サービス	WebLogic Server により提供されるサービスの上に構築され、時限イベントやその他のタイマベースのアクティビティをサポートする。

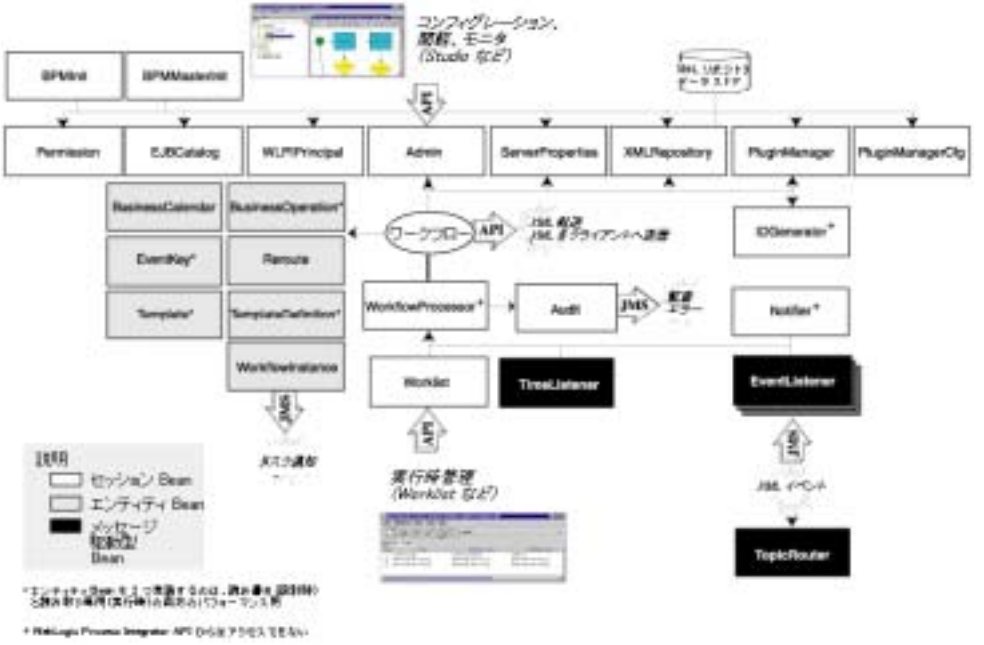
プロセス エンジンのコンポーネント アーキテクチャ

WebLogic Integration プロセス エンジンは以下のコンポーネントから構成されています。

- セッション EJB
- エンティティ EJB
- メッセージ駆動型 Bean

クライアントと JMS の会話を含むプロセス エンジンのアーキテクチャを次の図に示します。

図 1-2 プロセス エンジンのコンポーネント アーキテクチャ



上の図では以下のことが示されています。

- プロセス エンジンにより、ビジネス プロセスの設計、管理および実行が管理されます。
- Admin と Worklist のセッション EJB はそれぞれ、設計クライアントと実行時管理クライアントに対するプライマリ インタフェースです。
- WorkflowProcessor セッション EJB により、プロセス エンジンにプライマリ インタフェースが提供されます（これは BPM API の一部ではない）。
- Audit セッション EJB により、監査、トレースおよびエラーのメッセージが管理されます。
- EventListener メッセージ駆動型 Bean により、XML イベントが管理されます。
- TopicRouter メッセージ駆動型 Bean は、リリース 2.0 より前の BEA WebLogic Process Integrator 製品との下位互換性のためにサポートされています。TopicRouter により、wlpiEvent JMS トピック（BEA WebLogic Process Integrator リリース 2.0 以降ではプロセス エンジンによりサポートされていない）に送信されたすべてのメッセージが WLI_BPM_Event メッセージ駆動型 Bean に再ルーティングされます。
- TimeListener メッセージ駆動型 Bean により、時限イベントが管理されます。
- 複数のエンティティ EJB (BusinessOperation、EventKey、Template、および TemplateDefinition) により、コンフィグレーションおよび設計中に使用される読み書き対応実装と、実行時に使用される読み取り専用実装の 2 つの実装が保持されます。

プロセス エンジンのコンポーネントについて以下でさらに詳しく説明します。

セッション EJB

BPM セッション EJB は、クライアントに対して、要求に応じてアプリケーション サービスを提供します。また BPM セッション EJB により、プロセス エンジンにプライマリ インタフェースが提供されます。

BPM セッション EJB の機能を次の表にまとめます。これらのセッション EJB は、BPM API 経由でクライアント アプリケーションにアクセスできます（別途記載のない限り）。この表には、各 EJB がステートフルまたはステートレスのどちらであるかも示されています。

表 1-2 セッション EJB

セッション EJB	説明	EJB のタイプ
com.bea.wlpi.server.admin.Admin	設計クライアント（Studio 設計クライアントとカスタム設計クライアントを含む）のプライマリ インタフェースを提供する。	ステートフル
com.bea.wlpi.server.audit.Audit	監査とエラーの出力を WebLogic JMS に送信する。	ステートレス
com.bea.wlpi.server.catalog.EJBCatalog	WebLogic Server にデプロイされている EJB をカタログに入れる。	ステートフル
com.bea.wlpi.server.idgenerator.IDGenerator[1]	さまざまな WLPI システム オブジェクト クラスに一意な ID を生成する（内部使用のみ）。	ステートレス
com.bea.wlpi.server.init.BPMInit	プラグイン マネージャとライセンス チェックを初期化。クラスタ環境では、この EJB がクラスタ全体にデプロイされる。	ステートレス
com.bea.wlpi.server.init.BPMMasterInit	メッセージのリーパ（時限トリガ型クリーンアップ サービス）を設定し起動する。クラスタ環境では、この EJB が 1 台のシステム上で実行される。	ステートレス
com.bea.wlpi.server.notify.Notifier[1]	JMS キューと発行サービスをサポートする（内部使用のみ）。	ステートレス
com.bea.wlpi.server.permission.Permission	ユーザ パーミッションとロール パーミッションの取得と設定をサポートする。	ステートレス

表 1-2 セッション EJB (続き)

セッション EJB	説明	EJB のタイプ
com.bea.wlpi.server.plugin.PluginManager	ワークフロー実行中にプラグインの実行時管理を行う。 BPM プラグインのプログラミングの詳細については、『*WebLogic Integration BPM プラグイン プログラミング ガイド*』を参照。	ステートレス
com.bea.wlpi.server.plugin.PluginManagerCfg	プラグインのコンフィグレーション時および設計時サポートと管理を行う。 BPM プラグインのプログラミングの詳細については、『*WebLogic Integration BPM プラグイン プログラミング ガイド*』を参照。	ステートレス
com.bea.wlpi.server.serverproperties.ServerProperties	BPM サーバ プロパティを提供する。	ステートレス
com.bea.wlpi.server.principal.WLPIPrincipal	プロセス エンジンを WebLogic Server セキュリティ レルムにリンクする。	ステートレス
com.bea.wlpi.server.workflowprocessor.WorkflowProcessor[1]	プロセス エンジンにプライマリ インタフェースを提供する。	ステートフル
com.bea.wlpi.server.worklist.Worklist	実行時管理クライアント（Worklist クライアントとカスタム実行時管理クライアントを含む）のプライマリ インタフェースを提供する。	ステートフル
com.bea.eci.repository.ejb.XMLRepository	XML リポジトリ データベースへのアクセスを提供する。	ステートレス

1. BPM API では選択不可。

ステートフル セッション EJB は、セッション中であれば、クライアントがリソースを解放するまで使用可能です。

一方、ステートレス セッション EJB は、共有される EJB です。つまり、ある EJB に複数のクライアントがアクセスできます。したがって、プロセス エンジンによる EJB への複数のメソッド呼び出しを行うために、同じ EJB が使用される保証はありません。ステートレス セッション EJB はクラスタ化を容易にします。後続のメソッド呼び出しを任意の数のサーバ上の EJB の別のインスタンスによって処理できるからです。

BPM API 経由で使用できるセッション EJB の詳細については、1-13 ページの「BPM API」 を参照してください。

エンティティ EJB

エンティティ EJB は、データ オブジェクトを表わし、状態情報を提供します。

BPM エンティティ EJB には BPM API からは直接アクセスできませんが、状態情報を取得するためのクラスが `com.bea.wlpi.common` パッケージによって提供されます。`com.bea.wlpi.common` パッケージの詳細については、1-18 ページの「クライアント / サーバ共通パッケージ」 を参照してください。

BPM エンティティ EJB を次の表に示します。

表 1-3 エンティティ EJB

エンティティ EJB	説明
`com.bea.wlpi.server.businesscalendar.BusinessCalendar`	ビジネス指向の時間計算を定義する。
`com.bea.wlpi.server.businessoperation.BusinessOperation[RO,RW]`	ユーザ定義の EJB または Java クラス アクションを定義する。 パフォーマンスの向上のため、この EJB の 2 つの実装、すなわちコンフィグレーションおよび設計中に使用される読み書き対応実装と、実行時に使用される読み取り専用実装が保持されている。

表 1-3 エンティティ EJB (続き)

エンティティ EJB	説明
com.bea.wlpi.server.eventkey.EventKey[RO,RW]	イベントをフィルタ処理するためのイベントキー式を定義する。 パフォーマンスの向上のため、この EJB の 2 つの実装、すなわちコンフィグレーションおよび設計中に使用される読み書き対応実装と、実行時に使用される読み取り専用実装が保持されている。
com.bea.wlpi.server.reroute.Reroute	事前定義済みの時間間隔でユーザのタスク再ルーティングを定義する。
com.bea.wlpi.server.template.Template[RO,RW]	ワークフロー テンプレートの設計時表現を提供する。実行中のインスタンスを作成するために実行時に使用される。 パフォーマンスの向上のため、この EJB の 2 つの実装、すなわちコンフィグレーションおよび設計中に使用される読み書き対応実装と、実行時に使用される読み取り専用実装が保持されている。
com.bea.wlpi.server.template.TemplateDefinition[RO,RW]	ワークフロー テンプレート定義の設計時表現を提供する。実行中のインスタンスを作成するために実行時に使用される。 パフォーマンスの向上のため、この EJB の 2 つの実装、すなわちコンフィグレーションおよび設計中に使用される読み書き対応実装と、実行時に使用される読み取り専用実装が保持されている。
com.bea.wlpi.server.instance.WorkflowInstance	ワークフロー インスタンスの実行時表現を提供する。

メッセージ駆動型 Bean

BPM メッセージ駆動型 Bean を次の表に示します。

表 1-4 メッセージ駆動型 Bean

メッセージ駆動型 Bean	説明
`com.bea.wlpi.server.eventlistener.EventListener`	イベント メッセージを管理する。
`com.bea.wlpi.server.timelistener.TimeListener`	時限イベントを管理する。
`com.bea.wlpi.server.topicrouter.TopicRouter`	リリース 2.0 以前の BEA WebLogic Process Integrator 製品に搭載されているプロセス エンジンとの下位互換性を保つためにサポートされている。 `wlpiEvent` JMS *トピック*（WebLogic Process Integrator リリース 2.0 以降ではサポートしていない）に送信されたすべてのメッセージを、新たにサポートされた `WLI_BPM_Event` キューのメッセージを処理する `EventListener` メッセージ駆動型 Bean に再ルーティングする。JMS キューおよびトピックの詳細については、6-1 ページの「JMS コネクタの確立」を参照。 **注意：** メッセージ伝送を高速化したい場合は、`wlpiEvent` JMS トピックの代わりに `WLI_BPM_Event` JMS キュー（`com.bea.wli.bpm.Event`）を使うようコードを更新することもできます。

BPM API

Business Process Management (BPM) API は、1-8 ページの「セッション EJB」の表にまとめられている 9 つの EJB と以下の 7 つのパッケージから構成されています。

- `com.bea.wlpi.client.common`
- `com.bea.wlpi.client.util`
- `com.bea.wlpi.common`
- `com.bea.wlpi.common.plugin`
- `com.bea.wlpi.common.security`
- `com.bea.wlpi.util`
- `com.bea.eci.repository.helper`

`IDGenerator`、`Notifier` および `WorkflowProcessor` EJB は BPM API に含まれていません。

API コンポーネントについて、以下に詳しく説明します。詳細については、『*BEA WebLogic Integration Javadoc*』を参照してください。API セッション EJB とのインタフェースに関しては、3-1 ページの「プロセス エンジンへの接続」を参照してください。

Admin セッション EJB

`com.bea.wlpi.server.admin.Admin` セッション EJB は、Studio クライアントやカスタム設計クライアントなどの設計クライアント用のプライマリ インタフェースとして機能する*ステートフル* EJB です。

`Admin` EJB に含まれている public メソッドを使用すると、以下の操作が可能です。

- テンプレート、テンプレート定義、および関連付けられているビジネス オペレーションの定義
- ビジネス カレンダーとイベント キーの作成、およびタスクの再ルーティング

- タスクのライフサイクルにおける段階の制御（割り当て、完了マークの付与など）
- クラス記述子、タスク、変数、インスタンス、統計レポートの取得
- テンプレート定義およびワークフロー パッケージ全体のインポート / エクスポート

Audit セッション EJB

`com.bea.wlpi.server.audit.Audit` セッション EJB は、WebLogic JMS トピック（`wlpiAudit`）をカプセル化する*ステートレス* EJB であり、プロセス エンジンは監査およびエラー メッセージをクライアントに代わってこの EJB に送信します。`Audit` EJB は、エラー メッセージの送信先である `wlpiError` トピックへのアクセスもカプセル化します。ユーザは、『*WebLogic Integration Studio ユーザーズ ガイド*』の説明に従い、ワークフロー テンプレート定義の設計時にタスク アクションを使用して監査エントリを定義します。

各監査メッセージは、`Audit` DTD に定義されているフォーマットに準拠した XML ドキュメントから構成されています。`Audit` DTD フォーマットの詳細については、A-2 ページの「監査 DTD」または『*BEA WebLogic Integration Javadoc*』を参照してください。

監査情報メッセージ（`Info`）およびエラー メッセージ（`Error`）は、サブシステム名 `WLPI` を使用して WebLogic Server ログ ファイルに送信されます。

JMS トピック `wlipAudit` および `wlpiError` への JMS コネクタを設定する詳細については、6-1 ページの「JMS コネクタの確立」を参照してください。

EJBCatalog セッション EJB

`com.bea.wlpi.server.catalog.EJBCatalog` セッション EJB は、特定のサーバにデプロイされた EJB を分類する*ステートフル* EJB です。EJBCatalog セッション EJB は JNDI ツリーをスキャンし、EJB メタデータを収集してカタログに格納します。`EJBCatalog` は、ビジネス オペレーションを定義するために設計クライアントによって使用されます。

EJBCatalog EJB に含まれている public メソッドを使用すると、以下の操作が可能です。

- WebLogic Server にデプロイされている全 EJB の JNDI 名のリストアップ
- WebLogic Server にインストールされている全 EJB の内部情報を説明する EJB メタデータのリストアップ
- カタログ作成が開始される WebLogic JNDI ツリー内のルート コンテキストの設定
- リスト要求ごとにカタログを再生成するかどうかの指定

Permission セッション EJB

com.bea.wlpi.server.permission.Permission セッション EJB は、ロール アクションとユーザ アクションの両方に関連するセキュリティ パーミッションの取得および設定を可能にする*ステートレス* EJB です。

PluginManager セッション EJB

com.bea.wlpi.server.plugin.PluginManager セッション EJB は、ワークフロー実行中にプラグインの実行時管理を行う*ステートレス* EJB です。

プラグインのプログラミングと PluginManager EJB の詳細については、『*WebLogic Integration BPM プラグイン プログラミング ガイド*』を参照してください。

PluginManagerCfg セッション EJB

com.bea.wlpi.server.plugin.PluginManagerCfg セッション EJB は、ユーザ定義プラグインの実装の管理を可能にする*ステートレス* EJB です。

プラグインのプログラミングと PluginManagerCfg EJB の詳細については、『*WebLogic Integration BPM プラグイン プログラミング ガイド*』を参照してください。

ServerProperties セッション EJB

`com.bea.wlpi.server.serverproperties.ServerProperties` セッション EJB は、BPM を規定するプロパティに関する情報をクライアントが取得できるようにする*ステートレス* EJB です。

`ServerProperties` EJB に含まれている public メソッドを使用すると、以下の情報を取得できます。

- ソフトウェア バージョン（メジャー / マイナー バージョン番号、ビルド番号、製品名を含む）
- ソフトウェアのプロパティ
- テンプレートに定義されている機能のサポートに必要なソフトウェア バージョン
- サーバがサポートするパッケージのバージョン

WLPIPrincipal セッション EJB

`com.bea.wlpi.server.principal.WLPIPrincipal` セッション EJB は、クライアントが WebLogic Server セキュリティ レルムと会話を行い、BPM オーガニゼーション、ロール、ユーザ、その他のセキュリティ プロパティにアクセスできようにする*ステートレス* EJB です。BPM ロールは WebLogic Server セキュリティ グループにマッピングされ、BPM ユーザは WebLogic Server ユーザにマッピングされます。

`WLPIPrincipal` EJB に含まれている public メソッドを使用すると、システムへの確実なアクセスに必要なオーガニゼーション、ロールおよびユーザに関する情報のコンフィグレーション、管理、および取得が可能です。

注意： WebLogic Server セキュリティ レルムに格納されている WebLogic Server 固有の情報以外のロールおよびユーザに関する情報は、BPM データベース内に保持されます。

Worklist セッション EJB

com.bea.wlpi.server.worklist.Worklist セッション EJB は、Worklist クライアントやカスタム実行時管理クライアントなどの実行時管理クライアント用のプライマリ インタフェースとして機能するステートフル EJB です。Worklist EJB により、実行中のインスタンスとの会話が可能になります。

Worklist EJB に含まれている public メソッドを使用すると、以下の操作が可能です。

- 開始可能ビジネス プロセスの取得
- ビジネス プロセスのインスタンス化
- タスクとタスク数の取得
- 実行中のタスクのライフサイクルにおける段階の制御（割り当て、完了マークの付与など）
- 通知への応答
- タスクのプロパティの設定
- 例外処理メソッドの呼び出し

XMLRepository セッション EJB

com.bea.eci.repository.ejb.XMLRepository セッション EJB は、XML リポジトリ データベースへのアクセスを提供するステートレス EJB です。

XMLRepository EJB に含まれている public メソッドを使用すると、以下の操作が可能です。

- XML リポジトリ フォルダの管理
- XML リポジトリ エンティティの管理
- EJB 環境変数値の取得

クライアント共通パッケージ

com.bea.wlpi.client.common パッケージにより、BPM クライアントで使用される共通のクライアントサイド クラスが提供されます。このパッケージには以下の用途のクラスが含まれています。

- プロセス エンジンへの接続
- クライアント GUI レイアウトおよび開発の簡素化と、ダイアログ ボックスの生成
- 例外、タスク、バージョン情報の取得
- リソースの管理
- 画像とアイコンのキャッシュ
- ファイル拡張子別のファイルのフィルタ処理
- クラスタ化された環境内で使用するユニークなトランザクション ID の生成

クライアント ユーティリティ パッケージ

com.bea.wlpi.client.util パッケージには、JMS トピックのパブリッシュ / サブスクライブ機能をテストする 2 つの JMS ユーティリティ、JMSTest および JMSTestAddr が含まれています。

クライアント / サーバ共通パッケージ

com.bea.wlpi.common パッケージには、BPM クライアントと WebLogic Integration プロセス エンジンの両方で使用されるクラスが含まれています。このパッケージのコンポーネントは以下のとおりです。

- 以下の項目を表す *値オブジェクト*
  - オーガニゼーション、ユーザ、ロール
  - テンプレートとテンプレート定義
  - タスクおよびタスク再ルーティング

- 変数
- 実行中のインスタンスおよび機能
- ビジネス カレンダー
- Java クラス
- イベント キー
- バージョン情報

- Java クラス、EJB、EJB 呼び出し、およびサーバサイド Java クラス メソッドの記述子
- JDK 1.3 より前の実行時環境で*長い文字列*（UTF8 コードで 64KB を超える文字列）をシリアライズするメカニズム
- 言語非依存サーバ メッセージング用のメッセージ インタフェース

パッケージ メンバーをシリアライズすると、クライアントとサーバ間の情報交換が容易になります。

プラグイン共通パッケージ

`com.bea.wlpi.common.plugin` パッケージには、ユーザ定義プラグインの管理に使用される共通クラスが含まれています。

詳細については、『*WebLogic Integration BPM プラグイン プログラミング ガイド*』を参照してください。

セキュリティ共通パッケージ

`com.bea.wlpi.common.security` パッケージには、セキュリティ パーミッションの定義に使用される共通クラスが含まれています。このパッケージのコンポーネントは以下のとおりです。

- ロール パーミッションとユーザ パーミッションを表す*値オブジェクト*
- セキュリティ パーミッション タイプ

ユーティリティ パッケージ

`com.bea.wlpi.util` パッケージには、メッセージ駆動型 Bean を生成するユーティリティなどの一般的な BPM ユーティリティが含まれています。

XML リポジトリ ヘルパー パッケージ

`com.bea.eci.repository.helper` パッケージには、XML リポジトリへのアクセスに使用される共通クラスが含まれています。このパッケージのコンポーネントは以下のとおりです。

- XML リポジトリ フォルダおよびエンティティを表す*値オブジェクト*
- フォルダとサブフォルダを作成および管理するためのメソッド
- エンティティを作成および管理するためのメソッド

BPM アプリケーション開発タスク

BPM API を使用すると、以下に説明するアプリケーション開発タスクを行うことができます。

コンフィグレーション

Studio クライアントまたはカスタム コンフィグレーション クライアントを介して BPM API にアクセスすると、以下のエンティティをコンフィグレーションできます。

- セキュリティ レルム（オーガニゼーション、ロール、ユーザ）
- ビジネス オペレーション
- イベント キー
- ビジネス カレンダー

このマニュアルの Part II「コンフィグレーション」では、これらの各タイプのエンティティを BPM API を使用してコンフィグレーションする方法を説明します。API の詳細については、『*BEA WebLogic Integration Javadoc*』を参照してください。

設計

Studio クライアントまたはカスタム設計クライアントを介して BPM API にアクセスすると、以下の設計タスクを実行できます。

- ワークフロー テンプレートとテンプレート定義の作成および管理
- タスクとタスク再ルーティングの作成および管理
- XML リポジトリ データベースの管理
- 設計データのインポート / エクスポート

このマニュアルの Part III「設計」では、これらの各タイプの設計関連タスクを BPM API で実行する方法を説明します。API の詳細については、『*BEA WebLogic Integration Javadoc*』を参照してください。

実行時の管理

Worklist クライアントまたはカスタム実行時管理クライアントを介して BPM API にアクセスすると、以下の実行時管理タスクを実行できます。

- アクティブなオーガニゼーションの管理
- ビジネス プロセス インスタンスの呼び出し
- 実行時タスク（以下参照）の管理
  - タスクの取得、割り当て、実行
  - クライアント要求への応答
  - タスクへの完了または未完了マークの付与
  - タスクのプロパティの設定
  - 変数の更新
  - 例外ハンドラの呼び出し

このマニュアルの Part IV「実行時の管理」では、これらの各タイプの実行時管理タスクを BPM API で実行する方法を説明します。API の詳細については、『*BEA WebLogic Integration Javadoc*』を参照してください。

モニタ

Studio クライアントまたはカスタム モニタ クライアントを介して BPM API にアクセスすると、以下のモニタ タスクを実行できます。

- 実行時インスタンスの管理
- 実行時変数の管理
- ワークフロー、タスク、ユーザまたはロール、およびタスク状態に基づいたグラフィカル レポートの生成
- ワークフロー、タスク、ユーザまたはロール、および日付に基づいた統計レポートの生成

このマニュアルの Part V「モニタ」では、これらの各タイプのモニタ タスクを BPM API で実行する方法を説明します。API の詳細については、『*BEA WebLogic Integration Javadoc*』を参照してください。

プラグイン開発

BPM API を使用すると、カスタム プラグインを設計および統合できます。

詳細については、『*WebLogic Integration BPM プラグイン プログラミング ガイド*』を参照してください。

BPM API のサンプル

このマニュアル全体を通して使用されているサンプルは、*SAMPLES_HOME*/integration/samples/bpm_api ディレクトリのソフトウェアと共に提供される BPM クライアント サンプルから抜粋されたものです。ソフトウェアには以下のサンプルが含まれています。

- コマンドライン管理サンプル
- コマンドライン Studio サンプル
- コマンドライン Worklist サンプル
- コマンドライン SAX パーサ サンプル
- JSP Worklist サンプル

これらのクライアント サンプルについて、以下に詳しく説明します。

コマンドライン管理サンプル

WebLogic Integration には、カスタム コマンドライン クライアントでサポートされる可能性のある基本管理タスクを示すサンプルが含まれています。主なタスクは以下のとおりです。

- プロセス エンジンへの接続
- オーガニゼーション、ロール、ユーザの管理（追加、削除など）
- セキュリティ レルムの管理（セキュリティ レルム クラス名の取得、セキュリティ レルムが管理可能かどうかの判断）
- ビジネス オペレーションの管理（追加および削除や、EJB/Java クラス記述子の取得など）
- イベント キーの管理（追加、削除など）
- ビジネス カレンダーの管理（追加、削除など）
- デプロイされた EJB の管理（デプロイされた EJB の名前と記述子のリストアップや、自動再生成の有効化 / 無効化）
- サーバ プロパティ（サーバ バージョン、サポートされているテンプレート定義バージョン、システム プロパティなど）の表示

コマンドライン管理サンプルは、*SAMPLES_HOME*/integration/samples/bpm_api/commandline ディレクトリにある Java ファイル CLAdmin.java から構成されています。このサンプルをコンパイルおよび実行する方法については、このディレクトリに含まれている Readme.txt ファイルを参照してください。

Part I「API 開発の基本」および Part II「コンフィグレーション」に記載のサンプルの一部は、コマンドライン管理サンプルから抜粋されたものです。

コマンドライン Studio サンプル

WebLogic Integration には、カスタム コマンドライン Studio クライアントでサポートされる可能性のある以下の基本設計タスクを示すサンプルが含まれています。

- テンプレートの管理（作成、削除、リストアップ）
- タスク再ルーティングの管理（追加、削除、リストアップ）

コマンドライン Studio サンプルは、*SAMPLES_HOME*/integration/samples/bpm_api/commandline ディレクトリにある Java ファイル CLStudio.java から構成されています。このサンプルをコンパイルおよび実行する方法については、このディレクトリに含まれている Readme.txt ファイルを参照してください。

Part III「設計」に記載のサンプルの一部は、コマンドライン Studio サンプルから抜粋されたものです。

コマンドライン Worklist サンプル

WebLogic Integration には、カスタム コマンドライン Worklist クライアントでサポートされる可能性のある基本 Worklist 管理タスクを示すサンプルが含まれています。主なタスクは以下のとおりです。

- プロセス エンジンへの接続
- オーガニゼーションの管理（全オーガニゼーションのリストアップ、現在のアクティブなオーガニゼーションのリストアップおよび設定）
- ビジネス プロセスの管理（すべての開始可能ビジネス プロセスのリストアップ、1 つのビジネス プロセスの開始）
- タスクの管理（タスクのリストアップおよび実行、タスク数の取得）

コマンドライン Worklist サンプルは、*SAMPLES_HOME*`/integration/samples/bpm_api/commandline` ディレクトリにある Java ファイル `CLWorklist.java` から構成されています。このサンプルをコンパイルおよび実行する方法については、このディレクトリに含まれている `Readme.txt` ファイルを参照してください。

Part IV「実行時の管理」に記載のサンプルの一部は、コマンドライン Worklist サンプルから抜粋されたものです。

コマンドライン SAX パーサ サンプル

WebLogic Integration には、BPM サーバから受信したクライアント要求を SAX パーサで解析する方法を示すサンプルが含まれています。具体的には、このサンプルにより以下の操作方法が示されます。

- サーバからのクライアント要求を Xerces SAX パーサで解析する方法
- サーバへの応答の作成および送信方法

コマンドライン SAX パーサ サンプルは、*SAMPLES_HOME*`/integration/samples/bpm_api/commandline` ディレクトリにある Java ファイル `CLSaxParser.java` から構成されています。このサンプルをコンパイルおよび実行する方法については、このディレクトリに含まれている `Readme.txt` ファイルを参照してください。

Part IV「実行時の管理」に記載のサンプルの一部は、コマンドライン Worklist サンプルから抜粋されたものです。

JSP Worklist サンプル

WebLogic Integration には、カスタム JSP (Java Server Page: Java サーバ ページ) ベースの Worklist クライアントでサポートされる可能性のある基本実行時管理タスクを示すサンプルが含まれています。このサンプルのメイン コンポーネントは `worklist.jsp` ファイルであり、このファイルによって JSP Worklist へのプライマリ インタフェースが提供されます。

JSP クライアントによって実行されるタスクと、詳細が記載されているサンプル ファイルを次の表に示します。

表 1-5 JSP Worklist サンプル

タスク	関連するサンプル ファイル
プロセス エンジンへのログオンおよび接続と、ログイン、パスワード、URL の指定。 プロセス エンジンへの接続の詳細については、3-1 ページの「プロセス エンジンへの接続」を参照。	`logon.html`、`logon.jsp`
選択可能なワークフロー テンプレートのリストからのワークフローの開始。	`startworkflow.jsp`
タスクの取得	`worklist.jsp`
タスクの実行	`worklist.jsp`

表 1-5 JSP Worklist サンプル (続き)

タスク	関連するサンプル ファイル
応答 XML (responseParser) の解析。 responseParser により、SAX (Simple API for XML ñ XML 用の簡単な API) を使用してドキュメント ハンドラが実装され、XML ドキュメント内で発生した特定のイベントをアプリケーションが処理できるようにする。 XML の解析の詳細については、次の URL から O'Reilly & Associates 社の XML.com Web サイトを参照。http://www.xml.com/	ResponseParser.java
変数の設定（query.jsp）またはメッセージの確認応答（message.jsp）を行うために入力が必要な場合の応答。	query.jsp、message.jsp
別のユーザまたはロールへのタスクの再割り当て。	reassign.jsp
タスクへの完了または未完了マークの付与。	worklist.jsp
実行中インスタンスの実行時特性を決定するタスク プロパティの設定。	taskproperties.jsp
プロセス エンジンからのログアウトおよび切断。 切断の詳細については、8-1 ページの「プロセス エンジンからの切断」を参照。	logoff.jsp

プライマリ JSP インタフェース worklist.jsp を次の図で示します。

図 1-3　JSP Worklist へのプライマリ インタフェース

JSP Worklist を Web アプリケーションとしてデプロイする手順については、次の URL から BEA WebLogic Server ドキュメンテーション内の『*Web アプリケーションのアセンブルとコンフィグレーション*』の「Web アプリケーションのデプロイメント」を参照してください。

`http://edocs.beasys.co.jp/e-docs/wls/docs70/webapp/deployment.html`

デプロイが完了した後、JSP Worklist を実行する手順は、次のとおりです。

1. `CLASSPATH` に次のパスを追加します。`WLI_HOME/lib/wlpi-worklist.jar.`

   たとえば、`StartWebLogic` ファイルの `REM Start weblogic` コメントの上に次の情報を追加する方法もあります。

   Windows: `set SVRCP=%WLI_HOME%\lib\wlpi-worklist.jar;%SVRCP%`
   UNIX: `setenv SVRCP $WLI_HOME/lib/wlpi-worklist.jar;$SVRCP`

2. WebLogic Integration の起動

3. ブラウザを使用して、次のような URL を入力し、JSP ページを呼び出します。

   `http://WebLogicURL:WebLogicPort/example_name`

   たとえば、ご使用の WebLogic Server と同じ NT ホスト上で、ポート 7001 使用して実行されているブラウザで、Worklist サンプルをロードするには、次のように入力してください。

   `http://localhost:7001/worklist`

4. ログインしてサンプルの使用を開始します。

Part IV「実行時の管理」に記載のサンプルの一部は、JSP Worklist サンプルから抜粋されたものです。

2 パッケージおよびインタフェースのインポート

この章では、アプリケーションにインポートできる以下のパッケージとインタフェースについて説明します。

- BPM パッケージおよびインタフェース
- 汎用 Java パッケージ

BPM パッケージおよびインタフェース

設計および実行時クライアント アプリケーションで使用される Business Process Management (BPM) API パッケージおよびインタフェースについて説明し、各パッケージおよびインタフェースのインポートが有効な状況を次の表に示します。

表 2-1 BPM パッケージおよびインタフェース

パッケージまたはインタフェース	内容
com.bea.wlpi.server.admin.Admin	BPM 管理機能をサポートするインタフェース。 ビジネス プロセスのコンフィグレーション時、設計時、モニタ時にインポートする。
com.bea.wlpi.server.catalog.EJBCatalog	アプリケーション サーバにインストールされている EJB (Enterprise Java Bean ―エンタープライズ Java Bean) のカタログをサポートするインタフェース。 ビジネス オペレーションのコンフィグレーション時にインポートする。

表 2-1 BPM パッケージおよびインタフェース (続き)

パッケージまたはインタフェース	内容
com.bea.wlpi.server.permission.Permission	ロールおよびユーザのパーミッションを設定することでセキュリティ コンフィグレーションをサポートするインタフェース。 ロールおよびユーザのパーミッションのコンフィグレーション時にインポートする。
com.bea.wlpi.server.plugin.PluginManager	ワークフロー実行中にプラグインの実行時管理をサポートするインタフェース。 プラグインのプログラミングと PluginManager EJB の詳細については、『*WebLogic Integration BPM プラグイン プログラミング ガイド*』 を参照。
com.bea.wlpi.server.plugin.PluginManagerCfg	ユーザ定義プラグインの実装の管理を可能にするインタフェース。 プラグインのプログラミングと PluginManager EJB の詳細については、『*WebLogic Integration BPM プラグイン プログラミング ガイド*』 を参照。
com.bea.wlpi.server.serverproperties.ServerProperties	BPM サーバ情報へのアクセスと Java システム プロパティのリモート検査を可能にするインタフェース。 **注意 :** クラスタ化された環境では、コンフィグレーションされているルーティング アルゴリズムに従ってメソッド要求がサーバに送信されるので、Java システム プロパティのリモート検査機能が制限されます。メソッド呼び出しによって返される情報は変化します (特に、クラスタ内のシステムの仕様が一意の場合)。 テンプレートやテンプレート定義などの適合性を調べる必要がある場合にインポートする。

表 2-1 BPM パッケージおよびインタフェース (続き)

パッケージまたはインタフェース	内容
com.bea.wlpi.server.principal.WLPIPrincipal	BPM セキュリティ機能をサポートするインタフェース。 オーガニゼーション、ロールおよびユーザのコンフィグレーション時にインポートする。
com.bea.wlpi.server.worklist.Worklist	BPM 実行時管理機能をサポートするインタフェース。 ビジネス プロセスの実行時にインポートする。
com.bea.eci.repository.ejb.XMLRepository	XML リポジトリ データベースをサポートするインタフェース。 XML リポジトリ データベースへのアクセスが必要なときにインポートする。
com.bea.wlpi.client.common.*	共通のクライアントサイド クラス（さまざまなコンビニエンス メソッドなど）を含むパッケージ。 必要に応じてインポートする。
com.bea.wlpi.client.util.*	JMS (Java Message System ― Java メッセージ システム) テスト ユーティリティを含むパッケージ。 JMS のテスト時に必要に応じてインポートする。
com.bea.wlpi.common.*	BPM クライアントと BPM サーバの両方で使用されるクラス（値オブジェクト）を含むパッケージ。 必要に応じてインポートする。
com.bea.wlpi.common.plugin.*	ユーザ定義プラグインの管理に使用されるクラスを含むパッケージ。 ユーザ定義プラグインの管理時にインポートする。 プラグインのプログラミングと PluginManager EJB の詳細については、『*WebLogic Integration BPM プラグイン プログラミング ガイド*』を参照。

表 2-1 BPM パッケージおよびインタフェース (続き)

パッケージまたはインタフェース	内容
com.bea.wlpi.common.security.*	セキュリティ パーミッションの定義に使用される共通クラスを含むパッケージ。 セキュリティ パーミッションの定義時にインポートする。
com.bea.wlpi.util.*	一般的な BPM ユーティリティ（メッセージ駆動型 Bean を生成するユーティリティなど）を含むパッケージ。 メッセージ駆動型 Bean の生成時にインポートする。
com.bea.eci.repository.helper.*	XML リポジトリへのアクセスに使用される共通クラスを含むパッケージ。 XML リポジトリの使用時にインポートする。

汎用 Java パッケージ

設計および実行時クライアント アプリケーションで使用される汎用 Java パッケージと、各パッケージのインポートが有効な状況を次の表に示します。

表 2-2 汎用 Java パッケージ

パッケージ	サポート対象
java.io.*	システム入出力。 ファイルの入出力時にインポートする。
java.lang.*	基本の設計クラス。 必要に応じてインポートする。
java.net.*	ネットワーク アプリケーション実装。 さまざまなネットワーク固有の機能の実行時に必要に応じてインポートする。

表 2-2 汎用 Java パッケージ (続き)

パッケージ	サポート対象
java.sql.*	Java プログラミング言語を使用するデータ アクセス / 処理 API。 データベース固有の機能の実行時に必要に応じてインポートする。
java.text.*	自然言語から独立したテキスト。 必要に応じてインポートする。
java.util.*	日付ユーティリティや時間ユーティリティなどの標準 API。 必要に応じてインポートする。
javax.ejb.*	EJB。 さまざまな EJB オペレーションの実行時に必要に応じてインポートする。
javax.jms.*	JMS。 JMS への接続時にインポートする。
javax.naming.*	サーバおよび送り先のルックアップに必要な JNDI インタフェース。 JDNI ルックアップ実行時にインポートする。
javax.rmi.*	RMI (Remote Method Invocation ― リモート メソッド呼び出し)。 リモート メソッド呼び出しに関連する例外や、RMI 関連のその他の会話などをサポートするために必要に応じてインポートする。
javax.xml.parsers.*	JAXP (Java API for XML parsing ― XML 解析用 Java API)。 JAXP API は、SAX (Simple API for XML: XML 用の簡単な API) も DOM API も置換しない。代わりに、SAX および DOM API を使いやすくするコンビニエンス メソッドを追加する。 XML の解析時に必要に応じてインポートする。

表 2-2 汎用 Java パッケージ (続き)

パッケージ	サポート対象
`org.xml.sax.*`	SAX インタフェース。 XML の解析時に必要に応じてインポートする。
`weblogic.apache.xerces.parsers.*`	Apache Xerces XML パーサ。 XML の解析時に必要に応じてインポートする。

ユーザ インタフェースの要件によっては、次の表に記載のパッケージをインポートすると便利な場合もあります。

サポート対象	インポートするパッケージ
Java アプレット	`java.applet.*`
Java ユーザ インタフェース	`java.awt.*`
Java Swing GUI アプリケーション	`javax.swing.*`

3　プロセス エンジンへの接続

この章では、WebLogic Integration プロセス エンジンに接続して Business Process Management (BPM) フレームワークの機能にアクセスする方法について説明します。この章の内容は以下のとおりです。

- API セッション EJB へのアクセス
- コンビニエンス メソッドを使用した EJB へのアクセス

API セッション EJB へのアクセス

EJB はすべて、ホーム インタフェースとリモート インタフェースを使用して BPM API セッション EJB にアクセスする必要があります。この目的で使用できるホーム インタフェースとリモート インタフェースを次の表に示します。

表 3-1 API セッション EJB のホーム インタフェースとリモート インタフェース

EJB 名	ホーム インタフェース	リモート インタフェース
com.bea.wlpi.server.admin.Admin	AdminHome	Admin
com.bea.wlpi.server.audit.Audit	AuditHome	Audit
com.bea.wlpi.server.catalog.EJBCatalog	EJBCatalogHome	EJBCatalog
com.bea.wlpi.server.permission.Permission	PermissionHome	Permission
com.bea.wlpi.server.plugin.PluginManager	PluginManagerHome	PluginManager
com.bea.wlpi.server.plugin.PluginManagerCfg	PluginManagerCfgHome	PluginManagerCfg

表 3-1 API セッション EJB のホーム インタフェースとリモート インタフェース (続き)

EJB 名	ホーム インタフェース	リモート インタフェース
com.bea.wlpi.server.serverproperties.ServerProperties	ServerPropertiesHome	ServerProperties
com.bea.wlpi.server.principal.WLPIPrincipal	WLPIPrincipalHome	WLPIPrincipal
com.bea.wlpi.server.worklist.Worklist	WorklistHome	Worklist
com.bea.eci.repository.ejb.XMLRepository	XMLRespositoryHome	XMLRepository

BPM API セッション EJB およびそのメソッドにアクセスする手順は、次のとおりです。

1. セッション EJB のホーム インタフェースを JNDI でルックアップします。
2. そのホーム インタフェースを使用してリモート セッション オブジェクト（EJBObject）を作成します。

以下の節では、この 2 つの手順を詳しく説明します。

手順 1 ーセッション EJB のホーム インタフェースを JNDI でルックアップする

BPM セッション EJB は、WebLogic Server JNDI ネームスペースの一部としてクライアント アプリケーションにエクスポーズされます。

セッション EJB のホーム インタフェースをルックアップするには、まず、JNDI コンテキスト（java.naming.Context）を構築します。この最も一般的な方法は、javax.naming.InitialContext オブジェクトをインスタンス化することです。EJB とそのメソッドにアクセスするために認可用の特定のセキュリティ コンテキストを必要とするクライアント アプリケーションに対しては、セキュリティ証明（ユーザ名やパスワードなど）も InitialContext() コンストラクタに渡す必要があります。

たとえば、JSP Worklist サンプルからの抜粋である以下のメソッドでは、初期コンテキストが作成され、指定の URL、ユーザ ID、およびパスワードがコンテキスト環境として渡されます。

```
public Context getInitialContext(
  String user,
  String password,
  String url
) throws NamingException
{
     // 初期コンテキストを取得する
     Properties h = new Properties();
     h.put(Context.INITIAL_CONTEXT_FACTORY,"weblogic.jndi.WLInitialContextFactory");
     h.put(Context.PROVIDER_URL, url);
     if (user != null) {
         h.put(Context.SECURITY_PRINCIPAL, user);
         if (password == null)
           password = "";
     }
         h.put(Context.SECURITY_CREDENTIALS, password);

     return new InitialContext(h);
}
```

詳細については、Javadoc の `javax.naming.InitialContext()` を参照してください。JSP Worklist の詳細については、1-27 ページの「JSP Worklist サンプル」を参照してください。

JNDI コンテキストの定義後は、JNDI コンテキスト `lookup()` メソッドを使用してセッション EJB のホーム インタフェースにアクセスできます。

たとえば、`Worklist` セッション EJB のホーム インタフェースをルックアップしてオブジェクト参照を `worklistHome` 変数に格納するには、次の文を実行します。

```
Context context = getInitialContext(url, userId, password);
Object result = context.lookup("com.bea.wlpi.Worklist");
WorklistHome worklistHome =(WorklistHome)
  PortableRemoteObject.narrow(result, WorklistHome.class);
```

同様に、`WLPIPrincipal` セッション EJB のホーム インタフェースをルックアップしてオブジェクト参照を `principalHome` 変数に格納するには、次の文を実行します。

```
Context context = getInitialContext(url, userId, password);
object result = context.lookup("com.bea.wlpi.WLPIPrincipal");
PrincipalHome prinicipalHome=(PrincipalHome)
```

```
PortableRemoteObject.narrow(result,
WLPIPrinicipalHome.class);
```

注意： 上記のサンプルで使用されている `PortableRemoteObject.narrow()` メソッドにより、リモート オブジェクトが目的のタイプに確実にキャストされます。このメソッドは、システムが RMI-IIOP を使用するようコンフィグレーションされている場合に使用することが必須となります。これ以外の場合は、任意ですが、使用することをお勧めします。詳細については、Javadoc の `javax.rmi.PortableRemoteObject` クラスを参照してください。

手順 2 ーホーム インタフェースを使用してリモート セッション オブジェクトを作成する

ホーム インタフェースへの参照の取得後は、ホーム インタフェース オブジェクト `create()` メソッドを使用してリモート セッション オブジェクト（`EJBObject`）にアクセスできます。

たとえば、`Worklist` セッション EJB の `EJBObject` を作成してオブジェクト参照を `worklist` 変数に格納するには、次の文を実行します。

```
Worklist worklist = worklistHome.create();
```

同様に、`WLPIPrincipal` セッション EJB の `EJBObject` を作成してオブジェクト参照を `principal` 変数に格納するには、次の文を実行します。

```
WLPIPrincipal wlpiprincipal = principalHome.create();
```

コンビニエンス メソッドを使用した EJB へのアクセス

com.bea.wlpi.client.common.WLPI クラスには、セッション EJB にアクセスするためのコンビニエンス メソッド群が含まれています。これらのメソッドを次の表にまとめます。

表 3-2 EJB にアクセスするためのコンビニエンス メソッド

メソッド	説明
`public com.bea.wlpi.server.admin.Admin getAdmin( ) throws java.lang.IllegalStateException, com.bea.wlpi.common.WorkflowException`	Admin EJB オブジェクトを返す。
`public com.bea.wlpi.server.catalog.EJBCatalog getCatalog( ) throws java.lang.IllegalStateException, com.bea.wlpi.common.WorkflowException`	EJBCatalog EJB オブジェクトを返す。
`public com.bea.wlpi.server.permission.Permission getPermission( ) throws java.lang.IllegalStateException, com.bea.wlpi.common.WorkflowException`	Permission EJB オブジェクトを返す。
`public com.bea.wlpi.server.plugin.PluginManager getPluginManager( ) throws java.lang.IllegalStateException`	PluginManager EJB オブジェクトを返す。
`public com.bea.wlpi.server.serverproperties.ServerProperties getServerProperties( ) throws IllegalStateException, WorkflowException`	ServerProperties EJB オブジェクトを返す。
`public com.bea.wlpi.server.principal.WLPIPrincipal getPrincipal( ) throws java.lang.IllegalStateException`	WLPIPrincipal EJB オブジェクトを返す。

表 3-2 EJB にアクセスするためのコンビニエンス メソッド (続き)

メソッド	説明
`public com.bea.wlpi.server.worklist.Worklist getWorklist( ) throws java.lang.IllegalStateException, com.bea.wlpi.common.WorkflowException`	`Worklist` EJB オブジェクトを返す。
`public com.bea.eci.repository.ejb.XMLRepository getRepository( ) throws java.lang.IllegalStateException, com.bea.wlpi.common.WorkflowException`	`XMLRepository` EJB オブジェクトを返す。

表 3-2 EJB にアクセスするためのコンビニエンス メソッド (続き)

メソッド	説明

`public boolean connect(`
`  java.awt.Frame owner`
`  java.lang.String url,`
`  java.lang.String userId,`
`  java.lang.String password`
`) throws java.lang.IllegalStateException`
`public boolean connect(`
`  java.awt.Frame owner`
`  java.lang.String url,`
`  java.lang.String userId,`
`  java.lang.String password,`
`  int maxRetries,`
`  boolean autoLogon,`
`  boolean graphical`
`) throws java.lang.IllegalStateException`
	以下のタスクを実行してプロセス エンジンに接続する。
	■ WLPIPrincipal セッション EJB にアクセスする。
	■ ユーザ ID と URL を環境内に設定する。
	■ ログインが失敗した場合に自動再試行を 3 回実行する。
	指定が必要な引数は以下のとおり。
	■ *owner* ― メッセージ ボックスとダイアログ ボックスが表示されるウィンドウ
	■ *url* ― BPM URL
	■ *userId* ―ユーザ ID
	■ *password* ―パスワード
	■ *maxRetries* ―許可される再試行の最大回数
	■ *autoLogon* ―提供されている値を使用してログオンする（`true`）のか、ユーザに確認を求める（`false`）のかを指定するフラグ
	■ *graphical* ―ユーザとの会話にグラフィックスを使用する（`true`）のか、しない（`false`）のかを指定するフラグ
	ユーザ ID、パスワード、サーバ URL は、これらの情報をユーザに要求して取得する。操作を簡単にするため、ログイン ダイアログの生成とユーザ入力の取り込みに使用できる `client.common.logon` クラスが用意されている。このクラスの詳細については、Javadoc の `com.bea.wlpi.client.common.logon` を参照。

表 3-2 EJB にアクセスするためのコンビニエンス メソッド (続き)

メソッド	説明
`public boolean isConnected()`	`WLPI` インスタンスが接続されているかどうか、つまり `WLPIPrincipal` セッション EJB への参照が作成されていることを確認する。
`public javax.naming.Context getInitialContext()`	ホスト アプリケーションの JNDI コンテキストを返す。

詳細については、Javadoc の `com.bea.wlpi.client.common.WLPI` を参照してください。

4 プロセス エンジン情報へのアクセス

この章では、WebLogic Integration プロセス エンジンに関する情報にアクセスする方法について説明します。この章の内容は以下のとおりです。

- サーバ バージョンの取得
- パッケージ バージョンの取得
- テンプレート定義バージョンの取得
- サーバ プロパティの取得
- コンビニエンス メソッドの使い方
- プロセス エンジンに関する情報へのアクセス例

この章に記載されているメソッドの詳細については、Javadoc の `com.bea.wlpi.server.serverproperties.ServerProperties` または `com.bea.wlpi.client.common.WLPI` を参照してください。

サーバ バージョンの取得

サーバ バージョンを取得するには、次の `com.bea.wlpi.server.serverproperties.ServerProperties` メソッドを使用します。

```
public com.bea.wlpi.common.VersionInfo getServerVersion(
) throws java.rmi.RemoteException
```

このメソッドにより、サーバ バージョンが `VersionInfo` オブジェクトとして返されます。バージョンに関する情報にアクセスするには、B-31 ページの「VersionInfo オブジェクト」 に記載の `VersionInfo` オブジェクト メソッドを使用します。

たとえば、次のコードは、バージョンを取得して version オブジェクトに保存します。このコード例では properties は ServerProperties EJB への EJBObject 参照を表しています。

```
VersionInfo version = properties.getServerVersion();
```

getServerVersion() メソッドの詳細については、Javadoc の com.bea.wlpi.server.serverproperties.ServerProperties を参照してください。

パッケージ バージョンの取得

サーバがサポートするパッケージ バージョンを取得するには、次の com.bea.wlpi.server.serverproperties.ServerProperties メソッドを使用します。

```
public com.bea.wlpi.common.VersionInfo getPackageVersion(
) throws java.rmi.RemoteException
```

このメソッドにより、パッケージ バージョンが VersionInfo オブジェクトとして返されます。バージョンに関する情報にアクセスするには、B-31 ページの「VersionInfo オブジェクト」に記載の VersionInfo オブジェクト メソッドを使用します。

たとえば、次のコードは、パッケージ バージョンを取得して version オブジェクトに保存します。このコード例では properties は ServerProperties EJB への EJBObject 参照を表しています。

```
VersionInfo version = properties.getPackageVersion();
```

getPackageVersion() メソッドの詳細については、Javadoc の com.bea.wlpi.server.serverproperties.ServerProperties を参照してください。

テンプレート定義バージョンの取得

サーバがサポートするテンプレート定義バージョンを取得するには、次の com.bea.wlpi.server.serverproperties.ServerProperties メソッドを使用します。

```
public com.bea.wlpi.common.VersionInfo getTemplateDefinitionVersion(
) throws java.rmi.RemoteException
```

このメソッドによって、サーバがサポートするテンプレート定義バージョンが VersionInfo オブジェクトとして返されます。バージョンに関する情報にアクセスするには、B-31 ページの「VersionInfo オブジェクト」 に記載の VersionInfo オブジェクト メソッドを使用します。

たとえば次のコードは、サーバがサポートするテンプレート定義バージョンを取得して version オブジェクトに保存します。このコード例では properties は ServerProperties EJB への EJBObject 参照を表しています。

```
VersionInfo version = properties.getTemplateDefinitionVersion();
```

getTemplateDefinitionVersion() メソッドの詳細については、Javadoc の com.bea.wlpi.server.serverproperties.ServerProperties を参照してください。

サーバ プロパティの取得

サーバ プロパティを取得するには、次の com.bea.wlpi.server.serverproperties.ServerProperties メソッドを使用します。

```
public java.util.Properties getProperties(
) throws java.rmi.RemoteException
```

このメソッドにより、Java システム プロパティやその他の定義済みプロパティを含むサーバ プロパティが返されます。

注意： クラスタ化された環境では、コンフィグレーションされているルーティング アルゴリズムに従ってメソッド要求がサーバに送信されるので、Java システム プロパティのリモート検査機能が制限されます。メソッド呼び出しによって返される情報は変化します（特に、クラスタ内のシステムの仕様が一意の場合）。

たとえば次のコードは、サーバ プロパティを取得して props オブジェクトに保存します。このコード例では properties は ServerProperties EJB への EJBObject 参照を表しています。

```
Properties props = properties.getProperties();
```

getProperties() メソッドの詳細については、Javadoc の com.bea.wlpi.server.serverproperties.ServerProperties を参照してください。

コンビニエンス メソッドの使い方

com.bea.wlpi.client.common.WLPI クラスには、サーバ情報にアクセスするためのコンビニエンス メソッド群が含まれています。これらのメソッドを次の表にまとめます。

表 4-1 サーバ情報にアクセスするためのコンビニエンス メソッド

メソッド	説明
`public com.bea.wlpi.common.VersionInfo getServerVersion( ) throws java.lang.IllegalStateException, com.bea.wlpi.common.WorkflowException`	ServerProperties セッション EJB にアクセスして BPM サーバ バージョンを返す。このメソッドによって、テンプレートに対応する com.bea.wlpi.common.VersionInfo オブジェクトが返される。バージョンに関する情報にアクセスするには、B-31 ページの「VersionInfo オブジェクト」に記載の VersionInfo オブジェクト メソッドを使用する。

表 4-1 サーバ情報にアクセスするためのコンビニエンス メソッド (続き)

メソッド	説明
`public com.bea.wlpi.common.VersionInfo getServerTemplateDefinitionVersion( ) throws java.lang.IllegalStateException, com.bea.wlpi.common.WorkflowException`	`ServerProperties` セッション EJB にアクセスし、現バージョンの WebLogic Integration でサポートされているテンプレート定義バージョンを返す。このメソッドによって、テンプレートに対応する `com.bea.wlpi.common.VersionInfo` オブジェクトが返される。バージョンに関する情報にアクセスするには、B-31 ページの「VersionInfo オブジェクト」 に記載の `VersionInfo` オブジェクト メソッドを使用する。

詳細については、Javadoc の `com.bea.wlpi.client.common.WLPI` を参照してください。

プロセス エンジンに関する情報へのアクセス例

この節では、コマンドライン管理クライアント サンプルから抜粋して、サーバ プロパティへのアクセス方法を示します。

ユーザと通信を行うための入力ストリームが定義されており、ユーザはどのアクションを実行するかを指定するよう求められます。ユーザが `Server Properties` オプションを選択した場合、システムによってサーバ バージョンとテンプレート定義バージョンが取得され、返された `VersionInfo` オブジェクトにより、サーバとテンプレート定義に関する他の情報（名前、メジャー / マイナー / ビルド バージョンなど）が取得されます。さらに、サーバ プロパティの名前と値も表示されます。

重要なコード行は、**太字**で強調されています。このサンプルでは、文字列 `serverProperties` は `ServerProperties` EJB への `EJBObject` 参照を表しています。

```
/* ツール タイトルを表示する */
   System.out.print( "\n---  Command Line Administration v1.1  ---" );

/* メイン メニューを表示してユーザと交信する */
while( true ) {
/* メニューを表示する */
   System.out.println( "\n---              Main Menu            ---" );
   System.out.println( "\nEnter choice:" );
   System.out.println( "1) Organizations" );
   System.out.println( "2) Roles" );
   System.out.println( "3) Users" );
   System.out.println( "4) Security Realm" );
   System.out.println( "5) Business Operations" );
   System.out.println( "6) Event Keys" );
   System.out.println( "7) Business Calendars" );
   System.out.println( "8) EJB Catalog" );
   System.out.println( "9) Server Properties" );
   System.out.println( "Q) Quit" );
   System.out.print( ">> " );
      .
      .
      .

/**
 * ServerProperties インタフェースで使用可能な
 * 公開 API メソッドを示すメソッド（ユーザとの交信なし、表示モードのみ）
 */
public static void mngServerProperties( ) {
   Properties serverSystemProperties;
   String answer;

   try {
      /* システム プロパティの表示が必要かどうかをユーザに求める */
      if( ( answer = askQuestion( "\nList system properties (y/n)?" ) )
         == null ) {
         /* ユーザによる操作の取り消し */
         System.out.println( "*** Cancelled" );
         return;
      }

      /* 応答を解析する */
      boolean isListSysProperties = ( answer.equals( "y" ) ||
         answer.equals( "Y" ) );
      /* すべてのサーバ プロパティと属性を表示する */
      System.out.println( "\nServer Version:" );

      /* WLPI 公開 API メソッド */
      /* サーバ属性を取り出す */
```

```
      serverVersion = serverProperties.getServerVersion( );

      /* WLPI v1.2.1 で使用可能な全属性を表示する */
      System.out.println( "- Release Name: " + serverVersion.getName( ) );
      System.out.println( "- Version: " + serverVersion +
         " (Major=" + serverVersion.getMajorVersion( ) +
         " Minor=" + serverVersion.getMinorVersion( ) +
         " Build=" + serverVersion.getBuild( ) + ")" );

      /* すべてのテンプレート定義プロパティと属性を表示する */
      System.out.println( "\nTemplate Definition version supported:" );

      /* WLPI 公開 API メソッド */
      /* テンプレート定義属性を取り出す */
      serverDTDVersion = serverProperties.getTemplateDefinitionVersion( );

      /* 使用可能なすべての属性を表示する */
      System.out.println( "- Release Name: " + serverDTDVersion.getName( ) );
      System.out.println( "- Version: " + serverDTDVersion +
         " (Major=" + serverDTDVersion.getMajorVersion( ) +
         " Minor=" + serverDTDVersion.getMinorVersion( ) +
         " Build=" + serverDTDVersion.getBuild( ) + ")" );

      if( isListSysProperties ) {
         System.out.println( "\nSystem Properties: " );



      /* WLPI 公開 API メソッド */
      /* サーバ システム プロパティを取り出す */
      serverSystemProperties = serverProperties.getProperties( );

      /* すべてのプロパティを表示するループ */
      for( Enumeration e = serverSystemProperties.propertyNames( );
         e.hasMoreElements( ); ) {
         String propertyName = e.nextElement( ).toString( );
         System.out.println( "- Name: '" + propertyName + "' Value: '" +
         serverSystemProperties.getProperty( propertyName ) + "'\n" );
      }
   }
}
   catch( Exception e ) {
      System.out.println( "*** Failed to retrieve properties" );
      System.err.println( e );
   }
   return;
}
```

5　値オブジェクトの使い方

この章では、Business Process Management (BPM) 値オブジェクトを使用してオブジェクト データにアクセスする方法について説明します。この章の内容は以下のとおりです。

- 値オブジェクトの概要
- 値オブジェクトの作成
- 値オブジェクトによるオブジェクト データへのアクセス
- 値オブジェクトのソート
- 値オブジェクトの使用例

値オブジェクトの概要

`com.bea.wlpi.common`、`com.bea.wlpi.common.security`、および `com.bea.eci.repository.helper` という 3 つの BPM パッケージには、定義時と実行時にオブジェクト データを取得するためのクラス、すなわち*値オブジェクト*が含まれています。これらの各パッケージの詳細については、1-13 ページの「BPM API」を参照してください。

各値オブジェクトには、以下の共通の特性があります。

- セッション EJB (テンプレート、テンプレート定義、ビジネス カレンダーなど) や内部で使用されるエンティティ EJB を含むさまざまな BPM サーバサイド オブジェクトを保守します。また、これらのオブジェクトからのデータ値の取得を可能にします。
- 別々の Java クラスにより表されます（クラスのメンバーはまとめて*値*と呼ばれる）。
- シリアライズ可能で、クライアントとサーバ間で交換できます。

- 同じタイプの 2 つのオブジェクトの等価チェックを行うために `equals()` メソッドを以下のとおりオーバーライドします。

  ```
  public boolean equals(Object obj)
  ```

- 同じタイプの 2 つのオブジェクトを比較するために `java.lang.comparable` インタフェースを次のとおり実装します。

  ```
  public int compareTo(Object obj)
  ```

- 均一リストの一部である場合、クラスは以下のメソッドで検索およびソートできます。
  - `java.util.Collection.contains(Object o)`
  - `java.util.List.indexOf(Object o)`
  - `java.util.Collections.sort(List list)`
  - `java.util.Collections.sort(List list, Comparator c)`

  これらのクラスの詳細については、次の Sun Microsystems 社の Java Web サイトを参照してください。

  `http://java.sun.com`

- オブジェクトの自然順序（`boolean compareTo(Object o)` メソッドにより実装される）が `boolean equals(Object o)` メソッドで使用されるのと同じフィールドに基づく場合は、次のメソッド

  ```
  int java.util.Collections.binarySearch(List list, Object o)
  ```

  によって、以前に `java.util.Collections.sort(List list)` メソッドでソートされたリストをすばやく検索できます。

- オブジェクト データのインポートとエクスポートがサポートされている場合は、`com.bea.wlpi.common.Publishable` インタフェースを実装します。

  注意： データのインポートとエクスポートがサポートされている場合は、`com.bea.wlpi.common.Publishable` インタフェースも実装します。詳細については、18-1 ページの「ワークフロー オブジェクトの発行」を参照してください。

オブジェクト データへのアクセスに使用できる値オブジェクトを次の表に示します。

表 5-1 値オブジェクト

使用する値オブジェクト	アクセス対象
com.bea.wlpi.common.BusinessCalendarInfo	ビジネス カレンダー データ
com.bea.wlpi.common.EventKeyInfo	イベント キー データ
com.bea.wlpi.common.InstanceInfo	ワークフロー インスタンス データ
com.bea.wlpi.common.OrganizationInfo	オーガニゼーション データ
com.bea.wlpi.common.security.PermissionInfo	パーミッション データ
com.bea.wlpi.common.RepositoryFolderHelperInfo	XML リポジトリ フォルダ データ
com.bea.eci.repository.helper.RepositoryFolderInfo	XML リポジトリ フォルダ データ
com.bea.wlpi.common.RerouteInfo	タスク再ルーティング データ
com.bea.wlpi.common.RoleInfo	ロール データ
com.bea.wlpi.common.RolePermissionInfo	ロール パーミッション データ
com.bea.wlpi.common.TaskInfo	ワークフロー タスク データ
com.bea.wlpi.common.TemplateDefinitionInfo	テンプレート定義データ
com.bea.wlpi.common.TemplateInfo	テンプレート データ
com.bea.wlpi.common.UserInfo	ユーザ データ
com.bea.wlpi.common.security.UserPermissionInfo	ユーザ パーミッション データ
com.bea.wlpi.common.VariableInfo	変数データ
com.bea.wlpi.common.VersionInfo	バージョン番号データ
com.bea.wlpi.common.XMLEntityHelperInfo	XML リポジトリ エンティティ データ

表 5-1 値オブジェクト (続き)

使用する値オブジェクト	アクセス対象
com.bea.eci.repository.helper.XMLEntityInfo	XML リポジトリ エンティティ データ

値オブジェクトの作成

値オブジェクトを作成するには、関連付けられているコンストラクタを使用します。5-3 ページの「値オブジェクト」の表に記載のそれぞれの BPM 値オブジェクトには、オブジェクト データ作成用のコンストラクタが任意の数含まれています。値オブジェクトを作成するコンストラクタは、B-1 ページの「値オブジェクトのまとめ」に記載しています。

たとえば、次のコードでは OrganizationInfo オブジェクトが作成され、オーガニゼーション ID が ORG1 に設定され、生成されたオブジェクトが organization に割り当てられます。

```
OrganizationInfo organization = new OrganizationInfo(“ORG1”);
```

値オブジェクトによるオブジェクト データへのアクセス

5-3 ページの「値オブジェクト」の表に記載のそれぞれの BPM 値オブジェクトには、オブジェクト データにアクセスするためのさまざまなメソッドが含まれています。各値オブジェクトのオブジェクト データを取得および設定するメソッドは、B-1 ページの「値オブジェクトのまとめ」に記載されています。

たとえば、次のコードは、指定した OrganizationInfo オブジェクト、organization のオーガニゼーション ID を取得します。

```
String id = getOrgId(organization);
```

値オブジェクトのソート

5-1 ページの「値オブジェクトの概要」に記載のとおり、`java.util.Collections` クラスに使用可能な `sort()` メソッドを使用すると、`java.lang.Comparable` インタフェースを実装する値オブジェクトのリストを以下のようにソートすることが可能です。

- `java.util.Collections.sort(java.util.List` *`list`*`)`
- `java.util.Collections.sort(java.util.List` *`list`*`, java.util.Comparator` *`c`*`)`

第 1 のメソッドでは、`java.lang.Comparable` の場合と同様に、指定の要素リストが要素の *自然順序*に基づいて昇順にソートされます。

`http://java.sun.com/j2se/1.3/docs/api/java/lang/Comparable.html`

第 2 のメソッドでは、デフォルト以外のキーに基づいたソートを行うカスタム演算子を使用して、指定の要素リストがソートされます。

注意： `com.bea.wlpi.client.common.SortTableModel` クラスには、JTable のカラム ヘッダをクリックすることでカラム内の値オブジェクト データをソートするメソッド群が含まれています。詳細については、Javadoc の `com.bea.wlpi.client.common.SortableTableModel` を参照してください。

値オブジェクトの使用例

この節では、コマンドライン Worklist サンプルから抜粋して、値オブジェクトを使用してタスク オブジェクト データにアクセスする方法を示します。また、`TaskInfo` クラス メソッドを使用してタスク名、テンプレート定義 ID、インスタンス ID、およびタスク ID を取得する方法も示されています。

get アクションは**太字**で強調されています。

```
/* 割り当て済みのタスクがあるか ? */
if( taskList.size( ) == 0 )
   System.out.println( "\nNo task assigned" );
else
```

```
      System.out.print( "\nAssigned Tasks:" );

   /* リストを処理して、タスクを表示する */
   for( int i = 0; i < taskList.size( ); i++ ) {
      /* リストから要素を取り出す */
      TaskInfo taskInfo = ( TaskInfo )taskList.get( i );

      /* WLPI 公開 API メソッド */
      /* 使用可能な属性のサブセットを取り出して表示する */
      System.out.println( "\n- Name: " + taskInfo.getName( ) );
      System.out.println( "  Template Definition ID: " +
         taskInfo.getTemplateDefinitionId( ) );
      System.out.println( "  Workflow Instance ID: " +
         taskInfo.getInstanceId( ) );
      System.out.println( "  Task ID: " + taskInfo.getTaskId( ) );
      System.out.print( "  Status: " + taskInfo.getStatus( ) );

      /* タスク状態を取り出して表示する */
      if( taskInfo.getStatus( ) == taskInfo.STATUS_PENDING )
         System.out.println( " (Pending)" );
      else if( taskInfo.getStatus( ) == taskInfo.STATUS_COMPLETE )
         System.out.println( " (Complete)" );
      else if( taskInfo.getStatus( ) == taskInfo.STATUS_OVERDUE )
         System.out.println( " (Overdue)" );
      else if( taskInfo.getStatus( ) == taskInfo.STATUS_INACTIVE )
         System.out.println( " (Inactive)" );
      else
         System.out.println( " (Unknown)" );
   }
   break;|
   .
   .
   .
```

6 JMS コネクタの確立

この章では、JMS (Java Message Service: Java メッセージ サービス) コネクタの確立方法について説明します。この章の内容は以下のとおりです。

- JMS の概要
- プロセス エンジンで使用される JMS 送り先
- JMS への接続
- メッセージの非同期受信
- 複数のイベント キューに対するメッセージ駆動型 Bean の生成
- メッセージ配信の保証
- メッセージの順次処理の保証
- JMS トピックへの接続例

注意 : JMS の詳細については、次の URL の BEA WebLogic Server ドキュメント群に含まれている『*WebLogic JMS プログラマーズ ガイド*』を参照してください。

http://edocs.beasys.co.jp/e-docs/wls/docs70/jms/index.html

または、Sun Microsystems 社提供の次の URL の *JavaSoft JMS specification version 1.0.2* を参照してください。

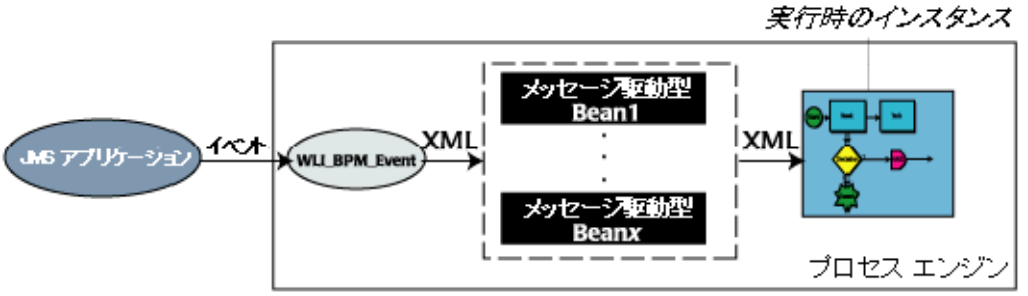
http://www.javasoft.com/products/jms/docs.html

JMS の概要

WebLogic Integration プロセス エンジンと Business Process Management (BPM) フレームワークが実行されている WebLogic Server によって JMS が実装され、JMS が XML コンテンツの伝送をサポートします。プロセス エンジンは WebLogic JMS を使用してワークリスト、時間、イベント通知、およびエラー / 監査メッセージの通信を行います。

実行中の BPM ワークフロー インスタンスと外部クライアント アプリケーション間で、JMS が XML メッセージを使用して通信を可能にする方法を次の図に示します。

図 6-1 JMS の概要



この図に示すように、JMS アプリケーションによって発行された XML イベントは以下のとおり処理されます。

1. 事前定義済みのイベント キュー（以下の節で説明する `WLI_BPM_Event` など）に配信されます。
2. 適切なメッセージ駆動型 Bean に送信されて処理されます。
3. 実行中のインスタンスに配信されます。

プロセス エンジンによって使用される JMS 送り先についての説明と、送り先を接続して使用する方法を以下の節で紹介します。また、JMS トピック `WLI_BPM_Notify` に接続する方法を示したコード例についても説明します。

JMS の詳細については、次の URL の BEA WebLogic Server ドキュメント群に含まれている『*WebLogic JMS プログラマーズ ガイド*』を参照してください。

http://edocs.beasys.co.jp/e-docs/wls/docs70/jms/index.html

プロセス エンジンで使用される JMS 送り先

プロセス エンジンで使用される JMS 送り先（キューとトピック）を次の表に示します。

表 6-1 プロセス エンジンで使用される JMS 送り先

JMS 送り先	JMS のタイプ	送られる項目
名前 : WLI_BPM_Audit JNDI 名 : com.bea.wli.bpm.Audit	トピック	監査メッセージ。
名前 : WLI_BPM_Error JNDI 名 : com.bea.wli.bpm.Error	トピック	エラー メッセージ。 詳細については、24-1 ページの「ワークフロー例外のモニタリング」を参照

表 6-1 プロセス エンジンで使用される JMS 送り先 (続き)

JMS 送り先	JMS のタイプ	送られる項目
名前 : `WLI_BPM_Event` JNDI 名 : `com.bea.wli.bpm.Event`	キュー	イベント。 すべての入力メッセージはメッセージ駆動型 Bean により処理される。メッセージ駆動型 Bean とは、WebLogic JMS メッセージ システム内でメッセージ コンシューマとして機能する EJB のことである。標準 JMS メッセージ コンシューマと同様に、メッセージ駆動型 Bean はメッセージを JMS キューまたはトピックから受信し、メッセージのコンテンツに基づいてビジネス ロジックを実行する。さらに、メッセージ駆動型 Bean は JMS 送り先の並行処理もサポートする。 システム管理者は、必要に応じて複数のイベント キューを定義できる (6-6 ページの「JMS への接続」参照)。この場合、システム管理者は 6-9 ページの「複数のイベント キューに対するメッセージ駆動型 Bean の生成」に記載のユーティリティを使用して、関連付けられているメッセージ駆動型 Bean も生成する必要があり、アプリケーション開発者は特定のイベントに対する適切な送り先を指定する必要がある。 6-11 ページの「メッセージ配信の保証」と 6-13 ページの「メッセージの順次処理の保証」に記載のメソッドを使用して、メッセージの配信と順次処理を適切に実行できる。 **注意 :** 下位互換性を保つため、`TopicRouter` Bean は、WebLogic Process Integrator リリース 1.2.1 以前でサポートされている `wlpiEvent` トピックを宛先とするすべてのメッセージを `WLI_BPM_Event` キューに自動的に再ルーティングします。

表 6-1 プロセス エンジンで使用される JMS 送り先 (続き)

JMS 送り先	JMS のタイプ	送られる項目
名前 : `WLI_BPM_Notify` JNDI 名 : `com.bea.wli.bpm.Notify`	トピック	Worklist 通知。 Worklist クライアントによる表示の動的な更新と、配信された `com.bea.wlpi.common.TaskInfo` オブジェクトを使用したタスク オブジェクト データの取得を可能にする。 `WLI_BPM_Notify` トピックにポスティングされたメッセージに対しては、適切なサブスクライバに確実に配信するために以下のメッセージ プロパティを定義する必要がある。 ■ `orgId` ―オーガニゼーションの ID を指定する文字列 ■ `assigneeId` ―*割り当て先*（ユーザまたはロール）の ID を指定する文字列 ■ `role` ―割り当て先がロールかどうかを指定するブール フラグ（`true` または `false`） ■ `action` ―以下のいずれかのアクションがタスクに対して実行されたことを指定する文字列 - `assigned` ―ユーザに割り当てられたタスク - `update` ―タスク状態の変更 - `remove` ―ユーザ タスクのリストから削除されるタスク Worklist クライアントは、メッセージをそのプロパティに基づいてフィルタ処理するセレクタを定義できる。セレクタを使用してメッセージをフィルタ処理するコード例については、6-16 ページの「JMS トピックへの接続例」 を参照。 JMS の詳細については、次の URL から BEA WebLogic Server ドキュメント内の『*WebLogic JMS プログラマーズ ガイド*』を参照 `http://edocs.beasys.co.jp/e-docs/wls/docs70/jms/index.html`

表 6-1 プロセス エンジンで使用される JMS 送り先 (続き)

JMS 送り先	JMS のタイプ	送られる項目
名前 : `WLI_BPM_Timer` JNDI 名 : `com.bea.wli.bpm.Timer`	キュー	タイム プロセッサ トリガ（内部使用のみ）

名前 :
`WLI_BPM_ValidatingEvent`
JNDI 名 :
`com.bea.wli.bpm.ValidatingEvent`
	キュー	DTD または スキーマ検証を必要とするイベント
		XML メッセージが DTD またはスキーマを参照する場合は、コンテンツを検証する必要があります。

名前 : `WLI_FailedEvent` JNDI 名 : `com.bea.wli.FailedEvent`	キュー	正常に送信されなかったメッセージで、再試行回数の最大値に達したもの。 再試行回数の最大値は `Redelivery-Limit` 属性を使用して JMS 宛先ごとに設定されている。詳細については、次の URL の『*BEA WebLogic Server コンフィグレーション リファレンス*』にある JMS テンプレート要素の説明を参照 http://edocs.beasys.co.jp/e-docs/wls/docs70/config_xml/index.html 失敗したメッセージを受信した場合は、エントリがログ ファイルに記録されます。必要に応じて、カスタム イベント ハンドラを開発して失敗したメッセージを処理することもできます。

JMS への接続

6-3 ページの「プロセス エンジンで使用される JMS 送り先」の表に定義されているいずれかの JMS 送り先に接続するには（および、XML メッセージをポスティングして受信できるようにするには）、WebLogic Server 管理者が各送り先に対して以下の手順を実行する必要があります。

1. JNDI で JMS コネクタ ファクトリをルック アップします。

システム管理者が任意の数のコネクタ ファクトリを定義してコンフィグレーションすると、起動中にそのファクトリが WebLogic Server によって JNDI スペースに追加されます。コネクタ ファクトリにより、コネクタ コンフィグレーション情報がカプセル化され、JMS アプリケーションが接続を作成できます。

2. コネクタ ファクトリを使用してコネクタを作成します。

   コネクタとは、アプリケーションとメッセージ システム間のオープンな通信チャネルのことです。接続は、メッセージを生成して使用するセッションの作成に使われます。

3. コネクタを使用してセッションを作成します。

   セッションにより、メッセージが生成および使用される順序が定義され、複数のメッセージ プロデューサとメッセージ コンシューマの作成が可能になります。

4. JNDI で送り先をルック アップします。

   送り先はキューかトピックのどちらかになり、特定のプロバイダのアドレス構文をカプセル化します。管理者が送り先を定義してコンフィグレーションすると、起動中にその送り先が WebLogic Server によって JNDI スペースに追加されます。

   クライアント サイドでは、送り先はサーバ上のオブジェクトへのハンドルになります。メソッドにより、送り先名のみが返されます。メッセージ用の送り先にアクセスするには、送り先にアタッチ可能なメッセージ プロデューサとコンシューマを作成します。

5. セッションと送り先を使用してメッセージ プロデューサとメッセージ コンシューマを作成します。

   メッセージ プロデューサはメッセージをキューまたはトピックに送信します。メッセージ コンシューマはメッセージをキューまたはトピックから受信します。

6. 以下のいずれかの手順を実行します。

   a. メッセージ プロデューサを作成している場合は、メッセージ オブジェクトを作成します。

      メッセージにより、アプリケーション間で交換される情報がカプセル化されます。

b. メッセージ コンシューマを作成している場合は、6-8 ページの「メッセージの非同期受信」に説明されているとおり、非同期メッセージ リスナを登録すると便利な場合があります。

7. コネクタを開始します。

JMS の詳細については、次の URL の BEA WebLogic Server ドキュメント群に含まれている『*WebLogic JMS プログラマーズ ガイド*』を参照してください。

http://edocs.beasys.co.jp/e-docs/wls/docs70/jms/index.html

メッセージの非同期受信

メッセージを送り先から非同期に受信するには、以下の手順を実行して非同期メッセージ リスナを登録する必要があります。

1. `javax.jms.MessageListener` インタフェースを実装します（このインタフェースには `onMessage()` メソッドが含まれている）。

   注意： この手順は、WebLogic Server 6.0 以上ではメッセージ駆動型 Bean を使用して実行することも可能です。EJB の詳細については、次の URL の BEA WebLogic Server ドキュメント群に含まれている『*WebLogic Enterprise JavaBeans プログラマーズ ガイド*』を参照してください。

   http://edocs.beasys.co.jp/e-docs/wls/docs70/ejb/index.html

2. 次の `javax.jms.MessageConsumer` メソッドを使用してリスナ情報を引数として渡してメッセージ リスナを設定します。

```
public void setMessageListener(
  javax.jms.MessageListener listener
) throws javax.jms.JMSException
```

JMS の詳細については、次の URL から BEA WebLogic Server ドキュメント内の『*WebLogic JMS プログラマーズ ガイド*』を参照してください。

http://edocs.beasys.co.jp/e-docs/wls/docs70/jms/index.html

複数のイベント キューに対するメッセージ駆動型 Bean の生成

前述のとおり、システム管理者は、必要に応じて複数のイベント キューを定義できます（6-6 ページの「JMS への接続」参照）。この場合、システム管理者は関連付けられているメッセージ駆動型 Bean も生成する必要があります。

複数のイベント キューに対するメッセージ駆動型 Bean を生成するには、`com.bea.wlpi.util.MDBGenerator` ユーティリティを次のように使用します。

```
java com.bea.wlpi.util.MDBGenerator -queue queue_name
   [-min number] [-max number] [-order number] [-transact]
   [-validate] [-timeout seconds] [-help]
```

`MDBGenerator` ユーティリティの引数を次の表に示します。

表 6-2 MDBGenerator ユーティリティの引数

引数	説明
`-queue queue_name`	メッセージ駆動型 Bean を生成するカスタム キュー名。 必須の引数。
`-min minimum`	*順序なし*メッセージの処理に割り当てられるメッセージ リスナの最低数。順序なしメッセージとは、特定の順序で処理される必要のないメッセージを指す（6-13 ページの「メッセージの順次処理の保証」参照）。*minimum* によって最大値より小さい整数値を指定する必要がある。 この引数は省略可能で、デフォルトでは 0。
`-max maximum`	*順序なし*メッセージの処理に割り当てられるメッセージ リスナの最大数。*maximum* によって指定される整数値は 100 以下でかつ指定最小値より大きな整数値を指定する必要がある。 この引数は省略可能で、デフォルトでは 5。

表 6-2 MDBGenerator ユーティリティの引数 (続き)

引数	説明
-order *ordernum*	*順序付き*メッセージの処理に割り当てられるメッセージリスナの数。順序付きメッセージとは、特定の順序で処理される必要のあるメッセージを指す（6-13 ページの「メッセージの順次処理の保証」参照）。*ordernum* により 31 以下の素数が指定される。 この引数は省略可能で、デフォルトでは 0。
-transact	トランザクションが必須であることを示すフラグ。
-validate	キューで受信したすべてのメッセージを XML JMS メッセージ内の DOC-TYPE タグに照らして検証することを指定するフラグ。 省略可能な引数。
-timeout *seconds*	トランザクションのタイムアウト値（秒単位）。この引数のデフォルト値は 30 秒。 この値は、transact フラグが設定されていない場合にのみ使用する。transact フラグが設定されている場合は、WebLogic Server トランザクション タイムアウト値が採用される。この値の設定は Administration Console を使用して行うが、デフォルト値は 30 秒になっている。
-help	コマンド解説構文を表示することを指定するフラグ。 省略可能な引数。

このユーティリティにより、特定のキューに対してメッセージ駆動型 Bean デプロイメントが jar ファイル *qname*-mdb.jar (*qname* は関連付けられているキュー名) として生成されます。

WebLogic Server にメッセージ駆動型 Bean をデプロイするには、ファイル ejb-jar.xml と weblogic-ejb.xml を編集し、コンフィグレーション済みの JMS 送り先と EJB を関連付けます。詳細については、次の URL の BEA WebLogic Server ドキュメント群に含まれている『*WebLogic Enterprise JavaBeans プログラマーズ ガイド*』を参照してください。

http://edocs.beasys.co.jp/e-docs/wls/docs70/ejb/index.html

メッセージ配信の保証

メッセージは、永続または非永続のいずれかとして指定できます。永続メッセージは必ず、少なくとも 1 回は配信されることが保証されています。このメッセージは、ファイルまたはデータベースに安全に書き込まれるまで、配信されたと見なされません。WebLogic JMS はコンフィグレーション中に、各 JMS サーバに割り当てられた永続バックアップ ストア（ファイルまたは JDBC データベース）に永続メッセージを書き込みます。非永続メッセージは保存されません。非永続メッセージは、システムに障害が無い限り、必ず 1 回は配信されます。ただし、障害の場合、メッセージは失われます。コネクタが閉じ、また回復すると、通知が済んでいない非永続メッセージはすべて、再配信されます。非永続メッセージはいったん通知されると、再配信されることはありません。

メッセージの配信時に 1 人または複数の受信者が対応しない場合は、*アドレス指定*メッセージを使用して確実にメッセージを配信できます。アドレス指定メッセージの場合、着信イベント メッセージはそのすべての受信者によって使用されるまで、または指定の期間（*存続時間*）が経過するまでのどちらか早い方まで保持されます。メッセージ配信はワークフロー インスタンスまたはテンプレートに基づいて保証できます。

アドレス指定メッセージを使用してメッセージ配信を保証するには、送信側アプリケーション（メッセージの作成者）が次の情報を定義する必要があります。

1. メッセージ ヘッダの一部である `WLPIInstanceIDs` や `WLPITemplateNames` フィールド（ヘッダはメッセージ コンシューマの*アドレス*を定義する）。

   具体的には、これらのフィールドにより、メッセージを受信するインスタンスの ID やテンプレートの名前がそれぞれ指定されます。次の `javax.jms.Message` クラス メソッドを使用すると、JMS ヘッダ フィールド `WLPIInstanceIDs` と `WLPITemplateNames` を定義できます。

   ```
   public void setStringProperty(
     java.lang.String name,
     java.lang.String value
   ) throws javax.jms.JMSException
   ```

   たとえば、`msg` メッセージ インスタンスの JMS ヘッダ フィールドを定義するには次のメソッドを使用します。

   ```
   String instanceID;
   //instanceID set somewhere
   msg.setStringProperty("WLPIInstanceIDs", instanceID);
   ```

```
String templateName="MyTemplate";
msg.setStringProperty("WLPITemplateNames", templateName);
```

これらのフィールドは文字列値として指定される必要があります。各フィールドには、複数の ID をカンマで区切って指定できます。インスタンスや、既にインスタンス化されているテンプレートのみを対象としてください。

注意： メッセージをテンプレートに送信するときは、テンプレート名にカンマを含めないでください。

JMS ヘッダ フィールドは常にメッセージと一緒に伝送され、メッセージ コンシューマが使用できます（メッセージ駆動型 Bean を含む）。JMS メッセージ ヘッダ フィールドの定義の詳細については、次の URL の BEA WebLogic Server ドキュメント群に含まれている『*WebLogic JMS プログラマーズ ガイド*』の「WebLogic JMS アプリケーションの開発」の「メッセージ ヘッダ フィールドおよびメッセージ プロパティ フィールドの設定と参照」を参照してください。

http://edocs.beasys.co.jp/e-docs/wls/docs70/jms/implement.html

2. アドレス指定メッセージを保持する秒数をメッセージ送信時に指定するパラメータとしての存続時間値。プロセス エンジンは、存続時間値が時間切れになるまで、またはメッセージが全アドレス指定受信者に使用されるまでのどちらか早い方までメッセージを保持します。このパラメータは整数値で指定する必要があります。メッセージ送信の詳細については、次の URL から BEA WebLogic Server ドキュメント内の『WebLogic JMS プログラマーズ ガイド』の「WebLogic JMS アプリケーションの開発」の「メッセージの送信」を参照してください。

   http://edocs.beasys.co.jp/e-docs/wls/docs70/jms/implement.html

   存続時間値は、『*WebLogic Integration Studio ユーザーズ ガイド*』の「アクションの定義」に説明されているとおり、[Studio XML イベントをポスト] ダイアログ ボックスでも設定できます。

   設定された期限は、次の `javax.jms.Message` クラス メソッドを使用して `JMSExpiration` JMS ヘッダ フィールドで参照できます。

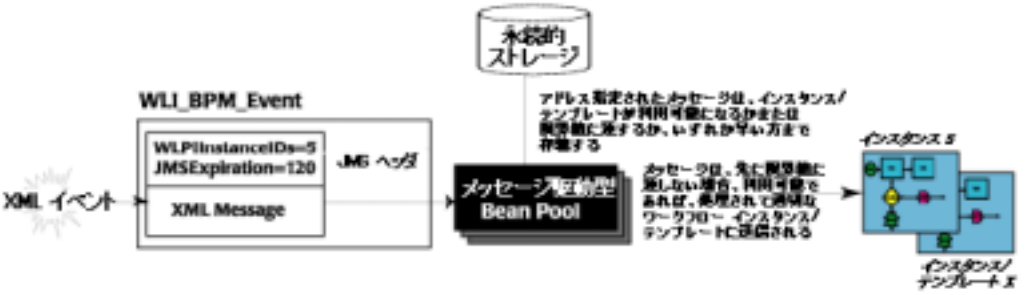
   ```
   public long getJMSExpiration(
   ) throws javax.jms.JMSException
   ```

   JMS メッセージ ヘッダ フィールドの参照の詳細については、次の URL から BEA WebLogic Server ドキュメント内群の『*WebLogic JMS プログラマーズ ガイド*』の「WebLogic JMS アプリケーションの開発」の「メッセージ ヘッダ フィールドおよびメッセージ プロパティ フィールドの設定と参照」を参照してください。

http://edocs.beasys.co.jp/e-docs/wls/docs70/jms/implement.html

アドレス指定メッセージ送信を使用してメッセージ配信を保証する方法を次の図に示します。

図 6-2　メッセージ配信の保証



上の図には以下のことが示されています。

- アドレス指定メッセージは、送信側アプリケーションの JMS ヘッダ フィールド（`WLPIInstanceIDs` または `WLPITemplateNames`）で指定されるインスタンスまたはテンプレートに配信されます。
- 受信者が使用不可能であるために配信できないアドレス指定メッセージは、受信者が使用可能になるまで、または存続時間値（`JMSExpiration` ヘッダ フィールド値）に達するまでのどちらか早い方まで保持されます。

メッセージの順次処理の保証

メッセージ駆動型 Bean の利点の 1 つは、ランダムなメッセージ駆動型 Bean インスタンスによって着信メッセージを並列処理できることです。ただし、この場合、メッセージが処理される*順序*は保証されません。

メッセージが受信されて処理される順序が重要である場合は、すべてのメッセージを同じメッセージ駆動型 Bean インスタンスに送信することでその順序を保証できます。

メッセージが同じメッセージ駆動型 Bean インスタンスに配信されて順次処理されるためには、送信側アプリケーションと受信側アプリケーションで以下の手順を実行する必要があります。

1. メッセージを送信する前に、送信側メッセージ（メッセージ プロデューサ）は順序キー（`WLPIOrderKey`）を JMS メッセージ ヘッダ内のフィールドとして定義する必要があります。順序キーの値は、インスタンス ID などの長整数でなければなりません。

   次の `javax.jms.Message` クラス メソッドを使用すると、JMS ヘッダ フィールド `WLPIOrderKey` を定義できます。

   ```
   public void setLongProperty(
     java.lang.String name,
     long value
   ) throws javax.jms.JMSException
   ```

   たとえば、`msg` メッセージ インスタンスの `WLPIOrderKey` JMS ヘッダ フィールドを定義するには次のメソッドを使用します。

   ```
   msg.setLongProperty("WLPIOrderKey",
      Long.parseLong(instanceID));
   ```

   順次処理を必要とする関連メッセージには、同じ順序キー値が割り当てられる必要があります。

   JMS ヘッダ フィールドは常にメッセージと一緒に伝送され、メッセージ コンシューマが使用できます（メッセージ駆動型 Bean を含む）。JMS メッセージ ヘッダ フィールドの定義の詳細については、次の URL の BEA WebLogic Server ドキュメント群に含まれている『*WebLogic JMS プログラマーズ ガイド*』の「WebLogic JMS アプリケーションの開発」の「メッセージ ヘッダ フィールドおよびメッセージ プロパティ フィールドの設定と参照」を参照してください。

   `http://edocs.beasys.co.jp/e-docs/wls/docs70/jms/implement.html`

2. 受信側アプリケーション（メッセージ コンシューマ）は、`WLPIOrderKey` 値に基づいて、処理が必要なメッセージを *フィルタ処理する* JMS メッセージ セレクタを定義する必要があります。

   メッセージ セレクタはブール式です。このセレクタは、`SQL select` 文の `where` 文節に似た構文を持つ文字列から構成されています。

   たとえば、`WLPIOrderKey=1`

   JMS メッセージ セレクタの定義の詳細については、次の URL から BEA WebLogic Server ドキュメント内の『*WebLogic JMS プログラマーズ ガイド*』の「WebLogic JMS アプリケーションの開発」の「メッセージのフィルタ処理」を参照してください。

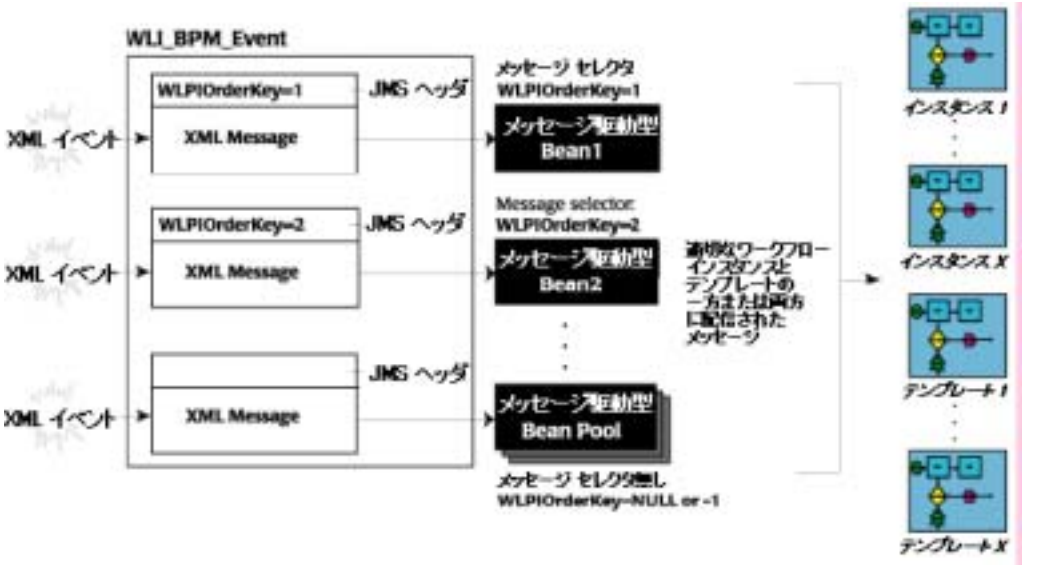
   `http://edocs.beasys.co.jp/e-docs/wls/docs70/jms/implement.html`

注意： 順次処理されるすべてのメッセージ グループは、同じ JMS イベント キューに送信される必要があります。

特定の順序キーを持つすべてのメッセージは、同じメッセージ駆動型 Bean インスタンスにより、受信された順序で処理されるため、メッセージの順次処理が保証されます。

順序キーを使用したメッセージの順次処理を次の図に示します。

図 6-3 メッセージの順次処理の保証



上の図には以下のことが示されています。

- 順次処理されるすべてのメッセージは同じイベント キューである WLI_BPM_Event、または他のユーザ定義キューに送信されます。
- WLPIOrderKey ヘッダ フィールドが 1 に設定されているすべてのメッセージはメッセージ駆動型 Bean 1 に処理され、WLPIOrderKey ヘッダ フィールドが 2 に設定されているメッセージはメッセージ駆動型 Bean 2 に処理されます。
- WLPIOrderKey ヘッダ フィールド情報が定義されていないすべてのメッセージは、メッセージ駆動型 Bean のプールによって処理されます。

- 各 XML メッセージは、ワークフロー テンプレートやインスタンスに影響を与える場合があります。

JMS トピックへの接続例

この節には、JMS 送り先（この節には Worklist 通知トピックである WLPI_BPM_Notify）への接続方法を示すコード例が記載されています。WLPI_BPM_Notify トピックは、表示を更新するために Worklist クライアントにより動的に（つまり、メッセージの受信時に）使用されます。このサンプルでは、通知の受信にメッセージ セレクタ（フィルタ）が使用されています。続いて、受信されたメッセージのプロパティがオーガニゼーション、ロール、ユーザ、およびアクションの現在値に一致する場合は、表示の更新にメッセージ セレクタが使用されます。

コードの各セクションについて以下で詳しく説明します。JMS の詳細については、次の URL から BEA WebLogic Server ドキュメント内の『*WebLogic JMS プログラマーズ ガイド*』を参照してください。

http://edocs.beasys.co.jp/e-docs/wls/docs70/jms/index.html

JNDI コンテキスト、JMS コネクタ ファクトリ、およびトピック静的変数を含む必須の変数を定義します。

```
protected static final String JNDI_FACTORY= "weblogic.jndi.WLInitialContextFactory";
protected static final String JMS_FACTORY= "javax.jms.TopicConnectionFactory";
protected static final String NOTIFY_TOPIC="com.bea.wli.bpm.Notify";
```

メッセージを JMS トピックに送信するのに必要なすべてのオブジェクトを作成します。

```
private TopicConnectionFactory tconFactory;
private TopicConnection tcon;
private TopicSession tsession;
private Topic topic;
private TopicSubscriber tsubscriber;
```

JNDI 初期コンテキストを次のとおり設定します。

```
try {
   Hashtable env = new Hashtable();
   env.put(Context.PROVIDER_URL, wlpi.getUrl());
   env.put(Context.INITIAL_CONTEXT_FACTORY, JNDI_FACTORY);
   env.put("weblogic.jndi.createIntermediateContexts", "true");
```

```
Context ctx = new InitialContext(env);
```

1. JNDI を使用してコネクタ ファクトリをルック アップします。

   ```
   tconFactory = (TopicConnectionFactory) ctx.lookup(JMS_FACTORY);
   ```

2. コネクタ ファクトリを使用してコネクタを作成します。

   ```
   tcon = tconFactory.createTopicConnection();
   ```

3. コネクタを使用してセッションを作成します。次のメソッドにより、セッションが未処理と定義され、メッセージが自動的に確認応答されるよう指定されます。

   セッション トランザクションと確認応答モードの詳細については、次の URL から BEA WebLogic Server ドキュメント内の『*WebLogic JMS プログラマーズ ガイド*』を参照してください。

   http://edocs.beasys.co.jp/e-docs/wls/docs70/jms/index.html

   ```
   tsession = tcon.createTopicSession(
      false, Session.AUTO_ACKNOWLEDGE
   );
   ```

4. 送り先（トピック）を JNDI で次のとおりルック アップします。

   ```
   topic = (Topic)ctx.lookup(NOTIFY_TOPIC);
   ```

5. セッションと送り先（トピック）を使用して、メッセージ プロデューサ（トピック発行者）への参照を作成します。サブスクライバが既に存在する場合は、閉じられます。オーガニゼーションと参加コンポーネントに関する情報がセレクタとして使用されます。セレクタの定義の詳細については、次の URL から BEA WebLogic Server ドキュメント内の『*WebLogic JMS プログラマーズ ガイド*』を参照してください。

   http://edocs.beasys.co.jp/e-docs/wls/docs70/jms/index.html

   wlpi が com.bea.wlpi.client.common.WLPI クラスのインスタンスを次のコード例で表しています。

```
if (tsubscriber != null) tsubscriber.close();
String selector = "orgId = '" +
wlpi.getWorklist().getActiveOrganization() + "' AND assigneeId = '" +
   wlpi.getUserId() + "' AND NOT role" + " AND action = 'assigned'";
tsubscriber = tsession.createSubscriber(topic, selector, false);
```

6. 非同期メッセージ リスナを登録します。

   ```
   tsubscriber.setMessageListener(this);
   ```

7. コネクタを開始します。

```
    tcon.start();
  }
```

メッセージは、トピック セッションに配信された後で、`onMessage()` メソッドに渡されます。このメソッドは、6-8 ページの「メッセージの非同期受信」 に記載の手順に従って実装する必要があります。

次のコードは、通知を使用してタスクリスト表示を更新するメッセージ リスナクライアントの例です。

注意： AWT/Swing アプリケーションの場合は、`onMessage()` メソッド呼び出しへの呼び出しと同じスレッド上のユーザ インタフェースは更新しないでください。更新すると、デッドロックや例外が発生します。呼び出しは AWT Event Dispatcher Thread にマーシャリングしてください。Swing ユーティリティには、実行可能オブジェクトを AWT Event Dispatcher Thread にエンキューする `invokeLater()` メソッドが用意されていて、更新ロジックを実行可能オブジェクト `run()` メソッドにインクルードできます。

```
import javax.swing.SwingUtilities;
public void onMessage(Message msg) {
   String action;
   TaskInfo task;
   try {
   // TaskInfo ペイロードを持つ ObjectMessages
   // が該当。
      if (!(msg instanceof ObjectMessage))
         return;
      Object data = ((ObjectMessage)msg).getObject();
      if (!(data instanceof TaskInfo))
         return;
      action = msg.getStringProperty("action");
      task = (TaskInfo)data;
   } catch (JMSException e) {
      e.printStackTrace();
      return;
   }

   if (action.equals("update"))
      taskUpdated(task);
   else if (action.equals("remove"))
      taskDeleted(task);
}

private void taskDeleted(final TaskInfo task) {
   SwingUtilities.invokeLater(new Runnable() {
      public void run() {
         vTasks.remove(task);
         int row = vDisplayedTasks.indexOf(task);
```

```
            if (row != -1) {
               vDisplayedTasks.remove(row);
               model.fireTableRowsDeleted(row, row);
            }
         }
      });
}
```

JMS の詳細については、次の URL から BEA WebLogic Server ドキュメント内の『*WebLogic JMS プログラマーズ ガイド*』を参照してください。

http://edocs.beasys.co.jp/e-docs/wls/docs70/jms/index.html

7 BPM トランザクション モデルの理解

この章では、トランザクションが Business Process Management (BPM) アプリケーション内で処理される方法について説明します。内容は以下のとおりです。

- トランザクションの開始方法
- トランザクションのコミット方法
- 例外の処理方法
- 新しいトランザクションの強制開始方法
- トランザクションのサンプル

トランザクションの概要や、トランザクション処理と EJB (Enterprise JavaBean: エンタープライズ JavaBean) の詳細については、次の URL から WebLogic Server ドキュメント内の『*WebLogic JTA プログラマーズ ガイド*』を参照してください。

http://edocs.beasys.co.jp/e-docs/wls/docs70/jta/index.html

トランザクションの開始方法

WebLogic Integration プロセス エンジンによるワークフロー実行の開始または再開をトリガする次の 3 つのアクションでは、新しいトランザクションの開始をマークする場合もあります。

- API メソッド呼び出し
- XML イベント
- 時間ベースのイベント

API メソッド呼び出しによって新しいトランザクションも開始されるかどうかは、呼び出し側アプリケーションにおけるトランザクション コンテキストの有無により決定されます。*外部*トランザクション コンテキストが存在する場合（つまり、アプリケーションが JTA を使用してトランザクションを作成済み、または JTA トランザクションを継承済みの場合）は、WebLogic Integration EJB によってそのコンテキストが使用されます。それ以外の場合は、EJB コンテナによって新しいトランザクションが自動的に作成され、その中で呼び出しが実行されます。`Audit` EJB 以外の公開 API のすべての EJB メソッドは、`TX_REQUIRED` トランザクション属性を持つコンテナ管理によるトランザクションを使用します。

XML イベントと時間ベースのイベントの両方は常に新しいトランザクションを開始します。それぞれの場合で、トランザクションが開始されてから、キューからメッセージが出されます。

注意： WebLogic Integration Worklist クライアント アプリケーションは、個々のトランザクション内の `Worklist` EJB 上のメソッドを呼び出すことで、ユーザ コマンドを実行します。WebLogic Integration Worklist アプリケーション内のタスクを操作すると、内部で各コマンドが実行されるトランザクションが EJB コンテナにより作成されます。たとえば EJB コンテナにより、タスクを実行したり、タスクに完了マークを付けたり、タスクを別のユーザまたはロールに再割り当てたりする、内部で実行されるトランザクションが作成されます。

Worklist クライアント アプリケーションは、このリリースの WebLogic Integration からは非推奨になります。Worklist クライアント アプリケーションに代わる機能については、『*WebLogic Integration リリース ノート*』を参照してください。

トランザクションのコミット方法

トランザクションのコミット方法を理解するには、次を理解する必要があります。

- ワークフロー インスタンスが WebLogic Integration で処理される方法
- ワークフロー インスタンスが静止状態になる方法

各方法について、この後の節で詳しく説明します。

ワークフロー インスタンスの処理方法

ワークフローを最初にインスタンス化するときは、WebLogic Integration によりワークフロー内のすべてのタスクに 作成時アクションが実行されます。次に、開始ノードが以下のとおり処理されます。

- イベントまたは時間でトリガされるワークフローの場合は、イベントに対応する開始ノードがアクティブ化されます。
- 手動で開始または呼び出されるワークフロー（サブワークフロー）の場合は、すべての手動で開始または呼び出された開始ノードがそれぞれアクティブ化されます。

初期インスタンス化の後のトランザクションの進行状況は、テンプレート定義により異なります。アクティブ化イベントは、各ノード タイプに対して定義済みの処理ルールに応じて、起点から始まるパスに沿って外へ向かって伝播されます。

各ノード タイプの処理ルールを次の表に示します。

表 7-1 ノード タイプ別の処理ルール

ノード タイプ	処理ルール
分岐	アクティブ化された分岐ノードによってまず条件が評価され、リストにある true アクションまたは false アクションが適宜実行され、最後に true または false サクセサ ノードがアクティブ化される。

表 7-1 ノード タイプ別の処理ルール (続き)

ノード タイプ	処理ルール
完了	アクティブ化された完了ノードによって、すべての完了ノードへの到達とは無関係に、関連付けられているアクションが実行され、現在のワークフローに完了マークが付けられる。
イベント	アクティブ化されたイベント ノードによって前に受信されたアドレス指定メッセージが確認され、メッセージがない場合は、自身が Event Watch テーブルに登録される。アドレス指定メッセージの詳細については、6-11 ページの「メッセージ配信の保証」を参照。 トリガされたイベント ノードによって以下のタスクが順次実行される。 ■ 変数の初期化 ■ 関連付けられているアクションの実行 ■ サクセサ ノードのアクティブ化 サクセサ ノードのアクティブ化は、各ノード タイプのルールに従って行われる。
結合	いずれかの単一入力パスによりアクティブ化される場合は OR 結合ノードによって OR 条件が評価される。すべての入力パスによってアクティブ化される場合は AND 結合ノードによって AND 条件が評価される。条件が満たされると、結合ノードによってそのサクセサ ノードがアクティブ化される。
開始	アクティブ化された開始ノードによって以下のタスクが順次実行される。 ■ 変数の初期化 ■ 関連付けられているアクションの実行 ■ サクセサ ノードのアクティブ化 サクセサ ノードのアクティブ化は、各ノード タイプのルールに従って行われる。

表 7-1 ノード タイプ別の処理ルール (続き)

ノード タイプ	処理ルール
タスク	アクティブ化されたタスク ノードによって、アクティブ時タスクのリストが順次実行される。行われる可能性のある後続アクションをまとめると次のようになる。 ■ アクションまたは API 要求によってタスクが実行されると、リストにある 実行アクションが実行される。 ■ アクションまたは API 要求によってタスクに完了マークが付けられると、完了マーク時アクションが実行される。 ■ アクションによってワークフローに完了マークが付けられるか、またはワークフローがアボートされると、それ以降の処理は行われない。

ワークフロー インスタンスが静止状態になる方法

ワークフローの処理時は、同期実行するアクションがなくなると、コントロールがプロセス エンジンからシステムに戻されます。つまり、ワークフロー インスタンスが*静止*状態に入り、以下の要求待ちになります。

ワークフロー インスタンスは、以下の条件が true の場合に静止状態に入ります。

- すべてのタスク ノードがアクティブ化、実行、または完了のマークが付けられるのを待っている場合。
- すべてのイベント ノードがアクティブ化を待っている場合、または既にトリガされている場合。
- すべての 分岐および結合ノードがアクティブ化されるための最終入力を待っている場合、または既にアクティブ化されている場合。
- すべての開始ノードがアクティブ化を待っているか、または既に実行されている場合。
- すべての完了ノードがアクティブ化されるのを待っている場合。

プロセス エンジンが最後にトリガされた時点で新しいトランザクションが開始された場合は、静止状態によってトランザクションの最後もマークされます（その結果、トランザクションがコミットされる）。外部トランザクション コンテキストが API 呼び出しトリガの前にエクスポートされた場合は、外部トランザクションが有効なままとなり、コントロールは呼び出し側アプリケーションに戻されます。詳細については、7-2 ページの「トランザクションの開始方法」を参照してください。

あるアクションによってサブワークフローがインスタンス化されると、サブワークフローが静止状態に入るまでに実行するすべてのワークは、呼び出し側ワークフローと同じトランザクション内で実行されます。このワークには、通常、呼び出されたすべての開始ノード内のアクションの実行、各開始ノードと関連付けられているすべてのサクセサ ノードのアクティブ化、および各サクセサと関連付けられているアクションなどがあります。この処理はサブワークフローを介して伝播され、事前定義済みのルールに従って、完了または静止状態になります。サブワークフローの完了へ向けて実行中（つまり、静止状態ではない）の場合は、親ワークフロー内の 完了アクションも同じトランザクション内で実行されます。タスク（たとえば親）に完了マークを付けるアクションが 完了アクションに含まれている場合は、処理が続行され、完了マーク時アクションが実行され、サクセサ ノードがアクティブ化されます。

例外の処理方法

プロセス エンジンによって例外が受け取られた場合、または送出された場合に、トランザクションのコミットとロールバックのどちらが行われるかは次の 2 つの要因によって決定されます。

- 例外の発生場所
- アクティブな例外ハンドラの動作

アクティブな例外ハンドラは、関連付けられている是正措置を実行でき、次のいずれかの処置を指定できます。

- ユーザ トランザクションのロールバックおよび例外の再送出
- 処理の停止および呼び出し側への戻し処理のコミット
- 失敗したオペレーションの再試行
- エラーの無視および処理の続行

ユーザ トランザクションが「ロールバックのみ」に設定されておらず、例外が回復可能であり（つまり、例外がワークフロー警告である場合）、かつアクションの処理中またはイベント変数の設定中に例外が発生した場合は、例外ハンドラが処理結果を決定できます。それ以外の場合は、ユーザ トランザクションが「ロールバックのみ」に自動設定され、アクティブな例外ハンドラが呼び出され、例外が再送出されます。

コンテナによってトランザクションが作成された場合は、コントロールがプロセス エンジンからコンテナに戻されたときにトランザクションがコミットまたはロールバックされます（コントロールは、7-6 ページの「ワークフロー インスタンスが静止状態になる方法」に記載のとおり、ワークフロー インスタンスが静止状態に入った際に戻される）。

アプリケーションが BPM EJB を外部トランザクションに入れ、ユーザ トランザクションを「ロールバックのみ」に設定しない例外がこの EJB によって送出される場合は、トランザクションの動作はアプリケーションによって定義されます。一方で、BPM 呼び出しが正常に終了した場合は、トランザクションが外部アプリケーション（または EJB）によってコミットされるという保証はありません。その結果、例外またはその他のエラー条件が発生することがあり、トランザクション全体がロールバックされます。

XML および時間ベースのイベントの場合、トランザクションがロールバックされると、メッセージはキューに戻されます。指定した再試行回数に達するまで、指定した間隔でメッセージの再配信が試みられます。再試行の最大回数および再試行間隔は、JMSTemplate 要素の Redelivery-Limit 属性および RedeliveryDelayOverride 属性を使用して JMS 送り先ごとに設定できます。たとえば、WebLogic Integration サンプル ドメインの場合、WLI_JMSTemplate は次のように定義されます。

```
<JMSTemplate ErrorDestination="WLI_FailedEvent"
Name="WLI_JMSTemplate"
RedeliveryDelayOverride="60000"
RedeliveryLimit="10"/>
```

これらの属性の詳細については、次の URL から、『*WebLogic Server コンフィグレーション リファレンス*』に掲載されている「JMSTemplate」の要素の説明を参照してください。

http://edocs.beasys.co.jp/e-docs/wls/docs70/config_xml/index.html

新しいトランザクションの強制開始方法

現在のワークフロー インスタンスを静止状態にするダミーの タスク アクションを定義することで、新しいトランザクションを強制できます（ワークフロー インスタンスを静止状態にする方法については、7-6 ページの「ワークフロー インスタンスが静止状態になる方法」を参照）。

たとえば、以下のいずれかの タスク アクションを定義することで、ワークフロー インスタンスを強制的に静止状態にできます。

- アクションを実行せず、1 秒後のトリガを含む時限イベント
- ダミーのトリガ メッセージを自身に送信する XML イベントのポスティング

トランザクションのサンプル

ビジネス オペレーションがワークフロー インスタンス内で実行されるとき、そのビジネス オペレーションが単一トランザクションまたは複数トランザクションのどちらとして処理されるかは以下の要素により決定されます。

- ワークフローの定義方法

  具体的には、すべてのビジネス オペレーションの実行前に既にあるトランザクションが静止状態に入るようにワークフローが定義されているかどうか。

- EJB ビジネス オペレーションの場合は、EJB のデプロイ方法

  アプリケーション定義 EJB を現在の BPM トランザクションに入れる場合は、`Mandatory`、`Required`、または `Supports` のいずれかのコンテナ管理によるトランザクション属性を使用して EJB をデプロイする*必要があります*。これらの設定の詳細については、Sun Microsystems 社発行の『Enterprise JavaBeans Specification 2.0』の「Section 16.7.2」を参照。

ビジネス オペレーションを単一タスクから構成されるアクションの一部として指定するか、複数タスクから構成されるアクションとして指定するかを定義するワークフローのサンプルを以下の節に記載しています。各サンプルでは、単一トランザクションと複数トランザクションの両方のシナリオを示しています。

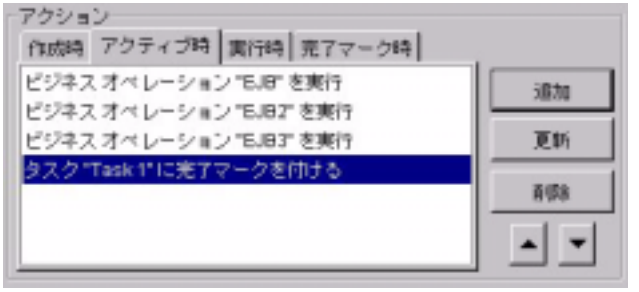
サンプル 1 ー アクションが単一タスクとして定義されているビジネス オペレーション

最初のサンプルでは、3 つの EJB を個別のビジネス オペレーションとして、または単一タスクから構成されるアクションの一部として実行すると仮定します。タスク プロパティの定義方法に基づいて、3 つのビジネス オペレーションを単一トランザクションまたは複数トランザクションとして処理できます。

単一トランザクション

指定のビジネス オペレーションを*単一*トランザクションとして処理するよう [タスクのプロパティ] ダイアログ ボックスの [アクティブ時] タブで タスク アクション を定義する方法を次のサンプルに示しています。

図 7-1 トランザクション サンプル－単一タスク、単一トランザクション

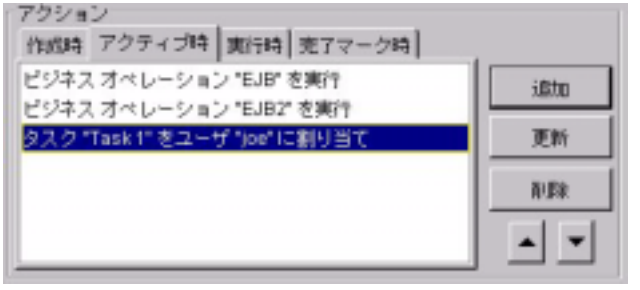


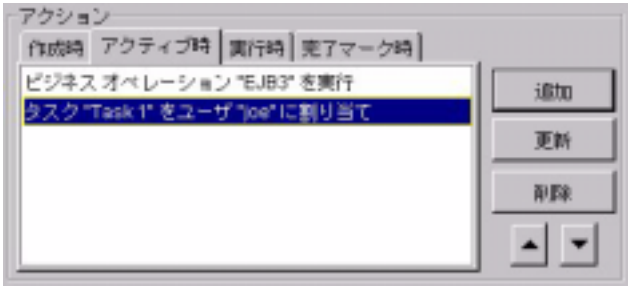
このサンプルでは、すべてのビジネス オペレーションがタスクのアクティブ化時に呼び出され、単一トランザクションとして処理されます。

複数トランザクション

指定のビジネス オペレーションを*複数*トランザクションとして処理するよう [タスクのプロパティ] ダイアログ ボックスの [アクティブ時] および [実行時] タブで タスク アクション を定義する方法を次のサンプルに示しています。

図 7-2 トランザクション サンプル－単一タスク、複数トランザクション





このサンプルでは、ビジネス オペレーション（EJB1 および EJB2）のサブセットがタスク アクティブ化時に 1 つのトランザクションとして呼び出されています。Assign task 摘 JB1î to user 屠 oeî アクションが実行されると、ワークフロー インスタンスが静止状態に入り、次の要求を待ちます。最初のトランザクションに完了マークが付けられ、実行時アクションが実行されます。次に EJB3 ビジネス オペレーションが別のトランザクションとして呼び出されます。

このサンプルでは、EJB3 の処理中に例外が発生すると、関連付けられている是正措置がアクティブな例外ハンドラによって実行され、行われる処置が指定されます。最初のトランザクションは、第 2 のトランザクションで発生した例外による影響を受けません。詳細については、7-7 ページの「例外の処理方法」を参照してください。

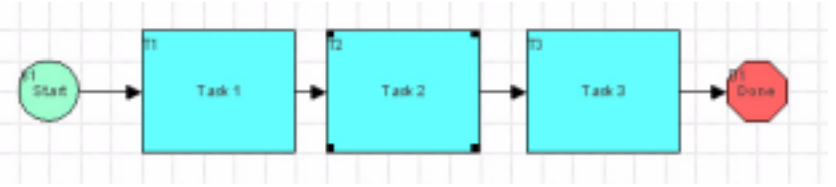
サンプル 2 ーアクションが複数タスクとして定義されているビジネス オペレーション

第 2 のサンプルでは、3 つの EJB を個別のビジネス オペレーションとして、または 3 つの個別のタスク内のアクションとして呼び出すと仮定します。ここでも、タスク プロパティの定義方法に基づいて、3 つのビジネス オペレーションを単一トランザクションまたは複数トランザクションとして処理できます。

単一トランザクション

このサンプルでは、以下のワークフロー テンプレートを想定します。

図 7-3 トランザクション サンプル－複数タスク、単一トランザクション



次の文が true であると仮定します。

- Task1 によって EJB1 ビジネス オペレーションが実行され、自身に完了マークが付けられます。
- Task2 によって EJB2 ビジネス オペレーションが実行され、自身に完了マークが付けられます。
- Task3 によって EJB3 ビジネス オペレーションが実行され、自身に完了マークが付けられます。

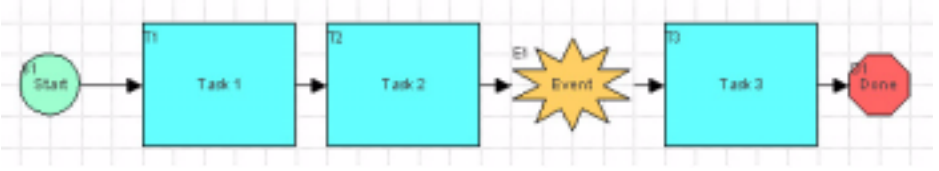
これらの文が true である場合、いずれかのビジネス オペレーションの実行前にワークフロー インスタンスは静止状態にはならず、3 つのすべてのビジネス オペレーションが単一トランザクションとして処理されます。

複数トランザクション

このサンプルでは、ダミーの Post XML イベントを定義して静止状態を強制（および新しいトランザクションを強制開始）する方法を示します。新しいトランザクションの強制開始に関する詳細については、7-8 ページの「新しいトランザクションの強制開始方法」を参照してください。

このサンプルでは、以下のワークフロー テンプレートを想定します。

図 7-4 トランザクション サンプル－複数タスク、複数トランザクション



次の文が true であると仮定します。

- Task1 によって EJB1 ビジネス オペレーションが実行され、自身に完了マークが付けられます。
- Task2 によって EJB2 ビジネス オペレーションが実行され、自身に完了マークが付けられます。
- イベント ノードがトリガされ、ワークフローが静止状態に入ります。

  注意： 入力イベント通知の配信時に Event Watch がトリガされず、関連付けられているメッセージが欠落または無視される場合があります。アドレス指定メッセージ送信を使用すると、6-11 ページの「メッセージ配信の保証」に記載のとおり、必須のメッセージ ヘッダ フィールドと存続時間値を定義することで、メッセージ配信を保証できます。

- Task3 によって EJB3 ビジネス オペレーションが実行され、自身に完了マークが付けられます。

これらの文が true である場合、EJB1 と EJB2 が単一トランザクションとして処理され、EJB3 が別のトランザクションとして処理されます。

このサンプルでは、EJB3 の処理中に例外が発生すると、関連付けられている是正措置がアクティブな例外ハンドラによって実行され、行われる処置が指定されます。最初のトランザクションは、第 2 のトランザクションで発生した例外による影響を受けません。詳細については、7-7 ページの「例外の処理方法」を参照してください。

8　プロセス エンジンからの切断

この章では、WebLogic Integration のプロセス エンジンからの切断方法について説明します。この章の内容は以下のとおりです。

- セッション EJB 参照の削除
- 他のリソースの解放

セッション EJB 参照の削除

すべてのアプリケーション タスクを完了後、セッション EJB 参照を削除して、システム スペースを他の EJB が使用できるようにすることが可能です。

3-1 ページの「API セッション EJB のホーム インタフェースとリモート インタフェース」の表に定義されているセッション EJB のホーム インタフェースとリモート インタフェースは、それぞれインタフェース `javax.ejb.EJBHome` と `javax.ejb.EJBObject` からメソッドを継承します。

EJB 参照の削除に使用できる継承済みメソッドを次の表で定義します。

EJB インタフェース	メソッド	機能
ホーム	`public void remove(javax.ejb.Handle handle ) throws java.rmi.RemoteException, javax.ejb.RemoveException`	指定のハンドルと関連付けられている EJB オブジェクトを削除する。
	`public javax.ejb.HomeHandle getHomeHandle() throws java.rmi.RemoteException`	ホーム インタフェース オブジェクトのハンドルを取得する。

EJB インタフェース	メソッド	機能
リモート	`public void remove() throws java.rmi.RemoteException, javax.ebj.RemoveException`	EJB オブジェクトを削除する。
	`public javax.ejb.EJBHome getEJBHome() throws java.rmi.RemoteException`	EJB ホーム インタフェースを取得する。
	`public javax.ejb.Handle getHandle() throws java.rmi.RemoteException`	EJB オブジェクトのハンドルを取得する。 この値は、この表で前述のホーム インタフェース `remove()` メソッド内の *`handle`* パラメータとして使用可能。

たとえば、JSP ワークリスト サンプルから抜粋された次のコードは、セッション変数を含む `Worklist` EJB 参照を削除する方法を示します。このコード例では、`worklist` は `Worklist` EJB への `EJBObject` 参照を表します。`session` 変数は、`worklist` リモート オブジェクト参照を事前に格納した `HttpSession` オブジェクトです。

```
Worklist worklist = null;
worklist = (Worklist)session.getValue("worklist");

if (worklist != null) {
    try {
worklist.remove();
    } catch (Exception e) {
    }

    worklist = null;
    session.removeValue("worklist");
}
```

ホーム インタフェースとリモート インタフェースによって継承されるメソッドの詳細については、Javadoc の `javax.ejb.EJBHome` および `javax.ejb.EJBObject` をそれぞれ参照してください。

他のリソースの解放

セッション EJB 参照の削除に加えて、以下の節で説明するタスクを実行する必要があります。

- JMS コネクタの停止および終了
- コンテキストの終了

JMS コネクタの停止および終了

一般に、JMS プロバイダはコネクタの作成時に多量のリソースを割り当てます。コネクタが使用されなくなった後に、コネクタを終了し、それに関連付けられているリソースを解放してください。コネクタは、次の `javax.jms.Connection` メソッドを使用して終了できます。

```
public void close(
) throws javax.jms.JMSException
```

コネクタを終了して JMS から切断する方法を次のコード例に示します。

```
try {
    tcon.close();
} catch (Exception e) {
    ExceptionHandler.reportException(e, this);
  }
  tcon = null;
  tsession = null;
  tsubscriber = null;
}
```

コネクタが終了した後、アプリケーションにより、コネクタ、セッション、およびトピック サブスクライバの変数が `null` に設定され、これらのオブジェクトで使用されていたリソースをガベージ コレクタが解放できます。

コンテキストの終了

JNDI コンテキスト リソースを（ガベージ コレクタによって自動的に解放されるのを待つのではなく）終了するには、次の `javax.naming.Context` メソッドを使用します。

```
public void close() throws javax.naming.NamingException
```

たとえば、次のコードでは JNDI コンテキスト リソースが終了します。

```
try {
    ctx.close();
    {
    catch(Exception exp)
    {
    }
}
```

詳細については、Javadoc の `javax.naming.Context` を参照してください。

9　セキュリティ レルムのコンフィグレーション

この章では、セキュリティ レルムのコンフィグレーション方法について説明します。内容は以下のとおりです。

- セキュリティに関する基本情報の取得
- オーガニゼーション、ロールおよびユーザのコンフィグレーション
- セキュリティ情報のマッピング
- パーミッションのコンフィグレーション

ビジネス プロセスの管理に関連するタスクに対するセキュリティ実装の詳細については、『*WebLogic Integration Studio ユーザーズ ガイド*』の「データの管理」を参照してください。

セキュリティに関する基本情報の取得

com.bea.wlpi.server.principal.WLPIPrincipal メソッド、およびコンビニエンス com.bea.wlpi.client.common.WLPI クラス メソッドの一方または両方を使用して、以下のセキュリティ関連情報を取得できます。

- WebLogic Server セキュリティ レルムのクラス名。たとえば、デフォルトの WebLogic Server セキュリティ レルムが使用中かどうかを調べるときに役立つ
- WebLogic Server セキュリティ レルムが管理可能かどうか、永続的かどうか
- プロセス エンジンの URL
- ユーザ ID

これらのタイプのセキュリティ情報を取得する方法とコード例を以下に示します。

セキュリティ レルムのクラス名の取得

WebLogic Server セキュリティ レルムのクラス名を取得するには、次の com.bea.wlpi.server.principal.WLPIPrincipal メソッドを使用します。

```
public java.lang.String getSecurityRealmClassName(
) throws java.rmi.RemoteException,
  com.bea.wlpi.common.WorkflowException
```

このメソッドは、セキュリティ レルムのクラスの完全修飾名を返します。

たとえば、次のコードでは、WebLogic Server セキュリティ レルムのクラス名が取得されて、文字列 security_class に保存されます。このコード例では、principal は WLPIPrincipal EJB への EJBObject 参照を表しています。

```
String security_class = principal.getSecurityRealmClassName();
```

getSecurityRealmClassName() メソッドの詳細については、Javadoc の com.bea.wlpi.server.principal.WLPIPrincipal を参照してください。

セキュリティ レルムの管理性および永続性のチェック

*管理可能*セキュリティ レルムでは、アプリケーションからグループおよびユーザのセキュリティ情報を更新できます。*管理不可能*セキュリティ レルムでは、アプリケーションからグループおよびユーザのセキュリティ情報を表示することのみができます。詳細については、BEA WebLogic Server のドキュメント セットに含まれている『*WebLogic Server 管理者ガイド*』の「セキュリティの管理」を参照してください。このマニュアルは、BEA WebLogic Server マニュアル セットの次の URL にあります。

http://edocs.beasys.co.jp/e-docs/wls/docs70/adminguide/index.html

セキュリティ レルムが管理可能かどうか調べるには、次の `com.bea.wlpi.server.principal.WLPIPrincipal` メソッドを使用します。

```
public boolean isManageableSecurityRealm(
) throws java.rmi.RemoteException,
  com.bea.wlpi.common.WorkflowException
```

このメソッドは、セキュリティ レルムが管理可能であれば（そのレルムで `ManageableRealm` インタフェースが実装されているのであれば）`true` を返し、管理不可能であれば `false` を返します。

セキュリティ レルムが管理可能*かつ*永続的であるかどうかを調べるには、次の `com.bea.wlpi.client.common.WLPI` メソッドを使用します。

```
public boolean allowSecurityRealmUpdates(
) throws java.lang.IllegalStateException
```

このメソッドは、セキュリティ レルムが管理可能*かつ*永続的であれば `true` を返し、いずれかの特徴を欠く場合は `false` を返します。

たとえば、次のコードでは、セキュリティ レルムが管理可能かどうか調べます。このコード例では、`principal` は `WLPIPrincipal` EJB への `EJBObject` 参照を表しています。

```
Boolean ismanageable = principal.isManageableSecurityRealm();
```

`isManageableSecurityRealm()` メソッドの詳細については、Javadoc の `com.bea.wlpi.server.principal.WLPIPrincipal` を参照してください。`allowSecurityRealmUpdates()` メソッドの詳細については、Javadoc の `com.bea.wlpi.client.common.WLPI` を参照してください。

サーバの URL の取得

WebLogic Integration プロセス エンジンの URL を取得するには、次の com.bea.wlpi.client.common.WLPI メソッドを使用します。

```
public java.lang.String getURL()
```

このメソッドはプロセス エンジンの URL を返します。現在ログオンしていない場合は、null を返します。

たとえば、次のコードは、プロセス エンジンの URL を取得して、url に保存します。

```
String url = com.bea.wlpi.client.common.WLPI.getURL();
```

getURL() メソッドの詳細については、Javadoc の com.bea.wlpi.client.common.WLPI を参照してください。

ユーザ ID の取得

現在のユーザ ID を取得するには、次の com.bea.wlpi.client.common.WLPI メソッドを使用します。

```
public java.lang.String getUserId()
```

このメソッドは、ユーザ ID を返します。現在ログオンしていない場合は、null を返します。

たとえば、次のコードは、ユーザ ID を取得して、user に保存します。

```
String user = com.bea.wlpi.client.common.WLPI.getUserId();
```

getUserId() メソッドの詳細については、Javadoc の com.bea.wlpi.client.common.WLPI を参照してください。

セキュリティに関する基本情報の取得例

ここでは、コマンドライン管理のサンプルから抜粋して、セキュリティ レルムの基本情報を取得する方法を示します。

注意： コマンドライン管理サンプルの詳細については、1-23 ページの「コマンドライン管理サンプル」を参照してください。

このサンプルでは、ユーザと通信を行うための入力ストリームが定義されており、ユーザはメニューからアクションを選択するよう求められます。ユーザが Security Realm オプションを選択すると、セキュリティ クラス名が表示され、セキュリティ レルムが管理可能かどうか示されます。

重要なコード行は、**太字**で強調されています。このコード例では、principal は WLPIPrincipal EJB への EJBObject 参照を表しています。

```
/* ユーザとの通信のための入力ストリームを作成する */
stdIn = new BufferedReader( new InputStreamReader( System.in ) );

/* ツール タイトルを表示する */
   System.out.print( "\n---  Command Line Administration v1.1  ---" );

/* メイン メニューを表示してユーザと交信する */
while( true ) {
/* メニューを表示する */
   System.out.println( "\n---              Main Menu            ---" );
   System.out.println( "\nEnter choice:" );
   System.out.println( "1) Organizations" );
   System.out.println( "2) Roles" );
   System.out.println( "3) Users" );
   System.out.println( "4) Security Realm" );
   System.out.println( "5) Business Operations" );
   System.out.println( "6) Event Keys" );
   System.out.println( "7) Business Calendars" );
   System.out.println( "8) EJB Catalog" );
   System.out.println( "9) Server Properties" );
   System.out.println( "Q) Quit" );
   System.out.print( ">> " );
      .
      .
      .
/* セキュリティ レルム */
         case '4' :
            /* WLPI セキュリティ レルムのプロパティと属性を
             * 表示する */
```

```
System.out.println( "\nSecurity Realm:" );
/* WLPI で現在使用中の セキュリティ レルムのクラスを
 * 取り出して表示する */
System.out.println( "- Class Name: " +
/* WLPI 公開 API メソッド */
/* 注意 : 発生した例外を取り込むコードを
 * 追加するとよい */
principal.getSecurityRealmClassName( ) );
/* WLPI 公開 API メソッド */
/* WebLogic Process Integration で現在使用中のセキュリティ レルムの
 * 管理可能性に関する情報を取り出して表示する */
/* 注意 : 発生した例外を取り込むコードを
 * 追加するとよい */
if( principal.isManageableSecurityRealm( ) )
   System.out.println( "- This realm is manageable" );
else
   System.out.println( "- This realm is not manageable" );

   break;
   .
   .
   .
```

オーガニゼーション、ロールおよびユーザのコンフィグレーション

オーガニゼーションは、Studio クライアントまたはカスタム定義のクライアントを使用して定義され、さまざまな業務エンティティ、地域、その他その企業の特定の事業に関連する分類を表します。オーガニゼーション内のロールおよびユーザに対して、セキュリティ パーミッションをさらに詳細に設定してコンフィグレーションを行うことができます。

この節の内容は以下のとおりです。

- オーガニゼーションのコンフィグレーション
- ロールのコンフィグレーション
- ユーザのコンフィグレーション

オーガニゼーションのコンフィグレーション

ここでは、オーガニゼーションのコンフィグレーション方法について説明します。内容は以下のとおりです。

- オーガニゼーションを追加する
- オーガニゼーションにユーザを追加する
- すべてのオーガニゼーション名を取得する
- 特定のオーガニゼーションに対して定義されているロールを取得する
- 特定のオーガニゼーションに対して定義されているユーザを取得する
- オーガニゼーション情報を取得する
- オーガニゼーション情報を設定する
- オーガニゼーションからユーザを削除する
- オーガニゼーションを削除する
- オーガニゼーションのコンフィグレーション例

オーガニゼーションを追加する

セキュリティ レルムにオーガニゼーションを追加するには、以下の `com.bea.wlpi.server.principal.WLPIPrincipal` メソッドを使用します。

```
public void addOrganization(
  com.bea.wlpi.common.OrganizationInfo orgInfo
) throws java.rmi.RemoteException,
  com.bea.wlpi.common.WorkflowException
```

`addOrganization()` メソッドのパラメータを次の表に示します。パラメータには値を指定する必要があります。

表 9-1 addOrganization() メソッドのパラメータ

パラメータ	説明	有効な値
orgInfo	新しいオーガニゼーションの情報	OrganizationInfo オブジェクト。 OrganizationInfo オブジェクトの定義方法については、B-9 ページの「OrganizationInfo オブジェクト」を参照。

たとえば、次のコードでは、指定された orgInfo オブジェクトの内容に基づいてオーガニゼーションが追加されます。このコード例では、principal は WLPIPrincipal EJB への EJBObject 参照を表しています。

```
principal.addOrganization(orgInfo);
```

addOrganization() メソッドの詳細については、Javadoc の com.bea.wlpi.server.principal.WLPIPrincipal を参照してください。

オーガニゼーションにユーザを追加する

オーガニゼーションにユーザを追加するには、次の com.bea.wlpi.server.principal.WLPIPrincipal メソッドを使用します。

```
public void addUserToOrganization(
  java.lang.String userId,
  java.lang.String orgId
) throws java.rmi.RemoteException,
  com.bea.wlpi.common.WorkflowException
```

addUserToOrganization() メソッドのパラメータを次の表に示します。パラメータには値を指定する必要があります。

表 9-2 addUserToOrganization() メソッドのパラメータ

パラメータ	説明	有効な値
userId	オーガニゼーションに追加するユーザの ID	有効なユーザ ID を指定する文字列。 ユーザのリストの取得方法については、9-52 ページの「すべてのユーザを取得する」を参照。

表 9-2 addUserToOrganization() メソッドのパラメータ (続き)

パラメータ	説明	有効な値
orgId	ユーザの追加先オーガニゼーションの ID	有効なオーガニゼーション ID を指定する文字列。 すべてのオーガニゼーション ID のリストの取得方法については、9-9 ページの「すべてのオーガニゼーション名を取得する」を参照。

たとえば、次のコードでは、ユーザ joe が指定されたオーガニゼーション ORG1 に追加されます。このコード例では、principal は WLPIPrincipal EJB への EJBObject 参照を表しています。

```
principal.addUserToOrganization(“joe”, “ORG1”);
```

addUserToOrganization() メソッドの詳細については、Javadoc の com.bea.wlpi.server.principal.WLPIPrincipal を参照してください。

すべてのオーガニゼーション名を取得する

セキュリティ レルムに対して定義されているすべてのオーガニゼーションのリストを取得するには、次の com.bea.wlpi.server.principal.WLPIPrincipal メソッドを使用します。

```
public java.util.List getAllOrganizations(
   boolean obtainAttributes
) throws java.rmi.RemoteException,
  com.bea.wlpi.common.WorkflowException
```

getAllOrganizations() メソッドのパラメータを次の表に示します。パラメータには値を指定する必要があります。

表 9-3 getAllOrganizations() メソッドのパラメータ

パラメータ	説明	有効な値
obtainAttributes	すべての属性を取得するのか、オーガニゼーション ID のみを取得するのかを指定するブール型フラグ	true (すべての属性) または false (オーガニゼーション ID のみ)。

このメソッドは、com.bea.wlpi.common.OrganizationInfo オブジェクトのリストを返します。各オーガニゼーションについての情報にアクセスするには、B-9 ページの「OrganizationInfo オブジェクト」に記載の OrganizationInfo オブジェクト メソッドを使用します。

たとえば、次のコードは、オーガニゼーション ID のみを取得して（obtainAttributes パラメータが false に設定されている）、orgList リスト変数に保存します。このコード例では、principal は WLPIPrincipal EJB への EJBObject 参照を表しています。

```
List orgList = principal.getAllOrganizations(false);
```

getAllOrganizations() メソッドの詳細については、Javadoc の com.bea.wlpi.server.principal.WLPIPrincipal を参照してください。

特定のオーガニゼーションに対して定義されているロールを取得する

特定のオーガニゼーションに対して定義されているロールのリストは、以下に記載のメソッドを使用して取得できます。また、特定のロールが任意のオーガニゼーションに対して定義されているかどうかを調べます。

特定のオーガニゼーションに対して定義されているロールのリストを取得する

特定のオーガニゼーションに対して定義されているすべてのロールのリストを取得するには、次の com.bea.wlpi.server.principal.WLPIPrincipal メソッドを使用します。

```
public java.util.List getRolesInOrganization(
  java.lang.String orgId,
  boolean obtainAttributes
) throws java.rmi.RemoteException,
  com.bea.wlpi.common.WorkflowException
```

getRolesInOrganization() メソッドのパラメータを次の表に示します。パラメータには値を指定する必要があります。

表 9-4 getRolesInOrganization() メソッドのパラメータ

パラメータ	説明	有効な値
orgId	ロールを取得するオーガニゼーションの ID	有効なオーガニゼーション ID を指定する文字列。 すべてのオーガニゼーション ID のリストの取得方法については、9-9 ページの「すべてのオーガニゼーション名を取得する」を参照。
obtainAttributes	すべての属性を取得するのか、ロール ID のみを取得するのかを指定するフラグ	true (すべての属性) または false (ロール ID のみ)。

このメソッドは、com.bea.wlpi.common.OrganizationInfo オブジェクトのリストを返します。各ロールについての情報にアクセスするには、B-18 ページの「RoleInfo オブジェクト」に記載の RoleInfo オブジェクト メソッドを使用します。

たとえば、次のコードは、オーガニゼーション ORG1 内のロールを取得して、すべての属性を返します（obtainAttributes パラメータが true に設定されているため）。このコード例では、principal は WLPIPrincipal EJB への EJBObject 参照を表しています。

```
List roles = principal.getRolesInOrganization(“ORG1”, true);
```

getRolesInOrganization() メソッドの詳細については、Javadoc の com.bea.wlpi.server.principal.WLPIPrincipal を参照してください。

特定のオーガニゼーションに対してロールが定義されているかどうか調べる

特定のロールが任意のオーガニゼーションに対して定義されているかどうか調べるには、次の com.bea.wlpi.server.principal.WLPIPrincipal メソッドを使用します。

```
public boolean isRoleInOrganization(
  java.lang.String roleId,
  java.lang.String orgId
) throws java.rmi.RemoteException,
  com.bea.wlpi.common.WorkflowException
```

isRoleInOrganization() メソッドのパラメータを次の表に示します。パラメータには値を指定する必要があります。

表 9-5 isRolesInOrganization() メソッドのパラメータ

パラメータ	説明	有効な値
roleId	確認するロールの ID	有効なロール ID を指定する文字列。 ロールのリストの取得方法については、9-10 ページの「特定のオーガニゼーションに対して定義されているロールを取得する」を参照。
orgId	確認するオーガニゼーションの ID	有効なオーガニゼーション ID を指定する文字列。 すべてのオーガニゼーション ID のリストの取得方法については、9-9 ページの「すべてのオーガニゼーション名を取得する」を参照。

このメソッドは、ロールがオーガニゼーションに対して定義されている場合は true を、そうでない場合は false を返します。

たとえば、次のコードでは、role1 ロールがオーガニゼーション ORG1 に対して指定されているかどうか調べます。このコード例では、principal は WLPIPrincipal EJB への EJBObject 参照を表しています。

```
List roles = principal.isRoleInOrganization(“role1”, “ORG1”);
```

isRoleInOrganization() メソッドの詳細については、Javadoc の com.bea.wlpi.server.principal.WLPIPrincipal を参照してください。

特定のオーガニゼーションに対して定義されているユーザを取得する

特定のオーガニゼーションに対して定義されているすべてのユーザのリストは、以下に記載のメソッドを使用して、取得できます。また、特定のユーザが任意のオーガニゼーションに対して定義されているかどうかを調べることもできます。

特定のオーガニゼーションに対して定義されているユーザのリストを取得する

特定のオーガニゼーションに対して定義されているユーザのリストを取得するには、次の com.bea.wlpi.server.principal.WLPIPrincipal メソッドを使用します。

```
public java.util.List getUsersInOrganization(
  java.lang.String orgId,
  boolean obtainAttributes
) throws java.rmi.RemoteException,
  com.bea.wlpi.common.WorkflowException
```

getUsersInOrganization() メソッドのパラメータを次の表に示します。パラメータには値を指定する必要があります。

表 9-6 getUsersInOrganization() メソッドのパラメータ

パラメータ	説明	有効な値
orgId	ユーザのリストを取得するオーガニゼーションの ID	有効なオーガニゼーション ID を指定する文字列。 すべてのオーガニゼーション ID のリストの取得方法については、9-9 ページの「すべてのオーガニゼーション名を取得する」を参照。
obtainAttributes	すべての属性を取得するのか、ユーザ ID のみを取得するのかを指定するフラグ	true (すべての属性) または false (ユーザ ID のみ)。

このメソッドは、com.bea.wlpi.common.UserInfo オブジェクトのリストを返します。各ロールについての情報にアクセスするには、B-28 ページの「UserInfo オブジェクト」に記載の UserInfo オブジェクト メソッドを使用します。

たとえば、次のコードでは、オーガニゼーション ORG1 内のユーザのリストを取得して、すべての属性を返します（obtainAttributes パラメータが true に設定されているため）。このコード例では、principal は WLPIPrincipal EJB への EJBObject 参照を表しています。

```
List users = principal.getUsersInOrganization(“ORG1”, true);
```

getUsersInOrganization() メソッドの詳細については、Javadoc の com.bea.wlpi.server.principal.WLPIPrincipal を参照してください。

特定のオーガニゼーションに対してユーザが定義されているかどうか調べる

特定のユーザが任意のオーガニゼーションに対して定義されているかどうか調べるには、次の com.bea.wlpi.server.principal.WLPIPrincipal メソッドを使用します。

```
public void isUserInOrganization(
  java.lang.String userId,
  java.lang.String orgId
) throws java.rmi.RemoteException,
  com.bea.wlpi.common.WorkflowException
```

isUserInOrganization() メソッドのパラメータを次の表に示します。パラメータには値を指定する必要があります。

表 9-7 isUserInOrganization() メソッドのパラメータ

パラメータ	説明	有効な値
userId	確認するユーザの ID	有効なユーザ ID を指定する文字列。 ユーザのリストの取得方法については、9-52 ページの「すべてのユーザを取得する」を参照。
orgId	確認するオーガニゼーションの ID	有効なオーガニゼーション ID を指定する文字列。 すべてのオーガニゼーション ID のリストの取得方法については、9-9 ページの「すべてのオーガニゼーション名を取得する」を参照。

このメソッドは、ユーザがオーガニゼーションに対して定義されている場合は true を、そうでない場合は false を返します。

たとえば、次のコードでは、user1 ユーザがオーガニゼーション ORG1 に対して定義されているかどうかを調べます。このコード例では、principal は WLPIPrincipal EJB への EJBObject 参照を表しています。

```
List roles = principal.isUserInOrganization(“user1”, “ORG1”);
```

isUserInOrganization() メソッドの詳細については、Javadoc の com.bea.wlpi.server.principal.WLPIPrincipal を参照してください。

オーガニゼーション情報を取得する

オーガニゼーション情報を取得するには、次の com.bea.wlpi.server.principal.WLPIPrincipal メソッドを使用します。

```
public com.bea.wlpi.common.OrganizationInfo getOrganizationInfo(
  java.lang.String orgId
) throws java.rmi.RemoteException,
  com.bea.wlpi.common.WorkflowException
```

getOrganizationInfo() メソッドのパラメータを次の表に示します。パラメータには値を指定する必要があります。

表 9-8 getOrganizationInfo() メソッドのパラメータ

パラメータ	説明	有効な値
orgId	情報を取得するオーガニゼーションの ID	有効なオーガニゼーション ID を指定する文字列。 すべてのオーガニゼーション ID のリストの取得方法については、9-9 ページの「すべてのオーガニゼーション名を取得する」を参照。

このメソッドは、com.bea.wlpi.common.OrganizationInfo オブジェクトを返します。オーガニゼーションについての情報にアクセスするには、B-9 ページの「OrganizationInfo オブジェクト」に記載の OrganizationInfo オブジェクト メソッドを使用します。

このコード例では、principal は WLPIPrincipal EJB への EJBObject 参照を表しています。

```
OrganizationInfo orgInfo = principal.getOrganizationInfo(orgId)
```

getOrganizationInfo() メソッドの詳細については、Javadoc の com.bea.wlpi.server.principal.WLPIPrincipal を参照してください。

オーガニゼーション情報を設定する

オーガニゼーション情報を設定するには、次の `com.bea.wlpi.server.principal.WLPIPrincipal` メソッドを使用します。

```
public void setOrganizationInfo(
  com.bea.wlpi.common.OrganizationInfo orgInfo
) throws java.rmi.RemoteException
```

`setOrganizationInfo()` メソッドのパラメータを次の表に示します。パラメータには値を指定する必要があります。

表 9-9 setOrganizationInfo() メソッドのパラメータ

パラメータ	説明	有効な値
`orgInfo`	更新するオーガニゼーション情報	`OrganizationInfo` オブジェクト。 すべての `OrganizationInfo` オブジェクトのリストを取得方法については、9-9 ページの「すべてのオーガニゼーション名を取得する」を参照（オーガニゼーション属性が誤って消去されることがないように、ブール型パラメータ `obtainAttributes` を必ず `true` に設定すること）。`OrganizationInfo` オブジェクトを更新方法については、B-9 ページの「OrganizationInfo オブジェクト」 を参照。

たとえば、次のコードでは、指定された `orgInfo` オブジェクトの内容に基づいてオーガニゼーションが更新されます。このコード例では、`principal` は `WLPIPrincipal` EJB への `EJBObject` 参照を表しています。

```
principal.updateOrganization(orgInfo);
```

`setOrganizationInfo()` メソッドの詳細については、Javadoc の `com.bea.wlpi.server.principal.WLPIPrincipal` を参照してください。

オーガニゼーションからユーザを削除する

オーガニゼーションからユーザを削除するには、次の `com.bea.wlpi.server.principal.WLPIPrincipal` メソッドを使用します。

```
public void removeUserFromOrganization(
  java.lang.String userId,
  java.lang.String orgId
) throws java.rmi.RemoteException,
  com.bea.wlpi.common.WorkflowException
```

`removeUserFromOrganization()` メソッドのパラメータを次の表に示します。パラメータには値を指定する必要があります。

表 9-10 removeUserFromOrganization() メソッドのパラメータ

パラメータ	説明	有効な値
`userId`	削除するユーザの ID	有効なユーザ ID を指定する文字列。 ユーザのリストの取得方法については、9-52 ページの「すべてのユーザを取得する」を参照。
`orgId`	ユーザに関連付けられたオーガニゼーションの ID	有効なオーガニゼーション ID を指定する文字列。 すべてのオーガニゼーション ID のリストの取得方法については、9-9 ページの「すべてのオーガニゼーション名を取得する」を参照。

たとえば、次のコードは、ユーザ `user1` をオーガニゼーション `ORG1` から削除します。このコード例では、`principal` は `WLPIPrincipal` EJB への `EJBObject` 参照を表しています。

```
principal.removeUserFromOrganization(“user1”, “ORG1”);
```

`removeUserFromOrganization()` メソッドの詳細については、Javadoc の `com.bea.wlpi.server.principal.WLPIPrincipal` を参照してください。

オーガニゼーションを削除する

オーガニゼーションを削除するには、次の com.bea.wlpi.server.principal.WLPIPrincipal メソッドを使用します。

```
public void deleteOrganization(
  java.lang.String orgId
) throws java.rmi.RemoteException,
  com.bea.wlpi.common.WorkflowException
```

deleteOrganization() メソッドのパラメータを次の表に示します。パラメータには値を指定する必要があります。

表 9-11 deleteOrganization() メソッドのパラメータ

パラメータ	説明	有効な値
orgId	削除するオーガニゼーションの ID	有効なオーガニゼーション ID を指定する文字列。 すべてのオーガニゼーション ID のリストの取得方法については、9-9 ページの「すべてのオーガニゼーション名を取得する」を参照。

次のサンプル コードでは、オーガニゼーション ID で指定されたオーガニゼーションを削除します。この例では、principal は WLPIPrincipal EJB への EJBObject 参照を表しています。

```
principal.deleteOrganization(organization.getId())
```

com.bea.wlpi.common.OrganizationInfo オブジェクトである Organization と関連付けられたメソッドを使用してオーガニゼーション ID を取得します。9-9 ページの「すべてのオーガニゼーション名を取得する」に記載のメソッドを使用して、organization オブジェクトを取得できます。

deleteOrganization() メソッドの詳細については、Javadoc の com.bea.wlpi.server.principal.WLPIPrincipal を参照してください。

オーガニゼーションのコンフィグレーション例

この節では、コマンドライン管理のコード例から抜粋して、オーガニゼーションのコンフィグレーション方法を示します。

注意： コマンドライン管理サンプルの詳細については、1-23 ページの「コマンドライン管理サンプル」を参照してください。

このサンプルでは、ユーザと通信するための入力ストリームが定義されており、ユーザは以下のアクションのいずれかを指定するよう求められます。

- オーガニゼーションを追加する
- オーガニゼーションを削除する
- オーガニゼーション情報を設定する
- オーガニゼーションにユーザを追加する
- オーガニゼーションからユーザを削除する
- すべてのオーガニゼーション名を取得する
- 特定のオーガニゼーションに対して定義されているユーザを取得する
- 特定のオーガニゼーションに対してユーザが定義されているかどうか調べる
- 特定のオーガニゼーションに対して定義されているロールを取得する
- 特定のオーガニゼーションに対してロールが定義されているかどうか調べる

重要なコード行は、**太字**で強調されています。このコード例では、`principal` は `WLPIPrincipal` EJB への `EJBObject` 参照を表しています。

```
/* ユーザとの通信のための入力ストリームを作成する */
stdIn = new BufferedReader( new InputStreamReader( System.in ) );

/* ツール タイトルを表示する */
   System.out.print( "\n---  Command Line Administration v1.1  ---" );

/* メイン メニューを表示してユーザと交信する */
while( true ) {
/* メニューを表示する */
   System.out.println( "\n---              Main Menu            ---" );
   System.out.println( "\nEnter choice:" );
   System.out.println( "1) Organizations" );
   System.out.println( "2) Roles" );
   System.out.println( "3) Users" );
   System.out.println( "4) Security Realm" );
   System.out.println( "5) Business Operations" );
   System.out.println( "6) Event Keys" );
   System.out.println( "7) Business Calendars" );
   System.out.println( "8) EJB Catalog" );
   System.out.println( "9) Server Properties" );
```

```
   System.out.println( "Q) Quit" );
   System.out.print( ">> " );
      .
      .
      .
public static void mngOrganizations( ) {
   String answer;
   String calendarId;
   String orgId;
   String userId;
   String eMail;
   String defaultOrgId;

   /* ユーザとの通信のための入力ストリームを作成する */
   BufferedReader stdIn = new BufferedReader( new InputStreamReader( System.in )
);

   try {
   /* WLPI 公開 API メソッド */
      boolean isRealmManageable = principal.isManageableSecurityRealm( );

      /* メニューを表示してユーザと交信する */
      while( true ) {
         /* メニューを表示する */
         System.out.println( "\n\n---        WLPI Organizations        ---" );
         System.out.println( "\nEnter choice:" );
         /* このレルムは管理可能か ? */
         if( isRealmManageable ) {
            /* このレルムは管理可能レルム。管理可能レルムを
             * 必要とする表示メニュー オプション */
            System.out.println( "0) Add a new Organization" );
            System.out.println( "1) Delete an Organization" );
            System.out.println( "2) Update Organization Info" );
            System.out.println( "3) Assign a User to an Organization" );
            System.out.println( "4) Remove a User from an Organization" );
         }
         System.out.println( "5) List all Organizations" );
         System.out.println( "6) List Organization Info" );
         System.out.println( "7) List Users assigned to an Organization" );
         System.out.println( "8) Is User assigned to an Organization" );
         System.out.println( "9) List Roles defined in an Organization" );
         System.out.println( "A) Is Role in an Organization" );
         System.out.println( "B) Back to previous menu" );
         System.out.println( "Q) Quit" );
         System.out.print( ">> " );

         /* ユーザの選択を取得する */
         String line = stdIn.readLine( );
```

```
/* ユーザが選択を行わないで〔Enter〕を押したか ? */
if( line.equals( "" ) )
   continue;


/* ユーザが複数の文字を入力したか ? */
else if( line.length( ) > 1 ) {
   System.out.println( "*** Invalid selection" );
   continue;
}
/* レルムが管理不可能で、ユーザが非表示の選択肢を入力したか ? */
else if( !isRealmManageable && line.charAt( 0 ) < '5' ) {
   System.out.println( "*** Invalid selection" );
   continue;
}

/* 大文字および文字へ変換する */
char choice = line.toUpperCase( ).charAt( 0 );

/* ユーザの選択を処理する */
switch( choice ) {
.
.
.
```

オーガニゼーションを追加する

オーガニゼーションを追加する方法を示します。

```
/* 新しいオーガニゼーションを追加する */
case '0' :
   /* 追加する新しいオーガニゼーションのオーガニゼーション ID を取得する */
   if( ( orgId = askQuestion( "\nEnter new Organization ID" ) )
      == null ) {
      /* ユーザによる操作の取り消し */
      System.out.println( "*** Cancelled" );
      break;
   }

   /* このオーガニゼーションに割り当てるカレンダー ID を取得する ; ( 省略可能 ) */
   calendarId = askQuestion( "Enter Calendar ID (press
      enter for none)" );

   /* OrganizationInfo オブジェクトを作成する ;
    * 新しいオーガニゼーションの追加に必要 */
```

```
OrganizationInfo orgInfo =
  new OrganizationInfo( orgId, calendarId );

try {
   /* WLPI 公開 API メソッド */
   /* 新しいオーガニゼーションを追加する */
   principal.addOrganization( orgInfo );

   /* 正常終了（例外の発生なし）*/
   System.out.println( "- Success" );
}
catch( Exception e ) {
   System.out.println( "*** Unable to add the organization\n" );
   System.err.println( e );
}
   break;
.
.
.
```

オーガニゼーションを削除する

オーガニゼーションを削除する方法を示します。

```
/* オーガニゼーションを削除する */
case '1' :
   /* 削除するオーガニゼーションのオーガニゼーション ID を取得する */
   if( ( orgId = askQuestion( "\nEnter Organization
      ID to delete" ) ) == null ) {
   /* ユーザによる操作の取り消し */
   System.out.println( "*** Cancelled" );
   break;
}

try {
   /* WLPI 公開 API メソッド */
   /* オーガニゼーションを削除する */
   principal.deleteOrganization( orgId );

   /* 正常終了（例外の発生なし）*/
   System.out.println( "- Success" );
}
catch( Exception e ) {
   System.out.println( "*** Unable to delete the organization\n" );
   System.err.println( e );
}
```

```
   break;
   .
   .
   .
```

オーガニゼーション情報を設定する

オーガニゼーションについての情報を設定する方法を示します。

```
/* オーガニゼーション情報を更新する */
case '2' :
/* 更新するオーガニゼーションのオーガニゼーション ID を取得する */
   if( ( orgId = askQuestion(
      "\nEnter Organization ID to update" ) ) == null ) {
   /* ユーザによる操作の取り消し */
   System.out.println( "*** Cancelled" );
   break;
}

  /* カレンダー ID を取得(WLPI v1.2.1 で定義されているオーガニゼーション属性のみ、
    * つまり、更新可能な属性のみ) */
   calendarId = askQuestion(
      "Enter new Calendar ID (press enter for none)" );

   /* OrganizationInfo オブジェクトを作成する
    * オーガニゼーションの更新に必要 */
   orgInfo = new OrganizationInfo( orgId, calendarId );

   try {
      /* WLPI 公開 API メソッド */
      /* オーガニゼーションを更新する(カレンダーの読み込み) */
      principal.setOrganizationInfo( orgInfo );

      /* 正常終了(例外の発生なし) */
      System.out.println( "- Success" );
   }
   catch( Exception e ) {
      System.out.println( "*** Unable to update the organization\n" );
      System.err.println( e );
   }
   break;            .
   .
   .
```

オーガニゼーションにユーザを追加する

ユーザをオーガニゼーションに割り当てる方法を示します。

```
/* オーガニゼーションにユーザを割り当てる */
        case '3' :
           /* オーガニゼーションに割り当てるユーザのユーザ ID を取得する */
           if( ( userId = askQuestion( "\nEnter User ID to assign" )
              ) == null ) {
           /* ユーザによる操作の取り消し */
           System.out.println( "*** Cancelled" );
           break;
           }

           /* ユーザを割り当てるオーガニゼーションのオーガニゼーション ID を取得する */
           if( ( orgId = askQuestion( "Enter Organization ID" ) ) == null ) {
              /* ユーザによる操作の取り消し */
              System.out.println( "*** Cancelled" );
              break;
           }

           try {
              /* WLPI 公開 API メソッド */
              /* ユーザをオーガニゼーションに割り当てる */
              principal.addUserToOrganization( userId, orgId );

              /* 正常終了（例外の発生なし）*/
              System.out.println( "- Success" );
           }
           catch( Exception e ) {
              System.out.println( "*** Unable to
                 assign user to the organization\n" );
              System.err.println( e );
           }
           break;
           .
           .
           .
```

オーガニゼーションからユーザを削除する

ユーザをオーガニゼーションから削除する方法を示します。

```
/* オーガニゼーションからユーザを削除する */
case '4' :
   /* オーガニゼーションから削除するユーザのユーザ ID を取得する */
```

```
   if( ( userId = askQuestion(
      "\nEnter User ID to remove" ) ) == null ) {
      /* ユーザによる操作の取り消し */
      System.out.println( "*** Cancelled" );
      break;
    }

   /* ユーザを削除するオーガニゼーションのオーガニゼーション ID を取得する */
   if( ( orgId = askQuestion( "Enter Organization ID" ) ) == null ) {
      /* ユーザによる操作の取り消し */
      System.out.println( "*** Cancelled" );
      break;
   }

   try {
      /* WLPI 公開 API メソッド */
      /* このユーザをオーガニゼーションから削除する */
      principal.removeUserFromOrganization( userId, orgId );

      /* 正常終了（例外の発生なし）*/
      System.out.println( "- Success" );
   }
   catch( Exception e ) {
      System.out.println(
         "*** Unable to remove user from organization\n" );
      System.err.println( e );
   }
   break;
   .
   .
   .
```

すべてのオーガニゼーション名を取得する

すべてのオーガニゼーションのリストを取得する方法を示します。

```
/* すべてのオーガニゼーションをリスト アップする */
case '5' :
   /* オーガニゼーション属性の表示が必要かどうかを
    * 選択するようにユーザに求める */
   if( ( answer = askQuestion(
      "\nList all attributes (y/n)?" ) ) == null ) {
      /* ユーザによる操作の取り消し */
      System.out.println( "*** Cancelled" );
      break;
   }
```

```
/* 応答を解析する */
boolean isGetAttributes = ( answer.equals( "y" ) ||
   answer.equals( "Y" ) );

/* WLPI 公開 API メソッド */
/* すべてのオーガニゼーションを検索する */
/* 注意 : 発生した例外を取り込むコードを
 * 追加するとよい */
List orgList = principal.getAllOrganizations( isGetAttributes );

/* 定義されているオーガニゼーションはあるか ? */
if( orgList.size( ) == 0 )
   System.out.println( "\nNo Organization defined" );
else
   System.out.println( "\nDefined organizations:" );

/* リストを処理してオーガニゼーションと属性を表示する */
for( int i = 0; i < orgList.size( ); i++ ) {
   /* リストから要素を取り出す */
   orgInfo = ( OrganizationInfo )orgList.get( i );
   /* オーガニゼーション ID を取り出して表示する */
   System.out.println( "- ID: " + orgInfo.getOrgId( ) );

   /* 属性を表示するか ? */
   if( isGetAttributes ) {
      /* カレンダー ID を取得する */
      if( ( calendarId = orgInfo.getCalendarId( ) ) == null )
         /* このオーガニゼーションに対して定義された ID がない場合は、*/
         'None' という ID が割り当てられる

      /* カレンダー ID を表示する */
      System.out.println( "
         Attributes: Calendar ID=" + calendarId + "\n" );
   }
}
break;
.
.
.
```

オーガニゼーション情報を取得する

次のコードの抜粋では、オーガニゼーションについての情報を取得する方法を示します。

```
/* オーガニゼーション情報をリストアップする */
case '6' :
   /* 表示するオーガニゼーションのオーガニゼーション ID を取得する */
   if( ( orgId = askQuestion( "\nEnter Organization ID" ) ) == null ) {
      /* ユーザによる操作の取り消し */
      System.out.println( "*** Cancelled" );
      break;
   }

   try {
      /* WLPI 公開 API メソッド */
      /* このオーガニゼーションについての情報を取り出す */
      orgInfo = principal.getOrganizationInfo( orgId );

      /* オーガニゼーション ID を取り出して表示する */
      System.out.println( "- ID: " + orgInfo.getOrgId( ) );

      /* カレンダー ID を取得する */
      if( ( calendarId = orgInfo.getCalendarId( ) ) == null )
         /* このオーガニゼーションに対して定義された ID がない場合は、'None' という ID が割り当てられる */
         calendarId = "None";

      /* カレンダー ID を表示する */
      System.out.println( "
         Attributes: Calendar ID=" + calendarId + "\n" );
   }
   catch( Exception e ) {
      System.out.println(
         "*** Unable to retrieve organization info\n" );
      System.err.println( e );
   }
   break;
   .
   .
   .
```

特定のオーガニゼーションに対して定義されているユーザを取得する

次のコードの抜粋では、あるオーガニゼーションに対して定義されているユーザのリストを取得する方法を示します。

```
/* オーガニゼーションに割り当てられたユーザをリストアップする */
case '7' :
   /* クエリするオーガニゼーションのオーガニゼーション ID を取得する */
   if( ( orgId = askQuestion( "\nEnter Organization ID" ) ) == null ) {
```

```
            /* ユーザによる操作の取り消し */
            System.out.println( "*** Cancelled" );
            break;
         }

         /* ユーザ属性の表示が必要かどうかを
          * 選択するようにユーザに求める */
         if( ( answer = askQuestion(
            "List user attributes (y/n)?" ) ) == null ) {
            /* ユーザによる操作の取り消し */
            System.out.println( "*** Cancelled" );
            break;
         }

         /* 応答を解析する */
         isGetAttributes = ( answer.equals( "y" ) || answer.equals( "Y" ) );

         /* WLPI 公開 API メソッド */
         /* このオーガニゼーションに割り当てられたすべてのユーザを取り出す */
         /* 注意 : 発生した例外を取り込むコードを
          * 追加するとよい */
         List userList = principal.getUsersInOrganization(
            orgId, isGetAttributes );

         /* 割り当てられているユーザはあるか ? */
         if( userList.size( ) == 0 )
            System.out.println( "\nNo user assigned" );
         else
            System.out.println( "\nAssigned Users:" );

         /* リストを処理して、ユーザと属性を表示する */
         for( int i = 0; i < userList.size( ); i++ ) {
         /* リストから要素を取り出す */
            UserInfo userInfo = ( UserInfo )userList.get( i );
            /* ユーザ ID を取り出して表示する */
            System.out.println( "- User ID: " + userInfo.getUserId( ) );

            /* 属性を表示するか ? */
            if( isGetAttributes ) {
               /* 電子メール アドレスを取り出す */
               if( ( eMail = userInfo.getEMailAddress( ) ) == null )
                  /* このユーザに対して定義された ID がない場合は、'None' という ID が
割り当てられる */
                  eMail = "None";

               /* デフォルトのオーガニゼーション ID を取り出す */
               if( ( defaultOrgId = userInfo.getDefaultOrgId( ) ) == null )
```

```
                    /* このユーザに対して定義された ID がない場合は、'None' という ID が
割り当てられる */
                    defaultOrgId = "None";

                /* カレンダー ID を取り出す */
                if( ( calendarId = userInfo.getCalendarId( ) ) == null )
                    /* このユーザに対して定義された ID がない場合は、'None' という ID が
割り当てられる */
                    calendarId = "None";

                /* 電子メール アドレスを表示（デフォルトのオーガニゼーション ID とカレン
ダー ID）する */
                System.out.println( "  Attributes:\n  - eMail: " + eMail );
                System.out.println( "  - Default ORG ID: " + defaultOrgId );
                System.out.println( "  - Calendar ID=" + calendarId + "\n" );
            }
        }
        break;
        .
        .


        .
```

特定のオーガニゼーションに対してユーザが定義されているかどうか調べる

ユーザがオーガニゼーションに対して定義されているかどうかを調べる方法を示します。

```
/* ユーザはオーガニゼーションに割り当てられているか */
case '8' :
   /* クエリするユーザのユーザ ID を取得する */
   if( ( userId = askQuestion( "\nEnter User ID" ) ) == null ) {
      /* ユーザによる操作の取り消し */
      System.out.println( "*** Cancelled" );
      break;
   }

   /* クエリするオーガニゼーションのオーガニゼーション ID を取得する */
   if( ( orgId = askQuestion( "Enter Organization ID" ) ) == null ) {
   /* ユーザによる操作の取り消し */
      System.out.println( "*** Cancelled" );
      break;
   }

   /* WLPI 公開 API メソッド */
```

```
   /* ユーザはこのオーガニゼーションに割り当てられているか ? */
   /* 注意 : 発生した例外を取り込むコードを
    * 追加するとよい */
   if( principal.isUserInOrganization( userId, orgId ) )
      System.out.println( "User is assigned to the organization" );
   else
      System.out.println( "User is not assigned to the organization" );
   break;
   .
   .
   .
```

特定のオーガニゼーションに対して定義されているロールを取得する

特定のオーガニゼーションに対して定義されているロールのリストを取得する方法を示します。

```
/* オーガニゼーションに対して定義されたロールをリスト アップする */
case '9' :
   /* クエリするオーガニゼーションのオーガニゼーション ID を取得する */
   if( ( orgId = askQuestion( "Enter Organization ID" ) ) == null ) {
      /* ユーザによる操作の取り消し */
      System.out.println( "*** Cancelled" );
      break;
   }

   /* ユーザ属性の表示が必要かどうかを選択するようにユーザに求める */
  if( ( answer = askQuestion(
     "List role attributes (y/n)?" ) ) == null ) {
      /* ユーザによる操作の取り消し */
      System.out.println( "*** Cancelled" );
      break;
   }

   /* 応答を解析する */
     isGetAttributes = ( answer.equals( "y" ) || answer.equals( "Y" ) );

   /* WLPI 公開 API メソッド */
   /* このオーガニゼーションで定義されているすべてのロールを取り出す */
   /* 注意 : 発生した例外を取り込むコードを
    * 追加するとよい */
   List roleList = principal.getRolesInOrganization(
      orgId, isGetAttributes );

   /* 定義されているロールはあるか ? */
```

```
            if( roleList.size( ) == 0 )
               System.out.println( "\nNo roles defined" );
            else
               System.out.println( "\nRoles Defined:" );

            /* リストを処理してロールと属性を表示する */
            for( int i = 0; i < roleList.size( ); i++ ) {
               /* リストから要素を取り出す */
               RoleInfo roleInfo = ( RoleInfo )roleList.get( i );
               /* ロール ID を取り出して表示する */
               System.out.println( "- Role ID: " + roleInfo.getRoleId( ) );

               /* 属性を表示するか ? */
               if( isGetAttributes ) {
                  /* カレンダー ID を取り出す */
                  if( ( calendarId = roleInfo.getCalendarId( ) ) == null )
                     /* このロールに対して定義された ID がない場合は、'None' という ID が
割り当てられる */
                     calendarId = "None";

                  /* カレンダー ID を表示する */
                  System.out.println(
                    "  Attributes: Calendar ID=" + calendarId + "\n" );
               }
            }
            break;
            .


            .
            .
```

特定のオーガニゼーションに対してロールが定義されているかどうか調べる

ユーザがオーガニゼーションに対して定義されているかどうかを調べる方法を示します。

```
         /* ロールがオーガニゼーションにあるか */
         case 'A' :
            /* クエリするロールのロール ID を取得する */
            if( ( roleId = askQuestion( "\nEnter Role ID" ) ) == null ) {
               /* ユーザによる操作の取り消し */
               System.out.println( "*** Cancelled" );
               break;
            }
```

```
   /* クエリするオーガニゼーションのオーガニゼーション ID を取得する */
   if( ( orgId = askQuestion( "Enter Organization ID" ) ) == null ) {
      /* ユーザによる操作の取り消し */
      System.out.println( "*** Cancelled" );
      break;
   }

   /* WLPI 公開 API メソッド */
   /* このオーガニゼーションでロールは定義されているか ? */
   /* 注意 : 発生した例外を取り込むコードを
      追加するとよい */
   if( principal.isRoleInOrganization( roleId, orgId) )
      System.out.println( "Role defined in the organization" );
   else
      System.out.println( "Role not defined in the organization" );
break;
.
.
.
```

ロールのコンフィグレーション

ここでは、ロールのコンフィグレーション方法について説明します。内容は以下のとおりです。

- ロールを追加する
- ロールにユーザを追加する
- 特定のロールに対して定義されているユーザを取得する
- ロール情報を取得する
- ロール情報を設定する
- ロールからユーザを削除する
- ロールを削除する
- ロールのコンフィグレーション例

ロールを追加する

セキュリティ レルムにロールを追加するには、以下の com.bea.wlpi.server.principal.WLPIPrincipal メソッドを使用します。

```
public void addRole(
  com.bea.wlpi.common.RoleInfo roleInfo
) throws java.rmi.RemoteException,
  com.bea.wlpi.common.WorkflowException
```

addRole() メソッドのパラメータを次の表に示します。パラメータには値を指定する必要があります。

表 9-12 addRole() メソッドのパラメータ

パラメータ	説明	有効な値
roleInfo	新しいロール情報	RoleInfo オブジェクト。 RoleInfo オブジェクトの定義方法については、B-18 ページの「RoleInfo オブジェクト」を参照。

たとえば、次のコードでは、指定された roleInfo オブジェクトの内容に基づいてロールが追加されます。このコード例では、principal は WLPIPrincipal EJB への EJBObject 参照を表しています。

```
principal.addRole(roleInfo);
```

addRole() メソッドの詳細については、Javadoc の com.bea.wlpi.server.principal.WLPIPrincipal を参照してください。

ロールにユーザを追加する

ロールにユーザを追加するには、次の com.bea.wlpi.server.principal.WLPIPrincipal メソッドを使用します。

```
public void addUserToRole(
  java.lang.String userId,
  java.lang.String orgId,
  java.lang.String roleId
) throws java.rmi.RemoteException,
  com.bea.wlpi.common.WorkflowException
```

addUserToRole() メソッドのパラメータを次の表に示します。パラメータには値を指定する必要があります。

表 9-13 addUserToRole() メソッドのパラメータ

パラメータ	説明	有効な値
userId	ロールに追加するユーザの ID	有効なユーザ ID を指定する文字列。 ユーザのリストの取得方法については、9-52 ページの「すべてのユーザを取得する」を参照。
orgId	ユーザに関連付けられたオーガニゼーションの ID	有効なオーガニゼーション ID を指定する文字列。 すべてのオーガニゼーション ID のリストの取得方法については、9-9 ページの「すべてのオーガニゼーション名を取得する」を参照。
roleId	ユーザの追加先ロールの ID	有効なロール ID を指定する文字列。 ロールのリストの取得方法については、9-36 ページの「ロール情報を取得する」を参照。

たとえば、次のコードでは、ユーザ user1 がオーガニゼーション ORG1 の role1 ロールに追加されます。このコード例では、principal は WLPIPrincipal EJB への EJBObject 参照を表しています。

```
principal.addUserToRole(“user1”, “ORG1”, “role1”);
```

addUserToRole() メソッドの詳細については、Javadoc の com.bea.wlpi.server.principal.WLPIPrincipal を参照してください。

特定のロールに対して定義されているユーザを取得する

特定のロールに対して定義されているユーザのリストを取得するには、次の com.bea.wlpi.server.principal.WLPIPrincipal メソッドを使用します。

```
public java.util.List getUsersInRole(
  java.lang.String roleId,
  java.lang.String orgId,
  boolean obtainAttributes
```

```
) throws java.rmi.RemoteException,
  com.bea.wlpi.common.WorkflowException
```

getUsersInRole() メソッドのパラメータを次の表に示します。パラメータには値を指定する必要があります。

表 9-14 getUsersInRole() メソッドのパラメータ

パラメータ	説明	有効な値
roleId	取得するユーザが定義されているロールの ID	有効なロール ID を指定する文字列。 すべてのロール ID のリストを取得する方法については、9-10 ページの「特定のオーガニゼーションに対して定義されているロールを取得する」を参照。
orgId	ロールに関連付けられたオーガニゼーションの ID	有効なオーガニゼーション ID を指定する文字列。 すべてのオーガニゼーション ID のリストの取得方法については、9-9 ページの「すべてのオーガニゼーション名を取得する」を参照。
obtainAttributes	すべての属性を取得するのか、ユーザ ID のみを取得するのかを指定するフラグ	true (すべての属性) または false (ユーザ ID のみ)。

このメソッドは、com.bea.wlpi.common.UserInfo オブジェクトのリストを返します。各ユーザについての情報にアクセスするには、B-28 ページの「UserInfo オブジェクト」に記載の UserInfo オブジェクト メソッドを使用します。

たとえば、次のコードでは、オーガニゼーション ORG1 内のロール role1 に対して定義されているユーザのリストが取得され、すべての属性が返されます (obtainAttributes パラメータが true に設定されているため)。このコード例では、principal は WLPIPrincipal EJB への EJBObject 参照を表しています。

```
List users = principal.getUsersInRole("role1", "ORG1", true);
```

getUsersInRole() メソッドの詳細については、Javadoc の com.bea.wlpi.server.principal.WLPIPrincipal を参照してください。

ロール情報を取得する

ロールについての情報を取得するには、次の com.bea.wlpi.server.principal.WLPIPrincipal メソッドを使用します。

```
public com.bea.wlpi.common.RoleInfo getRoleInfo(
  java.lang.String roleId,
  java.lang.String orgId
) throws java.rmi.RemoteException
```

getRoleInfo() メソッドのパラメータを次の表に示します。パラメータには値を指定する必要があります。

表 9-15 getRoleInfo() メソッドのパラメータ

パラメータ	説明	有効な値
roleId	情報を取得するロールの ID	有効なロール ID を指定する文字列。 ユーザのリストの取得方法については、9-10 ページの「特定のオーガニゼーションに対して定義されているロールを取得する」を参照。
orgId	ロールに関連付けられたオーガニゼーションの ID	有効なオーガニゼーション ID を指定する文字列。 すべてのオーガニゼーション ID のリストの取得方法については、9-9 ページの「すべてのオーガニゼーション名を取得する」を参照。

このメソッドは、com.bea.wlpi.common.RoleInfo オブジェクトを返します。そのロールについての情報にアクセスするには、B-18 ページの「RoleInfo オブジェクト」に記載の RoleInfo オブジェクト メソッドを使用します。

たとえば、次のコードでは、オーガニゼーション ORG1 のロール role1 についての情報を取得します。このコード例では、principal は WLPIPrincipal EJB への EJBObject 参照を表しています。

```
principal.addUserToRole(“role1”, “ORG1”);
```

getRoleInfo() メソッドの詳細については、Javadoc の com.bea.wlpi.server.principal.WLPIPrincipal を参照してください。

ロール情報を設定する

ロールについての情報を設定するには、次の `com.bea.wlpi.server.principal.WLPIPrincipal` メソッドを使用します。

```
public void setRoleInfo(
  com.bea.wlpi.common.RoleInfo roleInfo
) throws java.rmi.RemoteException
```

`setRoleInfo()` メソッドのパラメータを次の表に示します。パラメータには値を指定する必要があります。

表 9-16 setRoleInfo() メソッドのパラメータ

パラメータ	説明	有効な値
`roleInfo`	更新するロール情報	`RoleInfo` オブジェクト。 すべての `RoleInfo` オブジェクトのリストを取得する方法については、9-10 ページの「特定のオーガニゼーションに対して定義されているロールのリストを取得する」を参照（オーガニゼーション属性が誤って消去されることがないように、ブール型パラメータ `obtainAttributes` を必ず `true` に設定すること）。`RoleInfo` オブジェクトの更新方法については、B-18 ページの「RoleInfo オブジェクト」を参照。

たとえば、次のコードでは、指定された `roleInfo` オブジェクトの内容に基づいて、ロールについての情報が設定されます。このコード例では、`principal` は `WLPIPrincipal` EJB への `EJBObject` 参照を表しています。

```
principal.setRoleInfo(roleInfo);
```

`setRoleInfo()` メソッドの詳細については、Javadoc の `com.bea.wlpi.server.principal.WLPIPrincipal` を参照してください。

ロールからユーザを削除する

ロールからユーザを削除するには、次の `com.bea.wlpi.server.principal.WLPIPrincipal` メソッドを使用します。

```
public void removeUserFromRole(
  java.lang.String userId,
  java.lang.String orgId,
  java.lang.String roleId
) throws java.rmi.RemoteException,
  com.bea.wlpi.common.WorkflowException
```

`removeUserFromRole()` メソッドのパラメータを次の表に示します。パラメータには値を指定する必要があります。

表 9-17 removeUserFromRole() メソッドのパラメータ

パラメータ	説明	有効な値
`userId`	削除するユーザの ID	有効なユーザ ID を指定する文字列。 ユーザのリストの取得方法については、9-52 ページの「すべてのユーザを取得する」を参照。
`orgId`	ユーザに関連付けられたオーガニゼーションの ID	有効なオーガニゼーション ID を指定する文字列。 すべてのオーガニゼーション ID のリストの取得方法については、9-9 ページの「すべてのオーガニゼーション名を取得する」を参照。
`roleId`	削除するユーザを含むロールの ID	有効なロール ID を指定する文字列。 すべてのロール ID のリストの取得方法については、9-10 ページの「特定のオーガニゼーションに対して定義されているロールを取得する」を参照。

たとえば、次のコードでは、ユーザ `user1` がオーガニゼーション `ORG1` のロール `role1` から削除されます。このコード例では、`principal` は `WLPIPrincipal` EJB への `EJBObject` 参照を表しています。

```
principal.removeUserFromOrganization(“user1”, “ORG1”, “role1”);
```

removeUserFromRole() メソッドの詳細については、Javadoc の com.bea.wlpi.server.principal.WLPIPrincipal を参照してください。

ロールを削除する

ロールを削除するには、次の com.bea.wlpi.server.principal.WLPIPrincipal メソッドを使用します。

```
public void deleteRole(
  java.lang.String orgId,
  java.lang.String roleId
) throws java.rmi.RemoteException,
  com.bea.wlpi.common.WorkflowException
```

deleteRole() メソッドのを次の表に示します。パラメータには値を指定する必要があります。

表 9-18 deleteRole() メソッドのパラメータ

パラメータ	説明	有効な値
orgId	削除するロールに関連付けられたオーガニゼーションの ID	有効なオーガニゼーション ID を指定する文字列。 すべてのオーガニゼーション ID のリストの取得方法については、9-9 ページの「すべてのオーガニゼーション名を取得する」を参照。
roleId	削除するロールの ID	有効なロール ID を指定する文字列。 すべてのロール ID のリストの取得方法については、9-10 ページの「特定のオーガニゼーションに対して定義されているロールを取得する」を参照。

たとえば、次のコードでは、オーガニゼーション ORG1 の指定されたロール ID に対応するロールが削除されます。このコード例では、principal は WLPIPrincipal EJB への EJBObject 参照を表しています。

```
principal.deleteRole(role.getId(), “ORG1”)
```

com.bea.wlpi.common.RoleInfo オブジェクトである role と関連付けられたメソッドを使用してロール ID を取得します。9-10 ページの「特定のオーガニゼーションに対して定義されているロールのリストを取得する」に記載のメソッドを使用して role オブジェクトを取得します。

deleteRole() メソッドの詳細については、Javadoc の com.bea.wlpi.server.principal.WLPIPrincipal を参照してください。

ロールのコンフィグレーション例

この節では、コマンドライン管理のコード例から抜粋して、ロールのコンフィグレーション方法を示します。

注意： コマンドライン管理サンプルの詳細については、1-23 ページの「コマンドライン管理サンプル」を参照してください。

このサンプルでは、ユーザと通信するための入力ストリームが定義されており、ユーザは以下のアクションのいずれかを指定するよう求められます。

- ロールを追加する
- ロールにユーザを追加する
- ロールを削除する
- ロールからユーザを削除する
- ロール情報を設定する
- ロール情報を取得する
- 特定のロールに対して定義されているユーザを取得する

重要なコード行は、**太字**で強調されています。このコード例では、principal は WLPIPrincipal EJB への EJBObject 参照を表しています。

```
/* ユーザとの通信のための入力ストリームを作成する */
stdIn = new BufferedReader( new InputStreamReader( System.in ) );

/* ツール タイトルを表示する */
   System.out.print( "\n---  Command Line Administration v1.1  ---" );

/* メイン メニューを表示してユーザと交信する */
while( true ) {
/* メニューを表示する */
```

```
   System.out.println( "\n---              Main Menu            ---" );
   System.out.println( "\nEnter choice:" );
   System.out.println( "1) Organizations" );
   System.out.println( "2) Roles" );
   System.out.println( "3) Users" );
   System.out.println( "4) Security Realm" );
   System.out.println( "5) Business Operations" );
   System.out.println( "6) Event Keys" );
   System.out.println( "7) Business Calendars" );
   System.out.println( "8) EJB Catalog" );
   System.out.println( "9) Server Properties" );
   System.out.println( "Q) Quit" );
   System.out.print( ">> " );
      .
      .
      .
public static void mngRoles( ) {
   String answer;
   String calendarId;
   String orgId;
   String roleId;
   String userId;
   String eMail;
   String defaultOrgId;

   /* ユーザとの通信のための入力ストリームを作成する */
   BufferedReader stdIn = new BufferedReader(
      new InputStreamReader( System.in ) );

   try {
      /* WLPI 公開 API メソッド */
      boolean isRealmManageable = principal.isManageableSecurityRealm( );

      /* メニューを表示してユーザと交信する */
      while( true ) {
      /* メニューを表示する */
         System.out.println( "\n\n---             WLPI Roles            ---" );
         System.out.println( "\nEnter choice:" );
         /* このレルムは管理可能か ? */
         if( isRealmManageable ) {
            /* このレルムは管理可能レルムであるため、管理可能レルムが
             * 必要な [Display] メニューのオプションを表示する */
            System.out.println( "1) Add a new Role" );
            System.out.println( "2) Assign a User to a Role" );
            System.out.println( "3) Delete a Role" );
            System.out.println( "4) Remove a User from a Role" );
            System.out.println( "5) Update Role Info" );
         }
```

```
System.out.println( "6) List Role Info" );
System.out.println( "7) List Users in a Role" );
System.out.println( "B) Back to previous menu" );
System.out.println( "Q) Quit" );
System.out.print( ">> " );

/* ユーザの選択を取得する */
String line = stdIn.readLine( );

/* ユーザが選択を行わないで〔Enter〕を押したか ? */
if( line.equals( "" ) )


   continue;
/* ユーザが複数の文字を入力したか ? */
else if( line.length( ) > 1 ) {
   System.out.println( "*** Invalid selection" );
   continue;
}
/* レルムが管理不可能で、ユーザが非表示の選択肢を入力したか ? */
else if( !isRealmManageable && line.charAt( 0 ) < '6' ) {
   System.out.println( "*** Invalid selection" );
   continue;
}

/* 大文字および文字へ変換する */
char choice = line.toUpperCase( ).charAt( 0 );

/* ユーザの選択を処理する */
switch( choice ) {
   .
   .
   .
```

ロールを追加する

ロールを追加する方法を示します。

```
/* 新しいロールを追加する */
case '1' :
   /* 追加する新しいロールのロール ID を取得する */
   if( ( roleId = askQuestion(
      "\nEnter a new Role ID" ) ) == null ) {
      /* ユーザによる操作の取り消し */
      System.out.println( "*** Cancelled" );
      break;
```

```
            }

            /* 新しいユーザの追加先オーガニゼーションのオーガニゼーション ID を取得する ; 必
須 */
            if( ( orgId = askQuestion(
               "Enter Organization ID" ) ) == null ) {
               /* ユーザによる操作の取り消し */
               System.out.println( "*** Cancelled" );
               break;
            }

            /* このロールに割り当てるカレンダー ID を取得する（省略可能）*/
            calendarId = askQuestion(
               "Enter Calendar ID (press enter for none)" );

            /* RoleInfo オブジェクトを作成する ; 新しいロールの追加に必要 */
            RoleInfo roleInfo = new RoleInfo( roleId, orgId, calendarId );

            try {
               /* WLPI 公開 API メソッド */
               /* オーガニゼーションに新しいロールを追加する */
               principal.addRole( roleInfo );

               /* 正常終了（例外の発生なし）*/
               System.out.println( "- Success" );
            }
            catch( Exception e ) {
               System.out.println( "*** Unable to add role\n" );
               System.err.println( e );
            }
             break;
            .
            .
            .
```

ロールにユーザを追加する

ロールにユーザを追加する方法を示します。

```
         /* ロールにユーザを割り当てる */
         case '2' :
            /* ロールに割り当てるユーザのユーザ ID を取得する */
            if( ( userId = askQuestion(
               "\nEnter User ID to assign" ) ) == null ) {
               /* ユーザによる操作の取り消し */
               System.out.println( "*** Cancelled" );
```

```
            break;
         }

         /* ユーザを追加するロールのロール ID を取得する */
         if( ( roleId = askQuestion( "Enter Role ID" ) ) == null ) {
            /* ユーザによる操作の取り消し */
            System.out.println( "*** Cancelled" );
            break;
         }

      /* ロールが所属するオーガニゼーションのオーガニゼーション ID を取得する */
         if( ( orgId = askQuestion( "Enter Organization ID" ) ) == null ) {
            /* ユーザによる操作の取り消し */
            System.out.println( "*** Cancelled" );
            break;
         }

         try {
            /* WLPI 公開 API メソッド */
            /* このオーガニゼーション内のこのロールにユーザを追加する */
            principal.addUserToRole( userId, orgId, roleId );

            /* 正常終了（例外の発生なし）*/
            System.out.println( "- Success" );
         }
         catch( Exception e ) {
            System.out.println( "*** Unable to add user to role\n" );
            System.err.println( e );
         }
         break;
         .
         .
         .
```

ロールを削除する

ロールを削除する方法を示します。

```
      /* ロールを削除する */
      case '3' :
         /* 削除するロールのロール ID を取得する */
         if( ( roleId = askQuestion(
            "\nEnter Role ID to delete" ) ) == null ) {
            /* ユーザによる操作の取り消し */
            System.out.println( "*** Cancelled" );
            break;
```

```
    }

    /* ロールを削除するオーガニゼーションのオーガニゼーション ID を取得する */
    if( ( orgId = askQuestion( "Enter Organization ID" ) ) == null ) {
       /* ユーザによる操作の取り消し */
       System.out.println( "*** Cancelled" );
       break;
    }

    try {
       /* WLPI 公開 API メソッド */
       /* ロールをこのオーガニゼーションから削除する */
       principal.deleteRole( orgId, roleId );

       /* 正常終了（例外の発生なし）*/
       System.out.println( "- Success" );
    }
    catch( Exception e ) {
       System.out.println( "*** Unable to delete role\n" );
       System.err.println( e );
    }
    break;
    .
    .
    .
```

ロールからユーザを削除する

ロールからユーザを削除する方法を示します。

```
/* ロールからユーザを削除する */
case '4' :
   /* ロールから削除するユーザのユーザ ID を取得する */
   if( ( userId = askQuestion(
      "\nEnter User ID to remove" ) ) == null ) {
      /* ユーザによる操作の取り消し */
      System.out.println( "*** Cancelled" );
      break;
   }

   /* ユーザを削除するロールのロール ID を取得する */
   if( ( roleId = askQuestion( "Enter Role ID" ) ) == null ) {
   /* ユーザによる操作の取り消し */
      System.out.println( "*** Cancelled" );
      break;
   }
```

```
   /* ロールが所属するオーガニゼーションのオーガニゼーション ID を取得する */
   if( ( orgId = askQuestion( "Enter Organization ID" ) ) == null ) {
      /* ユーザによる操作の取り消し */
      System.out.println( "*** Cancelled" );
      break;
   }

   try {
      /* WLPI 公開 API メソッド */
      /* このオーガニゼーションのこのロールからユーザを削除する */
      principal.removeUserFromRole( userId, orgId, roleId);

      /* 正常終了（例外の発生なし）*/
      System.out.println( "- Success" );
   }
   catch( Exception e ) {
      System.out.println( "*** Unable to remove user from role\n" );
      System.err.println( e );
   }
   break;
   .
   .
   .
```

ロール情報を設定する

ロール情報を設定する方法を示します。

```
/* ロール情報を更新する */
case '5' :
   /* 更新するロールのロール ID を取得する */
   if( ( roleId = askQuestion(
      "\nEnter Role ID to update" ) ) == null ) {
      /* ユーザによる操作の取り消し */
      System.out.println( "*** Cancelled" );
      break;
   }

   /* ロールが所属するオーガニゼーションのオーガニゼーション ID を取得する */
   if( ( orgId = askQuestion(
      "Enter Organization ID" ) ) == null ) {
   /* ユーザによる操作の取り消し */
      System.out.println( "*** Cancelled" );
      break;
   }
```

```
/* このロールに割り当てる新しいカレンダー ID を取得する ; (WLPI v1.2.1 で定義
 * されているロール属性のみ、つまり、更新可能な属性のみ）*/
calendarId = askQuestion(
   "Enter Calendar ID (press enter for none)" );

/* RoleInfo オブジェクトを作成する ; ロールの更新に必要 */
roleInfo = new RoleInfo( roleId, orgId, calendarId );

try {
   /* WLPI 公開 API メソッド */
   /* このオーガニゼーションのこのロールを更新する（カレンダーの読み込み）*/
   principal.setRoleInfo( roleInfo );

   /* 正常終了（例外の発生なし）*/
   System.out.println( "- Success" );
}
catch( Exception e ) {
   System.out.println( "*** Unable to update role info\n" );
   System.err.println( e );
}
break;
.
.
.
```

ロール情報を取得する

ロールについての情報を取得する方法を示します。

```
/* ロール情報をリスト アップする */
case '6' :
   /* 表示するロールのロール ID を取得する */
   if( ( roleId = askQuestion( "\nEnter Role ID" ) ) == null ) {
   /* ユーザによる操作の取り消し */
      System.out.println( "*** Cancelled" );
      break;
   }

   /* ロールが所属するオーガニゼーションのオーガニゼーション ID を取得する */
   if( ( orgId = askQuestion( "Enter Organization ID" ) ) == null ) {
   /* ユーザによる操作の取り消し */
      System.out.println( "*** Cancelled" );
      break;
   }
```

```
            try {
               /* WLPI 公開 API メソッド */
               /* このロールについての情報を取り出す */
               roleInfo = principal.getRoleInfo( roleId, orgId );

               System.out.println( "\nRole Info:" );
               /* ロール ID を取り出して表示 */
               System.out.println( "- Role ID: " + roleInfo.getRoleId( ) );
              /* ロールが所属するオーガニゼーションのオーガニゼーション ID を取り出して表示
する */
               System.out.println( "  Org ID: " + roleInfo.getOrgId( ) );

               /* カレンダー ID を取り出す */
               if( ( calendarId = roleInfo.getCalendarId( ) ) == null )
               /* このロールに対して定義された ID がない場合は、'None' という ID が割り当
てられる */
                  calendarId = "None";

               /* カレンダー ID を表示する */
               System.out.println( "  Calendar ID: " + calendarId );
            }
            catch( Exception e ) {
               System.out.println( "*** Unable to retrieve role info\n" );
               System.err.println( e );
            }
            break;
            .
            .
            .
```

特定のロールに対して定義されているユーザを取得する

特定のロールに対して定義されているユーザのリストを取得する方法を示します。

```
         /* ロール内のユーザをリスト アップする */
         case '7' :
            /* クエリするロールのロール ID を取得する */
            if( ( roleId = askQuestion( "\nEnter Role ID" ) ) == null ) {
            /* ユーザによる操作の取り消し */
               System.out.println( "*** Cancelled" );
               break;
            }

            /* ロールが所属するオーガニゼーションのオーガニゼーション ID を取得する */
            if( ( orgId = askQuestion( "Enter Organization ID" ) ) == null ) {
```

```
            /* ユーザによる操作の取り消し */
            System.out.println( "*** Cancelled" );
            break;
         }

         /* ユーザ属性の表示が必要かどうか
          * 選択するようにユーザに求める */
         if( ( answer = askQuestion(
            "List all attributes (y/n)?" ) ) == null ) {
            /* ユーザによる操作の取り消し */
            System.out.println( "*** Cancelled" );
            break;
         }

         /* 応答を解析する */
         boolean isGetAttributes = ( answer.equals( "y" ) ||
            answer.equals( "Y" ) );

         /* WLPI 公開 API メソッド */
         /* このロールに割り当てられたすべてのユーザを取り出す */
         /* 注意 : 発生した例外を取り込むコードを
          * 追加するとよい */
         List userList =
            principal.getUsersInRole( roleId, orgId, isGetAttributes );

         /* 割り当てられたユーザはあるか ? */
         if( userList.size( ) == 0 )
            System.out.println( "\nNo users assigned" );
         else
            System.out.println( "\nAssigned Users:" );

         /* リストを処理して、ユーザと属性を表示する */
         for( int i = 0; i < userList.size( ); i++ ) {
            /* リストから要素を取り出す */
            UserInfo userInfo = ( UserInfo )userList.get( i );
            /* ユーザ ID を取り出して表示する */
            System.out.println( "- User ID: " + userInfo.getUserId( ) );

            /* 属性を表示するか ? */
            if( isGetAttributes ) {
               if( ( eMail = userInfo.getEMailAddress( ) ) == null )
                  /* このユーザに対して定義された ID がない場合は、'None' という ID が
割り当てられる */
                  eMail = "None";

               if( ( defaultOrgId = userInfo.getDefaultOrgId( ) ) == null )
               /* このユーザに対して定義された ID がない場合は、'None' という ID が割
```

```
り当てられる */
                    defaultOrgId = "None";

                /* カレンダー ID を取り出す */
                if( ( calendarId = userInfo.getCalendarId( ) ) == null )
                /* このユーザに対して定義された ID がない場合は、'None' という ID が割
り当てられる */
                    calendarId = "None";

                /* 電子メール アドレスを表示（デフォルトのオーガニゼーション ID とカレン
ダー ID）する */
                System.out.println( "  Attributes:\n  - eMail: " + eMail );
                System.out.println( "  - Default ORG ID: " + defaultOrgId );
                System.out.println( "  - Calendar ID=" + calendarId + "\n" );
              }
            }
            break;
            .
            .
            .
```

ユーザのコンフィグレーション

ここでは、ユーザのコンフィグレーション方法について説明します。内容は以下のとおりです。

- ユーザを追加する
- すべてのユーザを取得する
- ユーザ オーガニゼーションを取得する
- ユーザ ロールを取得する
- ユーザ情報を取得する
- ユーザ情報を設定する
- ユーザを削除する
- ユーザのコンフィグレーション例

ユーザを追加する

新しいユーザを作成して wlpiUsers グループに追加するには、次の com.bea.wlpi.server.principal.WLPIPrincipal メソッドを使用します。

```
public com.bea.wlpi.common.UserInfo createUser(
  java.lang.String userId,
  java.lang.String pswd
) throws java.rmi.RemoteException,
  com.bea.wlpi.common.WorkflowException
```

createUser() メソッドのパラメータを次の表に示します。パラメータには値を指定するが必要があります。

表 9-19 createUser() メソッドのパラメータ

パラメータ	説明	有効な値
userId	追加するユーザの ID	ユニークなユーザ ID を指定する文字列。
pswd	指定されたユーザ ID に対するクリアテキスト パスワード	パスワードを指定する文字列。

たとえば、次のコードでは、パスワード password を持つ sam という名前のユーザが wlpiUsers グループ内に作成されます。このコード例では、principal は WLPIPrincipal EJB への EJBObject 参照を表しています。

```
UserInfo userInfo = principal.createUser("sam", "password")
```

createUser() メソッドの詳細については、Javadoc の com.bea.wlpi.server.principal.WLPIPrincipal を参照してください。

すべてのユーザを取得する

すべてのユーザのリストを取得するには、次の com.bea.wlpi.server.principal.WLPIPrincipal メソッドを使用します。

```
public java.util.List getAllUsers(
   boolean obtainAttributes
) throws java.rmi.RemoteException,
  com.bea.wlpi.common.WorkflowException
```

getAllUsers() メソッドのパラメータを次の表に示します。パラメータには値を指定する必要があります。

表 9-20 getAllUsers() メソッドのパラメータ

パラメータ	説明	有効な値
obtainAttributes	すべての属性を取得するのか、ユーザ ID のみを取得するのかを指定するフラグ	true (すべての属性) または false (ユーザ ID のみ)。

このメソッドは、com.bea.wlpi.common.UserInfo オブジェクトのリストを返します。各ユーザについての情報にアクセスするには、B-28 ページの「UserInfo オブジェクト」に記載の UserInfo オブジェクト メソッドを使用します。

たとえば、次のコードでは、ユーザ ID を取得して（obtainAttributes パラメータが false に設定されている）userList リスト変数に保存します。このコード例では、principal は WLPIPrincipal EJB への EJBObject 参照を表しています。

```
List userList = principal.getAllUsers(false)
```

getAllUsers() メソッドの詳細については、Javadoc の com.bea.wlpi.server.principal.WLPIPrincipal を参照してください。

ユーザ オーガニゼーションを取得する

ユーザが所属するオーガニゼーションのリストを取得するには、次の com.bea.wlpi.server.principal.WLPIPrincipal メソッドを使用します。

```
public java.util.List getOrganizationsForUser(
  java.lang.String userId,
  boolean obtainAttributes
) throws java.rmi.RemoteException,
  com.bea.wlpi.common.WorkflowException
```

getOrganizationsForUser() メソッドのパラメータを次の表に示します。パラメータには値を指定する必要があります。

表 9-21 getOrganizationsForUser() メソッドのパラメータ

パラメータ	説明	有効な値
userId	オーガニゼーションを取得するユーザの ID	有効なユーザ ID を指定する文字列。 すべてのオーガニゼーション ID のリストの取得方法については、9-52 ページの「すべてのユーザを取得する」を参照。
obtainAttributes	すべての属性を取得するのか、オーガニゼーション ID のみを取得するのかを指定するフラグ	true (すべての属性) または false (オーガニゼーション ID のみ)。

このメソッドは、com.bea.wlpi.common.OrganizationInfo オブジェクトのリストを返します。各オーガニゼーションについての情報にアクセスするには、B-9 ページの「OrganizationInfo オブジェクト」に記載の OrganizationInfo オブジェクト メソッドを使用します。

たとえば、次のコードでは、user1 に対するオーガニゼーションのリストを取得して、すべての属性を返します（obtainAttributes パラメータが true に設定されているため）。このコード例では、principal は WLPIPrincipal EJB への EJBObject 参照を表しています。

```
List roles = principal.getOrganizationsForUser(“user1”, true);
```

getOrganizationsForUser() メソッドの詳細については、Javadoc の com.bea.wlpi.server.principal.WLPIPrincipal を参照してください。

ユーザ ロールを取得する

ユーザが所属するロールのリストを取得するには、次の com.bea.wlpi.server.principal.WLPIPrincipal メソッドを使用します。

```
public java.util.List getRolesForUser(
  java.lang.String orgId,
  java.lang.String userId,
  boolean obtainAttributes
) throws java.rmi.RemoteException,
  com.bea.wlpi.common.WorkflowException
```

getRolesForUser() メソッドのパラメータを次の表に示します。パラメータには値を指定する必要があります。

表 9-22 getRolesForUser() メソッドのパラメータ

パラメータ	説明	有効な値
orgId	ユーザに関連付けられたオーガニゼーションの ID	有効なオーガニゼーション ID を指定する文字列。 すべてのオーガニゼーション ID のリストの取得方法については、9-9 ページの「すべてのオーガニゼーション名を取得する」を参照。
userId	ロールのリストを取得するユーザの ID	有効なユーザ ID を指定する文字列。 すべてのオーガニゼーション ID のリストの取得方法については、9-52 ページの「すべてのユーザを取得する」を参照。
obtainAttributes	すべての属性を取得するのか、ロール ID のみを取得するのかを指定するフラグ	true (すべての属性) または false (オーガニゼーション ID のみ)。

このメソッドは、com.bea.wlpi.common.RoleInfo オブジェクトのリストを返します。各ロールについての情報にアクセスするには、B-18 ページの「RoleInfo オブジェクト」に記載の RoleInfo オブジェクト メソッドを使用します。

たとえば、次のコードでは、オーガニゼーション ORG1 内の user1 に対して定義されているロールのリストを取得し、すべての属性を返します（obtainAttributes パラメータが true に設定されているため）。このコード例では、principal は WLPIPrincipal EJB への EJBObject 参照を表しています。

```
List roles = principal.getRolesForUser("ORG1", "user1", true);
```

getRolesForUser() メソッドの詳細については、Javadoc の com.bea.wlpi.server.principal.WLPIPrincipal を参照してください。

ユーザ情報を取得する

ユーザについての情報を取得するには、次の com.bea.wlpi.server.principal.WLPIPrincipal メソッドを使用します。

```
public com.bea.wlpi.common.UserInfo getUserInfo(
  java.lang.String userId
) throws java.rmi.RemoteException
```

getUserInfo() メソッドのパラメータを次の表に示します。パラメータには値を指定する必要があります。

表 9-23 getUserInfo() メソッドのパラメータ

パラメータ	説明	有効な値
userId	情報を取得するユーザの ID	有効なユーザ ID を指定する文字列。 ユーザのリストの取得方法については、9-52 ページの「すべてのユーザを取得する」を参照。

このメソッドは、com.bea.wlpi.common.UserInfo オブジェクトを返します。ユーザについての情報にアクセスするには、B-28 ページの「UserInfo オブジェクト」に記載の UserInfo オブジェクト メソッドを使用します。

たとえば、次のコードでは、ユーザ user1 の情報を取得します。このコード例では、principal は WLPIPrincipal EJB への EJBObject 参照を表しています。

```
UserInfo user = principal.getUserInfo("user1");
```

getUserInfo() メソッドの詳細については、Javadoc の com.bea.wlpi.server.principal.WLPIPrincipal を参照してください。

ユーザ情報を設定する

ユーザについての情報を設定するには、次の com.bea.wlpi.server.principal.WLPIPrincipal メソッドを使用します。

```
public void setUserInfo(
  com.bea.wlpi.common.UserInfo userInfo
) throws java.rmi.RemoteException,
```

setUserInfo() メソッドのパラメータを次の表に示します。パラメータには値を指定する必要があります。

表 9-24 setUserInfo() メソッドのパラメータ

パラメータ	説明	有効な値
userInfo	更新するユーザ情報	UserInfo オブジェクト。 すべての UserInfo オブジェクトのリストの取得方法については、9-13 ページの「特定のオーガニゼーションに対して定義されているユーザのリストを取得する」を参照（オーガニゼーション属性が誤って消去されることがないように、ブール型パラメータ obtainAttributes を必ず true に設定すること）。UserInfo オブジェクトの更新方法については、B-28 ページの「UserInfo オブジェクト」を参照。

たとえば、次のコードでは、指定された userInfo オブジェクトの内容に基づいてユーザについての情報が設定されます。このコード例では、principal は WLPIPrincipal EJB への EJBObject 参照を表しています。

```
principal.setUserInfo(userInfo);
```

setUserInfo() メソッドの詳細については、Javadoc の com.bea.wlpi.server.principal.WLPIPrincipal を参照してください。

ユーザを削除する

オーガニゼーション、ロール、またはコンフィグレーション データベースからユーザを削除するには、以下のメソッドを使用します。

オーガニゼーションからユーザを削除する

ユーザをオーガニゼーションから削除する方法については、「オーガニゼーションのコンフィグレーション」節の 9-17 ページの「オーガニゼーションからユーザを削除する」を参照してください。

ロールからユーザを削除する

ユーザをロールから削除する方法については、「ロールのコンフィグレーション」節の 9-38 ページの「ロールからユーザを削除する」を参照してください。

データベースからユーザを削除する

コンフィグレーション データベースからユーザを削除するには、次の `com.bea.wlpi.server.principal.WLPIPrincipal` メソッドを使用します。

```
public void deleteUser(
  java.lang.String userId
) throws java.rmi.RemoteException,
  com.bea.wlpi.common.WorkflowException
```

`deleteUser()` メソッドのパラメータを次の表に示します。パラメータには値を指定する必要があります。

表 9-25 deleteRole() メソッドのパラメータ

パラメータ	説明	有効な値
`userId`	削除するユーザの ID	有効なユーザ ID を指定する文字列。 ユーザのリストの取得方法については、9-52 ページの「すべてのユーザを取得する」を参照。

たとえば、次のコードでは、ユーザ `user1` がデータベースから削除されます。このコード例では、`principal` は `WLPIPrincipal` EJB への `EJBObject` 参照を表しています。

```
principal.deleteUser("user1");
```

deleteUser() メソッドの詳細については、Javadoc の com.bea.wlpi.server.principal.WLPIPrincipal を参照してください。

ユーザのコンフィグレーション例

この節では、コマンドライン管理サンプルから抜粋して、ユーザのコンフィグレーション方法を示します。

注意： コマンドライン管理サンプルの詳細については、1-23 ページの「コマンドライン管理サンプル」を参照してください。

このサンプルでは、ユーザと通信するための入力ストリームが定義されており、ユーザは以下のアクションのいずれかを指定するよう求められます。

- ユーザを追加する
- ユーザを削除する
- ユーザ情報を設定する
- すべてのユーザを取得する
- ユーザ情報を取得する
- ユーザオーガニゼーションを取得する
- ユーザ ロールを取得する

重要なコード行は、**太字**で強調されています。このコード例では、principal は WLPIPrincipal EJB への EJBObject 参照を表しています。

```
/* ユーザとの通信のための入力ストリームを作成する */
stdIn = new BufferedReader( new InputStreamReader( System.in ) );

/* ツール タイトルを表示する */
   System.out.print( "\n---  Command Line Administration v1.1  ---" );

/* メイン メニューを表示してユーザと交信する */
while( true ) {
/* メニューを表示する */
   System.out.println( "\n---              Main Menu            ---" );
   System.out.println( "\nEnter choice:" );
   System.out.println( "1) Organizations" );
   System.out.println( "2) Roles" );
```

```
   System.out.println( "3) Users" );
   System.out.println( "4) Security Realm" );
   System.out.println( "5) Business Operations" );
   System.out.println( "6) Event Keys" );
   System.out.println( "7) Business Calendars" );
   System.out.println( "8) EJB Catalog" );
   System.out.println( "9) Server Properties" );
   System.out.println( "Q) Quit" );
   System.out.print( ">> " );
      .
      .
      .

   System.out.print( ">> " ); ...
```

mngUsers() メソッドは、ユーザと交信して必要な情報を取り出しながらユーザを管理する方法を示しています。

```
public static void mngUsers( ) {
   String answer;
   String calendarId;
   String orgId;
   String userId;
   String password;
   String eMail;
   String defaultOrgId;

   /* ユーザとの通信のための入力ストリームを作成する */
   BufferedReader stdIn = new BufferedReader(
      new InputStreamReader( System.in ) );

   try {
      /* WLPI 公開 API メソッド */
      boolean isRealmManageable = principal.isManageableSecurityRealm( );

      /* メニューを表示してユーザと交信する */
      while( true ) {
         /* メニューを表示する */
         System.out.println( "\n\n---            WLPI Users            ---" );
         System.out.println( "\nEnter choice:" );
         /* このレルムは管理可能か ? */
         if( isRealmManageable ) {
            /* このレルムは管理可能レルムであるため、管理可能レルムが
             * 必要な [Display] メニューのオプションを表示する */
            System.out.println( "1) Add a new User" );
            System.out.println( "2) Delete a User" );
            System.out.println( "3) Update User Info" );
         }
```

```
System.out.println( "4) List All Users" );
System.out.println( "5) List User Info" );
System.out.println( "6) List Organizations for a User" );
System.out.println( "7) List Roles for a User" );
System.out.println( "B) Back to previous menu" );
System.out.println( "Q) Quit" );
System.out.print( ">> " );

/* ユーザの選択を取得する */
String line = stdIn.readLine( );

/* ユーザが選択を行わないで〔Enter〕を押したか ? */
if( line.equals( "" ) )
   continue;
/* ユーザが複数の文字を入力したか ? */
else if( line.length( ) > 1 ) {
   System.out.println( "*** Invalid selection" );
   continue;
}
/* レルムが管理不可能で、ユーザが非表示の選択肢を入力したか ? */
else if( !isRealmManageable && line.charAt( 0 ) < '4' ) {
   System.out.println( "*** Invalid selection" );
   continue;
}

/* 大文字および文字へ変換する */
char choice = line.toUpperCase( ).charAt( 0 );

/* ユーザの選択を処理する */
switch( choice ) {
   .
   .
   .
```

ユーザを追加する

ユーザを追加する方法を示します。

```
/* 新しいユーザを追加する */
case '1' :
   /* 作成する新しいユーザのユーザ ID を取得する */
   if( ( userId = askQuestion( "\nEnter new User ID" ) ) == null ) {
   /* ユーザによる操作の取り消し */
      System.out.println( "*** Cancelled" );
      break;
   }
```

```
   /* 新しいユーザのパスワードを取得する */
   if( ( password = askQuestion( "Enter Password" ) ) == null ) {
      /* ユーザによる操作の取り消し */
      System.out.println( "*** Cancelled" );
      break;
   }

   try {
      /* WLPI 公開 API メソッド */
      /* このパスワードを使用して新しいユーザの ID を作成する */
      principal.createUser( userId, password );

      /* 正常終了（例外の発生なし）*/
      System.out.println( "- Success" );
   }
   catch( Exception e ) {
      System.out.println( "*** Unable to create user\n" );
      System.err.println( e );
   }
   break;
   .
   .
   .
```

ユーザを削除する

ユーザを削除する方法を示します。

```
/* ユーザを削除する */
case '2' :
   /* 削除するユーザのユーザ ID を取得する */
   if( ( userId = askQuestion( "\nEnter User ID to delete" ) )
      == null ) {
      /* ユーザによる操作の取り消し */
      System.out.println( "*** Cancelled" );
      break;
   }

   try {
      /* WLPI 公開 API メソッド */
      /* ユーザを削除する */
      principal.deleteUser( userId );

      /* 正常終了（例外の発生なし）*/
      System.out.println( "- Success" );
```

```
   }
   catch( Exception e ) {
      System.out.println( "*** Unable to delete user\n" );
      System.err.println( e );
   }




   break;
   .
   .
   .
```

ユーザ情報を設定する

ユーザについての情報を設定する方法を示します。

```
/* ユーザ情報を更新する */
case '3' :
   /* 更新するユーザのユーザ ID を取得する */
   if( ( userId = askQuestion( "\nEnter User ID to update" ) )
      == null ) {
      /* ユーザによる操作の取り消し */
      System.out.println( "*** Cancelled" );
      break;
   }

   /* このユーザの電子メール アドレスを取得する（省略可能）*/
   eMail = askQuestion(
      "Enter new eMail Address (press enter for none)" );

 /* このユーザに割り当てるデフォルトのオーガニゼーション ID を取得する(省略可能）*/
   defaultOrgId = askQuestion(
      "Enter new default Org ID (press enter for none)" );

   /* このユーザに割り当てるカレンダー ID を取得する（省略可能）*/
   calendarId = askQuestion(
      "Enter new Calendar ID (press enter for none)" );

   /* UserInfo オブジェクトを作成する。ユーザの更新に必要 */
   UserInfo userInfo = new UserInfo(
      userId, eMail, defaultOrgId, calendarId );

   try {
      /* WLPI 公開 API メソッド */
```

```
            principal.setUserInfo( userInfo );

            /* 正常終了（例外の発生なし）*/
            System.out.println( "- Success" );
         }
         catch( Exception e ) {
            System.out.println( "*** Unable to update user\n" );
            System.err.println( e );
         }

         break;
         .
         .
         .
```

すべてのユーザを取得する

すべてのユーザのリストを取得する方法を示します。

```
/* すべてのユーザをリスト アップする */
case '4' :
   /* ユーザ属性の表示が必要かどうかを
    * 選択するようにユーザに求める */
   if( ( answer = askQuestion(
      "\nList all attributes (y/n)?" ) ) == null ) {
      /* ユーザによる操作の取り消し */
      System.out.println( "*** Cancelled" );
      break;
   }

   /* 応答を解析する */
   boolean isGetAttributes = ( answer.equals( "y" ) ||
      answer.equals( "Y" ) );

   /* WLPI 公開 API メソッド */
   /* すべてのユーザを検索する */
   /* 注意 : 発生した例外を取り込むコードを
    * 追加するとよい */
   List userList = principal.getAllUsers( isGetAttributes );

   /* 定義されているユーザはあるか ? */
   if( userList.size( ) == 0 )
      System.out.println( "\nNo user defined" );
   else
      System.out.println( "\nDefined Users:" );
```

```
            /* リストを処理して、ユーザと属性を表示する */
            for( int i = 0; i < userList.size( ); i++ ) {
               /* リストから要素を取り出す */
               userInfo = ( UserInfo )userList.get( i );
               /* ユーザ ID を取り出して表示する */
               System.out.println( "- User ID: " + userInfo.getUserId( ) );

               /* 属性を表示するか ? */
               if( isGetAttributes ) {
                  /* 電子メール アドレスを取り出す */
                  if( ( eMail = userInfo.getEMailAddress( ) ) == null )
                     /* このユーザに対して定義された ID がない場合は、'None' という ID が
割り当てられる */
                     eMail = "None";

                  /* デフォルトのオーガニゼーション ID を取り出す */
                  if( ( defaultOrgId = userInfo.getDefaultOrgId( ) ) == null )
                     /* このユーザに対して定義された ID がない場合は、'None' という ID が
割り当てられる */
                     defaultOrgId = "None";

                  /* カレンダー ID を取り出す */
                  if( ( calendarId = userInfo.getCalendarId( ) ) == null )
                     /* このユーザに対して定義された ID がない場合は、'None' という ID が
割り当てられる */
                     calendarId = "None";

                  /* 電子メール アドレスを表示（デフォルトのオーガニゼーション ID とカレン
ダー ID）する */
                  System.out.println( "  Attributes:\n  - eMail: " + eMail );
                  System.out.println( "  - Default ORG ID: " + defaultOrgId );
                  System.out.println( "  - Calendar ID=" + calendarId + "\n" );
               }
            }
            break;
            .
            .
            .
```

ユーザ情報を取得する

ユーザについての情報を取得する方法を示します。

```
/* ユーザ情報をリスト アップする */
case '5' :
```

```
            /* 表示するユーザのユーザ ID を取得する */
            if( ( userId = askQuestion( "\nEnter User ID" ) ) == null ) {
               /* ユーザによる操作の取り消し */
               System.out.println( "*** Cancelled" );
               break;
            }

            try {
               /* WLPI 公開 API メソッド */
               /* このユーザについての情報を取り出す */
               userInfo = principal.getUserInfo( userId );

               /* ユーザ ID を取り出して表示する */
               System.out.println( "- User ID: " + userInfo.getUserId( ) );

               /* 電子メール アドレスを取り出す */
               if( ( eMail = userInfo.getEMailAddress( ) ) == null )
                  /* このユーザに対して定義された ID がない場合は、'None' という ID が割
り当てられる */
                  eMail = "None";

               /* デフォルトのオーガニゼーション ID を取り出す */
               if( ( defaultOrgId = userInfo.getDefaultOrgId( ) ) == null )
                  /* このユーザに対して定義された ID がない場合は、'None' という ID が割
り当てられる */
                  defaultOrgId = "None";

               /* カレンダー ID を取り出す */
               if( ( calendarId = userInfo.getCalendarId( ) ) == null )
                  /* このユーザに対して定義された ID がない場合は、'None' という ID が割
り当てられる */
                  calendarId = "None";

               /* 電子メール アドレスを表示(デフォルトのオーガニゼーション ID とカレンダー
ID) する */
               System.out.println( "  Attributes:\n  - eMail: " + eMail );
               System.out.println( "  - Default ORG ID: " + defaultOrgId );
               System.out.println( "  - Calendar ID=" + calendarId + "\n" );
            }
            catch( Exception e ) {
               System.out.println( "*** Unable to retrieve user info\n" );
               System.err.println( e );
            }
            break;
            .
            .
```

.

ユーザオーガニゼーションを取得する

すべてのユーザ オーガニゼーションのリストを取得する方法を示します。

```
/* あるユーザに対するオーガニゼーションをリスト アップする */
case '6' :
   /* クエリするユーザのユーザ ID を取得する */
   if( ( userId = askQuestion( "\nEnter User ID" ) ) == null ) {
      /* ユーザによる操作の取り消し */
      System.out.println( "*** Cancelled" );
      break;
   }

   /* オーガニゼーション属性の表示が必要かどうかを
    * 選択するようにユーザに求める */
   if( ( answer = askQuestion(
      "List Organization attributes (y/n)?" ) ) == null ) {
      /* ユーザによる操作の取り消し */
      System.out.println( "*** Cancelled" );
      break;
   }

   /* 応答を解析する */
   isGetAttributes = ( answer.equals( "y" ) || answer.equals( "Y" ) );

   /* WLPI 公開 API メソッド */
   /* このユーザに割り当てられたすべてのオーガニゼーションを検索する */
   /* 注意 : 発生した例外を取り込むコードを
    * 追加するとよい */
   List orgList = principal.getOrganizationsForUser(
      userId, isGetAttributes );

   /* 割り当てられているオーガニゼーションはあるか ? */
   if( orgList.size( ) == 0 )
      System.out.println( "\nNot assigned to any organization" );
   else
      System.out.println( "\nAssigned to organizations:" );

   /* リストを処理して、オーガニゼーションと属性を表示する */
   for( int i = 0; i < orgList.size( ); i++ ) {
      /* リストから要素を取り出す */
      OrganizationInfo orgInfo = ( OrganizationInfo )orgList.get( i );
      /* オーガニゼーション ID を取り出して表示する */
```

```
                System.out.println( "- Org ID: " + orgInfo.getOrgId( ) );

                /* 属性を表示するか ? */
                if( isGetAttributes ) {
                   /* カレンダー ID を取り出す */
                   if( ( calendarId = orgInfo.getCalendarId( ) ) == null ))
                      /* このオーガニゼーションに対して定義された ID がない場合は、'None'
という ID が割り当てられる */
                      calendarId = "None";

                   /* カレンダー ID を表示する */
                   System.out.println(
                      "  Attributes: Calendar ID=" + calendarId + "\n" );
                }
             }
             break;
             .
             .
             .
```

ユーザ ロールを取得する

ユーザが割り当てられているすべてのロールを取得する方法を示します。

```
          /* 特定のユーザに対するロールをリスト アップする *
          case '7' :
             /* クエリするユーザのユーザ ID を取得する */
             if( ( userId = askQuestion( "\nEnter User ID" ) ) == null ) {
                /* ユーザによる操作の取り消し */
                System.out.println( "*** Cancelled" );
                break;
             }

             /* クエリするオーガニゼーションのオーガニゼーション ID を取得する */
             if( ( orgId = askQuestion( "Enter Organization ID" ) ) == null ) {
                /* ユーザによる操作の取り消し */
                System.out.println( "*** Cancelled" );
                break;
             }

//            /* ロール属性の表示が必要かどうかを選択するようにユーザに求める */
//            if( ( answer = askQuestion(
//               "List Role attributes (y/n)?" ) ) == null ) {
//               /* ユーザによる操作の取り消し */
//               System.out.println( "*** Cancelled" );
//               break;
```

```
//          }
//
//          /* 応答を解析する */
//          isGetAttributes = ( answer.equals( "y" ) ||
//             answer.equals( "Y" ) );
            isGetAttributes = false;

            /* WLPI 公開 API メソッド */
            /* このユーザが割り当てられているすべてのロールを検索する */
            /* 注意 : 発生した例外を取り込むコードを
             * 追加するとよい */
            List roleList = principal.getRolesForUser(
               orgId, userId, isGetAttributes );

            /* 割り当てられているロールはあるか ? */
            if( roleList.size( ) == 0 )
               System.out.println( "\nNot assigned to any role" );
            else
               System.out.println( "\nAssigned to roles:" );

            /* リストを処理してロールと属性を表示する */
            for( int i = 0; i < roleList.size( ); i++ ) {
               /* リストから要素を取り出す */
               RoleInfo roleInfo = ( RoleInfo )roleList.get( i );
               /* ロール ID を取り出して表示する */
               System.out.println( "- Role ID: " + roleInfo.getRoleId( ) );



               /* 属性を表示するか ? */
               if( isGetAttributes ) {
                  /* カレンダー ID を取り出す */
                  if( ( calendarId = roleInfo.getCalendarId( ) ) == null )
                     /* このロールに対して定義された ID がない場合は、'None' という ID が
割り当てられる */
                     calendarId = "None";

                  /* カレンダー ID を表示する */
                  System.out.println(
                     "  Attributes: Calendar ID=" + calendarId + "\n" );
               }
            }
            break;

         /* 前のメニューに戻る */
         case 'B' :
            return;
```

```
            /* ツールを終了する */
            case 'Q' :
               /* サーバから切断する */
               disconnect( );
               System.exit( 1 );

               default:
               System.out.println( "*** Invalid selection" );
            }
         }
      }
      /* 「Unhandled」例外 */
      catch( Exception e ) {
         System.err.println( e );
      }
      return;
}
```

セキュリティ情報のマッピング

ユーザとロールを定義した後、BEA WebLogic Server に対して定義された、これらのユーザとロールの関係およびユーザとグループの関係をそれぞれ定義する必要があります。これは、各ロールを BEA WebLogic Server のセキュリティ レルムにマッピングすることによって行います。

この節では、マッピングに必要なタスクのを実行方法について説明します。内容は以下のとおりです。

- セキュリティ レルム グループの取得
- グループへのロールのマッピング
- グループへの複数のロールのマッピング
- 特定のロールに対するグループ マッピングの取得
- 特定のオーガニゼーションに対して定義されているすべてのロールに対するグループ マッピングの取得

セキュリティ レルム グループの取得

BEA WebLogic Server のセキュリティ レルム グループのリストを取得するには、次の com.bea.wlpi.server.principal.WLPIPrincipal メソッドを使用します。

```
public java.util.List getGroups(
) throws java.rmi.RemoteException,
  com.bea.wlpi.common.WorkflowException
```

このメソッドは、セキュリティ レルム グループ名のリストを返します。

たとえば、次のコードは、セキュリティ レルム グループ名を取得して、groups リストに保存します。このコード例では、principal は WLPIPrincipal EJB への EJBObject 参照を表しています。

```
List groups = principal.getGroups();
```

getGroups() メソッドの詳細については、Javadoc の com.bea.wlpi.server.principal.WLPIPrincipal を参照してください。

グループへのロールのマッピング

BEA WebLogic Server のセキュリティ レルム グループに BPM ロールをマッピングするには、次の `com.bea.wlpi.server.principal.WLPIPrincipal` メソッドを使用します。

```
public void mapRoleToGroup(
  java.lang.String roleId,
  java.lang.String orgId,
  java.lang.String groupId
) throws java.rmi.RemoteException,
  com.bea.wlpi.common.WorkflowException
```

`mapRolesToGroups()` メソッドのパラメータを次の表に示します。パラメータには値を指定する必要があります。

表 9-26 mapRolesToGroups() メソッドのパラメータ

パラメータ	説明	有効な値
`roleId`	マッピングするロールの ID	有効なロール ID を指定する文字列。 すべてのロール ID のリストの取得方法については、9-10 ページの「特定のオーガニゼーションに対して定義されているロールを取得する」を参照。
`orgId`	ロールに関連付けられたオーガニゼーションの ID	有効なオーガニゼーション ID を指定する文字列。 すべてのオーガニゼーション ID のリストの取得方法については、9-9 ページの「すべてのオーガニゼーション名を取得する」を参照。
`groupId`	ロールのマッピング先セキュリティ レルム グループの ID	新しい、または既にあるグループ ID を指定する文字列。 指定されたセキュリティ レルム グループが存在しない場合は、このメソッドによって作成される。 グループ ID のリストの取得方法については、9-71 ページの「セキュリティ レルム グループの取得」を参照。

たとえば、次のコードでは、セキュリティ レルム admin にオーガニゼーション ORG1 のロール role1 がマッピングされます。このコード例では、principal は WLPIPrincipal EJB への EJBObject 参照を表しています。

```
principal.mapRoleToGroup(“role1”, “ORG1”, “admin”);
```

mapRoleToGroup() メソッドの詳細については、Javadoc の com.bea.wlpi.server.principal.WLPIPrincipal を参照してください。

グループへの複数のロールのマッピング

BEA WebLogic Server のセキュリティ レルム グループに複数の BPM ロールをマッピングするには、次の com.bea.wlpi.server.principal.WLPIPrincipal メソッドを使用します。

```
public void mapRolesToGroups(
  java.lang.String orgId,
  java.util.Map rolesToGroupMap
) throws java.rmi.RemoteException,
  com.bea.wlpi.common.WorkflowException
```

mapRolesToGroups() メソッドのパラメータを次の表に示します。パラメータには値を指定する必要があります。

表 9-27 mapRolesToGroups() メソッドのパラメータ

パラメータ	説明	有効な値
orgId	ロールに関連付けられたオーガニゼーションの ID	有効なオーガニゼーション ID を指定する文字列。 すべてのオーガニゼーション ID のリストの取得方法については、9-9 ページの「すべてのオーガニゼーション名を取得する」 を参照。
rolesToGroupMap	ロールとグループの関係を指定する Map オブジェクト	*キーと値*のペア（key-value ペア）を持つ Map オブジェクトで、キーとしてロール ID を、値としてグループ ID を指定する。 指定されたセキュリティ レルム グループが存在しない場合は、このメソッドによって作成される。

たとえば、次のコードでは、オーガニゼーション ORG1 に対するマップ map1 で定義されたとおりに、複数のグループに対して複数のロールがマッピングされます。このコード例では、principal は WLPIPrincipal EJB への EJBObject 参照を表しています。

```
principal.mapRolesToGroups(“ORG1”, “map1”);
```

mapRolesToGroups() メソッドの詳細については、Javadoc の com.bea.wlpi.server.principal.WLPIPrincipal を参照してください。

特定のロールに対するグループ マッピングの取得

ロールのマッピング先グループの名前を取得するには、次の com.bea.wlpi.server.principal.WLPIPrincipal メソッドを使用します。

```
public java.lang.String getMappedGroup(
  java.lang.String roleId,
  java.lang.String orgId
) throws java.rmi.RemoteException,
  com.bea.wlpi.common.WorkflowException
```

getMappedGroup メソッドのパラメータを次の表に示します。パラメータには値を指定する必要があります。

表 9-28 getMappedGroup() メソッドのパラメータ

パラメータ	説明	有効な値
roleId	取得するマッピングされたグループのマッピング元ロールの ID	有効なロール ID を指定する文字列。 すべてのオーガニゼーション ID のリストの取得方法については、9-10 ページの「特定のオーガニゼーションに対して定義されているロールを取得する」を参照。
orgId	ロールに関連付けられたオーガニゼーションの ID	有効なオーガニゼーション ID を指定する文字列。 すべてのオーガニゼーション ID のリストの取得方法については、9-9 ページの「すべてのオーガニゼーション名を取得する」を参照。

このメソッドは、指定されたロールのマッピング先セキュリティ レルム グループを返します。マッピングが存在しないときはヌルを返します。

たとえば、次のコードでは、オーガニゼーション ORG1 のロール role1 に関連付けられた、マッピングされたグループを返します。このコード例では、principal は WLPIPrincipal EJB への EJBObject 参照を表しています。

```
principal.getMappedGroups(“role1”, “ORG1”);
```

getMappedGroups() メソッドの詳細については、Javadoc の com.bea.wlpi.server.principal.WLPIPrincipal を参照してください。

特定のオーガニゼーションに対して定義されているすべてのロールに対するグループ マッピングの取得

特定のオーガニゼーションに対して定義されているすべてのロールのマッピング先グループのリストを取得するには、次の com.bea.wlpi.server.principal.WLPIPrincipal メソッドを使用します。

```
public java.util.Map getRoleMappingsInOrg(
  java.lang.String orgId
) throws java.rmi.RemoteException,
  com.bea.wlpi.common.WorkflowException
```

getRoleMappingsInOrg() メソッドのパラメータを次の表に示します。パラメータには値を指定する必要があります。

表 9-29 getRoleMappingsInOrg() メソッドのパラメータ

パラメータ	説明	有効な値
orgId	すべてのロール対グループのマッピングを取得するオーガニゼーションの ID	有効なオーガニゼーション ID を指定する文字列。 すべてのオーガニゼーション ID のリストの取得方法については、9-9 ページの「すべてのオーガニゼーション名を取得する」 を参照。

このメソッドは、*key-value* ペアのマップを返します。このペアでは、キーとしてロール ID、値としてグループ ID が指定されています。

たとえば、次のコードでは、オーガニゼーション `ORG1` に対するロール対グループのマップが返されます。このコード例では、`principal` は `WLPIPrincipal` EJB への `EJBObject` 参照を表しています。

```
Map map = principal.getRoleMappingsInOrg(“ORG1”);
```

`getRoleMappingsInOrg()` メソッドの詳細については、Javadoc の `com.bea.wlpi.server.principal.WLPIPrincipal` を参照してください。

パーミッションのコンフィグレーション

ロールおよびユーザに対して、特定の BPM 機能へのアクセスを保護する手段としてパーミッションのコンフィグレーションを行うことができます。

この節では、パーミッションおよびパーミッションのコンフィグレーションに関連するタスクの実行方法について説明します。内容は以下のとおりです。

- パーミッションの概要
- すべてのロールに対するパーミッションの取得
- 特定のロールに対するパーミッションの取得
- すべてのユーザに対するパーミッションの取得
- 特定のユーザに対するパーミッションの取得
- 特定のパーミッションの設定のチェック
- ロール固有のパーミッションの設定
- ユーザ固有のパーミッションの設定

パーミッションの概要

各ロール、ユーザの一方または両方に対して設定できるパーミッションおよびパーミッションの設定に使用できる、関連した `com.bea.wlpi.common.security.EnumPermission` 静的値を次の表に示します。特定のロールまたはユーザに対するパーミッションの設定方法については、それぞれ、9-83 ページの「ロール固有のパーミッションの設定」または 9-85 ページの「ユーザ固有のパーミッションの設定」を参照してください。

表 9-30 パーミッション

パーミッション	説明	EnumPermission 静的値
Configure System	ビジネス カレンダーの追加、更新、削除などによるアプリケーション コンフィグレーションの変更	`P_Configure_System`
Configure Components	■ ビジネス オペレーションの定義、更新、および削除 ■ プラグインのロードとコンフィグレーション	`P_Configure_Components`
Administer User	■ オーガニゼーション、ロール、およびユーザの管理 ■ ロールおよびユーザに対するパーミッション レベルの指定 ■ タスク ルーティングの更新	`P_Admininister_User`
Monitor Instance	インスタンス、ビジネス カレンダー、ワークロード レポート、および統計レポートのモニタ（編集はしない）	`P_Monitor_Instance`
Create Template	テンプレートの作成	`P_Create_Template`
Delete Template	テンプレートの削除	`P_Delete_Template`
Execute Template	テンプレートの実行	`P_Execute_Template`

パーミッションの詳細については、Javadoc の `com.bea.wlpi.common.security.EnumPermission` を参照してください。

すべてのロールに対するパーミッションの取得

すべてのロール パーミッションを取得するには、次の com.bea.wlpi.server.permission.Permission メソッドを使用します。

```
public java.util.List getAllRolePermissions(
) throws java.rmi.RemoteException,
  com.bea.wlpi.common.WorkflowException
```

注意： getAllRolePermissions() メソッドを実行するには、相当量のリソースが必要となる場合があります。実行時操作中には実行しないでください。

このメソッドは、com.bea.wlpi.common.security.RolePermissionInfo オブジェクトのリストを返します。ロールに対するパーミッションについての情報にアクセスするには、B-19 ページの「RolePermissionInfo オブジェクト」に記載の RolePermissionInfo オブジェクト メソッドを使用します。

たとえば、次のコードでは、すべてのロール パーミッションが返されます。このコード例では、principal は WLPIPrincipal EJB への EJBObject 参照を表しています。

```
List rolePermissions = principal.getAllRolePermissions();
```

getAllRolePermissions() メソッドの詳細については、Javadoc の com.bea.wlpi.server.permission.Permission を参照してください。

特定のロールに対するパーミッションの取得

ロールに割り当てられたパーミッションのリストを取得するには、次の com.bea.wlpi.server.permission.Permission メソッドを使用します。

```
public com.bea.wlpi.common.RolePermissionInfo getRolePermissions(
  java.lang.String roleName,
  java.lang.String orgId
) throws java.rmi.RemoteException,
  com.bea.wlpi.common.WorkflowException
```

getRolePermissions() メソッドのパラメータを次の表に示します。パラメータには値を指定する必要があります。

表 9-31 getRolePermissions() メソッドのパラメータ

パラメータ	説明	有効な値
roleName	パーミッションを返すロールの名前	有効なロール名を指定する文字列。 ロールのリストの取得方法については、9-10 ページの「特定のオーガニゼーションに対して定義されているロールを取得する」を参照。
orgId	すべてのロール対グループのマッピングを取得するオーガニゼーションの ID	有効なオーガニゼーション ID を指定する文字列。 すべてのオーガニゼーション ID のリストの取得方法については、9-9 ページの「すべてのオーガニゼーション名を取得する」を参照。

このメソッドは、com.bea.wlpi.common.security.RolePermissionInfo オブジェクトを戻します。特定のロールについての情報にアクセスするには、B-19 ページの「RolePermissionInfo オブジェクト」に記載の RolePermissionInfo オブジェクト メソッドを使用します。

たとえば、次のコードでは、オーガニゼーション ORG1 のロール role1 に対するパーミッションが返されます。このコード例では、permission は Permission EJB への EJBObject 参照を表しています。

```
RolePermissionInfo rolePermissions =
  permission.getRolePermissions(“role1”, “ORG1”);
```

getRolePermissions() メソッドの詳細については、Javadoc の com.bea.wlpi.server.permission.Permission を参照してください。

すべてのユーザに対するパーミッションの取得

すべてのユーザ パーミッションを取得するには、以下の com.bea.wlpi.server.permission.Permission メソッドのいずれかを使用します。

メソッド 1
```
public java.util.List getAllUserPermissions(
) throws java.rmi.RemoteException,
  com.bea.wlpi.common.WorkflowException
```

メソッド 2
```
public java.util.List getAllUserPermissions(
  boolean getRoles
) throws java.rmi.RemoteException,
  com.bea.wlpi.common.WorkflowException
```

注意： getAllUserPermissions() メソッドを実行するには、相当量のリソースが必要となる場合があります。実行時操作中には実行しないでください。

getAllUserPermissions() メソッドのパラメータを次の表に示します。パラメータには値を指定する必要があります。

表 9-32 getAllUserPermissions() メソッドのパラメータ

パラメータ	説明	有効な値
getRoles	ユーザが所属するロールから継承されたパーミッションを含むかどうかを指定するブール型フラグ	true (ロール パーミッションが返される) または false (ロール パミッションは返されない)。

各メソッドは、com.bea.wlpi.common.security.UserPermissionInfo オブジェクトのリストを返します。第 1 のメソッドでは、デフォルトでは、ユーザの割り当て先ロールから継承されたパーミッションが戻されます。第 2 のメソッドでは、getRoles ブール型フラグの値に基づいて、継承されたパーミッションを返すかどうかが決定されます。ユーザ固有パーミッションについての情報にアクセスするには、B-29 ページの「UserPermissionInfo オブジェクト」に記載の UserPermissionInfo オブジェクト メソッドを使用します。

たとえば、次のコードでは、特定のユーザに対して設定されたすべてのパーミッションが（デフォルトで返されるユーザの割り当て先ロールから継承されたパーミッションと共に）返されます。このコード例では、permission は Permission EJB への EJBObject 参照を表しています。

```
List userPermissions = principal.getAllUserPermissions();
```

getAllUserPermissions() メソッドの詳細については、Javadoc の com.bea.wlpi.server.permission.Permission を参照してください。

特定のユーザに対するパーミッションの取得

特定のユーザに関連付けられたパーミッションのリストを取得するには、以下の com.bea.wlpi.server.permission.Permission メソッドのいずれかを使用します。

メソッド 1

```
public com.bea.wlpi.common.UserPermissionInfo getUserPermissions(
  java.lang.String userName
) throws java.rmi.RemoteException,
  com.bea.wlpi.common.WorkflowException
```

メソッド 2

```
public com.bea.wlpi.common.UserPermissionInfo getUserPermissions(
  java.lang.String userName,
  boolean getRoles
) throws java.rmi.RemoteException,
  com.bea.wlpi.common.WorkflowException
```

getUserPermissions() メソッドのパラメータを次の表に示します。パラメータには値を指定する必要があります。

表 9-33 getUserPermissions() メソッドのパラメータ

パラメータ	説明	有効な値
userName	パーミッションを返すユーザの名前	有効なユーザ名を指定する文字列。ユーザのリストの取得方法については、9-12 ページの「特定のオーガニゼーションに対して定義されているユーザを取得する」を参照。
getRoles	ユーザの割り当て先ロールも返すかどうかを指定するブール型フラグ	true (ロールが返される) または false (ロールは返されない)。

各メソッドは、com.bea.wlpi.common.security.UserPermissionInfo オブジェクトを返します。第 1 のメソッドでは、デフォルトで、ユーザの割り当て先ロールのリストも返されます。第 2 のメソッドでは、getRoles ブール型フラグの値に基づいて、ロールのリストを返すかどうかが決定されます。ユーザ固有パーミッションについての情報にアクセスするには、B-29 ページの「UserPermissionInfo オブジェクト」に記載の UserPermissionInfo オブジェクト メソッドを使用します。

たとえば、次のコードでは、特定のユーザに対して設定されたすべてのパーミッションが（デフォルトで返される、ユーザの割り当て先ロールと共に）返されます。このコード例では、permission は Permission EJB への EJBObject 参照を表しています。

```
List userPermissions = permission.getUserPermissions();
```

getUserPermissions() メソッドの詳細については、Javadoc の com.bea.wlpi.server.permission.Permission を参照してください。

特定のパーミッションの設定のチェック

PermissionInfo オブジェクトに対して特定のパーミッションが設定されているかを調べるには、次の com.bea.wlpi.common.security.PermissionInfo メソッドを使用します。

```
public boolean hasPermission(
  com.bea.wlpi.common.security.EnumPermission permission
)
```

hasPermission() メソッドのパラメータを次の表に示します。パラメータには値を指定する必要があります。

表 9-34 hasPermission() メソッドのパラメータ

パラメータ	説明	有効な値
permission	設定の有無をチェックするパーミッションの名前	設定の有無をチェックするパーミッションを指定する EnumPermission オブジェクト。 設定可能なパーミッションについては、9-77 ページの「パーミッションの概要」 を参照。

このメソッドは、指定されたパーミッションがオブジェクトに含まれているかを示すブール値を返します。

たとえば、次のコードでは、指定されたパーミッションが設定されているかどうかを調べます。このコード例では、permissionInfo は PermissionInfo クラスへのオブジェクト参照を表しています。

```
boolean hasPermission =
  permissionInfo.hasPermission(P_Administer_User);
```

hasPermission() メソッドの詳細については、Javadoc の com.bea.wlpi.server.permission.Permission を参照してください。

ロール固有のパーミッションの設定

任意の数のロールに対してパーミッションまたはパーミッション グループを設定するには、以下のメソッドを使用します。

特定のロールに対してパーミッションを設定する

特定のロールに対するパーミッションを設定するには、次の com.bea.wlpi.common.security.PermissionInfo メソッドを使用します。

```
public void setPermission(
  com.bea.wlpi.common.security.EnumPermission permission,
  boolean value
)
```

setPermission() メソッドのパラメータを次の表に示します。パラメータには値を指定する必要があります。

表 9-35 setPermission() メソッドのパラメータ

パラメータ	説明	有効な値
permission	設定するロール固有のパーミッション	有効なパーミッション。 有効なパーミッションのリストについては、9-77 ページの「パーミッション」を参照。

たとえば、次のコードでは、PermissionInfo オブジェクトに対して Administer User パーミッションが設定されます。このコード例では、permissionInfo は PermissionInfo クラスへのオブジェクト参照を表しています。

```
permissionInfo.setPermission(P_Administer_User);
```

詳細については、B-10 ページの「PermissionInfo オブジェクト」を参照してください。

複数のロールに対してパーミッションのグループを設定する

任意の数のロールに対してパーミッション グループを設定するには、次の com.bea.wlpi.server.permission.Permission メソッドを使用します。

```
public void setRolePermissions(
  java.util.List roleInfo
) throws java.rmi.RemoteException,
  com.bea.wlpi.common.WorkflowException
```

setRolePermissions() メソッドのパラメータを次の表に示します。パラメータには値を指定する必要があります。

表 9-36 setRolePermissions() メソッドのパラメータ

パラメータ	説明	有効な値
roleInfo	ロールに対するパーミッションについての最新情報	RolePermissionInfo オブジェクトのリスト。 すべての RolePermissionInfo オブジェクトのリストの取得方法については、9-78 ページの「すべてのロールに対するパーミッションの取得」を参照。RolePermissionInfo オブジェクトを定義する方法については、B-19 ページの「RolePermissionInfo オブジェクト」を参照。

たとえば、次のコードでは、指定された roleInfo RolePermissionInfo オブジェクト内で定義された情報に基づいてロール固有のパーミッションが設定されます。このコード例では、permission は Permission EJB への EJBObject 参照を表しています。

```
permission.setRolePermissions(roleInfo);
```

setRolePermissions() メソッドの詳細については、Javadoc の com.bea.wlpi.server.permission.Permission を参照してください。

ユーザ固有のパーミッションの設定

任意の数のユーザに対して、特定のパーミッションまたはパーミッション グループを設定するには、以下のメソッドを使用します。

のユーザ 1 人に固有なパーミッションを設定する

特定のユーザに対する単一のパーミッションを設定するには、次の com.bea.wlpi.common.security.PermissionInfo メソッドを使用します。

```
public void setPermission(
  com.bea.wlpi.common.security.EnumPermission permission,
  boolean value
)
```

setPermission() メソッドのパラメータを次の表に示します。パラメータには値を指定する必要があります。

表 9-37 setPermission() メソッドのパラメータ

パラメータ	説明	有効な値
permission	設定するユーザ固有のパーミッション	有効なパーミッション。 有効なパーミッションのリストについては、9-77 ページの「パーミッション」を参照。

たとえば、次のコードでは、PermissionInfo オブジェクトに対して Administer User パーミッションが設定されます。このコード例では、permissionInfo は PermissionInfo クラスへのオブジェクト参照を表しています。

```
permissionInfo.setPermission(P_Administer_User);
```

詳細については、B-10 ページの「PermissionInfo オブジェクト」を参照してください。

複数のユーザに対してパーミッション グループを設定する

任意の数のユーザに対するパーミッション グループを取得するには、次の com.bea.wlpi.server.principal.WLPIPrincipal メソッドを使用します。

```
public void setUserPermissions(
  java.util.List userInfo
) throws java.rmi.RemoteException,
  com.bea.wlpi.common.WorkflowException
```

setUserPermissions() メソッドのパラメータを次の表に示します。パラメータには値を指定する必要があります。

表 9-38 setUserPermissions() メソッドのパラメータ

パラメータ	説明	有効な値
userInfo	ユーザ固有のパーミッションについての最新情報	UserPermissionInfo オブジェクトのリスト。 すべての UserPermissionInfo オブジェクトのリストの取得方法については、9-79 ページの「すべてのユーザに対するパーミッションの取得」を参照。UserPermissionInfo オブジェクトを定義する方法については、B-29 ページの「UserPermissionInfo オブジェクト」を参照。

たとえば、次のコードでは、指定された userInfo UserPermissionInfo オブジェクト内で定義された情報に基づいてユーザ パーミッションが設定されます。このコード例では、permission は Permission EJB への EJBObject 参照を表しています。

```
permission.setUserPermissions(userInfo);
```

setUserPermissions() メソッドの詳細については、Javadoc の com.bea.wlpi.server.permission.Permission を参照してください。

Part II　コンフィグレーション

10 ビジネス オペレーションのコンフィグレーション

ビジネス オペレーションは、カスタマイズされたアクションを作成することのできる、EJB または Java クラスのインスタンスで実装された、メソッド呼び出しを表します。詳細については『*WebLogic Integration Studio ユーザーズ ガイド*』の「ワークフロー リソースのコンフィグレーション」の「ビジネス オペレーションのコンフィグレーション」を参照してください。

この章ではビジネス オペレーションのコンフィグレーションと関連するタスクについて説明します。内容は以下のとおりです。

- ビジネス オペレーションの追加
- ビジネス オペレーションの取得
- ビジネス オペレーションの更新
- ビジネス オペレーションの削除
- EJB 記述子の取得
- Java クラス記述子の取得
- ビジネス オペレーションのコンフィグレーション例

この章に記載するメソッドの詳細については、Javadoc の `com.bea.wlpi.server.admin.Admin` および `com.bea.wlpi.server.catalog.EJBCatalog` を参照してください。

ビジネス オペレーションの追加

ビジネス オペレーションを追加するには、以下のいずれかの `com.bea.wlpi.server.admin.Admin` メソッドを使用します。

メソッド 1

```
public java.lang.String addBusinessOperation(
  com.bea.wlpi.common.EJBInvocationDescriptor descriptor
) throws java.rmi.RemoteException,
  com.bea.wlpi.common.WorkflowException
```

メソッド 2

```
public java.lang.String addBusinessOperation(
  com.bea.wlpi.common.ClassInvocationDescriptor descriptor
) throws java.rmi.RemoteException,
  com.bea.wlpi.common.WorkflowException
```

各メソッドにより、1 つのビジネス オペレーションが作成されます。1 つ目のメソッドはセッションまたはエンティティ EJB で実装されたビジネス オペレーションを作成し、2 つ目はサーバでアクセス可能な Java クラス インスタンスによって実装されたビジネス オペレーションを作成します。

注意： セッション EJB は、ワークフロー変数内に保管することができますが、ステートフル セッション EJB は、作成されたトランザクションが存在する期間のみ保管され、ワークフロー インスタンスの一部としては永続しません。トランザクション モデルの詳細については、7-1 ページの「BPM トランザクション モデルの理解」を参照してください。一方、エンティティ EJB、ステートレス EJB、および Java クラス インスタンスは、ワークフロー インスタンスの一部として永続します。

addBusinessOperation() メソッドのパラメータを次の表に示します。パラメータには値の指定が必要です。

表 10-1 addBusinessOperation() メソッド パラメータ

パラメータ	説明	有効な値
descriptor	ビジネス オペレーションの呼び出し記述子 呼び出し記述子には、ビジネス オペレーションを実装する EJB または Java クラス インスタンスのメソッド呼び出しを記述するメタデータが含まれる。	以下のいずれかの値が含まれる。 ■ EJB にメソッドが追加される場合は、com.bea.wlpi.common.EJBInvocationDescriptor が含まれる。 ■ Java クラス インスタンスにメソッドが追加される場合は、com.bea.wlpi.common.ClassInvocationDescriptor が含まれる。 EJB または Java クラス呼び出し記述子の取得については、それぞれ 10-8 ページの「EJB 記述子の取得」および 10-10 ページの「Java クラス記述子の取得」を参照。EJBInvocationDescriptor、または ClassInvocationDescriptor の定義方法については、それぞれ C-6 ページの「EJBInvocationDescriptor オブジェクト」または C-3 ページの「ClassInvocationDescriptor オブジェクト」を参照。

各メソッドは、新しいビジネス オペレーションの ID を返します。

たとえば、次のコードは指定された呼び出し記述子 descriptor と EJB ビジネス オペレーションを追加します。このコード例では、admin は Admin EJB への EJBObject 参照を表します。

```
String operationId = admin.addBusinessOperation(descriptor);
```

addBusinessOperation() メソッドの詳細については、Javadoc の com.bea.wlpi.server.admin.Admin を参照してください。

ビジネス オペレーションの取得

定義されたビジネス オペレーションのリストを取得するには、次の com.bea.wlpi.server.admin.Admin メソッドを使用します。

```
public java.util.List getBusinessOperations(
) throws java.rmi.RemoteException,
  com.bea.wlpi.common.WorkflowException
```

メソッドは、現在定義されているビジネス オペレーションが識別されている com.bea.wlpi.common.EJBInvocationDescriptor または com.bea.wlpi.common.ClassInvocationDescriptor オブジェクトのリストを返します。各オブジェクトについての情報にアクセスするには、それぞれ C-6 ページの「EJBInvocationDescriptor オブジェクト」、または C-3 ページの「ClassInvocationDescriptor オブジェクト」 に記載する EJBInvocationDescritor、または ClassInvocationDescriptor オブジェクト メソッドを使用します。

たとえば、次のコードは現在定義されているビジネス オペレーションを取得します。このコード例では、admin は Admin EJB への EJBObject 参照を表します。

```
String operationsList = admin.getBusinessOperations();
```

getBusinessOperations() メソッドの詳細については、Javadoc の com.bea.wlpi.server.admin.Admin を参照してください。

ビジネス オペレーションの更新

ビジネス オペレーションを更新するには、以下のいずれかの com.bea.wlpi.server.admin.Admin メソッドを使用します。

メソッド 1

```
public void updateBusinessOperation(
  java.lang.String busOpId,
  com.bea.wlpi.common.EJBInvocationDescriptor descriptor
) throws java.rmi.RemoteException,
  com.bea.wlpi.common.WorkflowException
```

メソッド 2

```
public void updateBusinessOperation(
  java.lang.String busOpId,
  com.bea.wlpi.common.EJBInvocationDescriptor descriptor
) throws java.rmi.RemoteException,
  com.bea.wlpi.common.WorkflowException
```

各メソッドにより 1 つのビジネス オペレーションが更新されます。1 つ目のメソッドはセッションまたはエンティティ EJB で実装されたビジネス オペレーションを更新し、2 つ目はサーバでアクセス可能な Java クラス インスタンスによって実装されたビジネス オペレーションを更新します。

`updateBusinessOperation()` メソッドのパラメータを次の表に示します。パラメータには値の指定が必要です。

表 10-2 updateBusinessOperation() メソッド パラメータ

パラメータ	説明	有効な値
`busOpId`	更新するビジネス オペレーションの ID	有効なビジネス オペレーション ID を指定する文字列。 ビジネス オペレーション ID を取得するには、以下の `com.bea.wlpi.common.EJBInvocationDescriptor` または `com.bea.wlpi.common.ClassInvocationDescriptor` メソッドを使用する。 `public java.lang.String getId()` EJB または Java クラス呼び出し記述子の取得については、それぞれ 10-8 ページの「EJB 記述子の取得」および 10-10 ページの「Java クラス記述子の取得」を参照。 `EJBInvocationDescriptor`、または `ClassInvocationDescriptor` の定義方法については、それぞれ C-6 ページの「EJBInvocationDescriptor オブジェクト」または C-3 ページの「ClassInvocationDescriptor オブジェクト」を参照。

表 10-2 updateBusinessOperation() メソッド パラメータ (Continued)

パラメータ	説明	有効な値
descriptor	ビジネス オペレーションの呼び出し記述子 呼び出し記述子には、ビジネス オペレーションを実装する EJB または Java クラス インスタンスのメソッド呼び出しを記述するメタデータが含まれる。	以下のいずれかの値が含まれる。 ■ EJB のメソッドが更新される場合は、com.bea.wlpi.common.EJBInvocationDescriptor が含まれる。 ■ Java クラス インスタンス のメソッドが更新される場合は、com.bea.wlpi.common.ClassInvocationDescriptor が含まれる。 EJB、または Java クラス記述子の取得については、それぞれ 10-8 ページの「EJB 記述子の取得」および 10-10 ページの「Java クラス記述子の取得」を参照。

たとえば、次のコードは指定された ID および呼び出し記述子 descriptor で EJB ビジネス オペレーションを更新します。このコード例では、admin は Admin EJB への EJBObject 参照を表します。

```
admin.updateBusinessOperation(ì12345î, descriptor);
```

updateBusinessOperation() メソッドの詳細については、Javadoc の com.bea.wlpi.server.admin.Admin を参照してください。

ビジネス オペレーションの削除

ビジネス オペレーションを削除するには、次の com.bea.wlpi.server.admin.Admin メソッドを使用します。

```
public void deleteBusinessOperation(
  java.lang.String busOpId
) throws java.rmi.RemoteException,
  com.bea.wlpi.common.WorkflowException
```

deleteBusinessOperation() メソッドのパラメータを次の表に示します。パラメータには値の指定が必要です。

表 10-3 deleteBusinessOperation() メソッド パラメータ

パラメータ	説明	有効な値
busOpId	削除するビジネス オペレーションの ID	有効なビジネス オペレーション ID を指定する文字列。 ビジネス オペレーション ID を取得するには、以下の com.bea.wlpi.common.EJBInvocationDescriptor または com.bea.wlpi.common.ClassInvocationDescriptor メソッドを使用する。 public java.lang.String getId() EJB または Java クラス呼び出し記述子の取得については、それぞれ 10-8 ページの「EJB 記述子の取得」および 10-10 ページの「Java クラス記述子の取得」を参照。 EJBInvocationDescriptor、または ClassInvocationDescriptor の定義方法については、それぞれ C-6 ページの「EJBInvocationDescriptor オブジェクト」または C-3 ページの「ClassInvocationDescriptor オブジェクト」を参照。

たとえば、次のコードでは、指定されたビジネス オペレーションが削除されます。このコード例では、admin は Admin EJB への EJBObject 参照を表します。

```
admin.deleteBusinessOperation(i124345i);
```

deleteBusinessOperation() メソッドの詳細については、Javadoc の com.bea.wlpi.server.admin.Admin を参照してください。

EJB 記述子の取得

EJB によって実装されたビジネス オペレーションを追加、更新、または削除するには、EJB 記述子を取得する必要があります。EJB 記述子を取得するには、com.bea.wlpi.server.catalog.EJBCatalog EJB を通じて選択可能なメソッドを使用します。

注意： また、EJBInvocationDescriptor クラスには EJB 記述子の取得、設定、および呼び出しメソッドも含まれます。詳細については、C-6 ページの「EJBInvocationDescriptor オブジェクト」を参照してください。

次の表に、EJB 記述子のプリファレンスを設定するために使用できる com.bea.wlpi.server.catalog.EJBCatalog メソッドを示します。

表 10-4 EJB 記述子取得のプリファレンスを設定するための EJB カタログ EJB メソッド

メソッド	説明

`public void setCatalogRoot(`
`  java.lang.String root`
`) throws java.rmi.RemoteException`
	インストールされた EJB をルックアップする場合に使用する JNDI コンテキストを設定する。
	`root` は、EJB バインドを含むコンテキストの有効 JNDI 名を指定する。

`public void setInspectAlways(`
`  boolean inspectAlways`
`) throws java.rmi.RemoteException`
	EJB 記述子または EJB 名の取得の際に、プロセス エンジンが `EJBMetaData` および JNDI 名を再生成するかを指定する。
	`inspectAlways` は、自動再生成を有効化（`true`）または無効化（`false`）するためのブール引数を指定する。

`public boolean inspectAlways(` `) throws java.rmi.RemoteException`	自動再生成が有効または無効になっているかを確認する。 プロセス エンジンが EJB 記述子または EJB 名を取得する際、JNDI 名および `EJBMetaData` が再生成されるかを指定するブール引数を返す。

次の表に、EJB 記述子を取得するために使用できる com.bea.wlpi.server.catalog.EJBCatalog メソッドを示します。

表 10-5 EJB 記述子を取得するための EJB カタログ EJB メソッド

メソッド	説明
`public java.util.List getEJBDescriptors( ) throws com.bea.wlpi.common.WorkflowException, java.rmi.RemoteException`	前に指定した JNDI コンテキストにインストールされた EJB を取得する。デフォルトの実装は、コンテキスト内のすべての JNDI バインドで再帰的に繰り返され、遭遇する各 `EJBHome` オブジェクト内に含まれる `EJBMetaData` オブジェクトのリストを作成する。 メソッドは、`javax.ejb.EJBMetaData` オブジェクトのリストを返す。
`public java.util.List getEJBNames( ) throws com.bea.wlpi.common.WorkflowException, java.rmi.RemoteException`	前に指定した JNDI コンテキストにインストールされた EJB の JNDI 名を取得する。デフォルトの実装は、コンテキスト内のすべての JNDI バインドを再帰的に繰り返され、遭遇するすべての `EJBHome` オブジェクトの JNDI 名を含むリストを作成する。 このメソッドは、JNDI 名のリストを返す。

たとえば、次のコードは EJB のホット デプロイメントを有効化し、すべての `EJBHome` オブジェクトの JNDI 名を取得します。このコード例では、`catalog` は、`EJBCatalog` EJB への `EJBObject` 参照を表します。

```
catalog.setInspectAlways(true);
List ejbList = catalog.getEJBNames();
```

`EJBCatalog` メソッドの詳細については、Javadoc の `com.bea.wlpi.server.catalog.EJBCatalog` を参照してください。

Java クラス記述子の取得

Java クラスによって実装されたビジネス オペレーションを追加、更新、または削除するには、クラス記述子を取得する必要があります。クラス記述子を取得するには、`com.bea.wlpi.server.admin.Admin` EJB を通じて選択可能なメソッドを使用します。

注意： また、ClassInvocationDescriptor クラスには、クラス記述子の取得、設定、および呼び出しメソッドも含まれます。詳細については、C-3 ページの「ClassInvocationDescriptor オブジェクト」を参照してください。

Java クラス記述子を取得するには、次の com.bea.wlpi.server.admin.Admin メソッドを使用します。

```
public com.bea.wlpi.common.ClassDescriptor getClassDescriptor(
  java.lang.String className
) throws java.rmi.RemoteException,
  com.bea.wlpi.common.WorkflowException
```

getClassDescriptor() メソッドのパラメータを次の表に示します。パラメータには値の指定が必要です。

表 10-6 getClassDescriptor() メソッド パラメータ

パラメータ	説明	有効な値
className	返却を行う記述子の完全修飾 Java クラス名	有効な完全修飾 Java クラス名を指定する文字列。

メソッドは、指定された Java クラスのメタデータを含む ClassDescriptor を返します。

たとえば、次のコードは指定された Java クラスのクラス記述子、com.somedomain.someproduct.CreateWidget を取得します。このコード例では、admin は Admin EJB への EJBObject 参照を表します。

```
ClassDescriptor descriptor = admin.getClassDescriptor
  (“com.somedomain.someproduct.CreateWidget”);
```

getClassDescriptor() メソッドの詳細については、Javadoc の com.bea.wlpi.server.admin.Admin を参照してください。

ビジネス オペレーションのコンフィグレーション例

この節では、コマンドライン管理クライアント例から抜粋して、環境のコンフィグレーションを示します。

- EJB 記述子の取得例
- ビジネス オペレーションのコンフィグレーション例

EJB 記述子の取得例

この節では、コマンドライン管理例から抜粋して、EJB 記述子およびセットと関連したプリファレンスを示します。

注意： コマンドライン管理サンプルの詳細については、1-23 ページの「コマンドライン管理サンプル」を参照してください。

このサンプルでは、ユーザと通信するための入力ストリームが定義されており、ユーザは以下のアクションのいずれかを指定するよう求められます。

- Inspect Always フラグを問い合わせる
- Inspect Always フラグを設定する
- デプロイ EJB 名を取得する
- EJB デプロイメント記述子を取得する

重要なコード行は、**太字**で強調されています。このコード例では、`catalog` は、`EJBCatalog` EJB への `EJBObject` 参照を表します。

```
/* ユーザとの通信のための入力ストリームを作成する */
stdIn = new BufferedReader( new InputStreamReader( System.in ) );

/* ツール タイトルを表示する */
   System.out.print( "\n---  Command Line Administration v1.1  ---" );

/* メイン メニューを表示してユーザと交信する */
while( true ) {
```

```
/* メニューを表示する */
   System.out.println( "\n---              Main Menu            ---" );
   System.out.println( "\nEnter choice:" );
   System.out.println( "1) Organizations" );
   System.out.println( "2) Roles" );
   System.out.println( "3) Users" );
   System.out.println( "4) Security Realm" );
   System.out.println( "5) Business Operations" );
   System.out.println( "6) Event Keys" );
   System.out.println( "7) Business Calendars" );
   System.out.println( "8) EJB Catalog" );
   System.out.println( "9) Server Properties" );
   System.out.println( "Q) Quit" );
   System.out.print( ">> " );
      .
      .
      .
```

mngEJBCatalog() メソッドは、EJB カタログの管理方法を表し、必要な情報を検索するためにユーザと交信します。

```
private static void mngEJBCatalog( ) {
   boolean isInspectAlways;
   List ejbList;
   String answer;

   /* ユーザとの通信のための入力ストリームを作成する */
   BufferedReader stdIn = new BufferedReader( new InputStreamReader( System.in )
);

   try {
   /* メニューを表示してユーザと交信する */
      while( true ) {
      /* メニューを表示する */
         System.out.println( "\n\n---             EJB Catalog           ---" );
         System.out.println( "\nEnter choice:" );
         System.out.println( "1) Query Inspect Always Flag" );
         System.out.println( "2) Set Inspect Always Flag" );
         System.out.println( "3) List names of deployed EJBs" );
         System.out.println( "4) List descriptors of deployed EJBs" );
         System.out.println( "B) Back to previous menu" );
         System.out.println( "Q) Quit" );
         System.out.print( ">> " );

         /* ユーザの選択を取得する */
         String line = stdIn.readLine( );

         /* ユーザが選択を行わないで〔Enter〕を押したか ? */
```

```
   if( line.equals( "" ) )
      continue;
   /* ユーザが複数の文字を入力したか ? */
   else if( line.length( ) > 1 ) {
      System.out.println( "*** Invalid selection" );
   continue;
}

/* 大文字および文字へのコンバート */
char choice = line.toUpperCase( ).charAt( 0 );

/* ユーザの選択を処理する */
switch( choice ) {
.
.
.
```

Inspect Always フラグを問い合わせる

Inspect Always フラグのクエリ方法を示します。

```
/* Inspect Always フラグを問い合わせる */
   case '1' :
      /* WLPI 公開 API メソッド */
      /* 注意： 発生した例外を取り込むコードを
       * 追加するとよい */
      isInspectAlways = catalog.inspectAlways( );

      if( isInspectAlways )
         System.out.println( "\nInspect Always flag is set " +
            "(EJB hot deployment is enabled)" );
      else
         System.out.println( "\nInspect Always flag is not set " +
            "(EJB hot deployment is disabled)" );
      break;
      .
      .
      .
```

Inspect Always フラグを設定する

Inspect Always フラグの設定方法を示します。

```
/* Inspect Always フラグを設定する */
case '2' :
   /* フラグ設定用の値を取得する */
   /* Inspect Always フラグの設定 / 解除が必要かどうかを
    * ユーザに求める */
   if( ( answer = askQuestion( "\nEnable Inspect Always (y/n)?" ) )
   == null ) {
      /* ユーザによる操作の取り消し */
      System.out.println( "*** Cancelled" );
      break;
}

/* 応答を解析する */
isInspectAlways = ( answer.equals( "y" ) || answer.equals( "Y" ) );

try {
   /* WLPI 公開 API メソッド */
   /* Inspect Always フラグを設定する */
   catalog.setInspectAlways( isInspectAlways );

   /* 正常終了（例外の発生なし）*/
   System.out.println( "- Success" );
}
catch( Exception e ) {
   System.out.println( "*** Failed to set Inspect Always flag" );
   System.err.println( e );
}
break;
   .
   .
   .
```

デプロイ EJB 名を取得する

デプロイ EJB 名の取得方法を示します。

```
/* デプロイ EJB 名をリストする */
case '3' :
   /* WLPI 公開 API メソッド */
   /* デプロイ EJB の JNDI 名をリストする */
   /* 注意 : 発生した例外を取り込むコードを

    * 追加するとよい */
   ejbList = catalog.getEJBNames( );
```

```
   /* デプロイ EJB はあるか ? */
   if( ejbList.size( ) == 0 )
      System.out.println( "\nNo EJB deployed" );
   else
      System.out.println( "\nCataloged EJBs:" );

   /* リストを処理して、名前を表示する */
   for( int i = 0; i < ejbList.size( ); i++ )
      System.out.println( "- JNDI Name: " + ejbList.get( i ) );
   break;
   .
   .
   .
```

EJB デプロイメント記述子を取得する

EJB デプロイメント記述子の取得方法を示します。

```
/* デプロイ EJB の記述子をリストする */
case '4' :
   /* WLPI 公開 API メソッド */
   /* デプロイ EJP をリストする */
   /* ejbList は、com.bea.wlpi.common.EJBDescriptor
    * オブジェクトのリストである。*/
   /* 注意 : 発生した例外を取り込むコードを
    * 追加するとよい */
   ejbList = catalog.getEJBDescriptors( );

   /* デプロイ EJB はあるか ? */
   if( ejbList.size( ) == 0 )
      System.out.println( "\nNo EJB deployed" );
   else
      System.out.println( "\nCataloged EJBs:" );

   /* リストを処理して、その他の属性を表示する */
   for( int i = 0; i < ejbList.size( ); i++ ) {
      /* リストから要素を取り出す */
      EJBDescriptor ejbDescriptor = ( EJBDescriptor )ejbList.get( i );

      /* 選択可能な属性のサブセットを表示する */
      System.out.println( "\n- Deployment Name: " +
         ejbDescriptor.getEJBDeploymentName( ) );
      System.out.println( "  Home Interface: " +
         ejbDescriptor.getEJBHomeName( ) );
      System.out.println( "  Remote Interface: " +
```

```
            ejbDescriptor.getEJBRemoteName( ) );
         /* など ...
          * 選択可能なすべてのメソッドのリストについては、
          * WLPI JavaDocs、Class EJBDescriptor を参照 */
      }
      break;
      .
      .
      .
```

ビジネス オペレーションのコンフィグレーション例

この節では、コマンドライン管理例から抜粋して、ビジネス オペレーションのコンフィグレーション方法を示します。

注意： コマンドライン管理サンプルの詳細については、1-23 ページの「コマンドライン管理サンプル」を参照してください。

このサンプルでは、ユーザと通信するための入力ストリームが定義されており、ユーザは以下のアクションのいずれかを指定するよう求められます。

- ビジネス オペレーションを削除する
- すべてのビジネス オペレーションを取得する

重要なコード行は、**太字**で強調されています。このコード例では、admin は Admin EJB への EJBObject 参照を表します。

```
/* ユーザとの通信のための入力ストリームを作成する */
stdIn = new BufferedReader( new InputStreamReader( System.in ) );

/* ツール タイトルを表示する */
   System.out.print( "\n---  Command Line Administration v1.1  ---" );

/* メイン メニューを表示してユーザと交信する */
while( true ) {
/* メニューを表示する */
   System.out.println( "\n---              Main Menu            ---" );
   System.out.println( "\nEnter choice:" );
   System.out.println( "1) Organizations" );
   System.out.println( "2) Roles" );
```

```
   System.out.println( "3) Users" );
   System.out.println( "4) Security Realm" );
   System.out.println( "5) Business Operations" );
   System.out.println( "6) Event Keys" );
   System.out.println( "7) Business Calendars" );
   System.out.println( "8) EJB Catalog" );
   System.out.println( "9) Server Properties" );
   System.out.println( "Q) Quit" );
   System.out.print( ">> " );
      .
      .
      .
/**
 * WLPI ビジネス オペレーションと関連する、Admin インタフェースで選択可能な公開 API メソッドを示す目的で
 * 必要な情報をすべて取得するために
 * ユーザと交信するメソッド
*/
public static void mngBusinessOperations() {
   ClassInvocationDescriptor operationClassInfo;
   EJBInvocationDescriptor operationEjbInfo;
   List operationList;
   Object o;
   String operationId;

   /* ユーザとの通信のための入力ストリームを作成する */
   BufferedReader stdIn = new BufferedReader(
      new InputStreamReader( System.in ) );

   try {
   /* メニューを表示してユーザと交信する */
      while( true ) {
      /* メニューを表示する */
         System.out.println( "\n\n---       Business Operations        ---" );
         System.out.println( "\nEnter choice:" );
         System.out.println( "1) Delete a Business Operation" );
         System.out.println( "2) List all Business Operations" );
         System.out.println( "B) Back to previous menu" );
         System.out.println( "Q) Quit" );
         System.out.print( ">> " );

         /* ユーザの選択を取得する */
         String line = stdIn.readLine();

         /* ユーザが選択を行わないで〔Enter〕を押したか ? */
         if( line.equals( "" ) )
            continue;
         /* ユーザが複数の文字を入力したか ? */
```

```
else if( line.length() > 1 ) {
   System.out.println( "*** Invalid selection" );
   continue;
}


/* 大文字および文字へのコンバート */
char choice = line.toUpperCase().charAt( 0 );

/* ユーザの選択を処理する */
switch( choice ) {
.
.
.
```

ビジネス オペレーションを削除する

ビジネス オペレーションの削除方法を示します。

```
/* ビジネス オペレーションを削除する */
case '1' :
   /* 削除するオペレーションのオペレーション ID を取得する */
   if( ( operationId = askQuestion( "\nEnter Business Operation
      ID to delete" ) ) == null ) {
      /* ユーザによる操作の取り消し */
      System.out.println( "*** Cancelled" );
      break;
   }

   try {
      /* WLPI 公開 API メソッド */
      /* ビジネス オペレーションを削除する */
      admin.deleteBusinessOperation( operationId );

      /* 正常終了（例外の発生なし）*/
      System.out.println( "- Deleted" );
   }
   catch( Exception e ) {
      System.out.println( "*** Unable to delete Business Operation" );
      System.err.println( e );
   }
   break;
   .
   .
```

.

すべてのビジネス オペレーションを取得する

すべてのビジネス オペレーションの取得方法を示します。

```
/* すべてのビジネス オペレーションをリストする */
case '2' :
   try {
      /* WLPI 公開 API メソッド */
      /* すべてのビジネス オペレーションを取り出す */
      operationList = admin.getBusinessOperations();

      /* 定義されているオペレーションはあるか ? */
      if( operationList.size() == 0 )
         System.out.println( "\nNo Business Operation defined" );
      else
         System.out.println( "\nDefined Business Operations:" );

      /* リストを処理して、オペレーションと属性を表示する */
      for( int i = 0; i < operationList.size(); i++ ) {
         /* リストから要素を取り出す */
         o = operationList.get( i );

         /* ビジネス オペレーションは、クラスであるか ? */
         if( o instanceof ClassInvocationDescriptor ) {
            operationClassInfo = ( ClassInvocationDescriptor ) o;

            /* ビジネス オペレーション タイプを表示する */
            System.out.println( "- Business Operation Type: Class" );

            /* オペレーション ID を取り出して表示する */
            /* WLPI 公開 API メソッド */
            System.out.println( "  ID: " +
               operationClassInfo.getId() );

            /* クラス名を取り出して表示する */
            /* WLPI 公開 API メソッド */
            System.out.println( "  Class Name: " +
               operationClassInfo.getClassName() );

            /* クラス記述を取り出して表示する */
            /* WLPI 公開 API メソッド */
            System.out.println( "  Description: " +
               operationClassInfo.getDescription() );

            /* など ...
             * 選択可能なすべてのメソッドのリスト
```

```
                * については、
            * WLPI JavaDocs、ClassInvocationDescriptor を参照 */
              }
              /* ビジネス オペレーションは、EJB であるか ? */
              else if( o instanceof EJBInvocationDescriptor ) {
                 operationEjbInfo = ( EJBInvocationDescriptor ) o;

                 /* ビジネス オペレーション タイプを表示する */
                 System.out.print( "- Business Operation Type: " );

                 /* セッションあるいは エンティティ EJB であるか */
                 /* WLPI 公開 API メソッド */
                 if( operationEjbInfo.isSessionEJB() )
                    System.out.println( "Session EJB" );
                 else
                    System.out.println( "Entity EJB" );

                 /* オペレーション ID を取り出して表示する */
                 /* WLPI 公開 API メソッド */
                 System.out.println( "  ID: " +
                    operationEjbInfo.getId() );

                 /* EJB デプロイメント名を取り出して表示する */
                 /* WLPI 公開 API メソッド */
                 System.out.println( "  EJB deployment name: " +
                    operationEjbInfo.getEJBDeploymentName() );

                 /* Bean の記述を取り出して表示する */
                 /* WLPI 公開 API メソッド */
                 System.out.println( "  Description: " +
                    operationEjbInfo.getDescription() );

                 /* など ...
                  * 選択可能なすべてのメソッドのリスト
                  * Class EJBInvocationDescriptor,
                  * WLPI JavaDocs、Class EJBInvocationDescriptor を参照 */
              }
           }
        }
        catch( Exception e ) {
           System.out.println( "*** Unable to list Business Operations" );
           System.err.println( e );
        }
        break;
        .
        .
        .
```

11 イベント キーのコンフィグレーション

この章では、イベント キーのコンフィグレーション方法について説明します。内容は以下のとおりです。

- イベント キーの概要
- イベント キーの追加
- イベント キー情報の取得
- イベント キーの更新
- イベント キーを削除する
- イベント キーのコンフィグレーション例

詳細については、『*WebLogic Integration Studio ユーザーズ ガイド*』の「ワークフロー リソースのコンフィグレーション」にある「イベント キーのコンフィグレーション」を参照してください。

イベント キーの概要

イベント キーを使用すると、イベント データに対してワークフロー式を評価することにより、イベント プロセッサでインバウンド イベント固有のキー値を計算できます。通常、ワークフロー式では、フィールド参照または関数呼び出しを使用して着信イベント データ内の名前付きフィールドから情報を抽出します。

イベントは コンテンツ タイプとイベント記述子という 2 つの属性によって特徴付けられており、データ型をサポートするプラグインのフィールドと特定のコンテンツ タイプおよびイベント記述子 プロパティから固有のキーを抽出するワークフロー式のコンテンツが含まれています。

イベント キー属性を次の表に示します。

表 11-1 イベント キー属性

イベント キー属性	説明
Content Type	潜在的なデータ型を識別するイベント データの MIME (Multipurpose Internet Mail Extensions) タイプ。WebLogic Integration でサポートするデフォルトのコンテンツ タイプは、`text/xml` である（ML ドキュメント）。 バイナリ COBOL Copybook レコード、CSV (comma-separated values)、所有権を主張できるファイル フォーマット (Microsoft Word や Excel など)、シリアライズされた Java オブジェクト、HTML、SGML、など追加のコンテンツ タイプを扱うためにプラグインを作成することができる。 BPM プラグインのプログラミングの詳細については、『*WebLogic Integration BPM プラグイン プログラミング ガイド*』を参照。
Event Descriptor	（省略可能）任意のデータ フォーマットを定義するための追加スキーマ レベル情報 **注意 :** コンテンツ タイプによっては、データ フォーマットを定義する追加情報は必要としない。 デフォルトのコンテンツ タイプ `text/xml` の場合、イベント記述子は XML ドキュメント タイプを反映し、着信ドキュメントで宣言されている場合には、DOCTYPE DTD (Document Type Definition : 文書型定義) 内の公開 ID または（公開 ID が定義されていない場合）システム ID と一致させる。DOCTYPE が宣言されていない場合は、イベント記述子はドキュメント ルート要素名を一致させる。 プラグインがサポートするコンテンツ タイプは、必要な場合において、十分な情報を与えるためにフィールド参照がイベント データから指定されたフィールドを抽出できるよう、イベント記述子を使用する。（未処理バイナリ レコード画像など）データが自己記述型ではない場合、イベント記述子は、フィールド参照が、たとえば、バイト オフセット、長さ、および参照されたフィールドのデータ型など関連するデータ ディクショナリ サービスを検索できるよう、十分な情報を提供する必要がある。

イベント プロセッサは、どの着信イベントの固有キーも計算できます。イベント ノードのように、イベントが特定のワークフロー インスタンスに送られる場合、計算されたキー値を対象であるワークフロー インスタンスの ID に一致させる必要があります。一致作業は、入力コンテンツ タイプ、イベント記述子、およびキー値を、待機中のインスタンスおよびイベント ノード ID と関連付けるために使用する、イベント監視表によって行われます。また、イベント監視表は、特定のコンテンツ タイプ、イベント記述子、およびキー値の組み合わせ（この場合インスタンス ID は null）の受信に対してインスタンス化され、トリガーされたワークフローを処理するためにも使用されます。

イベント キーの追加

イベント キーを追加するには、次の com.bea.wlpi.server.admin.Admin メソッドを使用します。

```
public void addEventKey(
  com.bea.wlpi.common.EventKeyInfo eventKeyInfo
) throws java.rmi.RemoteException,
  com.bea.wlpi.common.WorkflowException
```

addEventKey() メソッドのパラメータを次の表に示します。パラメータには値の指定が必要です。

表 11-2 addEventKey() メソッドのパラメータ

パラメータ	説明	有効な値
eventKeyInfo	新しいイベント キー情報	EventKeyInfo オブジェクト。 EventKeyInfo オブジェクトを定義する方法については、B-3 ページの「EventKeyInfo オブジェクト」を参照。

たとえば、以下のコードでは、指定した eventKeyInfo オブジェクトのコンテンツに基づきイベント キーが追加されます。このコード例では、admin は Admin EJB への EJBObject 参照を表します。

```
admin.addEventKey(eventKeyInfo);
```

addEventKey() メソッドの詳細については、Javadoc の com.bea.wlpi.server.admin.Admin を参照してください。

イベント キー情報の取得

イベント キーについての情報を取得するには、次の com.bea.wlpi.server.admin.Admin メソッドを使用します。

```
public java.util.List getEventKeyInfo(
) throws java.rmi.RemoteException,
  com.bea.wlpi.common.WorkflowException
```

このメソッドは、com.bea.wlpi.common.EventKeyInfo オブジェクトのリストを戻します。各イベント キーの情報にアクセスするには、B-3 ページの「EventKeyInfo オブジェクト」に記載の EventKeyInfo オブジェクト メソッドを使用します。

たとえば、以下のコードにより、イベント キー情報が取得されて、eventKeys リスト オブジェクトに保存されます。このコード例では、admin は Admin EJB への EJBObject 参照を表します。

```
List eventKeys = admin.getEventKeyInfo();
```

getEventKey() メソッドの詳細については、Javadoc の com.bea.wlpi.server.admin.Admin を参照してください。

イベント キーの更新

イベント キーを更新するには、次の com.bea.wlpi.server.admin.Admin メソッドを使用します。

```
public void updateEventKey(
  com.bea.wlpi.common.EventKeyInfo eventKeyInfo
) throws java.rmi.RemoteException,
  com.bea.wlpi.common.WorkflowException
```

updateEventKey() メソッドのパラメータを次の表に示します。パラメータには値の指定が必要です。

表 11-3 updateEventKey() メソッドのパラメータ

パラメータ	説明	有効な値
eventKeyInfo	更新するイベント キー情報	EventKeyInfo オブジェクト。 EventKeyInfo オブジェクトを定義する方法については、B-3 ページの「EventKeyInfo オブジェクト」 を参照。

たとえば、以下のコードでは、指定した eventKeyInfo オブジェクトのコンテンツに基づきイベント キーが更新されます。このコード例では、admin は Admin EJB への EJBObject 参照を表します。

```
admin.updateEventKey(eventKeyInfo);
```

updateEventKey() メソッドの詳細については、Javadoc の com.bea.wlpi.server.admin.Admin を参照してください。

イベント キーを削除する

イベント キーを削除するには、次の com.bea.wlpi.server.admin.Admin メソッドを使用します。

```
public void deleteEventKey(
  com.bea.wlpi.common.EventKeyInfo eventKeyInfo
) throws java.rmi.RemoteException,
  com.bea.wlpi.common.WorkflowException
```

deleteEventKey() メソッドのパラメータを次の表に示します。パラメータには値の指定が必要です。

表 11-4 deleteEventKey() メソッドのパラメータ

パラメータ	説明	有効な値
eventKeyInfo	削除するイベント キー	既存の EventKeyInfo オブジェクト。 EventKeyInfo オブジェクトを定義する方法については、B-3 ページの「EventKeyInfo オブジェクト」を参照。

たとえば、次のコードは指定されたイベント キーを削除します。このコード例では、admin は Admin EJB への EJBObject 参照を表します。

```
admin.deleteEventKey(eventKeyInfo);
```

deleteEventKey() メソッドの詳細については、Javadoc の com.bea.wlpi.server.admin.Admin を参照してください。

イベント キーのコンフィグレーション例

この節では、コマンドライン管理のコード例から抜粋して、イベント キーのコンフィグレーション方法を示します。

注意： コマンドライン管理サンプルの詳細については、1-23 ページの「コマンドライン管理サンプル」を参照してください。

このサンプルでは、ユーザと通信するための入力ストリームが定義されており、ユーザは以下のアクションのいずれかを指定するよう求められます。

- イベント キーの追加
- イベント キーの削除
- イベント キーの取得
- イベント キーの更新

重要なコード行は、**太字**で強調されています。このコード例では、admin は Admin EJB への EJBObject 参照を表します。

```
/* ユーザとの通信のための入力ストリームを作成する */
stdIn = new BufferedReader( new InputStreamReader( System.in ) );

/* ツール タイトルを表示する */
   System.out.print( "\n---  Command Line Administration v1.1  ---" );

/* メイン メニューを表示してユーザと交信する */
while( true ) {
/* メニューを表示する */
   System.out.println( "\n---              Main Menu            ---" );
   System.out.println( "\nEnter choice:" );
   System.out.println( "1) Organizations" );
   System.out.println( "2) Roles" );
   System.out.println( "3) Users" );
   System.out.println( "4) Security Realm" );
   System.out.println( "5) Business Operations" );
   System.out.println( "6) Event Keys" );
   System.out.println( "7) Business Calendars" );
   System.out.println( "8) EJB Catalog" );
   System.out.println( "9) Server Properties" );
   System.out.println( "Q) Quit" );
   System.out.print( ">> " );
      .
      .
      .
/**
 * WLPI イベント キーと関連する、Admin インタフェースで選択可能な公開 API メソッドを示す目的
で
 * 必要な情報をすべて取得するために
 * ユーザと交信するメソッド
 * イベント キーは、着信 XML ドキュメントと関連したワークフロー定義またはインスタンスを
 * 識別するために
 * 使用するイベント キーを使用すると、
 * 該当するものを取得するためにすべての XML ベース ワークフロー トリガーのシーケンス検索を行う
必要がなくなる
 */
public static void mngEventKeys() {
   EventKeyInfo eventKeyInfo;
   List eventKeyList;
   String eventKeyRoot;
   String eventKeyExpr;
   //boolean isGetDefinition;
   //BusinessCalendarInfo calendarInfo;
   //String answer;
   //String calendarDefinition;
   //String calendarId;
   //String calendarName;
   //String calendarTimezone;// XML Document
```

```
/* ユーザとの通信のための入力ストリームを作成する */
BufferedReader stdIn = new BufferedReader( new InputStreamReader(

   System.in ) );

try {
   /* メニューを表示してユーザと交信する */
   while( true ) {
     /* メニューを表示する */
      System.out.println( "\n\n---            Event Keys            ---" );
      System.out.println( "\nEnter choice:" );
      System.out.println( "1) Add an Event Key" );
      System.out.println( "2) Delete an Event Key" );
      System.out.println( "3) List all Event Keys" );
      System.out.println( "4) Update an Event Key" );
      System.out.println( "B) Back to previous menu" );
      System.out.println( "Q) Quit" );
      System.out.print( ">> " );

      /* ユーザの選択を取得する */
      String line = stdIn.readLine();

      /* ユーザが選択を行わないで〔Enter〕を押したか ? */
      if( line.equals( "" ) )
         continue;
      /* ユーザが複数の文字を入力したか ? */
      else if( line.length() > 1 ) {
         System.out.println( "*** Invalid selection" );
         continue;
      }

      /* 大文字および文字へのコンバート */
      char choice = line.toUpperCase().charAt( 0 );

      /* ユーザの選択を処理する */
      switch( choice ) {
      .
      .
      .
```

イベント キーの追加

イベント キーの追加方法を示します。

```
/* イベント キーを追加する */
case '1' :
   /* 新しいイベント キーのイベント キー XML ドキュメント ルートを取得する */
   if( ( eventKeyRoot = askQuestion( "\nEnter the Event Key XML
      Document Root" ) ) == null ) {
      /* ユーザによる操作の取り消し */
      System.out.println( "*** Cancelled" );
      break;
   }

   /* 新しいイベント キーの式を取得する */
  if( ( eventKeyExpr = askQuestion( "Enter the Event Key Expression" ) )
      == null ) {
      /* ユーザによる操作の取り消し */
      System.out.println( "*** Cancelled" );
      break;
   }

   try {
      /* EventKeyInfo オブジェクトを作成する ;
       * 追加メソッドを呼び出すために必要 */
      eventKeyInfo = new EventKeyInfo(
         "text/xml", eventKeyRoot, eventKeyExpr, null, 0 );

      /* WLPI 公開 API メソッド */
      /* イベント キーを追加する */
      admin.addEventKey( eventKeyInfo );

      /* 正常終了（例外の発生なし）*/
      System.out.println( "- Added" );
   }
   catch( Exception e ) {
      System.out.println( "*** Unable to add the Event Key" );
      System.err.println( e );
   }
   break;
   .
   .
   .
```

イベント キーの削除

イベント キーの削除方法を示します。

```
/* イベント キーを削除する */
case '2' :
   /* 削除するイベント キーのイベント キー XML ドキュメント ルートを取得する */
   if( ( eventKeyRoot = askQuestion( "\nEnter the root for
      the Event Key to delete" ) ) == null ) {
   /* ユーザによる操作の取り消し */
      System.out.println( "*** Cancelled" );
      break;
   }

   try {
      /* 削除メソッドを呼び出すには、
       * 基本的に XML ドキュメント ルートと式を含んだ
       * EventKeyInfo オブジェクトを作成する必要がある。式を削除する
       * 必要はない */
      eventKeyExpr = "";

      /* EventKeyInfo オブジェクトを作成する ;
       * 削除メソッドを呼び出すために必要 */
      eventKeyInfo = new EventKeyInfo(
         "text/xml", eventKeyRoot, eventKeyExpr, null, 0 );

      /* WLPI 公開 API メソッド */
      /* イベント キーを削除する */
      admin.deleteEventKey( eventKeyInfo );

      /* 正常終了（例外の発生なし）*/
      System.out.println( "- Deleted" );
   }
   catch( Exception e ) {
      System.out.println( "*** Unable to delete the Event Key" );
      System.err.println( e );
   }
   break;
   .
   .
   .
```

イベント キーの取得

イベント キーの取得方法を示します。

```
/* すべてのイベント キーをリストアップする */
case '3' :
   try {
      /* WLPI 公開 API メソッド */
      /* すべてのイベント キーを検索する */
     eventKeyList = admin.getEventKeyInfo();

      /* 定義されたイベント キーはあるか ? */
      if( eventKeyList.size() == 0 )
         System.out.println( "\nNo Event Keys defined" );
      else
         System.out.println( "\nDefined Event Keys:" );

      /* リストを処理して、イベント キーと属性を表示する */
      for( int i = 0; i < eventKeyList.size(); i++ ) {
         /* リストから要素を取り出す */
         eventKeyInfo = ( EventKeyInfo )eventKeyList.get( i );

         /* WLPI 公開 API メソッド */
         /* XML ドキュメント ルートを取り出して表示する */
         System.out.println( "- Content Type:       " +
           eventKeyInfo.getContentType() );

         /* WLPI 公開 API メソッド */
         /* XML ドキュメント ルートを取り出して表示する */
         System.out.println( "- Event Descriptor:       " +
           eventKeyInfo.getEventDescriptor() );

         /* WLPI 公開 API メソッド */
         /* カレンダー名を取り出して表示する */
         System.out.println( "  Expression: " +
            eventKeyInfo.getExpr() );
      }
   }
   catch( Exception e ) {
      System.out.println( "*** Unable to retrieve Event Keys" );
      System.err.println( e );
   }
   break;
   .
   .
```

.

イベント キーの更新

イベント キーの更新方法を示します。

```
/* イベント キーを更新する */
case '4' :
   /* 更新するイベント キーのイベント キー XML ドキュメント ルートを取得する */
   if( ( eventKeyRoot = askQuestion( "\nEnter the root for the
      Event Key to update" ) ) == null ) {
      /* ユーザによる操作の取り消し */
      System.out.println( "*** Cancelled" );
      break;
   }

   /* 新しいイベント キーの式を取得する */
   if( ( eventKeyExpr = askQuestion( "Enter the new Event Key
      Expression" ) ) == null ) {
      /* ユーザによる操作の取り消し */
      System.out.println( "*** Cancelled" );
      break;
   }

   try {
      /* EventKeyInfo オブジェクトを作成する ;
       * 更新メソッドを呼び出すために必要 */
      eventKeyInfo = new EventKeyInfo(
         "text/xml", eventKeyRoot, eventKeyExpr, null, 0 );

      /* WLPI 公開 API メソッド */
      /* イベント キーを更新する */
      admin.updateEventKey( eventKeyInfo );

      /* 正常終了（例外の発生なし）*/
      System.out.println( "- Updated" );
   }
   catch( Exception e ) {
      System.out.println( "*** Unable to update the Event Key" );
      System.err.println( e );
   }
   break;
   .
```

.
.

12 ビジネス カレンダーのコンフィグレーション

ビジネス カレンダーで、オーガニゼーションの稼働時間を定義できます。ビジネス カレンダーの管理方法の詳細については、『*WebLogic Integration Studio ユーザーズ ガイド*』の「データの管理」の「ビジネス カレンダーの管理」を参照してください。

この章では、ビジネス カレンダーのコンフィグレーションと関連したタスクについて説明します。内容は以下のとおりです。

- ビジネス カレンダーの追加
- ビジネス カレンダーの取得
- ビジネス カレンダー定義の取得
- ビジネス カレンダーの更新
- ビジネス カレンダーの削除
- ビジネス カレンダーのコンフィグレーション例

ビジネス カレンダーの追加

ビジネス カレンダーを追加するには、次の `com.bea.wlpi.server.admin.Admin` メソッドを使用します。

```
public java.lang.String addBusinessCalendar(
  com.bea.wlpi.common.BusinessCalendarInfo calendarInfo
) throws java.rmi.RemoteException,
  com.bea.wlpi.common.WorkflowException
```

`addBusinessCalendar()` メソッドのパラメータを次の表に示します。パラメータには値の指定が必要です。

表 12-1 addBusinessCalendar() メソッドのパラメータ

パラメータ	説明	有効な値
`calendarInfo`	新しいビジネス カレンダー情報	`BusinessCalendarInfo` オブジェクト。 `BusinessCalendarInfo` オブジェクトを定義する方法については、B-2 ページの「BusinessCalendarInfo オブジェクト」 を参照。

メソッドは、新しいビジネス カレンダーの ID を返します。

たとえば、以下のコードでは、指定された `calendarInfo` オブジェクトのコンテンツに基づいてビジネス カレンダーが追加されます。このコード例では、`admin` は `Admin` EJB への `EJBObject` 参照を表します。

```
admin.addBusinessCalendar(calendarInfo);
```

`addBusinessCalendar()` メソッドの詳細については、Javadoc の `com.bea.wlpi.server.admin.Admin` を参照してください。

ビジネス カレンダーの取得

現在定義されているすべてのビジネス カレンダーのコンテンツを取得するには、次の com.bea.wlpi.server.admin.Admin メソッドを使用します。

```
public java.util.List getAllBusinessCalendars(
  boolean includeDefinition
) throws java.rmi.RemoteException,
  com.bea.wlpi.common.WorkflowException
```

getAllBusinessCalendars() メソッドのパラメータを次の表に示します。パラメータには値の指定が必要です。

表 12-2 getAllBusinessCalendars() メソッドのパラメータ

パラメータ	説明	有効な値
includeDefintion	XML カレンダーの仕様を含むかどうかを指定するブール フラグ	true (XML を含む) または false (XML を含まない)。

このメソッドは、com.bea.wlpi.common.BusinessCalendarInfo オブジェクトのリストを返します。各ビジネス カレンダーの情報にアクセスするには、B-2 ページの「BusinessCalendarInfo オブジェクト」に記載の BusinessCalendarInfo オブジェクト メソッドを使用します。

たとえば、以下のコードにより、ビジネス カレンダー情報が取得されて、calendars リスト オブジェクトに保存されます。このコード例では、admin は Admin EJB への EJBObject 参照を表します。

```
List calendars = admin.getAllBusinessCalendars();
```

getAllBusinessCalendars() メソッドの詳細については、Javadoc の com.bea.wlpi.server.admin.Admin を参照してください。

ビジネス カレンダー定義の取得

ビジネス カレンダー定義を取得するには、次の `com.bea.wlpi.server.admin.Admin` メソッドを使用します。

```
public java.lang.String getBusinessCalendarDefinition(
  java.lang.String calendarId
) throws java.rmi.RemoteException,
  com.bea.wlpi.common.WorkflowException
```

`getBusinessCalendarDefinition()` メソッドのパラメータを次の表に示します。パラメータには値の指定が必要です。

表 12-3 getBusinessCalendarDefinition() メソッドのパラメータ

パラメータ	説明	有効な値
`calendarId`	定義の検索を行うビジネス カレンダーの ID	有効なビジネス カレンダー ID を指定する文字列。 カレンダー ID を取得するには、以下の `com.bea.wlpi.common.BusinessCalendarInfo` メソッドを使用する。 `public final java.lang.String getId()` `BusinessCalendarInfo` オブジェクトを取得する方法については、12-3 ページの「ビジネス カレンダーの取得」を参照。`BusinessCalendarInfo` オブジェクトで選択可能なメソッドの詳細については、B-2 ページの「BusinessCalendarInfo オブジェクト」を参照。

このメソッドは、A-11 ページの「ビジネス カレンダー DTD」に準拠するビジネス カレンダー定義を含む XML ファイルを返します。

たとえば、以下のコードにより、単一のビジネス カレンダー定義の情報が取得されて、`calendar` 文字列に保存されます。このコード例では、`admin` は `Admin` EJB への `EJBObject` 参照を表します。

```
String calendar = admin.getBusinessCalendarDefinition();
```

getBusinessCalendarDefinition() メソッドの詳細については、Javadoc の com.bea.wlpi.server.admin.Admin を参照してください。

ビジネス カレンダーの更新

ビジネス カレンダーを更新するには、次の com.bea.wlpi.server.admin.Admin メソッドを使用します。

```
public void updateBusinessCalendar(
  com.bea.wlpi.common.BusinessCalendarInfo calendarInfo
) throws java.rmi.RemoteException,
  com.bea.wlpi.common.WorkflowException
```

updateBusinessCalendar() メソッドのパラメータを次の表に示します。パラメータには値の指定が必要です。

表 12-4 updateBusinessCalendar() メソッドのパラメータ

パラメータ	説明	有効な値
calendarInfo	更新するカレンダー情報	BusinessCalendarInfo オブジェクト。 BusinessCalendarInfo オブジェクトを定義する方法については、B-2 ページの「BusinessCalendarInfo オブジェクト」を参照。

たとえば、以下のコードでは、指定された calendarInfo オブジェクトのコンテンツに基づいてビジネス カレンダーが更新されます。このコード例では、admin は Admin EJB への EJBObject 参照を表します。

```
admin.updateBusinessCalendar(calendarInfo);
```

updateBusinessCalendar() メソッドの詳細については、Javadoc の com.bea.wlpi.server.admin.Admin を参照してください。

ビジネス カレンダーの削除

ビジネス カレンダーを削除するには、次の `com.bea.wlpi.server.admin.Admin` メソッドを使用します。

```
public void deleteBusinessCalendar(
  java.lang.String calendarId
) throws java.rmi.RemoteException,
  com.bea.wlpi.common.WorkflowException
```

`deleteBusinessCalendar()` メソッドのパラメータを次の表に示します。パラメータには値の指定が必要です。

表 12-5 deleteBusinessCalendar() メソッドのパラメータ

パラメータ	説明	有効な値
`calendarId`	削除するビジネス カレンダーの ID	有効なビジネス カレンダー ID を指定する文字列。 カレンダー ID を取得するには、以下の `com.bea.wlpi.common.BusinessCalendarInfo` メソッドを使用する。 `public final java.lang.String getId()` `BusinessCalendarInfo` オブジェクトを取得する方法については、12-3 ページの「ビジネス カレンダーの取得」を参照。 `BusinessCalendarInfo` オブジェクトで選択可能なメソッドの詳細については、B-2 ページの「BusinessCalendarInfo オブジェクト」を参照。

たとえば、次のコードでは、指定されたビジネス カレンダーが削除されます。このコード例では、`admin` は `Admin` EJB への `EJBObject` 参照を表します。

```
admin.deleteBusinessCalendar(calendarInfo);
```

deleteBusinessCalendar() メソッドの詳細については、Javadoc の com.bea.wlpi.server.admin.Admin を参照してください。

ビジネス カレンダーのコンフィグレーション例

この節では、コマンドライン管理例から抜粋して、ビジネス カレンダーのコンフィグレーション方法を示します。

注意： コマンドライン管理サンプルの詳細については、1-23 ページの「コマンドライン管理サンプル」を参照してください。

このサンプルでは、ユーザと通信するための入力ストリームが定義されており、ユーザは以下のアクションのいずれかを指定するよう求められます。

- ビジネス カレンダーの追加
- ビジネス カレンダーの削除
- ビジネス カレンダー定義の取得
- ビジネス カレンダーの取得
- ビジネス カレンダーの更新

重要なコード行は、**太字**で強調されています。このコード例では、admin は Admin EJB への EJBObject 参照を表します。

```
/* ユーザとの通信のための入力ストリームを作成する */
stdIn = new BufferedReader( new InputStreamReader( System.in ) );

/* ツール タイトルを表示する */
   System.out.print( "\n---  Command Line Administration v1.1  ---" );

/* メイン メニューを表示してユーザと交信する */
while( true ) {
/* メニューを表示する */
   System.out.println( "\n---              Main Menu            ---" );
   System.out.println( "\nEnter choice:" );
   System.out.println( "1) Organizations" );
   System.out.println( "2) Roles" );
   System.out.println( "3) Users" );
   System.out.println( "4) Security Realm" );
   System.out.println( "5) Business Operations" );
   System.out.println( "6) Event Keys" );
   System.out.println( "7) Business Calendars" );
```

```
   System.out.println( "8) EJB Catalog" );
   System.out.println( "9) Server Properties" );
   System.out.println( "Q) Quit" );
   System.out.print( ">> " );
      .
      .
      .
/**
 * WLPI ビジネス カレンダーと関連する、Admin インタフェースで選択可能な公開 API メソッドを示す目的で
 * 必要な情報をすべて取得するために
 * ユーザと交信するメソッド
 */
public static void mngBusinessCalendars() {\
   boolean isGetDefinition;
   BusinessCalendarInfo calendarInfo;
   List calendarList;
   String answer;
   String calendarDefinition;
   String calendarId;
   String calendarName;
   String calendarTimezone;// XML Document

   /* ユーザとの通信のための入力ストリームを作成する */
   BufferedReader stdIn = new BufferedReader(
      new InputStreamReader( System.in ) );

   try {
      /* メニューを表示してユーザと交信する */
      while( true ) {
         /* メニューを表示する */
         System.out.println( "\n\n---        Business Calendars        ---" );
         System.out.println( "\nEnter choice:" );
         System.out.println( "1) Add a Business Calendar" );
         System.out.println( "2) Delete a Business Calendar" );
         System.out.println( "3) List a Business Calendar Definition" );
         System.out.println( "4) List all Business Calendars" );
         System.out.println( "5) Update a Business Calendar" );
         System.out.println( "B) Back to previous menu" );
         System.out.println( "Q) Quit" );
         System.out.print( ">> " );
         /* ユーザの選択を取得する */
         String line = stdIn.readLine();
         /* ユーザが選択を行わないで〔Enter〕を押したか ? */
         if( line.equals( "" ) )
            continue;
         else if( line.length() > 1 ) {
            System.out.println( "*** Invalid selection" );
```

```
    continue;


}
/* 大文字および文字へのコンバート */
char choice = line.toUpperCase().charAt( 0 );
/* ユーザの選択を処理する */
switch( choice ) {
.
.
.
```

ビジネス カレンダーの追加

ビジネス カレンダーの追加方法を示します。

```
/* ビジネス カレンダーを追加する *
case '1' :
   /* 新しいビジネス カレンダーを追加するため
        カレンダー ID はない。サーバが割り当てて
      新しい ID を戻す。値を指定しない場合には
      ワードを「null」に設定する必要がある ( ヌル文字列ではない ) */
      calendarId = new String( "null" );

   /* 追加する新しいカレンダーのカレンダー名を取得する */
  if( ( calendarName = askQuestion( "\nEnter new Calendar Name" ) )
     == null ) {
      /* ユーザによる操作の取り消し */
      System.out.println( "*** Cancelled" );
      break;
   }

  /* 追加する新しいカレンダーの Calendar TimeZone を取得する */
 if( ( calendarTimezone = askQuestion( "Enter Time Zone" ) ) == null ) {
      /* ユーザによる操作の取り消し */
      System.out.println( "*** Cancelled" );
      break;
   }

   /* ビジネス カレンダー定義は
     BusinessCalendar.dtd ドキュメント タイプ(WLPI javadocs を参照)に準拠した
      XML ドキュメントである。
      ビジネス カレンダーを定義する
```

```
      すべての情報が含まれる */
   /* XML ドキュメントを構築する必要がある */
   calendarDefinition = createCalendarDefinition( calendarId,
      calendarName, calendarTimezone );

   /* BusinessCalendarInfo オブジェクトを作成する */
   calendarInfo = new BusinessCalendarInfo( calendarId, calendarName,
      calendarTimezone, calendarDefinition );

   try {
      /* WLPI 公開 API メソッド */
      /* 新しいビジネス カレンダーを作成する */
      calendarId = admin.addBusinessCalendar( calendarInfo );

      /* 正常終了（例外の発生なし）*/
      System.out.println( "- Business Calendar added (ID=" +
         calendarId + ")" );
   }
   catch( Exception e ) {
      System.out.println( "*** Unable to add business calendar\n" );
      System.err.println( e );
   }
   break;
   .
   .
   .
```

ビジネス カレンダーの削除

ビジネス カレンダーの削除方法を示します。

```
/* ビジネス カレンダーを削除する */
case '2' :
   /* 削除するカレンダーのカレンダー ID を取得する */
   if( ( calendarId = askQuestion( "\nEnter Calendar ID to delete" ) )
      == null ) {
     /* ユーザによる操作の取り消し */
      System.out.println( "*** Cancelled" );
      break;
   }

   try {
      /* WLPI 公開 API メソッド */
```

```
      /* ビジネス カレンダーを削除する */
      admin.deleteBusinessCalendar( calendarId );

      /* 正常終了（例外の発生なし）*/
      System.out.println( "- Deleted" );
   }
   catch( Exception e ) {
      System.out.println( "*** Unable to delete Business Calendar" );
      System.err.println( e );
   }
   break;
   .
   .
   .
```

ビジネス カレンダー定義の取得

、ビジネス カレンダー定義の取得方法を示します。

```
/* ビジネス カレンダー定義をリストアップする */
case '3' :
  /* 削除するカレンダーのカレンダー ID を取得する */
   if( ( calendarId = askQuestion( "\nEnter Calendar ID" ) ) == null ) {
      /* ユーザによる操作の取り消し */
      System.out.println( "*** Cancelled" );
      break;
   }

   try {
      /* WLPI 公開 API メソッド */
      /* ビジネス カレンダーの定義を取り出す。*/
      calendarDefinition = admin.getBusinessCalendarDefinition(
         calendarId );

   /* 正常終了（例外の発生なし）*/
   System.out.println( "- Business Calendar definition:\n" +
      calendarDefinition );
}
catch( Exception e ) {
   System.out.println( "*** Unable to retrieve Business " +
      "Calendar definition\n" );
   System.err.println( e );
}
break;
```

.
.
.

ビジネス カレンダーの取得

ビジネス カレンダーの取得方法を示します。

```
/* すべてのビジネス カレンダーをリストアップする */
case '4' :
   /* カレンダー定義の表示が必要かどうかをユーザに求める */
  if( ( answer = askQuestion( "\nList Definition (y/n)?" ) ) == null ) {
      /* ユーザによる操作の取り消し */
      System.out.println( "*** Cancelled" );
      break;
   }

   /* 応答を解析する */
   isGetDefinition = ( answer.equals( "y" ) || answer.equals( "Y" ) );

   try {
      /* WLPI 公開 API メソッド */
      /* すべてのビジネス カレンダーを取り出す */
      //Bug1 calendarList = admin.getAllBusinessCalendars(
      //isGetDefinition );
      calendarList = admin.getAllBusinessCalendars( false );

      /* 定義されているカレンダーはあるか ? */
      if( calendarList.size() == 0 )
         System.out.println( "\nNo Business Calendar defined" );
      else
         System.out.println( "\nDefined Business Calendars:" );

      /* リストを処理してカレンダーと属性を表示する */
      for( int i = 0; i < calendarList.size(); i++ ) {
         /* リストから要素を取り出す */
         calendarInfo = ( BusinessCalendarInfo )calendarList.get( i );

         /* オーガニゼーション ID を取り出して表示する */
         System.out.println( "- ID:         " + calendarInfo.getId() );

         /* カレンダー名を取り出して表示する */
        System.out.println( "  Name:       " + calendarInfo.getName() );

         /* 時間帯を取り出して表示する */
       System.out.println( "  Time Zone: " + calendarInfo.getTimeZone() );

         /* ビジネス カレンダー定義を表示するか ? */
         if( isGetDefinition ) {
```

```
            /* カレンダー定義を取り出して表示する */
            //Bug 1 説明：      無効文字を含む戻された文字列に
            // getXML がプレフィックスされる */
            //Bug1    System.out.println( "  Business Calendar
            //definition:\n" +
            //Bug1    calendarInfo.getXML() );
            /* カレンダー定義を取り出すために回避する */
            calendarDefinition = admin.getBusinessCalendarDefinition(
               calendarInfo.getId() );
            /* カレンダー 定義を表示する */
            System.out.println( "  Business Calendar definition:\n" +
              calendarDefinition );
        }
      }
    }
    catch( Exception e ) {
       System.out.println( "*** Unable to retrieve Business Calendars" );
        System.err.println( e );
   }
    break;
    .
    .
    .
```

ビジネス カレンダーの更新

ビジネス カレンダーの更新方法を示します。

```
/* ビジネス カレンダーを更新する */
case '5' :
   /* 更新するカレンダーのカレンダー ID を取得する */
   if( ( calendarId = askQuestion( "\nEnter Calendar ID to update" ) )
      == null ) {
      /* ユーザによる操作の取り消し */
      System.out.println( "*** Cancelled" );
      break;
   }

   try {
      /* WLPI 公開 API メソッド */
      /* 更新するビジネス カレンダーを取り出すために、すべてのビジネス カレンダーを
         検索する */
      calendarList = admin.getAllBusinessCalendars( false );
```

```
   /* 定義されているカレンダーはあるか ? */
   if( calendarList.size() == 0 ) {
      System.out.println( "\nNo Business Calendar defined" );
      break;
   }

   boolean isNotDefined = true;
   calendarInfo = null;

   /* 更新するカレンダー情報を取り出すためのリストを処理する */
   for( int i = 0; i < calendarList.size(); i++ ) {
      /* リストから要素を取り出す */
      calendarInfo = ( BusinessCalendarInfo )calendarList.get( i );

      /* オーガニゼーション ID を取り出して表示する */
      if( calendarId.equals( calendarInfo.getId() ) ) {
         isNotDefined = false;
         break;
      }
   }

/* カレンダーは定義されているか ? */
   if( isNotDefined ) {
      System.out.println( "*** This Business Calendar is
         not defined" );
      break;
   }

   System.out.println( "\nUpdating Business Calendar" );

   /* カレンダー名を取り出して表示する */
   System.out.println( "- The current Name is: " +
      calendarInfo.getName() );

  /* 新しいカレンダー名を取得する */
 if( ( calendarName = askQuestion( "  Enter a new Name " ) ) != null )
      /* ユーザが値を入力したので値を更新する */
      /* WLPI 公開 API メソッド */
      calendarInfo.setName( calendarName );

   /* 時間帯を取り出して表示する */
   System.out.println( "- The current Time Zone is: " +
      calendarInfo.getTimeZone() );

   /* 追加する新しいカレンダーの Calendar TimeZone を取得する */
if( ( calendarTimezone = askQuestion( "  Enter a new Time Zone " ) )
      != null )
```

```
      /* ユーザが値を入力したので値を更新する */
      /* WLPI 公開 API メソッド */
      calendarInfo.setTimeZone( calendarTimezone );

   /* 新しいビジネス カレンダー定義を作成する */


   calendarDefinition = updateCalendarDefinition( calendarId,
      calendarInfo.getName(), calendarInfo.getTimeZone() );

   /* ビジネス カレンダー定義を更新する */
   calendarInfo.setXML( calendarDefinition );

   /* WLPI 公開 API メソッド */
   admin.updateBusinessCalendar( calendarInfo );

   /* 正常終了（例外の発生なし）*/
   System.out.println( "\nUpdated" );
}
catch( Exception e ) {
   System.out.println( "*** Unable to update Business Calendar" );
   System.err.println( e );
}
break;
.
.
.
```

Part III　設計

13 ワークフロー テンプレートの作成および管理

ワークフロー テンプレートは、任意の数のワークフロー テンプレート定義を含むコンテナです。

この章では、ワークフロー テンプレートの作成および管理方法について説明します。内容は以下のとおりです。

- テンプレートの作成
- テンプレートの取得
- オーガニゼーションのテンプレートの取得
- テンプレート オーガニゼーションの取得
- テンプレート オーガニゼーションの設定
- D テンプレートの削除
- テンプレートの管理例

この章に記載するメソッドの詳細については、Javadoc の `com.bea.wlpi.server.admin.Admin` を参照してください。WebLogic Integration Studio を使用してワークフロー テンプレートを管理する方法については、『*WebLogic Integration Studio ユーザーズ ガイド*』の「ワークフロー テンプレートの定義」を参照してください。

テンプレートの作成

ワークフロー テンプレートを作成するには、以下のいずれかの com.bea.wlpi.server.admin.Admin メソッドを使用します。

メソッド 1

```
public java.lang.String createTemplate(
  java.lang.String name,
  java.lang.String xml,
  java.util.Collection orgs
) throws java.rmi.RemoteException,
  com.bea.wlpi.common.WorkflowException
```

メソッド 2

```
public java.lang.String createTemplate(
  java.lang.String name,
  java.lang.String xml,
  java.util.Collection orgs,
  java.lang.Object transactionId
) throws java.rmi.RemoteException,
  com.bea.wlpi.common.WorkflowException
```

第 1 のメソッドは、非クラスタ化環境で使用できます。クラスタ化環境では第 2 のメソッドを使用することをお勧めします。この場合、トランザクションがコミットされた後、またはサーバ クラッシュやフェイルオーバーが発生した後に、メソッドが再発行されないよう、指定された ID を使用してメソッドの実行状態が追跡されます。

createTemplate() メソッドのパラメータを次の表に示します。パラメータには値を指定する必要があります。

表 13-1 createTemplate() メソッドのパラメータ

パラメータ	説明	有効な値
name	テンプレート名	非ヌル文字列。
xml	プラグイン データを定義し、A-53 ページの「テンプレート定義 DTD」に記載の Template Definition DTD に準拠する XML ドキュメント名 BPM プラグインのプログラミングの詳細については、『*WebLogic Integration BPM プラグイン プログラミング ガイド*』を参照。	非ヌル文字列。

表 13-1 createTemplate() メソッドのパラメータ (続き)

パラメータ	説明	有効な値
orgs	テンプレートにアクセスできるオーガニゼーション ID のコレクション	有効なオーガニゼーション ID を含む文字列のコレクション。オーガニゼーション ID のリストの取得については、9-9 ページの「すべてのオーガニゼーション名を取得する」 を参照。
transactionId	トランザクションの ID **注意 :** このパラメータは、クラスタ化環境においてのみ必要です。	ユニークなトランザクション ID を指定するオブジェクト。 ユニークなトランザクション ID を生成するには、次のコンストラクタを使用して新しい `com.bea.wlpi.client.common.GUID` オブジェクトを作成する。 `GUID transactionId = new GUID();` GUID クラスの詳細については、Javadoc の `com.bea.wlpi.client.common.GUID` を参照。

各メソッドは、新しいテンプレートの ID を返します。

たとえば、次のコードは指定されたオーガニゼーションのコレクションである `orgIds` からアクセスできる `Order Processing` という名の新しいテンプレートを作成します。このコード例では、`admin` は `Admin` EJB への `EJBObject` 参照を表します。

```
String id = admin.createTemplate(“Order Processing”, null, orgIds);
```

`createTemplate()` メソッドの詳細については、Javadoc の `com.bea.wlpi.server.admin.Admin` を参照してください。

テンプレートの取得

ワークフロー テンプレートを取得するには、次の `com.bea.wlpi.server.admin.Admin` メソッドを使用します。

```
public com.bea.wlpi.common.TemplateInfo getTemplate(
  java.lang.String templateId,
  boolean byName
) throws java.rmi.RemoteException,
  com.bea.wlpi.common.WorkflowException
```

getTemplate() メソッドのパラメータを次の表に示します。パラメータには値を指定する必要があります。

表 13-2 getTemplate() メソッドのパラメータ

パラメータ	説明	有効な値
templateId	検索するテンプレートの ID または名前	有効なテンプレート ID または名前を指定する文字列。
byName	templateId パラメータに指定された値がテンプレート名 (true) または ID (false) であるかを指定するブール フラグ	true (名前) または false (名前)。

各メソッドにより、テンプレートに対応する com.bea.wlpi.common.TemplateInfo オブジェクトが返されます。そのテンプレート情報にアクセスするには、B-27 ページの「TemplateInfo オブジェクト」に記載の TemplateInfo オブジェクト メソッドを使用します。

たとえば、次のコードは Order Processing という名前のテンプレートを取得します（*byName* パラメータが true に設定されていることに注意）。このコード例では、admin は Admin EJB への EJBObject 参照を表します。

```
TemplateInfo template = admin.getTemplate(
“Order Processing”, true);
```

getTemplate() メソッドの詳細については、Javadoc の com.bea.wlpi.server.admin.Admin を参照してください。

オーガニゼーションのテンプレートの取得

オーガニゼーションのワークフロー テンプレート リストを取得するには、次の com.bea.wlpi.server.admin.Admin メソッドを使用します。

```
public java.util.List getTemplates(
  java.lang.String orgId
) throws java.rmi.RemoteException,
  com.bea.wlpi.common.WorkflowException
```

getTemplates() メソッドのパラメータを次の表に示します。パラメータには値を指定する必要があります。

表 13-3 getTemplates() メソッドのパラメータ

パラメータ	説明	有効な値
orgId	テンプレートを取得するオーガニゼーションの ID	有効なオーガニゼーション ID を指定する文字列。 オーガニゼーション ID のリストの取得については、9-9 ページの「すべてのオーガニゼーション名を取得する」を参照。

このメソッドは、com.bea.wlpi.common.TemplateInfo オブジェクトのリストを返します。各テンプレートについての情報にアクセスするには、B-27 ページの「TemplateInfo オブジェクト」に記載の TemplateInfo オブジェクト メソッドを使用します。

たとえば、次のコードでは ORG1 オーガニゼーションと関連するすべてのテンプレートが取得されます。このコード例では、admin は Admin EJB への EJBObject 参照を表します。

```
List templates = admin.getTemplates(“ORG1”);
```

getTemplates() メソッドの詳細については、Javadoc の com.bea.wlpi.server.admin.Admin を参照してください。

テンプレート オーガニゼーションの取得

テンプレートにアクセスできるオーガニゼーションを取得するには、次の com.bea.wlpi.server.admin.Admin メソッドを使用します。

```
public List getTemplateOrgs(
  java.lang.String templateId
```

```
) throws java.rmi.RemoteException,
  com.bea.wlpi.common.WorkflowException
```

`getTemplateOrgs()` メソッドのパラメータを次の表に示します。パラメータには値を指定する必要があります。

表 13-4 getTemplateOrgs() メソッドのパラメータ

パラメータ	説明	有効な値
templateId	オーガニゼーションの検索を行うテンプレートの ID	有効なテンプレート ID を指定する文字列。 テンプレート ID を取得するには、次の `com.bea.wlpi.common.TemplateInfo` メソッドを使用する。 `public final java.lang.String getId()` `TemplateInfo` オブジェクトを取得する方法については、13-4 ページの「オーガニゼーションのテンプレートの取得」を参照。`TemplateInfo` オブジェクトで選択可能なメソッドの詳細については、B-27 ページの「TemplateInfo オブジェクト」を参照。

各メソッドは、オーガニゼーション ID のリストを返します。

たとえば、次のコードでは、テンプレートにアクセスできるオーガニゼーションのリストが取得され、その結果が `orgs` に割り当てられます。このコード例では、`admin` は `Admin` EJB への `EJBObject` 参照を表します。

```
List orgs = admin.getTemplateOrgs(template.getId());
```

テンプレート ID は、`com.bea.wlpi.common.TemplateInfo` オブジェクトである `template` と関連付けられたメソッドを使用して取得されます。`template` オブジェクトは、13-4 ページの「オーガニゼーションのテンプレートの取得」に記載のメソッドを使用して取得されます。

`getTemplateOrgs()` メソッドの詳細については、Javadoc の `com.bea.wlpi.server.admin.Admin` を参照してください。

テンプレート オーガニゼーションの設定

テンプレートにアクセスできるオーガニゼーションを設定するには、次の com.bea.wlpi.server.admin.Admin メソッドを使用します。

```
public void setTemplateOrgs(
  java.lang.String templateId,
  java.util.Collection orgs
) throws java.rmi.RemoteException,
  com.bea.wlpi.common.WorkflowException
```

setTemplateOrgs() メソッドのパラメータを次の表に示します。パラメータには値の指定の指定が必要です

表 13-5 setTemplateOrganizations() メソッドのパラメータ

パラメータ	説明	有効な値
templateId	オーガニゼーションの設定を行うテンプレートの ID	有効なテンプレート ID を指定する文字列。 テンプレート ID を取得するには、次の com.bea.wlpi.common.TemplateInfo メソッドを使用する。 public final java.lang.String getId() TemplateInfo オブジェクトを取得する方法については、13-4 ページの「オーガニゼーションのテンプレートの取得」を参照。TemplateInfo オブジェクトで選択可能なメソッドの詳細については、B-27 ページの「TemplateInfo オブジェクト」を参照。
orgs	テンプレートにアクセスできるオーガニゼーション ID のコレクション	有効なオーガニゼーション ID を含む文字列のコレクション。オーガニゼーション ID のリストの取得については、9-9 ページの「すべてのオーガニゼーション名を取得する」を参照。

たとえば、次のコードでは、コレクション organizations を使用するテンプレートにアクセスできるオーガニゼーションが設定されます。このコード例では、admin は Admin EJB への EJBObject 参照を表します。

```
List orgs = admin.setTemplateOrgs(
  template.getId(), template.getName(), organizations);
```

テンプレート ID および名前は、com.bea.wlpi.common.TemplateInfo オブジェクトである template と関連付けられたメソッドを使用して取得されます。template オブジェクトは、13-4 ページの「オーガニゼーションのテンプレートの取得」に記載のメソッドを使用して取得されます。

setTemplateOrgs() メソッドの詳細については、Javadoc の com.bea.wlpi.server.admin.Admin を参照してください。

テンプレートの更新

ワークフロー テンプレートを更新するには、次の com.bea.wlpi.server.admin.Admin メソッドを使用します。

```
public void updateTemplate(
  com.bea.wlpi.common.TemplateInfo info
) throws java.rmi.RemoteException,
  com.bea.wlpi.common.WorkflowException
```

updateTemplate() メソッドのパラメータを次の表に示します。パラメータには値を指定する必要があります。

表 13-6 updateTemplate() メソッドのパラメータ

パラメータ	説明	有効な値
info	テンプレート名、ID、および新情報	name および ID フィールドを含む TemplateInfo オブジェクトは、更新する既存テンプレートを識別するために設定され、残りのフィールドは新しい情報に設定される。

たとえば、次のコードでは、既存テンプレートが TemplateInfo オブジェクト、info で定義されたとおりに更新されます。このコード例では、admin は Admin EJB への EJBObject 参照を表します。

```
admin.updateTemplate(info);
```

updateTemplate() メソッドの詳細については、Javadoc の com.bea.wlpi.server.admin.Admin を参照してください。

D テンプレートの削除

テンプレートを削除するには、次の com.bea.wlpi.server.admin.Admin メソッドを使用します。

```
public void deleteTemplate(
  java.lang.String templateId
) throws java.rmi.RemoteException,
  com.bea.wlpi.common.WorkflowException
```

deleteTemplate() メソッドのパラメータを次の表に示します。パラメータには値を指定する必要があります。

表 13-7 deleteTemplate() メソッドのパラメータ

パラメータ	説明	有効な値
templateId	削除するテンプレートの ID	有効なテンプレート ID を指定する文字列。 テンプレート ID を取得するには、次の com.bea.wlpi.common.TemplateInfo メソッドを使用する。 public final java.lang.String getId() TemplateInfo オブジェクトを取得する方法については、13-4 ページの「オーガニゼーションのテンプレートの取得」を参照。TemplateInfo オブジェクトで選択可能なメソッドの詳細については、B-27 ページの「TemplateInfo オブジェクト」を参照。

たとえば、次のコードでは、指定されたテンプレートが削除されます。このコード例では、admin は Admin EJB への EJBObject 参照を表します。

```
admin.deleteTemplate(template.getId());
```

テンプレート ID は、`com.bea.wlpi.common.TemplateInfo` オブジェクトである `template` と関連付けられたメソッドを使用して取得されます。`template` オブジェクトは、13-4 ページの「オーガニゼーションのテンプレートの取得」に記載のメソッドを使用して取得されます。

`deleteTemplate()` メソッドの詳細については、Javadoc の `com.bea.wlpi.server.admin.Admin` を参照してください。

テンプレートの管理例

この節では、コマンドライン Studio 例から抜粋して、テンプレートの管理方法を示します。

注意： コマンドライン Studio 例の詳細については、1-24 ページの「コマンドライン Studio サンプル」を参照してください。

このサンプルでは、ユーザと通信するための入力ストリームが定義されており、ユーザは以下のアクションのいずれかを指定するよう求められます。

- テンプレートの作成
- テンプレートの削除
- オーガニゼーションのテンプレートの取得

重要なコード行は、**太字**で強調されています。このコード例では、`admin` は `Admin` EJB への `EJBObject` 参照を表します。

```
/* ユーザとの通信のための入力ストリームを作成する */
stdIn = new BufferedReader( new InputStreamReader( System.in ) );

/* ツール タイトルを表示する */
System.out.print( "\n---      Command Line Studio v1.0      ---" );

/* メイン メニューを表示してユーザと交信する */
while( true ) {
/* メニューを表示する */
   System.out.println( "\n---             Main Menu            ---" );
   System.out.println( "\nEnter choice:" );
   System.out.println( "1) Templates" );
   System.out.println( "2) Task Reroutes" );
   System.out.println( "Q) Quit" );
```

```
   System.out.print( ">> " );
      .
      .
      .
/**
 * WLPI ワークフロー テンプレートと関連する、Admin インタフェースで選択可能な公開 API メソッドを示す目的で
 * 必要な情報をすべて取得するために
 * ユーザと交信するメソッド
 */
public static void mngTemplates() {
   ArrayList orgsList = new ArrayList();
   String answer;
   String orgId;
   String templateId;
   String templateName;

   /* ユーザとの通信のための入力ストリームを作成する */
   BufferedReader stdIn = new BufferedReader( new InputStreamReader(
      System.in ) );

   try {
      /* メニューを表示してユーザと交信する */
      while( true ) {
         /* メニューを表示する */
         System.out.println( "\n\n---       Workflow Templates         ---" );
         System.out.println( "\nEnter choice:" );
         System.out.println( "1) Create a Template" );
         System.out.println( "2) Delete a Template" );
         System.out.println( "3) List Templates for an Organization" );
         System.out.println( "B) Back to previous menu" );
         System.out.println( "Q) Quit" );
         System.out.print( ">> " );

         /* ユーザの選択を取得する */
         String line = stdIn.readLine();

         /* ユーザが選択を行わないで〔Enter〕を押したか ? */
         if( line.equals( "" ) )
            continue;
         /* ユーザが複数の文字を入力したか ? */
         else if( line.length() > 1 ) {
            System.out.println( "*** Invalid selection" );
            continue;
         }

         /* 大文字および文字へのコンバート */
         char choice = line.toUpperCase().charAt( 0 );
```

```
/* ユーザの選択を処理する */
switch( choice ) {
```

テンプレートの作成

テンプレートの作成方法を示します。

```
/* テンプレートを作成する */
case '1' :
   /* テンプレート名を取得する */
   if( ( templateName = askQuestion( "\nEnter Template Name" ) )
      == null ) {
      /* ユーザによる操作の取り消し */
      System.out.println( "*** Cancelled" );
      break;
   }

   System.out.println( "\nDefining organizations where
      the template is accessible" );
   boolean isEnterMore = true;
   boolean isCancelled = false;

   /* オーガニゼーション ID リストを取得するループ */
   while( isEnterMore ) {
     /* アクティブに設定するオーガニゼーションのオーガニゼーション ID を取得する */
      if( ( orgId = askQuestion( "Enter an Organization ID" ) )
         == null ) {
         /* ユーザによる操作の取り消し */
         System.out.println( "*** Cancelled" );
         isCancelled = true;
         break;
      }

      orgsList.add( orgId );

      /* さらにオーガニゼーション ID を入力するためにループを継続するか ? */
      if( ( answer = askQuestion( "Enter more (y/n)?" ) ) == null ) {
         /* ユーザによる操作の取り消し */
         System.out.println( "*** Cancelled" );
         isCancelled = true;
         break;
      }

      /* 応答を評価する */
```

```
      isEnterMore = ( answer.equals( "y" ) || answer.equals( "Y" ) );
   }

   /* ユーザが操作をキャンセルしたか ? */
   if( isCancelled )
      break;

   try {
      /* WLPI 公開 API メソッド */
      /* 新しいテンプレートを作成する */
     templateId = admin.createTemplate( templateName, null, orgsList );

      /* 正常終了（例外の発生なし）*/
      System.out.println( "- Created (template Id = " +
         templateId + ")" );
   }
   catch( Exception e ) {
      System.out.println( "*** Unable to create Template" );
      System.err.println( e );
   }
   break;
      .
      .
      .
```

テンプレートの削除

テンプレートの削除方法を示します。

```
/* テンプレートを削除する */
case '2' :
   /* 削除するテンプレートのテンプレート ID を取得する */
   if( ( templateId = askQuestion( "\nEnter Template ID
      to delete" ) ) == null ) {
      /* ユーザによる操作の取り消し */
      System.out.println( "*** Cancelled" );
      break;
   }

   try {
      /* WLPI 公開 API メソッド */
      /* テンプレートを削除する */
      admin.deleteTemplate( templateId );
```

```
            /* 正常終了（例外の発生なし）*/
            System.out.println( "- Deleted" );
         }
         catch( Exception e ) {
            System.out.println( "*** Unable to delete Template" );
            System.err.println( e );
         }
         break;

            .
            .
            .
```

オーガニゼーションのテンプレートの取得

オーガニゼーションのテンプレートの取得方法を示します。

```
/* オーガニゼーションのテンプレートをリストアップする */
case '3' :
   /* クエリするオーガニゼーションのオーガニゼーション ID を取得する */
   if( ( orgId = askQuestion( "\nEnter Organization ID" ) ) == null ) {
      /* ユーザによる操作の取り消し */
      System.out.println( "*** Cancelled" );
      break;
   }

   /* WLPI 公開 API メソッド */
   /* このオーガニゼーションで定義されているすべてのテンプレートを取り出す */
   /* 注意 : 発生した例外を取り込むコードを
    * 追加するとよい */
   List templateList = admin.getTemplates( orgId );

   /* 定義されているテンプレートはあるか ? */
   if( templateList.size() == 0 )
      System.out.println( "\nNo template defined" );
   else
      System.out.println( "\nDefined Templates:" );

   /* リストを処理して、テンプレートを表示する */
   for( int i = 0; i < templateList.size(); i++ ) {
      /* リストから要素を取り出す */
      TemplateInfo templateInfo =
         ( TemplateInfo )templateList.get( i );
```

```
   /* テンプレート ID を取り出して表示する */
   System.out.println( "- Template ID: " + templateInfo.getId() );

   /* テンプレート名を取り出して表示する */
   System.out.println( "  Name:        " +
      templateInfo.getName() + "\n" );
}
break;
   .
   .
   .
```

14 ワークフロー テンプレート定義の作成および管理

ワークフロー テンプレート定義は、ビジネス プロセスのオペレーションを指定します。各ワークフロー テンプレートに対して、任意の数の定義を作成できます。

この章では、ワークフロー テンプレート定義の作成および管理方法について説明します。内容は以下のとおりです。

- テンプレート定義の作成
- テンプレート定義情報の取得
- テンプレートの定義の取得
- テンプレート定義のコンテンツの取得
- テンプレート定義のコンテンツの設定
- テンプレート定義オーナの取得
- 呼び出し可能なワークフローの取得
- 呼び出し可能なワークフローの検索
- テンプレート定義のロックおよびロック解除
- テンプレート定義の削除

この章に記載するメソッドの詳細については、Javadoc の `com.bea.wlpi.server.admin.Admin` を参照してください。WebLogic Integration Studio を使用してワークフロー テンプレート定義を管理する方法については、『*WebLogic Integration Studio ユーザーズ ガイド*』の「ワークフロー テンプレートの定義」を参照してください。

テンプレート定義の作成

ワークフロー テンプレート定義を作成するには、以下のいずれかの com.bea.wlpi.server.admin.Admin メソッドを使用します。

メソッド 1

```
public java.lang.String createTemplateDefinition(
  java.lang.String templateId,
  java.lang.String name,
  java.lang.String xml
) throws java.rmi.RemoteException,
  com.bea.wlpi.common.WorkflowException
```

メソッド 2

```
public java.lang.String createTemplateDefinition(
  java.lang.String templateId,
  java.lang.String name,
  java.lang.String xml,
  java.lang.Object transactionId
) throws java.rmi.RemoteException,
  com.bea.wlpi.common.WorkflowException
```

第 1 のメソッドは、非クラスタ化環境で使用できます。クラスタ化環境では第 2 のメソッドを使用することをお勧めします。この場合、トランザクションがコミットされた後、またはサーバ クラッシュやフェイルオーバーが発生した後に、メソッドが再発行されないよう、指定された ID を使用してメソッドの実行状態が追跡されます。

createTemplateDefinition() メソッドのパラメータを次の表に示します。パラメータには値を指定する必要があります。

表 14-1 createTemplateDefinition() メソッドのパラメータ

パラメータ	説明	有効な値
templateId	テンプレート定義を作成するテンプレートの ID	有効なテンプレート ID を指定する文字列。 テンプレート ID を取得するには、次の com.bea.wlpi.common.TemplateInfo メソッドを使用する。 `public final java.lang.String getId()` TemplateInfo オブジェクトを取得する方法については、13-3 ページの「テンプレートの取得」を参照。TemplateInfo オブジェクトで選択可能なメソッドの詳細については、B-27 ページの「TemplateInfo オブジェクト」を参照。
name	新しいテンプレート定義の名前。	非ヌル文字列。
xml	テンプレート定義を定義し、A-53 ページの「テンプレート定義 DTD」に記載の Template Definition DTD に準拠する XML ドキュメント名	非ヌル文字列。
transactionId	トランザクションの ID **注意：** このパラメータは、クラスタ化環境においてのみ必要です。	ユニークなトランザクション ID を指定するオブジェクト。 ユニークなトランザクション ID を生成するには、次のコンストラクタを使用して新しい com.bea.wlpi.client.common.GUID オブジェクトを作成する。 `GUID transactionId = new GUID();` GUID クラスの詳細については、Javadoc の com.bea.wlpi.client.common.GUID を参照。

このメソッドは、新しいテンプレート定義の ID を返します。

たとえば、次のコードは、orderprocessing.xml ファイルに基づき Order Processing という名の新しいテンプレート定義を作成します。このコード例では、admin は Admin EJB への EJBObject 参照を表します。

```
String id = admin.createTemplateDefinition(
  template.getId(),"Order Processing", "orderprocessing.xml");
```

テンプレート ID は、com.bea.wlpi.common.TemplateInfo オブジェクトである template と関連付けられたメソッドを使用して取得されます。template オブジェクトは、13-3 ページの「テンプレートの取得」に記載のメソッドを使用して取得されます。

createTemplateDefinition() メソッドの詳細については、Javadoc の com.bea.wlpi.server.admin.Admin を参照してください。

テンプレート定義情報の取得

ワークフロー テンプレート定義を取得するには、次の com.bea.wlpi.server.admin.Admin メソッドを使用します。

```
public com.bea.wlpi.common.TemplateDefinitionInfo getTemplateDefinition(
  java.lang.String templateDefinitionId
) throws java.rmi.RemoteException,
  com.bea.wlpi.common.WorkflowException
```

値 getTemplateDefinitions() メソッドのパラメータを次の表に示します。パラメータには値を指定する必要があります。

表 14-2 getTemplateDefinition() メソッドのパラメータ

パラメータ	説明	有効な値
templateDefinitionId	検索を行うテンプレート定義の ID	有効なテンプレート定義 ID を指定する文字列。

このメソッドは、com.bea.wlpi.common.TemplateDefinitionInfo オブジェクトを返します。テンプレート定義についての情報にアクセスするには、B-24 ページの「TemplateDefinitionInfo オブジェクト」に記載の TemplateDefinitionInfo オブジェクト メソッドを使用します。

たとえば、次のコードでは、指定されたインスタンス ID に対応するワークフロー インスタンスに対する変数が取得されます。このコード例では、admin は Admin EJB への EJBObject 参照を表します。

```
TemplateDefinitionInfo info =
  admin.getTemplateDefinitions(templateDefinitionID);
```

getTemplateDefinition() メソッドの詳細については、Javadoc の com.bea.wlpi.server.admin.Admin を参照してください。

テンプレートの定義の取得

特定のワークフロー テンプレート定義を取得するには、以下の com.bea.wlpi.server.admin.Admin メソッドを使用します。

```
public java.util.List getTemplateDefinitions(
  java.lang.String templateId
) throws java.rmi.RemoteException,
  com.bea.wlpi.common.WorkflowException
```

getTemplateDefinitions() メソッドのパラメータを次の表に示します。パラメータには値を指定する必要があります。

表 14-3 getTemplateDefinitions() メソッドのパラメータ

パラメータ	説明	有効な値
templateId	テンプレート定義を検索するテンプレートの ID	有効なテンプレート ID を指定する文字列。 テンプレート ID を取得するには、次の com.bea.wlpi.common.TemplateInfo メソッドを使用する。 public final java.lang.String getId() TemplateInfo オブジェクトを取得する方法については、13-3 ページの「テンプレートの取得」を参照。TemplateInfo オブジェクトで選択可能なメソッドの詳細については、B-27 ページの「TemplateInfo オブジェクト」を参照。

このメソッドは、com.bea.wlpi.common.TemplateDefinitionInfo オブジェクトのリストを返します。各テンプレートについての情報にアクセスするには、B-24 ページの「TemplateDefinitionInfo オブジェクト」に記載の TemplateDefinitinoInfo オブジェクト メソッドを使用します。

たとえば、次のコードでは Order Processing テンプレートと関連するすべてのテンプレートが取得されます。このコード例では、admin は Admin EJB への EJBObject 参照を表します。

```
List tempdefs = admin.getTemplateDefinitions(“Order Processing”);
```

getTemplateDefinitions() メソッドの詳細については、Javadoc の com.bea.wlpi.server.admin.Admin を参照してください。

テンプレート定義のコンテンツの取得

テンプレート定義のコンテンツを取得するには、次の com.bea.wlpi.server.admin.Admin メソッドを使用します。

```
public java.lang.String getTemplateDefinitionContent(
  java.lang.String templateDefinitionId
) throws java.rmi.RemoteException,
  com.bea.wlpi.common.WorkflowException
```

getTemplateDefinitionContent() メソッドのパラメータを次の表に示します。パラメータには値を指定する必要があります。

表 14-4 getTemplateDefinition() メソッドのパラメータ

パラメータ	説明	有効な値
templateDefinitionId	取得するコンテンツのテンプレート定義の ID	有効なテンプレート定義 ID を指定する文字列。 テンプレート定義 ID を取得するには、次の com.bea.wlpi.common.TemplateDefinitionInfo メソッドを使用する。 public final java.lang.String getId() TemplateDefinitionInfo オブジェクトを取得する方法については、14-5 ページの「テンプレートの定義の取得」を参照。TemplateDefinitionInfo オブジェクトで選択可能なメソッドの詳細については、B-24 ページの「TemplateDefinitionInfo オブジェクト」を参照。

このメソッドは、A-53 ページの「テンプレート定義 DTD」に記載の Template Definition DTD に準拠する XML ドキュメントを返します。

たとえば、次のコードでは、指定されたテンプレート定義のコンテンツが取得されます。このコード例では、admin は Admin EJB への EJBObject 参照を表します。

```
String content =
  admin.getTemplateDefinitionContent(tempDef.getId());
```

テンプレート定義 ID は、com.bea.wlpi.common.TemplateDefinitionInfo オブジェクトである definition と関連付けられたメソッドを使用して取得されます。definition オブジェクトは、14-5 ページの「テンプレートの定義の取得」に記載のメソッドを使用して取得されます。

getTemplateDefinitionContent() メソッドの詳細については、Javadoc の com.bea.wlpi.server.admin.Admin を参照してください。

テンプレート定義のコンテンツの設定

テンプレート定義のコンテンツを設定するには、次の `com.bea.wlpi.server.admin.Admin` メソッドを使用します。

```
public void setTemplateDefinitionContent(
  java.lang.String templateDefinitionId,
  java.lang.String xml
) throws java.rmi.RemoteException,
  com.bea.wlpi.common.WorkflowException
```

`setTemplateDefinitionContent()` メソッドのパラメータを次の表に示します。パラメータには値を指定する必要があります。

表 14-5 setTemplateDefinition() メソッドのパラメータ

パラメータ	説明	有効な値
`templateDefinitionId`	コンテンツを設定するテンプレート定義の ID	有効なテンプレート定義 ID を指定する文字列。 テンプレート定義 ID を取得するには、次の `com.bea.wlpi.common.TemplateDefinitionInfo` メソッドを使用する。 `public final java.lang.String getId()` `TemplateDefinitionInfo` オブジェクトを取得する方法については、14-5 ページの「テンプレートの定義の取得」を参照。 `TemplateDefinitionInfo` オブジェクトで選択可能なメソッドの詳細については、B-24 ページの「TemplateDefinitionInfo オブジェクト」を参照。
`xml`	新しいテンプレート定義のコンテンツ	A-53 ページの「テンプレート定義 DTD」に記載の Template Definition DTD に準拠する XML ファイル。

たとえば、次のコードでは、指定された XML ファイル、new.xml を使用して、指定されたテンプレート定義のコンテンツが設定されます。このコード例では、admin は Admin EJB への EJBObject 参照を表します。

```
admin.setTemplateDefinitionContent(tempDef.getId(), “new.xml”);
```

テンプレート定義 ID は、com.bea.wlpi.common.TemplateDefinitionInfo オブジェクトである definition と関連付けられたメソッドを使用して取得されます。definition オブジェクトは、14-5 ページの「テンプレートの定義の取得」に記載のメソッドを使用して取得されます。

setTemplateDefinitionContent() メソッドの詳細については、Javadoc の com.bea.wlpi.server.admin.Admin を参照してください。

テンプレート定義オーナの取得

ワークフロー テンプレート定義の現在のオーナを取得するには、次の com.bea.wlpi.server.admin.Admin メソッドを使用します。

```
public java.lang.String getTemplateOwner(
  java.lang.String templateDefinitionId
) throws java.rmi.RemoteException,
  com.bea.wlpi.common.WorkflowException
```

getTemplates() メソッドのパラメータを次の表に示します。パラメータには値を指定する必要があります。

表 14-6 getTemplateOwner() メソッドのパラメータ

パラメータ	説明	有効な値
templateDefinitionId	取得するオーナのテンプレート定義に対応する ID	有効なテンプレート定義 ID を指定する文字列。 テンプレート定義 ID を取得するには、次の `com.bea.wlpi.common.TemplateDefinitionInfo` メソッドを使用する。 `public final java.lang.String getId()` `TemplateDefinitionInfo` オブジェクトを取得する方法については、14-5 ページの「テンプレートの定義の取得」を参照。`TemplateDefinitionInfo` オブジェクトで選択可能なメソッドの詳細については、B-24 ページの「TemplateDefinitionInfo オブジェクト」を参照。

このメソッドは、現在のテンプレート定義オーナの ID を返します。

たとえば、次のコードはテンプレート定義オーナ ID を取得し、その結果を `owner` 文字列に割り当てます。このコード例では、`admin` は `Admin` EJB への `EJBObject` 参照を表します。

```
String owner = admin.getTemplateOwner(templateDef.getId());
```

`getTemplateOwner()` メソッドの詳細については、Javadoc の `com.bea.wlpi.server.admin.Admin` を参照してください。

呼び出し可能なワークフローの取得

呼び出し可能なワークフローのリストを取得するには、次の com.bea.wlpi.server.admin.Admin メソッドを使用します。

```
public java.util.List getCallableWorkflows(
  java.lang.String orgId
) throws java.rmi.RemoteException,
  com.bea.wlpi.common.WorkflowException
```

getCallableWorkflows() メソッドのパラメータを次の表に示します。パラメータには値を指定する必要があります。

パラメータ	説明	有効な値
orgId	呼び出し可能なワークフローの名前を取得するオーガニゼーションの ID	有効なオーガニゼーション ID を指定する文字列。 すべてのオーガニゼーション ID のリストの取得方法については、9-9 ページの「すべてのオーガニゼーション名を取得する」を参照。

このメソッドは、呼び出し可能なワークフローのリストを返します。

たとえば、次のコードを使用すると OrgID 変数の値で指定したオーガニゼーションの呼び出し可能ワークフローのリストを取得できます。このコード例では、admin は Admin EJB への EJBObject 参照を表します。

```
List callablewf = admin.getCallableWorkflows(orgId);
```

getCallableWorkflows() メソッドの詳細については、Javadoc の com.bea.wlpi.server.admin.Adm を参照してください。

呼び出し可能なワークフローの検索

次の com.bea.wlpi.server.admin.Admin メソッドは、最適な（アクティブでしかも効果的な）呼び出し可能なワークフローのリストを返します。

```
public java.util.List findCallableWorkflows(
  java.lang.String templateName
  java.lang.String templateID
  java.lang.String orgId
) throws java.rmi.RemoteException,
  com.bea.wlpi.common.WorkflowException
```

getTemplateOwner() メソッドのパラメータを次の表に示します。パラメータには値を指定する必要があります。

パラメータ	説明	有効な値
templateName	テンプレート名。このテンプレートに対して、呼び出し可能なワークフローを検索する。	有効なテンプレート名 テンプレート名を取得するには、次の com.bea.wlpi.common.TemplateInfo メソッドを使用する。 `public final java.lang.String getId()` TemplateInfo オブジェクトを取得する方法については、13-3 ページの「テンプレートの取得」を参照。TemplateInfo オブジェクトで選択可能なメソッドの詳細については、B-27 ページの「TemplateInfo オブジェクト」を参照。

パラメータ	説明	有効な値
templateID	呼び出し可能なワークフローを検索するテンプレートの ID	有効なテンプレート ID を指定する文字列。 テンプレート ID を取得するには、次の `com.bea.wlpi.common.TemplateInfo` メソッドを使用する。 `public final java.lang.String getId()` `TemplateInfo` オブジェクトを取得する方法については、13-3 ページの「テンプレートの取得」を参照。`TemplateInfo` オブジェクトで選択可能なメソッドの詳細については、B-27 ページの「TemplateInfo オブジェクト」を参照。
orgId	オーガニゼーションの ID。このオーガニゼーションを対象にして、呼び出し可能なワークフローを検索する。	有効なオーガニゼーション ID。 すべてのオーガニゼーション ID のリストの取得方法については、9-9 ページの「すべてのオーガニゼーション名を取得する」を参照。

このメソッドは、呼び出し可能なワークフローのリストを返す。

たとえば、次のコードを使用すると、指定したテンプレートについて最適な呼び出し可能ワークフローのリストを検索できる。このコード例では、`admin` は `Admin` EJB への `EJBObject` 参照を表します。

```
List callablewf = admin.findCallableWorkflows(templateName,
templateID, orgId);
```

`findCallableWorkflows()` メソッドの詳細については、Javadoc の `com.bea.wlpi.server.admin.Admin` を参照してください。

テンプレート定義のロックおよびロック解除

不要な編集をなくすために、ワークフロー テンプレート定義に占有書き込みロックをかけることができます。テンプレート定義のロックおよびロック解除には、それぞれ以下の `com.bea.wlpi.server.admin.Admin` メソッドを使用します。

```
public void lockTemplate(
  java.lang.String templateDefinitionId
) throws java.rmi.RemoteException,
  com.bea.wlpi.common.WorkflowException

public void unlockTemplate(
  java.lang.String templateDefinitionId
) throws java.rmi.RemoteException,
  com.bea.wlpi.common.WorkflowException
```

注意： 現在のテンプレート定義オーナを取得するには、14-9 ページの「テンプレート定義オーナの取得」で説明する `getTemplateOwner()` メソッドを使用します。

`lockTemplate()` および `unlockTemplate()` メソッドのパラメータを次の表に示します。パラメータには値を指定する必要があります。

表 14-7 lockTemplate() および unlockTemplate() メソッドのパラメータ

パラメータ	説明	有効な値
templateDefinitionId	ロックまたはロック解除するテンプレート定義に対応する ID	有効なテンプレート定義 ID を指定する文字列。 テンプレート定義 ID を取得するには、次の `com.bea.wlpi.common.TemplateDefinitionInfo` メソッドを使用する。 `public final java.lang.String getId()` `TemplateDefinitionInfo` オブジェクトを取得する方法については、14-5 ページの「テンプレートの定義の取得」を参照。`TemplateDefinitionInfo` オブジェクトで選択可能なメソッドの詳細については、B-24 ページの「TemplateDefinitionInfo オブジェクト」を参照。

たとえば、次のコードでは、指定されたテンプレート定義がロックされます。このコード例では、`admin` は `Admin` EJB への `EJBObject` 参照を表します。

```
admin.lockTemplate(templateDef.getId());
```

以下のコードは、指定されたテンプレート定義のロックを解除します。

```
admin.unlockTemplate(templateDef.getId());
```

`lockTemplate()` および `unlockTemplate()` メソッドの詳細については、Javadoc の `com.bea.wlpi.server.admin.Admin` を参照してください。

テンプレート定義の削除

テンプレート定義を削除するには、次の com.bea.wlpi.server.admin.Admin メソッドを使用します。

```
public void deleteTemplateDefinition(
  java.lang.String templateDefinitionId
) throws java.rmi.RemoteException,
  com.bea.wlpi.common.WorkflowException
```

deleteTemplateDefinition() メソッドのパラメータを次の表に示します。パラメータには値を指定する必要があります。

表 14-8 getTemplateDefinition() メソッドのパラメータ

パラメータ	説明	有効な値
templateDefinitionId	削除するテンプレート定義の ID	有効なテンプレート定義 ID を指定する文字列。 テンプレート定義 ID を取得するには、次の com.bea.wlpi.common.TemplateDefinitionInfo メソッドを使用する。 public final java.lang.String getId() TemplateDefinitionInfo オブジェクトを取得する方法については、14-5 ページの「テンプレートの定義の取得」を参照。 TemplateDefinitionInfo オブジェクトで選択可能なメソッドの詳細については、B-24 ページの「TemplateDefinitionInfo オブジェクト」を参照。

たとえば、次のコードでは、指定されたテンプレート定義が削除されます。このコード例では、admin は Admin EJB への EJBObject 参照を表します。

```
admin.deleteTemplateDefinition(definition.getId());
```

テンプレート定義 ID は、com.bea.wlpi.common.TemplateDefinitionInfo オブジェクトである definition と関連付けられたメソッドを使用して取得されます。definition オブジェクトは、14-5 ページの「テンプレートの定義の取得」に記載のメソッドを使用して取得されます。

deleteTemplateDefinition() メソッドの詳細については、Javadoc の com.bea.wlpi.server.admin.Admin を参照してください。

15　タスクの管理

タスクは、14-2 ページの「テンプレート定義の作成」に記載のように、テンプレート定義の作成プロセスの一部として定義されます。

この章では、タスクの管理方法について説明します。内容は以下のとおりです。

- タスクの取得
- タスクの割り当て
- タスク数の取得
- タスクへの完了または未完了マークの付与
- タスク プロパティの設定

この章に記載するメソッドの詳細については com.bea.server.admin.Admin を参照してください。WebLogic Integration Studio を使用してタスクを管理する方法については、『*WebLogic Integration Studio ユーザーズ ガイド*』の「ワークフロー テンプレートの定義」を参照してください。

タスクの取得

特定のワークフロー テンプレートで定義されたタスクのリストを取得するには、以下の com.bea.wlpi.server.admin.Admin メソッドのいずれかを使用します。

```
public javautil.List getTasks(
  java.lang.String assigneeId,
  java.lang.String orgId,
  boolean role
) throws java.rmi.RemoteException,
  com.bea.wlpi.common.WorkflowException

public javautil.List getTasks(
  java.lang.String assigneeId,
  java.lang.String orgId,
  boolean role,
  boolean incompleteonly,
  boolean sortAscending
```

```
) throws java.rmi.RemoteException,
  com.bea.wlpi.common.WorkflowException
```

getTasks() メソッドのパラメータを次の表に示します。パラメータには値を指定する必要があります。

表 15-1 getTasks() メソッドのパラメータ

パラメータ	説明	有効な値
assigneeId	取得するタスクのリストの *assignee* (ユーザまたはロール) の ID	有効なユーザまたはロールを指定する文字列。 現在のユーザまたはロールのリストを取得する方法については、9-1 ページの「セキュリティ レルムのコンフィグレーション」を参照。
orgId	タスクを取得するオーガニゼーションの ID	有効なオーガニゼーション ID を指定する文字列。 すべてのオーガニゼーション ID のリストの取得方法については、9-9 ページの「すべてのオーガニゼーション名を取得する」を参照。
role	assigneeId パラメータに指定された割り当て先がロール (true) とユーザ (false) のいずれであるかを指定するブール フラグ	true (ロール) または false (ユーザ)。
incompleteonly	未完了のタスクだけを返す（true）のか、すべてのタスクを返す（false）のかを指定するブール フラグ	true (未完了のタスクのみ) または false (すべてのタスク)。
sortAscending	タイムスタンプ（日および時）をソート キーとするタスクのソートを、昇順（true）で行うのか、降順（false）で行うのかを指定するブール フラグ	true (昇順) または false (降順)。

各メソッドは、com.bea.wlpi.common.TaskInfo オブジェクトのリストを返します。各タスクについての情報にアクセスするには、B-21 ページの「TaskInfo オブジェクト」に記載する TaskInfo オブジェクト メソッドを使用します。

たとえば、次のコードでは、オーガニゼーション ORG1 内のロール ROLE1 (role パラメータが true に設定してあることに注意) のタスクのリストが取得されます。このコード例では、admin は Admin EJB への EJBObject 参照を表します。

```
List taskList = admin.getTasks(“ROLE1”, “ORG1”, true);
```

getTasks() メソッドの詳細については、Javadoc の com.bea.wlpi.server.admin.Admin を参照してください。

タスクの割り当て

タスクに割り当てられているユーザまたはロールを変更する、またはタスクの割り当てを解除するには、それぞれ以下の com.bea.wlpi.server.admin.Admin メソッドを使用します。

```
public void taskAssign(
  java.lang.String templateDefinitionId,
  java.lang.String instanceId,
  java.lang.String taskId,
  java.lang.String assignTo,
  boolean bRole,
  boolean bLoadBalance
) throws java.rmi.RemoteException,
  com.bea.wlpi.common.WorkflowException

public void taskUnassign(
  java.lang.String templateDefinitionId,
  java.lang.String instanceId,
  java.lang.String taskId
) throws java.rmi.RemoteException,
  com.bea.wlpi.common.WorkflowException
```

taskAssign() および taskUnassign() メソッドのパラメータを次の表に示します。パラメータには値を指定する必要があります。

表 15-2 taskAssign() および taskUnassign() メソッドのパラメータ

パラメータ	説明	有効な値	対象	
			taskAssign()	taskUnassign()
templateDefinitionId	タスクを含むワークフロー インスタンスがインスタンス化されたテンプレート定義の ID	有効なテンプレート定義 ID を指定する文字列。 テンプレート定義 ID を取得するには、次の `com.bea.wlpi.common.TaskInfo` メソッドを使用する。 `public final java.lang.String getTemplateDefinitionId()` `TaskInfo` オブジェクトを取得する方法については、15-1 ページの「タスクの取得」を参照。`TaskInfo` オブジェクトで選択可能なメソッドの詳細については、B-21 ページの「TaskInfo オブジェクト」を参照。	+	+

表 15-2 taskAssign() および taskUnassign() メソッドのパラメータ (続き)

パラメータ	説明	有効な値	対象	
			taskAssign()	taskUnassign()
instanceId	タスクを含むワークフロー インスタンスの ID	有効なインスタンス ID を指定する文字列。 インスタンス ID を取得するには、次の `com.bea.wlpi.common.TaskInfo` メソッドを使用する。 `public final java.lang.String getInstanceId()` `TaskInfo` オブジェクトを取得する方法については、15-1 ページの「タスクの取得」を参照。`TaskInfo` オブジェクトで選択可能なメソッドの詳細については、B-21 ページの「TaskInfo オブジェクト」を参照。	+	+

表 15-2 taskAssign() および taskUnassign() メソッドのパラメータ (続き)

パラメータ	説明	有効な値	対象	
			taskAssign()	taskUnassign()
taskId	割り当てまたは割り当て解除を行うタスクの ID	有効なタスク ID を指定する文字列。 タスク ID を取得するには、次の `com.bea.wlpi.common.TaskInfo` メソッドを使用する。 `public final java.lang.String getTaskId()` `TaskInfo` オブジェクトを取得する方法については、15-1 ページの「タスクの取得」を参照。`TaskInfo` オブジェクトで選択可能なメソッドの詳細については、B-21 ページの「TaskInfo オブジェクト」を参照。	+	+
assignTo	タスクを割り当てる *assignee* (ユーザまたはロール) の ID	有効なユーザまたはロールを指定する文字列。 現在のユーザまたはロールのリストを取得する方法については、9-1 ページの「セキュリティ レルムのコンフィグレーション」を参照。	+	

表 15-2 taskAssign() および taskUnassign() メソッドのパラメータ (続き)

パラメータ	説明	有効な値	対象	
			taskAssign()	taskUnassign()
bRole	assigneeId パラメータに指定された割り当て先がロール（true）または ユーザ（false）であるかを指定するブール フラグ	true (ロール) または false (ユーザ)。	+	
bLoadBalance	指定されたロール内でロード バランスを行うかどうかを指定するブール フラグ（この設定は、bRole 値が true に設定されている場合のみ有効）。	true (ロード バランスを行う) または false (ロード バランスを行わない)。	+	

タスクが割り当てられる実際の割り当て先は、以下の条件で決定されます。

- 指定された割り当て先に対して現在タスクの再ルーティングが存在するか。
- bRole および bLoadBalance 引数は、ロード バランスが有効となっていることを表す true に設定してあるか。この場合、WebLogic Integration プロセス エンジンでは、（現在有効な再ルーティングがない）ロール内に割り当てられたすべてのユーザのタスク数が検討し、割り当てられたタスクが最も少ないユーザを識別し、そのユーザにタスクを割り当てます。

たとえば、次のコードでは、ユーザ joe にタスクが割り当てられます。このコード例では、admin は Admin EJB への EJBObject 参照を表します。

```
admin.taskAssign(
  task.getTemplateDefinitionId(),
  task.getInstanceId(),
  task.getTaskId(),
  “joe”,
  false,
  false
);
```

次のコードでは、同タスクの割り当てが解除されます。

```
admin.taskUnassign(
  task.getTemplateDefinitionId(),
  task.getInstanceId(),
  task.getTaskId(),
  “joe”,
  false,
  false
);
```

テンプレート定義、ワークフロー インスタンス、およびタスク ID は、`com.bea.wlpi.common.TaskInfo` オブジェクト、`task` と関連するメソッドを使用して取得されます。`task` オブジェクトは、15-1 ページの「タスクの取得」に記載のメソッドを使用して取得されます。

`com.bea.wlpi.common.TaskInfo` メソッドの詳細については、B-21 ページの「TaskInfo オブジェクト」または、Javadoc の `com.bea.wlpi.common.TaskInfo` を参照してください。

`taskAssign()` および `taskUnassign()` メソッドの詳細については、Javadoc の `com.bea.wlpi.server.admin.Admin` を参照してください。

タスク数の取得

特定のユーザに割り当てられているタスクの数を取得するには、次の com.bea.wlpi.server.admin.Admin メソッドを使用します。

```
public int[] getTaskCounts(
java.lang.String assigneeId,
java.lang.String orgId,
boolean isRole
) throws java.rmi.RemoteException,
  com.bea.wlpi.common.WorkflowException
```

getTaskCounts() メソッドのパラメータを次の表に示します。パラメータには値を指定する必要があります。

表 15-3 getTaskCounts() メソッドのパラメータ

パラメータ	説明	有効な値
assigneeId	タスクを取得する *assignee* (ロールまたはユーザ) の ID	有効なユーザまたはロールの ID。 現在のユーザまたはロールのリストを取得する方法については、9-1 ページの「セキュリティ レルムのコンフィグレーション」を参照。
orgId	タスクを取得するオーガニゼーションの ID	有効なオーガニゼーション ID を指定する文字列。 すべてのオーガニゼーション ID のリストを取得する方法については、19-1 ページの「アクティブなオーガニゼーションの管理」を参照。
isRole	assigneeId パラメータに指定された割り当て先がロールまたはユーザであるかを指定するブール フラグ	true (ロール) または false (ユーザ)。

このメソッドは、次の表で説明する、5 つの要素から成る配列を返します。

表 15-4 getTaskCounts() メソッドの配列要素

要素	説明
TASKCOUNT_TOTAL	ユーザやロールに割り当てられたタスクの総数
TASKCOUNT_PENDING	割り当てられているタスクのうちペンディング中のものの数
TASKCOUNT_INACTIVE	割り当てられているタスクのうちアクティブでないものの数
TASKCOUNT_COMPLETED	割り当てられているタスクのうち完了したものの数
TASKCOUNT_OVERDUE	割り当てられているタスクのうち期日超過のものの数

たとえば、次のコードでは、現在のユーザに割り当てられたタスクの数が取得され、配列 `taskCounts[ ]` にその数が格納されます。このコード例では、`admin` は `Worklist` EJB への `EJBObject` 参照を表します。

```
int taskCounts[] = admin.getTaskCounts(“ROLE1”, “ORG1”, true);
```

配列 `taskCounts[ ]` の各要素にタスク数が格納されます。たとえば、`taskCounts[TASKCOUNT_TOTAL]` は、タスクの総数、`taskCounts[TASKCOUNT_PENDING]` は、ペンディング中のタスクの総数を示します。

`getTaskCounts()` メソッドの詳細については、Javadoc の `com.bea.wlpi.server.admin.Admin` を参照してください。

タスクへの完了または未完了マークの付与

タスクに完了（done）または未完了（undone）としてマークするには、それぞれ以下の `com.bea.wlpi.server.admin.Admin` メソッドを使用します。

```
public void taskMarkDone(
  java.lang.String templateDefinitionId,
  java.lang.String instanceId,
  java.lang.String taskId
```

```
) throws java.rmi.RemoteException,
  com.bea.wlpi.common.WorkflowException

public void taskUnmarkDone(
  java.lang.String templateDefinitionId,
  java.lang.String instanceId,
  java.lang.String taskId
) throws java.rmi.RemoteException,
  com.bea.wlpi.common.WorkflowException
```

taskMarkDone() および taskUnmarkDone() メソッドのパラメータを次の表に示します。パラメータには値を指定する必要があります。

表 15-5 taskMarkDone() および taskUnmarkDone() メソッドのパラメータ

パラメータ	説明	有効な値
templateDefinitionId	タスクを含むワークフロー インスタンスがインスタンス化されたテンプレート定義の ID	有効なテンプレート定義 ID を指定する文字列。 テンプレート定義 ID を取得するには、次の com.bea.wlpi.common.TaskInfo メソッドを使用する。 public final java.lang.String getTemplateDefinitionId() TaskInfo オブジェクトを取得する方法については、15-1 ページの「タスクの取得」を参照。TaskInfo オブジェクトで選択可能なメソッドの詳細については、B-21 ページの「TaskInfo オブジェクト」を参照。

表 15-5 taskMarkDone() および taskUnmarkDone() メソッドのパラメータ (続き)

パラメータ	説明	有効な値
instanceId	タスクを含むワークフロー インスタンスの ID	有効なインスタンス ID を指定する文字列。 インスタンス ID を取得するには、次の `com.bea.wlpi.common.TaskInfo` メソッドを使用する。 `public final java.lang.String getInstanceId()` `TaskInfo` オブジェクトを取得する方法については、15-1 ページの「タスクの取得」を参照。`TaskInfo` オブジェクトで選択可能なメソッドの詳細については、B-21 ページの「TaskInfo オブジェクト」を参照。
taskId	完了または未完了マークを付けるタスクの ID	有効なタスク ID を指定する文字列。 タスク ID を取得するには、次の `com.bea.wlpi.common.TaskInfo` メソッドを使用する。 `public final java.lang.String getTaskId()` `TaskInfo` オブジェクトを取得する方法については、15-1 ページの「タスクの取得」を参照。`TaskInfo` オブジェクトで選択可能なメソッドの詳細については、B-21 ページの「TaskInfo オブジェクト」を参照。

`taskMarkDone()` メソッドは、現在の日付および時間にタスク `Completed` 値を設定します。さらに、指定されたタスクの `Marked Done` イベントと関連する、すべてのアクションの連続実行を行います。ただし、`taskMarkDone()` メソッドは完了マークの付けられたタスクには影響しません。タスクの `Marked Done` イベントの定義については、『*WebLogic Integration Studio ユーザーズ ガイド*』の「ワークフローのテンプレートの定義」を参照してください。

taskUnmarkDone() メソッドは Completed 日付を消去します。このメソッドは、指定されたタスクの Activated イベントと関連するアクションの実行にはつながりません。

たとえば、次のコードでは、指定されたタスクに完了マークが付けられ、Completed 値が現在の日付および時間に設定され、指定されたタスクの Marked Done イベントと関連するアクションが実行されます。このコード例では、admin は Admin EJB への EJBObject 参照を表します。

```
admin.taskMarkDone(
  task.getTemplateDefinitionId(),
  task.getInstanceId(),
  task.getTaskId()
);
```

次のコードは、同タスクに未完了マークを付け、Completed 日付を消去します。

```
admin.taskMarkUndone(
  task.getTemplateDefinitionId(),
  task.getInstanceId(),
  task.getTaskId()
);
```

テンプレート定義、ワークフロー インスタンス、およびタスク ID は、com.bea.wlpi.common.TaskInfo オブジェクト、task と関連するメソッドを使用して取得されます。task オブジェクトは、15-1 ページの「タスクの取得」に記載のメソッドを使用して取得されます。

com.bea.wlpi.common.TaskInfo メソッドの詳細については、B-21 ページの「TaskInfo オブジェクト」または、Javadoc の com.bea.wlpi.common.TaskInfo を参照してください。

taskMarkDone() および taskMarkUndone() メソッドの詳細については、Javadoc の com.bea.wlpi.server.admin.Admin を参照してください。

タスク プロパティの設定

タスク プロパティを設定するには、次の com.bea.wlpi.server.admin.Admin メソッドを使用します。

```
public void taskSetProperties(
  java.lang.String templateDefinitionId,
  java.lang.String instanceId,
  java.lang.String taskId,
```

```
  int priority,
  boolean doneWithoutDoit,
  boolean doitIfDone,
  boolean unmarkDone,
  boolean modify,
  boolean reassign
) throws java.rmi.RemoteException,
  com.bea.wlpi.common.WorkflowException
```

taskSetProperties() メソッドのパラメータを次の表に示します。パラメータには値を指定する必要があります。

表 15-6 taskSetProperties() メソッドのパラメータ

パラメータ	説明	有効な値
templateDefinitionId	タスクと対応するテンプレート定義の ID	有効なテンプレート定義 ID を指定する文字列。 テンプレート定義 ID を取得するには、次の com.bea.wlpi.common.TaskInfo メソッドを使用する。 public final java.lang.String getTemplateDefinitionId() TaskInfo オブジェクトを取得する方法については、15-1 ページの「タスクの取得」を参照。TaskInfo オブジェクトで選択可能なメソッドの詳細については、B-21 ページの「TaskInfo オブジェクト」を参照。

表 15-6 taskSetProperties() メソッドのパラメータ (続き)

パラメータ	説明	有効な値
instanceId	タスクと対応するワークフローインスタンスの ID	有効なインスタンス ID を指定する文字列。 インスタンス ID を取得するには、次の `com.bea.wlpi.common.TaskInfo` メソッドを使用する。 `public final java.lang.String getInstanceId()` `TaskInfo` オブジェクトを取得する方法については、15-1 ページの「タスクの取得」を参照。`TaskInfo` オブジェクトで選択可能なメソッドの詳細については、B-21 ページの「TaskInfo オブジェクト」を参照。
taskId	完了または未完了マークを付けるタスクの ID	有効なタスク ID を指定する文字列。 タスク ID を取得するには、次の `com.bea.wlpi.common.TaskInfo` メソッドを使用する。 `public final java.lang.String getTaskId()` `TaskInfo` オブジェクトを取得する方法については、15-1 ページの「タスクの取得」を参照。`TaskInfo` オブジェクトで選択可能なメソッドの詳細については、B-21 ページの「TaskInfo オブジェクト」を参照。
priority	タスクの優先順位	`0`= 低、`1`= 中、または `2`= 高。
doneWithoutDoIt	15-10 ページの「タスクへの完了または未完了マークの付与」に記載するメソッドを使用してタスクに完了マークを付けるパーミッション	`true` (有効化) または `false` (無効化)。

表 15-6 taskSetProperties() メソッドのパラメータ (続き)

パラメータ	説明	有効な値
doitIfDone	21-6 ページの「タスクの実行」に記載するメソッドを使用して、完了マークを付けた後にそのタスクを実行するパーミッション	true (有効化) または false (無効化)。
unmarkDone	15-10 ページの「タスクへの完了または未完了マークの付与」に記載するメソッドを使用して、タスクに未完了マークを付けるパーミッション	true (有効化) または false (無効化)。
modify	taskSetProperties() メソッドを使用して、ランタイム プロパティを変更するパーミッション	true (有効化) または false (無効化)。
reassign	15-3 ページの「タスクの割り当て」に記載するメソッドを使用して、タスクを別の参加コンポーネントに再び割り当てるパーミッション	true (有効化) または false (無効化)。

たとえば、次のコードは、デフォルトのタスク優先順位を 中（1）に設定し、他のすべてのタスク優先順位を有効化（true に設定）します。このコード例では、admin は Admin EJB への EJBObject 参照を表します。

```
admin.taskSetProperties(
   task.getTemplateDefinitionId(),
   task.getInstanceId(),
   task.getTaskId(),
   1,
   true,
   true,
   true,
   true,
   true
);
```

テンプレート定義、ワークフロー インスタンス、およびタスク ID は、com.bea.wlpi.common.TaskInfo オブジェクト、task と関連するメソッドを使用して取得されます。task オブジェクトは、15-1 ページの「タスクの取得」に記載のメソッドを使用して取得されます。

com.bea.wlpi.common.TaskInfo メソッドの詳細については、B-21 ページの「TaskInfo オブジェクト」または、Javadoc の com.bea.wlpi.common.TaskInfo を参照してください。

taskSetProperties() メソッドの詳細については、Javadoc の com.bea.wlpi.server.admin.Admin を参照してください。

16　タスクのルーティングの管理

指定された期間のタスクを、1 つのロールまたはユーザから別のロールまたはユーザへ再ルーティングできます。

この章では、タスクのルーティングの管理方法について説明します。内容は以下のとおりです。

- タスクの再ルーティングの追加
- タスクの再ルーティングの取得
- タスクの再ルーティングの更新
- タスクの再ルーティングの削除
- タスクのルーティング例

この章に記載するメソッドの詳細については、Javadoc の `com.bea.wlpi.server.admin.Admin` を参照してください。WebLogic Integration Studio を使用してのタスクのルーティングの管理方法については、『*WebLogic Integration Studio ユーザーズ ガイド*』の「データの管理」にある「タスクルーティングの管理」を参照してください。

タスクの再ルーティングの追加

タスクの再ルーティングを追加するには、次の `com.bea.wlpi.server.admin.Admin` メソッドを使用します。

```
public java.lang.String addReroute(
  java.lang.String orgId,
  java.lang.String from,
  java.lang.String to,
  int type,
  java.sql.Timestamp effective,
  java.sql.Timestamp expiry
) throws java.rmi.RemoteException,
  com.bea.wlpi.common.WorkflowException
```

`addReroute()` メソッドのパラメータを次の表に示します。パラメータには値の指定が必要です。

表 16-1 addReroute() メソッドのパラメータ

パラメータ	説明	有効な値
`orgId`	タスクの再ルーティングを行うオーガニゼーションの ID	有効なオーガニゼーション ID を指定する文字列。 オーガニゼーション ID のリストの取得については、9-9 ページの「すべてのオーガニゼーション名を取得する」を参照。
`from`	タスクの再ルーティング元である *assognee* (ロールまたはユーザ) の ID	有効なロールまたはユーザを指定する文字列。 現在のユーザまたはロールのリストを取得する方法については、9-1 ページの「セキュリティ レルムのコンフィグレーション」を参照。
`to`	タスクの再ルーティング先である *assignee* (ロールまたはユーザ) の ID	有効なロールまたはユーザを指定する文字列。 現在のユーザまたはロールのリストを取得する方法については、9-1 ページの「セキュリティ レルムのコンフィグレーション」を参照。

表 16-1 addReroute() メソッドのパラメータ (続き)

パラメータ	説明	有効な値
type	再ルーティングのタイプ	次の静的変数値のいずれかを指定する整数。 ■ TYPE_USER ―ユーザへのタスクの再ルーティング。 ■ TYPE_USERINROLE ―ロード バランスを使用したロールのユーザへのタスクの再ルーティング。 ■ TYPE_ROLE ―ロールへのタスクの再ルーティング。 静的変数値は、com.bea.wlpi.common.RerouteInfo オブジェクトの一部として定義されている。
effective	再ルーティングが有効となる日付および時間	有効 java.sql.Timestamp オブジェクト。
expiry	再ルーティングが終了となる日付および時間	有効 java.sql.Timestamp オブジェクト。

メソッドは、新しい再ルーティング オブジェクトの ID を戻します。

たとえば、次のコードでは、ユーザ joe からユーザ mary への再ルーティングが追加されます。このコード例では、admin は Admin EJB への EJBObject 参照を表します。

```
List rerouteId = admin.addReroute( “ORG1”, “joe”, “mary”,
  TYPE_USER, tsEffective, tsExpiry);
```

type パラメータは、TYPE_USER に設定されていますが、これはタスクがユーザへと再ルーティングされていることを表します。再ルーティングは、tsEffective タイムスタンプで指定された日付および時間に有効となり、tsExpiry タイムスタンプで指定された日付および時間に終了となります。

addReroute() メソッドの詳細については、Javadoc の com.bea.wlpi.server.admin.Admin を参照してください。

タスクの再ルーティングの取得

オーガニゼーション内で定義されたタスクの再ルーティングは、次の com.bea.wlpi.server.admin.Admin メソッドを使用します。

```
public java.util.List getReroutes(
  java.lang.String orgId
) throws java.rmi.RemoteException,
  com.bea.wlpi.common.WorkflowException
```

getReroutes() メソッドのパラメータを次の表に示します。パラメータには値の指定が必要です。

表 16-2 getReroutes() メソッドのパラメータ

パラメータ	説明	有効な値
orgId	再ルーティングを取得するオーガニゼーションの ID	有効なオーガニゼーション ID を指定する文字列。 オーガニゼーション ID のリストの取得については、9-9 ページの「すべてのオーガニゼーション名を取得する」を参照。

各メソッドは、com.bea.wlpi.common.TaskInfo オブジェクトのリストを返します。各タスクについての情報にアクセスするには、B-21 ページの「TaskInfo オブジェクト」に記載の TaskInfo オブジェクト メソッドを使用します。

たとえば、次のコードでは、ORG1 オーガニゼーションの再ルーティングされたタスクが取得されます。このコード例では、admin は Admin EJB への EJBObject 参照を表します。

```
List taskList = admin.getReroutes("ORG1");
```

getReroutes() メソッドの詳細については、Javadoc の com.bea.wlpi.server.admin.Admin を参照してください。

タスクの再ルーティングの更新

タスクの再ルーティングを更新するには、以下のいずれかの `com.bea.wlpi.server.admin.Admin` メソッドを使用します。

メソッド 1

```
public void updateReroute(
  java.lang.String rerouteId,
  java.lang.String to,
  int type,
  java.sql.Timestamp effective,
  java.sql.Timestamp expiry
) throws java.rmi.RemoteException,
  com.bea.wlpi.common.WorkflowException
```

メソッド 2

```
public void updateReroute(
  java.lang.String rerouteId,
  java.lang.String from,
  java.lang.String to,
  int type,
  java.sql.Timestamp effective,
  java.sql.Timestamp expiry
) throws java.rmi.RemoteException,
  com.bea.wlpi.common.WorkflowException
```

これらのメソッドは、2 つ目のメソッドでタスクの再ルーティング先である割り当て先を定義するために `from` パラメータを指定すること以外は同一です。

`updateReroute()` メソッドのパラメータを次の表に示します。パラメータには値の指定が必要です。

表 16-3 updateReroute() メソッドのパラメータ

パラメータ	説明	有効な値
rerouteId	更新するタスクの再ルーティングの ID	有効な再ルーティング ID を指定する文字列。 すべてのタスクの再ルーティング ID のリストを取得する方法については、16-4 ページの「タスクの再ルーティングの取得」を参照。

表 16-3 updateReroute() メソッドのパラメータ (続き)

パラメータ	説明	有効な値
from	タスクの再ルーティング元である *assognee* (ロールまたはユーザ) の ID	有効なロールまたはユーザを指定する文字列。 現在のユーザまたはロールのリストを取得する方法については、9-1 ページの「セキュリティ レルムのコンフィグレーション」を参照。
to	タスクの再ルーティング先である *assignee* (ロールまたはユーザ) の ID	有効なロールまたはユーザを指定する文字列。 現在のユーザまたはロールのリストを取得する方法については、9-1 ページの「セキュリティ レルムのコンフィグレーション」を参照。
type	再ルーティングのタイプ	次の静的変数値のいずれかを指定する整数値。 ■ TYPE_USER ―ユーザへのタスクの再ルーティング。 ■ TYPE_USERINROLE ―ロード バランスを使用したロールのユーザへのタスクの再ルーティング。 ■ TYPE_ROLE ―ロールへのタスクの再ルーティング。 静的変数値は、com.bea.wlpi.common.RerouteInfo オブジェクトの一部として定義されている。
effective	再ルーティングが有効となる日付および時間	有効 java.sql.Timestamp オブジェクト。
expiry	再ルーティングが終了となる日付および時間	有効 java.sql.Timestamp オブジェクト。

たとえば、以下のコードでは、指定されたタスクの再ルーティングが更新され、タスクはユーザ Mary から joe へと再ルーティングされます。このコード例では、admin は Admin EJB への EJBObject 参照を表します。

```
List rerouteId = admin.addReroute( “ORG1”, “mary”, “joe”,
  TYPE_USER, tsEffective, tsExpiry);
```

type パラメータは、TYPE_USER に設定されていますが、これはタスクがユーザへと再ルーティングされていることを表します。再ルーティングは、tsEffective タイムスタンプで指定された日付および時間から有効となり、tsExpiry タイムスタンプで指定された日付および時間に終了となります。

updateReroutes() メソッドの詳細については、Javadoc の com.bea.wlpi.server.admin.Admin を参照してください。

タスクの再ルーティングの削除

タスクの再ルーティングを削除するには、次の com.bea.wlpi.server.admin.Admin メソッドを使用します。

```
public void deleteReroute(
  java.lang.String rerouteId
) throws java.rmi.RemoteException,
  com.bea.wlpi.common.WorkflowException
```

deleteReroute() メソッドのパラメータを次の表に示します。パラメータには値の指定が必要です。

表 16-4 deleteReroute() メソッドのパラメータ

パラメータ	説明	有効な値
rerouteId	削除するタスクの再ルーティングの ID	有効な再ルーティング ID を指定する文字列。 すべてのタスクの再ルーティング ID のリストを取得する方法については、16-4 ページの「タスクの再ルーティングの取得」を参照。

たとえば、次のコードでは、指定された再ルーティングが削除されます。このコード例では、admin は Admin EJB への EJBObject 参照を表します。

```
admin.deleteReroute(reroute.getId());
```

再ルーティング ID は、com.bea.wlpi.common.RerouteInfo オブジェクトである reroute と関連付けられたメソッドを使用して取得されます。reroute オブジェクトは、16-4 ページの「タスクの再ルーティングの取得」に記載のメソッドを使用して取得されます。

deleteReroute() メソッドの詳細については、Javadoc の com.bea.wlpi.server.admin.Admin を参照してください。

タスクのルーティング例

この節では、コマンドライン Studio サンプルから抜粋して、タスクのルーティングの管理方法を示します。

注意： コマンドライン Studio サンプルの詳細については、1-24 ページの「コマンドライン Studio サンプル」を参照してください。

このサンプルでは、ユーザと通信するための入力ストリームが定義されており、ユーザは以下のアクションのいずれかを指定するよう求められます。

- タスクの再ルーティングの追加
- タスクの再ルーティングの削除
- タスクの再ルーティングの取得

重要なコード行は、**太字**で強調されています。このコード例では、admin は Admin EJB への EJBObject 参照を表します。

```
/* ユーザとの通信のための入力ストリームを作成する */
stdIn = new BufferedReader( new InputStreamReader( System.in ) );

/* ツール タイトルを表示する */
System.out.print( "\n---      Command Line Studio v1.0      ---" );

/* メイン メニューを表示してユーザと交信する */
while( true ) {
/* メニューを表示する */
```

```
   System.out.println( "\n---              Main Menu            ---" );
   System.out.println( "\nEnter choice:" );
   System.out.println( "1) Templates" );
   System.out.println( "2) Task Reroutes" );
   System.out.println( "Q) Quit" );
   System.out.print( ">> " );
      .
      .
      .
/**
 * WLPI タスクの再ルーティングと関連する、Admin インタフェースで選択可能な公開 API メソッドを示す目的で
 * 必要な情報をすべて取得するために
 * ユーザと交信するメソッド
 */
public static void mngReroutes() {
   int assigneeType;
   List rerouteList;
   RerouteInfo rerouteInfo;
   String orgId;
   String rerouteId;
   String answer;
   String fromAssignee;
   String toAssignee;
   Timestamp effectiveDate;
   Timestamp expiryDate;

   /* ユーザとの通信のための入力ストリームを作成する */
   BufferedReader stdIn = new BufferedReader( new InputStreamReader(
      System.in ) );

   try {
      /* メニューを表示してユーザと交信する */
      while( true ) {
         /* メニューを表示する */
         System.out.println( "\n\n---          Task Routings          ---" );
         System.out.println( "\nEnter choice:" );
         System.out.println( "1) Add a Task Reroute" );
         System.out.println( "2) Delete a Task Reroute" );
         System.out.println( "3) List all Task Reroutes" );
         System.out.println( "B) Back to previous menu" );
         System.out.println( "Q) Quit" );
         System.out.print( ">> " );

         /* ユーザの選択を取得する */
         String line = stdIn.readLine();

         /* ユーザが選択を行わないで〔Enter〕を押したか ? */
```

```
if( line.equals( "" ) )
   continue;
   /* ユーザが複数の文字を入力したか ? */
else if( line.length() > 1 ) {
   System.out.println( "*** Invalid selection" );
   continue;
}

/* 大文字および文字へのコンバート */
char choice = line.toUpperCase().charAt( 0 );

/* ユーザの選択を処理する */
switch( choice ) {
.
.
.
```

タスクの再ルーティングの追加

タスクの再ルーティングの追加方法を示します。

```
/* タスクの再ルーティングを追加する */
case '1' :
   /* 再ルーティングを追加するオーガニゼーション ID を取得する */
   if( ( orgId = askQuestion( "\nEnter Organization ID where to
      add the Reroute" ) ) == null ) {
      /* ユーザによる操作の取り消し */
      System.out.println( "*** Cancelled" );
      break;
   }

   /* 再ルーティング元のユーザ ID をユーザに求める */
   if( ( fromAssignee = askQuestion( "Enter User to reroute from" ) )
      == null ) {
      /* ユーザによる操作の取り消し */
      System.out.println( "*** Cancelled" );
      break;
   }

   /* 再ルーティング先の受託者をユーザに求める */
   if( ( toAssignee = askQuestion( "Enter Assignee to reroute to" ) )
      == null ) {
      /* ユーザによる操作の取り消し */
      System.out.println( "*** Cancelled" );
```

```
    break;
}

/* 受託者がユーザでであるかの確認をユーザに求める */
if( ( answer = askQuestion( "- Is this a user (y/n)?" ) ) == null ) {
    /* ユーザによる操作の取り消し */
    System.out.println( "*** Cancelled" );
    break;
}

/* 応答を解析する */
if( answer.equals( "y" ) || answer.equals( "Y" ) )
    assigneeType = RerouteInfo.TYPE_USER;
else {
    /* 受託者がロール内のユーザであるかの確認をユーザに求める */
    if( ( answer = askQuestion( "- Reroute to user in role (y/n)?" )
        ) == null ) {
        /* ユーザによる操作の取り消し */
        System.out.println( "*** Cancelled" );
        break;
    }

    /* 応答を解析する */
    if( answer.equals( "y" ) || answer.equals( "Y" ) )
        assigneeType = RerouteInfo.TYPE_USERINROLE;
    else
        assigneeType = RerouteInfo.TYPE_ROLE;
}

/* 有効日付を取得する */
/* 注意 : 有効日付はタイムスタンプである。便宜上
 * 日付のみを使用する。*/
if( ( answer = askQuestion( "Enter Effective Date (yyyy-mm-dd)" )
    ) == null ) {
    /* ユーザによる操作の取り消し */
    System.out.println( "*** Cancelled" );
    break;
}

/* タイムスタンプへのコンバート */
effectiveDate = Timestamp.valueOf( answer + " 0:0:0.0" );

/* 有効期限日付を取得する */
/* 注意 : 有効期限日付はタイムスタンプである。便宜上
 *        日付のみを使用する。*/
if( ( answer = askQuestion( "Enter Expiry Date (yyyy-mm-dd)" ) )
    == null ) {
```

```
   /* ユーザによる操作の取り消し */
      System.out.println( "*** Cancelled" );
      break;
   }

   /* タイムスタンプへのコンバート */
   expiryDate = Timestamp.valueOf( answer + " 0:0:0.0" );

   try {
      /* WLPI 公開 API メソッド */
      /* タスクの再ルーティングを追加する */
      admin.addReroute( orgId, fromAssignee, toAssignee,
         assigneeType, effectiveDate, expiryDate );

      /* 正常終了（例外の発生なし）*/
      System.out.println( "- Added" );
   }
   catch( Exception e ) {
      System.out.println( "*** Unable to add Task Reroute" );
      System.err.println( e );
   }


   break;
   .
   .
   .
```

タスクの再ルーティングの削除

タスクの再ルーティングの削除方法を示します。

```
/* タスクの再ルーティングを削除する */
case '2' :
   /* 削除するタスクの再ルーティング ID を取得する */
   if( ( rerouteId = askQuestion( "\nEnter Task Reroute ID to delete"
      ) ) == null ) {
      /* ユーザによる操作の取り消し */
      System.out.println( "*** Cancelled" );
      break;
   }
```

```
   try {
      /* WLPI 公開 API メソッド */
      /* タスクの再ルーティングを削除する */
      admin.deleteReroute( rerouteId );

      /* 正常終了（例外の発生なし）*/
      System.out.println( "- Deleted" );
   }
   catch( Exception e ) {
      System.out.println( "*** Unable to delete Task Reroute" );
      System.err.println( e );
   }
   break;
```

タスクの再ルーティングの取得

タスクのルーティングの取得方法を示します。

```
/* タスクの再ルーティングをすべてリストアップする */
case '3' :
   /* クエリするオーガニゼーションのオーガニゼーション ID を取得する */
   if( ( orgId = askQuestion( "\nEnter Organization ID to list Task
      Reroutes" ) ) == null ) {
      /* ユーザによる操作の取り消し */
      System.out.println( "*** Cancelled" );
      break;
   }
   try {
      /* WLPI 公開 API メソッド */
      /* タスクの再ルーティングをすべて取り出す */
      rerouteList = admin.getReroutes( orgId );

      /* 定義されているタスクの再ルーティングはあるか ? */
      if( rerouteList.size() == 0 )
         System.out.println( "\nNo Task Reroute defined" );
      else
         System.out.println( "\nDefined Task Reroutes:" );

      /* リストを処理して、タスクの再ルーティングと属性を表示する */
      for( int i = 0; i < rerouteList.size(); i++ ) {
         /* リストから要素を取り出す */
         rerouteInfo = ( RerouteInfo )rerouteList.get( i );
```

```
            /* WLPI 公開 API メソッド */
            /* タスクの再ルーティングの ID を取り出して表示する */
            System.out.println( "- ID:       " + rerouteInfo.getId() );

            /* WLPI 公開 API メソッド */
            /* タスクが再割り当てされる割り当て先を取り出して
             * 表示する */
            System.out.println( "  From user: " + rerouteInfo.getFrom() );

            /* 再ルーティングされたタスクが割り当てられる割り当て先を取り出して
             * 表示する */
            System.out.print( "  To" );

            /* WLPI 公開 API メソッド */
            /* 再ルーティングのタイプを取り出す */
            assigneeType = rerouteInfo.getType();

            if( assigneeType == RerouteInfo.TYPE_ROLE )
               System.out.print( " role: " );
            else if( assigneeType == RerouteInfo.TYPE_USER )
               System.out.print( " user: " );
            else if( assigneeType == RerouteInfo.TYPE_USERINROLE )
               System.out.print( " user in role: " );
            else
               System.out.print( ": " );



            /* WLPI 公開 API メソッド */
            /* 再ルーティングされたタスクが割り当てられる割り当て先を取り出して
             * 表示する */
            System.out.println( rerouteInfo.getTo() );

            /* WLPI 公開 API メソッド */
            /* この再ルーティングの有効期間を取り出して
             * 表示する */
            System.out.println( "  Valid From " +
               rerouteInfo.getEffective().toString() +
               " to " + rerouteInfo.getExpiry().toString() );
      }
   }
   catch( Exception e ) {
      System.out.println( "*** Unable to retrieve Task Reroutes" );
      System.err.println( e );
   }
   break;
   .
```

.
.

17 XML リポジトリの管理

XML リポジトリは、WebLogic Integration のビジネス プロセス コンポーネントに対し、データ ストレージ機能を提供します。

この章では、XML リポジトリの管理方法について説明します。内容は以下のとおりです。

- XML リポジトリ フォルダの管理
- XML リポジトリ エンティティの管理
- EJB 環境変数の値の取得

この章に記載する各メソッドの詳細については、Javadoc の `com.bea.eci.repository.ejb.XMLRepository` を参照してください。

XML リポジトリ フォルダの管理

XML リポジトリのフォルダやサブフォルダの作成、更新、表示、および削除の方法について以下に説明します。

フォルダまたはサブフォルダの作成

XML リポジトリ内で新しいフォルダまたはサブフォルダを作成するには、次の com.bea.wlpi.eci.repository.ejb.XMLRepository メソッドを使用します。

```
public com.bea.wlpi.repository.helper.RepositoryFolderInfo createFolder (
  java.lang.String type,
  java.lang.String name,
  java.lang.String desc,
  java.lang.String notes,
  com.bea.eci.repository.helper.RepositoryFolderInfo parent
) throws com.bea.eci.repository.helper.RepositoryException,
  java.rmi.RemoteException
```

createFolder() メソッドのパラメータを次の表に示します。パラメータには値を指定する必要があります。

表 17-1 createFolder() メソッドのパラメータ

パラメータ	説明	有効な値
type	作成するフォルダのタイプ	com.bea.eci.repository.helper.Types から有効なタイプを指定する文字列。 現在有効な型は、ObjectFolder のみ。
name	作成するフォルダの名前	すべての ObjectFolder オブジェクトに対して、リポジトリ内でユニークなヌルでない文字列。
desc	作成するフォルダの説明	ヌルに設定可能な文字列。
notes	作成するフォルダのメモ	ヌルに設定可能な文字列。
parent	作成する子の親フォルダ	トップ レベルでフォルダを作成するために、既にあるフォルダまたはヌルを指定する RepositoryFolderInfo オブジェクト。

このメソッドは、com.bea.eci.repository.helper.RepositoryFolderInfo オブジェクトのリストを返します。各フォルダについての情報にアクセスするには、B-12 ページの「RepositoryFolderInfo オブジェクト」で説明する RepositoryFolder オブジェクトのメソッドを使用します。

たとえば、次のコードでは、フォルダ factoryA 内に、inventory という名前の新しいフォルダが作成されます。このコード例では、xmlrepository は XMLRepository EJB への EJBObject 参照を表します。

```
RepositoryFolderInfo newFolder = xmlrepository.createFolder(
  ObjectFolder, inventory, ”Inventory of factory items.”,
”This is a note.”, “factoryA”);
```

createFolder() メソッドの詳細については、Javadoc の com.bea.eci.repository.ejb.XMLRepository を参照してください。

すべてのフォルダ名とサブフォルダ名の取得

XML リポジトリ内のすべてのフォルダおよびサブフォルダのリスト、または特定のフォルダに関連するサブフォルダのリストを取得するには、それぞれ以下の com.bea.wlpi.eci.repository.ejb.XMLRepository メソッドを使用します。

```
public java.util.List getAllFolders(
) throws com.bea.eci.repository.helper.RepositoryException,
  java.rmi.RemoteException
```

```
public java.util.List getChildFolders(
  com.bea.eci.repository.helper.RepositoryFolderInfo rfi
) throws com.bea.eci.repository.helper.RepositoryException,
  java.rmi.RemoteException
```

getChildFolders() メソッドのパラメータを次の表に示します。パラメータには値を指定する必要があります。

表 17-2 getChildFolders() メソッドのパラメータ

パラメータ	説明	有効な値
rfi	すべてのサブフォルダを表示する場合のフォルダ	既にある RepositoryFolderInfo オブジェクト。 ヌルに設定した場合、サブフォルダでないすべてのフォルダ（つまり、フォルダの階層構造の最上位に存在するフォルダ）が取り出される。 フォルダ リストの取得については、17-3 ページの「すべてのフォルダ名とサブフォルダ名の取得」 を参照。

各メソッドは com.bea.eci.repository.helper.RepositoryFolderInfo オブジェクトを返しますが、何も定義していない場合は空のリストが返されます。各フォルダについての情報にアクセスするには、B-12 ページの「RepositoryFolderInfo オブジェクト」で説明する RepositoryFolder オブジェクトのメソッドを使用します。

たとえば、次のコードでは、すべてのフォルダのリストが取得されます。このコード例では、xmlrepository は XMLRepository EJB への EJBObject 参照を表します。

```
List folders = xmlrepository.getAllFolders();
```

次のコードでは、folderA という指定された親フォルダに対するすべてのサブフォルダのリストを取得します。

```
List subfolders = xmlrepository.getChildFolders(“folderA”);
```

getAllFolders() および getChildFolders() メソッドの詳細については、Javadoc の com.bea.eci.repository.ejb.XMLRepository を参照してください。

フォルダ ツリーの取得

XML リポジトリ内のすべてのフォルダのツリー構造を取得するには、次の com.bea.wlpi.eci.repository.ejb.XMLRepository メソッドを使用します。

```
public javax.swing.tree.DefaultMutableTreeNode getObjectFolderTree(
  javax.swing.tree.DefaultMutableTreeNode node
) throws com.bea.eci.repository.helper.RepositoryException,
  java.rmi.RemoteException
```

getObjectFolderTree() メソッドのパラメータを次の表に示します。パラメータには値を指定する必要があります。

表 17-3 getObjectFolderTree() メソッドのパラメータ

パラメータ	説明	有効な値
node	XML リポジトリのルート ノード	XML リポジトリのルートを表す DefaultMutableTreeNode オブジェクト。

このメソッドは、ツリーの最上位の階層を表す javax.swing.tree.DefaultMutableTreeNode を返します。指定された node の子フォルダが、ツリー内で最上位のフォルダになります。これらのフォルダは、それぞれの子ノードによって再帰的に送出されます。各子ノードは、DefaultMutableTreeNode オブジェクトで構成されます。

たとえば、次のコードでは、root ノードからツリーが形成されます。root は、ツリーのルートとして表示する DefaultMutableTreeNode を表します。このコード例では、xmlrepository は XMLRepository EJB への EJBObject 参照を表します。

```
javax.swing.tree.DefaultMutableTreeNode node =
   xmlrepository.getObjectFolderTree(root);
```

getObjectFolderTree メソッドの詳細については、Javadoc の com.bea.eci.repository.ejb.XMLRepository を参照してください。

フォルダ情報の取得する

XML リポジトリ内のフォルダの情報を取得するには、以下のいずれかの com.bea.wlpi.eci.repository.ejb.XMLRepository メソッドを使用します。

メソッド 1

```
public com.bea.wlpi.eci.repository.helper.RepositoryFolderInfo
getFolderInfo(
  java.lang.String name
) throws com.bea.eci.repository.helper.RepositoryException,
  java.rmi.RemoteException
```

メソッド 2

```
public com.bea.wlpi.eci.repository.helper.RepositoryFolderInfo
getFolderInfo(
  java.lang.String type,
  java.lang.String name
) throws com.bea.eci.repository.helper.RepositoryException,
  java.rmi.RemoteException
```

getFolderInfo() メソッドのパラメータを次の表に示します。パラメータには値を指定する必要があります。

表 17-4 getFolderInfo() メソッドのパラメータ

パラメータ	説明	有効な値
type	情報を取得するフォルダのタイプ	com.bea.eci.repository.helper.Types から有効なタイプを指定する文字列。 現在有効な型は、ObjectFolder のみ。
name	情報を取得するフォルダの名前	既にある ObjectFolder オブジェクト。

各メソッドは、com.bea.eci.repository.helper.RepositoryFolderInfo オブジェクトを返し、指定されたフォルダが定義されていない場合は null を返します。フォルダについての情報にアクセスするには、B-12 ページの「RepositoryFolderInfo オブジェクト」に記載の RepositoryFolder オブジェクトのメソッドを使用します。

たとえば、次のコードでは、folderA のフォルダの情報を取得します。このコード例では、xmlrepository は XMLRepository EJB への EJBObject 参照を表します。

```
com.bea.eci.repository.helper.RepositoryFolderInfo folderInfo =
  xmlrepository.getFolderInfo(“folderA”);
```

getFolderInfo() メソッドの詳細については、Javadoc の com.bea.eci.repository.ejb.XMLRepository を参照してください。

フォルダの再編

XML リポジトリ内のフォルダを再編するには、以下の com.bea.wlpi.eci.repository.ejb.XMLRepository メソッドを使用します。

```
public void addChildFolder(
  com.bea.eci.repository.helper.RepositoryFolderInfo child,
  com.bea.eci.repository.helper.RepositoryFolderInfo parent
) throws com.bea.eci.repository.helper.RepositoryException,
  java.rmi.RemoteException
```

```
public void removeChildFolder(
  com.bea.eci.repository.helper.RepositoryFolderInfo child,
  com.bea.eci.repository.helper.RepositoryFolderInfo parent
) throws com.bea.eci.repository.helper.RepositoryException,
  java.rmi.RemoteException
```

第 1 のメソッドでは、子フォルダが親フォルダに追加されます。第 2 のメソッドでは、子フォルダが親フォルダから削除されます。

注意： 親フォルダからフォルダを削除すると、XML リポジトリからも削除されます。

addChildFolder() と removeChildFolder() メソッドのパラメータを次の表に示します。パラメータには値を指定する必要があります。

表 17-5 addChildFolder() および removeChildFolder() メソッドのパラメータ

パラメータ	説明	有効な値
child	指定された親フォルダに追加するフォルダ、またはその親フォルダから削除するフォルダ	既にある RepositoryFolderInfo オブジェクト。 フォルダ リストの取得については、17-3 ページの「すべてのフォルダ名とサブフォルダ名の取得」を参照。
parent	親フォルダ	既にある RepositoryFolderInfo オブジェクト。 フォルダ リストの取得については、17-3 ページの「すべてのフォルダ名とサブフォルダ名の取得」を参照。

たとえば、次のコードでは、folderA を folderB 内に移動します。このコード例では、xmlrepository は XMLRepository EJB への EJBObject 参照を表します。

```
xmlrepository.addChildFolder(“folderA”, “folderB”);
```

同様に、次のコードでは、folderA を folderB から削除します。

```
xmlrepository.removeChildFolder(“folderA”, “folderB”);
```

addChildFolder() および removeChildFolder() メソッドの詳細については、Javadoc の com.bea.eci.repository.ejb.XMLRepository を参照してください。

フォルダ名の変更

XML リポジトリ内のフォルダ名を変更するには、次の com.bea.wlpi.eci.repository.ejb.XMLRepository メソッドを使用します。

```
public void renameFolder(
  java.lang.String curname,
  java.lang.String newName
) throws com.bea.eci.repository.helper.RepositoryException,
  java.rmi.RemoteException
```

renameFolder() メソッドのパラメータを次の表に示します。パラメータには値を指定する必要があります。

表 17-6 renameFolder() メソッドのパラメータ

パラメータ	説明	有効な値
curName	フォルダの現在名	既にあるフォルダの名前を指定する文字列。
newName	フォルダの新しい名前	新しいフォルダの固有の名前を指定する文字列。

たとえば、次のコードでは、既にあるフォルダ名の folderA が folderB に変更されます。このコード例では、xmlrepository は XMLRepository EJB への EJBObject 参照を表します。

```
xmlrepository.renameFolder(folderA, folderB);
```

renameFolder() メソッドの詳細については、Javadoc の com.bea.eci.repository.ejb.XMLRepository を参照してください。

フォルダの更新

XML リポジトリ内のフォルダの説明やノート フィールドを更新するには、次の com.bea.wlpi.eci.repository.ejb.XMLRepository メソッドを使用します。

```
public void updateFolder(
  com.bea.eci.repository.helper.RepositoryFolderInfo rfi
) throws com.bea.eci.repository.helper.RepositoryException,
  java.rmi.RemoteException
```

updateFolder() メソッドのパラメータを示します。パラメータには値を指定する必要があります。

表 17-7 updateFolder() メソッドのパラメータ

パラメータ	説明	有効な値
rfi	フォルダの名前とタイプ、および更新する情報	RepositoryFolderInfo オブジェクトは、name と type フィールドで、ユーザが更新する既にあるフォルダを識別し、description と notes フィールドで、新しい情報を設定する。

たとえば、次のコードでは、RepositoryFolderInfo オブジェクトの folderInfo で定義したように、既にあるフォルダの description と notes フィールドが更新されます。このコード例では、xmlrepository は XMLRepository EJB への EJBObject 参照を表します。

```
xmlrepository.updateFolder(folderInfo);
```

updateFolder() メソッドの詳細については、Javadoc の com.bea.eci.repository.ejb.XMLRepository を参照してください。

フォルダの削除

XML リポジトリからフォルダを削除するには、次の com.bea.wlpi.eci.repository.ejb.XMLRepository メソッドを使用します。

```
public void deleteFolder(
  com.bea.eci.repository.helper.RepositoryFolderInfo rfi
) throws com.bea.eci.repository.helper.RepositoryException,
  java.rmi.RemoteException
```

deleteFolder() メソッドのパラメータを次の表に示します。パラメータには値を指定する必要があります。

表 17-8 deleteFolder() メソッドのパラメータ

パラメータ	説明	有効な値
rfi	削除するフォルダ	削除するフォルダに関連した既にある RepositoryFolderInfo オブジェクト。 指定するフォルダに、サブフォルダを含むことはできない。 フォルダ情報の取得については、17-5 ページの「フォルダ情報の取得する」 を参照。

削除するフォルダに含まれる XML エンティティは、XML リポジトリから*削除されません*。XML エンティティは、階層構造の最上位のエンティティになります。

たとえば、次のコードでは、RepositoryFolderInfo オブジェクトの folderInfo で定義されたフォルダが削除されます。このコード例では、xmlrepository は XMLRepository EJB への EJBObject 参照を表します。

```
xmlrepository.deleteFolder(folderInfo);
```

deleteFolder() メソッドの詳細については、Javadoc の com.bea.eci.repository.ejb.XMLRepository を参照してください。

XML リポジトリ エンティティの管理

XML リポジトリ エンティティの作成、更新、表示、および削除の方法について以下に説明します。

エンティティの作成

XML リポジトリ エンティティを作成するには、次の com.bea.wlpi.eci.repository.ejb.XMLRepository メソッドを使用します。

```
public com.bea.eci.repository.helper.XMLEntityInfo createEntity(
  int type,
  java.lang.String name,
  java.lang.String desc,
  java.lang.String notes,
  java.lang.String content,
  com.bea.eci.repository.helper.RepositoryFolderInfo parent
) throws com.bea.eci.repository.helper.RepositoryException,
  java.rmi.RemoteException
```

createEntity() メソッドのパラメータを次の表に示します。パラメータには値を指定する必要があります。

表 17-9 createEntity() メソッドのパラメータ

パラメータ	説明	有効な値
type	作成するエンティティのタイプ	com.bea.eci.repository.helper.Types から有効なタイプを指定する整数。 有効な値は、以下のとおり。 ■ NO_CONTENT ―コンテンツなし（デフォルト） ■ TEXT ―テキスト ■ DTD ―文書型定義（DTD） ■ XML_SCHEMA ― XML スキーマ ■ XML ― XML ■ XSLT ― XML トランスフォーメーション文書（XML Transformation Document: XSLT） ■ MFL ―メッセージ フォーマット言語（Message Format Language: MFL）

表 17-9 createEntity() メソッドのパラメータ (続き)

パラメータ	説明	有効な値
name	作成するエンティティの名前	すべての XMLEntityInfo オブジェクトに対して、リポジトリ内でユニークなヌルでない文字列。
desc	作成するエンティティの説明	ヌルに設定可能な文字列。
notes	作成するエンティティのメモ	ヌルに設定可能な文字列。
parent	作成するエンティティの親フォルダ	RepositoryFolderInfo オブジェクト。 ヌルに設定した場合、エンティティは最上位に作成される。 フォルダ情報の取得については、17-5 ページの「フォルダ情報の取得する」を参照。

このメソッドは、新しいエンティティに対応した com.bea.eci.repository.helper.XMLEntityInfo オブジェクトを返します。エンティティに関する情報にアクセスするには、B-33 ページの「XMLEntityInfo オブジェクト」で説明する XMLEntityInfo オブジェクトのメソッドを使用します。

たとえば、次のコードでは、widgets という名前の新しい XML エンティティが、inventory フォルダ内に作成されます。このコード例では、xmlrepository は XMLRepository EJB への EJBObject 参照を表します。

```
XMLEntityInfo entity = xmlrepository.createEntity(Types.XML,
“widgets”, “widgets inventory”, “This is a note.”, inventory);
```

createFolder() メソッドの詳細については、Javadoc の com.bea.eci.repository.ejb.XMLRepository を参照してください。

エンティティ名の取得

XML リポジトリ内のすべてのエンティティのリスト、または指定されたフォルダ内のエンティティのリストを取得するには、以下の com.bea.wlpi.eci.repository.ejb.XMLRepository メソッドをそれぞれ使用します。

```
public java.util.List getAllEntities(
) throws com.bea.eci.repository.helper.RepositoryException,
  java.rmi.RemoteException

public java.util.List getChildDocs(
  com.bea.eci.repository.helper.RepositoryFolderInfo rfi
) throws com.bea.eci.repository.helper.RepositoryException,
  java.rmi.RemoteException
```

getChildDocs() メソッドのパラメータを次の表に示します。パラメータには値を指定する必要があります。

表 17-10 getChildDocs() メソッドのパラメータ

パラメータ	説明	有効な値
rfi	エンティティをリストするフォルダ	エンティティをリストするフォルダに関連した RepositoryFolderInfo オブジェクト。 この値をヌルに設定すると、フォルダに*含まれ*ないすべてのエンティティが取り出される。 フォルダ情報の取得については、17-5 ページの「フォルダ情報の取得する」 を参照。

各メソッドは com.bea.eci.repository.helper.XMLEntityInfo オブジェクトのリストを返しますが、エンティティを定義しなかった場合は空きリストが返されます。XMLEntityInfo オブジェクトには、エンティティのオブジェクトの内容は含まれず、名前、型およびメタデータのみが含まれます。各エンティティの情報にアクセスするには、B-33 ページの「XMLEntityInfo オブジェクト」 で説明する XMLEntityInfo オブジェクトのメソッドを使用します。

たとえば、次のコードでは、すべてのエンティティのリストを取得します。このコード例では、xmlrepository は XMLRepository EJB への EJBObject 参照を表します。

```
List entities = xmlrepository.getAllEntities();
```

次のコードでは、folderA 内のすべてのエンティティのリストを取得します。

```
List entities = xmlrepository.getChildDocs("folderA");
```

getAllEntities() と getChildDocs() メソッドの詳細については、Javadoc の com.bea.eci.repository.ejb.XMLRepository を参照してください。

エンティティ情報の取得

XML リポジトリ内のエンティティに関する情報を取得するには、以下のいずれかの com.bea.wlpi.eci.repository.ejb.XMLRepository メソッドを使用します。

メソッド 1

```
public com.bea.wlpi.eci.repository.helper.XMLEntityInfo getEntity(
  java.lang.String name
) throws com.bea.eci.repository.helper.RepositoryException,
  java.rmi.RemoteException
```

メソッド 2

```
public com.bea.wlpi.eci.repository.helper.XMLEntityInfo getEntity(
  java.lang.String name,
  int type
) throws com.bea.eci.repository.helper.RepositoryException,
  java.rmi.RemoteException
```

第 1 のメソッドでは、単一コンテンツのフィールドのエンティティがサポートされます。複数コンテンツのフィールドが存在する場合、このメソッドは検出した最初のタイプのコンテンツのみを返します。第 2 のメソッドでは、複数コンテンツのフィールドのエンティティがサポートされます。これにより、取り出すコンテンツのタイプが指定可能になります。

getEntity() メソッドのパラメータを次の表に示します。パラメータには値を指定する必要があります。

表 17-11 getEntity() メソッドのパラメータ

パラメータ	説明	有効な値
name	情報を取得するエンティティの名前	有効な XMLEntityInfo オブジェクトの名前を指定する文字列。

表 17-11 getEntity() メソッドのパラメータ (続き)

パラメータ	説明	有効な値
type	情報を取得するエンティティのタイプ	com.bea.eci.repository.helper.Types から有効なタイプを指定する整数。 有効な値は、以下のとおり。 ■ NO_CONTENT ー コンテンツなし（デフォルト） ■ TEXT ー テキスト ■ DTD ー 文書型定義（DTD） ■ XML_SCHEMA ー XML スキーマ ■ XML ー XML ■ XSLT ー XML トランスフォーメーション文書（XSLT） ■ MFL ー メッセージ フォーマット言語（MFL）

各メソッドは、com.bea.eci.repository.helper.XMLEntityInfo オブジェクトを返し、指定されたエンティティが定義されていない場合は null を返します。エンティティに関する情報にアクセスするには、B-33 ページの「XMLEntityInfo オブジェクト」で説明する XMLEntityInfo オブジェクトのメソッドを使用します。

たとえば、以下のコードでは、entityA に関するエンティティの情報を取得します。このコード例では、xmlrepository は XMLRepository EJB への EJBObject 参照を表します。

```
com.bea.eci.repository.helper.XMLEntityInfo entityInfo =
  xmlrepository.getEntity("entityA");
```

getEntity() メソッドの詳細については、Javadoc の com.bea.eci.repository.ejb.XMLRepository を参照してください。

フォルダ内のエンティティの編成

フォルダ内のエンティティを XML リポジトリ内で編成するには、以下の `com.bea.wlpi.eci.repository.ejb.XMLRepository` メソッドを使用します。

```
public void addEntityToFolder(
  com.bea.eci.repository.helper.XMLEntityInfo xei,
  com.bea.eci.repository.helper.RepositoryFolderInfo rfi
) throws com.bea.eci.repository.helper.RepositoryException,
  java.rmi.RemoteException

public void removeEntityFromFolder(
  com.bea.eci.repository.helper.XMLEntityInfo xei,
  com.bea.eci.repository.helper.RepositoryFolderInfo rfi
) throws com.bea.eci.repository.helper.RepositoryException,
  java.rmi.RemoteException
```

第 1 のメソッドでは、エンティティがフォルダに追加されます。第 2 のメソッドでは、エンティティがフォルダから削除されます。

注意： あるエンティティをフォルダから削除すると、XML リポジトリからも削除されます。

`addEntityToFolder()` と `removeEntityFromFolder()` メソッドのパラメータを次の表に示します。パラメータには値を指定する必要があります。

表 17-12 addEntityToFolder() および removeEntityFromFolder() メソッドのパラメータ

パラメータ	説明	有効な値
`xei`	指定された親フォルダに追加するエンティティ、またはその親フォルダから削除するエンティティ	既にある `XMLEntityInfo` オブジェクト。 エンティティ リストの取得については、17-12 ページの「エンティティ名の取得」を参照。
`rfi`	親フォルダ	既にある `RepositoryFolderInfo` オブジェクト。 フォルダ リストの取得については、17-3 ページの「すべてのフォルダ名とサブフォルダ名の取得」を参照。

たとえば、次のコードでは、entityA を folderA 内に移動します。このコード例では、xmlrepository は XMLRepository EJB への EJBObject 参照を表します。

```
xmlrepository.addEntityToFolder(“entityA”, “folderA”);
```

同様に、次のコードでは、entityA を folderA から削除します。

```
xmlrepository.removeEntityFromFolder(“entityA”, “folderA”);
```

addEntityToFolder() と removeEntityFromFolder() メソッドの詳細については、Javadoc の com.bea.eci.repository.ejb.XMLRepository を参照してください。

エンティティ名の変更

XML リポジトリ内のエンティティ名を変更するには、次の com.bea.wlpi.eci.repository.ejb.XMLRepository メソッドを使用します。

```
public void renameEntity(
  java.lang.String curname,
  java.lang.String newName
) throws com.bea.eci.repository.helper.RepositoryException,
  java.rmi.RemoteException
```

renameEntity() メソッドのパラメータを次の表に示します。パラメータには値を指定する必要があります。

表 17-13 renameEntity() メソッドのパラメータ

パラメータ	説明	有効な値
curName	エンティティの現在名	既にあるエンティティの名前を指定する文字列。
newName	エンティティの新しい名前	新しいエンティティにユニークな名前を指定する文字列。

たとえば、次のコードでは、既にあるエンティティ名の entityA が entityB に変更されます。このコード例では、xmlrepository は XMLRepository EJB への EJBObject 参照を表します。

```
xmlrepository.renameEntity(entityA, entityB);
```

renameEntity() メソッドの詳細については、Javadoc の com.bea.eci.repository.ejb.XMLRepository を参照してください。

エンティティの更新

XML リポジトリ内のエンティティに対して、その説明、メモおよびコンテンツを更新するには、次の com.bea.wlpi.eci.repository.ejb.XMLRepository メソッドを使用します。

```
public void updateEntity(
  com.bea.eci.repository.helper.XMLEntityInfo xei
) throws com.bea.eci.repository.helper.RepositoryException,
  java.rmi.RemoteException
```

updateEntity() メソッドのパラメータを次の表に示します。パラメータには値を指定する必要があります。

表 17-14 updateEntity() メソッドのパラメータ

パラメータ	説明	有効な値
xei	更新するエンティティと情報	XMLRepositoryInfo オブジェクトは、name と type フィールドで、ユーザが更新する既にあるエンティティを識別し、description、notes および contents フィールドで新しい情報を設定する。

エンティティが複数のコンテンツのタイプを持つかどうかに関係なく、ユーザは XMLRepositoryInfo オブジェクトの type フィールドで指定したタイプに対応する単一コンテンツのフィールドのみを更新できます。

たとえば、次のコードでは、XMLRepository オブジェクトの entityInfo で定義したように、既にあるエンティティの description、notes および contents フィールドが更新されます。このコード例では、xmlrepository は XMLRepository EJB への EJBObject 参照を表します。

```
xmlrepository.updateEntity(entityInfo);
```

updateEntity() メソッドの詳細については、Javadoc の com.bea.eci.repository.ejb.XMLRepository を参照してください。

エンティティの削除

XML リポジトリからエンティティを削除するには、次の `com.bea.wlpi.eci.repository.ejb.XMLRepository` メソッドを使用します。

```
public void deleteEntity(
  com.bea.eci.repository.helper.XMLEntityInfo xei
) throws com.bea.eci.repository.helper.RepositoryException,
  java.rmi.RemoteException
```

`deleteEntity()` メソッドのパラメータを次の表に示します。パラメータには値を指定する必要があります。

表 17-15 deleteEntity() メソッドのパラメータ

パラメータ	説明	有効な値
xei	削除するエンティティ	削除するエンティティに関連した既にある `XMLEntityInfo` オブジェクト。エンティティ情報の取得については、17-14 ページの「エンティティ情報の取得」を参照。 `XMLEntityInfo` オブジェクト内で唯一の必要なフィールドは、エンティティをユニークに識別する `name` フィールドである。

たとえば、次のコードでは、`XMLEntityInfo` オブジェクトの `entityInfo` で定義されたフォルダが削除されます。このコード例では、`xmlrepository` は `XMLRepository` EJB への `EJBObject` 参照を表します。

```
xmlrepository.deleteEntity(entityInfo);
```

`deleteEntity()` メソッドの詳細については、Javadoc の `com.bea.eci.repository.ejb.XMLRepository` を参照してください。

EJB 環境変数の値の取得

XML リポジトリとの接続に用いられる EJB 環境変数のリストを取得するには、次の `com.bea.wlpi.eci.repository.ejb.XMLRepository` メソッドを使用します。

```
public java.util.List getEnvVars(
) throws java.rmi.RemoteException
```

このメソッドは、リポジトリへの接続に用いられる環境変数の値を、指定された順序に並べた以下のリストを返します。

- 接続
- トランザクション データ ソース名
- JDBC クラス
- JDBC URL
- JDBC サーバ

たとえば、以下のコードでは、使用している環境変数のリストが取得されます。このコード例では、`xmlrepository` は `XMLRepository` EJB への `EJBObject` 参照を表します。

```
List envVars = xmlrepository.getEnvVars();
```

`getEnvVars()` メソッドの詳細については、Javadoc の `com.bea.eci.repository.ejb.XMLRepository` を参照してください。

18 ワークフロー オブジェクトの発行

この章では、発行可能なオブジェクト、およびワークフロー オブジェクトの発行方法について説明します。内容は以下のとおりです。

- 発行可能オブジェクトについて
- パッケージ エントリの作成
- 発行可能オブジェクトのパッケージのエクスポート
- 発行可能オブジェクトのパッケージのインポート
- 発行可能オブジェクトのパッケージの読み取り

この章に記載するメソッドの詳細については、Javadoc の `com.bea.wlpi.server.admin.Admin` を参照してください。WebLogic Integration Studio を使用したワークフロー オブジェクトの発行については、『*WebLogic Integration Studio ユーザーズ ガイド*』の「ワークフロー テンプレートの定義」を参照してください。

発行可能オブジェクトについて

発行可能オブジェクトは、`com.bea.wlpi.common.Publishable` インタフェースを実装する任意のオブジェクトで構成されます。`Publishable` インタフェースは、パッケージの作成、エクスポート、およびインポートをサポートします。

WebLogic Integration Studio またはカスタム設計クライアントを使用することにより、有効な*発行可能*オブジェクトであるワークフロー オブジェクトの作成、エクスポート、およびインポートが可能になります。

`Publishable` インタフェースで定義された発行可能オブジェクトを、その対応するタイプと共に次の表に示します。

表 18-1 発行可能オブジェクトおよびタイプ

発行可能オブジェクト	タイプ
BusinessCalendarInfo	BUSINESS_CALENDAR
ClassInvocationDescriptor	BUSINESS_OPERATION
EJBInvocationDescriptor	BUSINESS_OPERATION
EventKeyInfo	EVENT_KEY
OrganizationInfo	ORG
RepositoryFolderInfoHelper	XML_REPOSITORY_FOLDER
RoleInfo	ROLE
TemplateDefinitionInfo	TEMPLATE_DEFINITION
TemplateInfo	TEMPLATE
XMLEntityInfoHelper	XML_REPOSITORY_ENTITY
UserInfo	USER

Publishable インタフェースが提供するメソッドの中で、発行可能オブジェクトの情報を取得するために使用できるメソッドを次の表に示します。

表 18-2 Publishable インタフェース メソッド

メソッド	説明
`public java.lang.Object getContents()`	発行可能オブジェクトの内容を取得する。
`public java.lang.String getEntryName()`	発行可能オブジェクトのエントリ名を取得する。
`public java.lang.String getOwnerName()`	発行可能オブジェクトのオーナ名を取得する。
`public java.util.List getReferencedPublishables( java.util.Map publishables )`	指定された発行可能エンティティのマップ オブジェクトを基本にして、参照するすべての発行可能エンティティを取得する。
`public int getType()`	発行可能オブジェクトのタイプを取得する。有効なタイプのリストについては、18-2 ページの「発行可能オブジェクトおよびタイプ」の表を参照。

詳細については、Javadoc の `com.bea.wlpi.common.Publishable` を参照してください。

パッケージ エントリの作成

エクスポートの準備でパッケージ エントリを作成するには、`com.bea.wlpi.common.PackageEntry` オブジェクトを作成する必要があります。

次のコード例では、コンストラクタに対して新しい PackageEntry オブジェクトを作成します。

```
public PackageEntry(
  com.bea.wlpi.common.Publishable p,
  java.util.Map r,
  boolean b
)
```

PackageEntry オブジェクトのデータ、データの定義に使用するコンストラクタ パラメータ、およびオブジェクトの定義後にそのデータへのアクセスに使用できる取得メソッドと設定メソッドを次の表に示します。

表 18-3 PackageEntry オブジェクト データ

オブジェクト データ	コンストラクタ パラメータ	取得メソッド	設定メソッド
発行可能オブジェクト 有効な発行可能オブジェクトのリストについては、18-2 ページの「発行可能オブジェクトについて」 を参照。	p	public com.bea.wlpi.common.Publishable getPublishable()	public void setPublishable(com.bea.wlpi.common.Publishable p)
参照する発行可能オブジェクト 発行可能オブジェクト データの収集については、18-3 ページの「Publishable インタフェース メソッド」 の表を参照。	r	public java.util.Map getReferences()	public void setReferences(java.util.Map r)
発行可能オブジェクトをロックするかどうかを指定するブール フラグ	b	public boolean getPublished()	public void setPublished(boolean *b*)
エントリのタイプ 有効なタイプのリストについては、18-2 ページの「発行可能オブジェクトおよびタイプ」の表を参照。	なし	public java.lang.String getEntryType()	なし

詳細については、Javadoc の com.bea.wlpi.common.PackageEntry を参照してください。

発行可能オブジェクトのパッケージのエクスポート

発行可能オブジェクトのパッケージを JAR ファイルにエクスポートするには、次の com.bea.wlpi.server.admin.Admin メソッドを使用します。

```
public byte[ ] exportPackage(
  com.bea.wlpi.common.PublishPackage publishables,
  java.lang.Object credential
) throws java.rmi.RemoteException,
  com.bea.wlpi.common.WorkflowException
```

exportPackage() メソッドのパラメータを次の表に示します。パラメータには値を指定する必要があります。

表 18-4 exportPackage() メソッドのパラメータ

パラメータ	説明	有効な値
publishables	エクスポートするワークフロー発行可能オブジェクト	Jar パッケージにエクスポートされる、エントリ タイプ com.bea.wlpi.common.PackageEntry を含むオブジェクト タイプ com.bea.wlpi.common.PublishPackage 発行可能エンティティの有効なタイプのリストについては、18-2 ページの「発行可能オブジェクトおよびタイプ」の表を参照。
credential	パスワードの値（必要な場合）	有効なパスワードを指定する、またはパスワードが不要な場合は null 値を指定する java.lang.Object オブジェクト。

このメソッドは、エクスポートしたすべてのオブジェクトを含む JAR ファイルのイメージである byte[ ] の配列を返します。

たとえば、次のコードでは、前回定義した発行可能な Map オブジェクトで指定された発行可能オブジェクトのパッケージをエクスポートします。このコード例では、admin は Admin EJB への EJBObject 参照を表します。

```
byte[ ] exportResults = admin.exportPackage(publishableMap);
```

exportPackage() メソッドの詳細については、Javadoc の com.bea.wlpi.server.admin.Admin を参照してください。

発行可能オブジェクトのパッケージのインポート

発行可能オブジェクトのパッケージをインポートするには、次の com.bea.wlpi.server.admin.Admin メソッドを使用します。

```
public java.lang.String importPackage(
  byte[ ] pkg,
  java.util.Map publishables,
  java.lang.String orgId,
  boolean activate,
  java.lang.Object credential
) throws java.rmi.RemoteException,
  com.bea.wlpi.common.WorkflowException
```

importPackage() メソッドのパラメータを次の表に示します。パラメータには値を指定する必要があります。

表 18-5 importPackage() メソッドのパラメータ

パラメータ	説明	有効な値
pkg	インポートする発行可能オブジェクトの JAR ファイル	発行可能オブジェクトを含む JAR ファイルのイメージで構成された、有効な byte[ ] オブジェクト。 発行可能オブジェクトのパッケージのエクスポートについては、18-5 ページの「発行可能オブジェクトのパッケージのエクスポート」を参照。

表 18-5 importPackage() メソッドのパラメータ (続き)

パラメータ	説明	有効な値
publishables	インポートするワークフロー発行可能オブジェクト	発行可能エンティティのタイプをキーとして指定し、PackageEntry オブジェクトを値として指定する「*key-value*」ペア付きの java.util.Map オブジェクト。 発行可能エンティティの有効なタイプのリストについては、18-2 ページの「発行可能オブジェクトおよびタイプ」の表を参照。
orgId	パッケージをインポートするオーガニゼーションの ID	有効なオーガニゼーション ID を指定する文字列。 オーガニゼーション ID のリストの取得については、9-9 ページの「すべてのオーガニゼーション名を取得する」を参照。
activate	インポート時にパッケージをアクティブ化するかどうかを指定するブール フラグ	true (アクティブ化する) または false (アクティブ化しない)。
credential	パスワードの値（必要な場合）	有効なパスワードを指定する、またはパスワードが不要の場合は null 値を指定する java.lang.Object オブジェクト。

このメソッドは、未解決の参照など、報告された問題を含む文字列の値を返します。

たとえば、次のコードでは、shipping.jar ファイルと publishPackage マップ内で指定したとおり、発行可能オブジェクトを ORG1 オーガニゼーションにインポートします。このコード例では、admin は Admin EJB への EJBObject 参照を表します。

```
//JAR ファイルをバイト配列に読み込む
File jarFile = new File(“shipping.jar”);
pkg = new byte[(int)jarFile.length()];
FileInputStream fin = new FileInputStream(jarFile);
fin.read(pkg);
fin.close();
```

```
// コンテンツをマッピングしてインポートする
Map contents = admin.readPackage(pkg, “password”);
String importResults = admin.importPackage(
  pkg, contents, “ORG1”, true, “password”);
```

発行可能オブジェクトはインポート時にアクティブ化され（アクティブ化の値は `true` に設定される）、`abcd` がパスワードとして使用されます。

`importPackage()` メソッドの詳細については、Javadoc の `com.bea.wlpi.server.admin.Admin` を参照してください。

発行可能オブジェクトのパッケージの読み取り

発行可能オブジェクトのパッケージを読むには、次の `com.bea.wlpi.server.admin.Admin` メソッドを使用します。

```
public java.util.Map readPackage(
  byte[ ] pkg,
  java.lang.Object credential
) throws java.rmi.RemoteException,
  com.bea.wlpi.common.WorkflowException
```

`readPackage()` メソッドのパラメータを次の表に示します。パラメータには値を指定する必要があります。

表 18-6 readPackage() メソッドのパラメータ

パラメータ	説明	有効な値
`pkg`	読み取りする発行可能オブジェクトの JAR ファイル	発行可能オブジェクトを含む JAR ファイルのイメージで構成された、有効な `byte[ ]` オブジェクト。 発行可能オブジェクトのパッケージのエクスポートについては、18-5 ページの「発行可能オブジェクトのパッケージのエクスポート」を参照。

表 18-6 readPackage() メソッドのパラメータ (続き)

パラメータ	説明	有効な値
credential	パスワードの値（必要な場合）	有効なパスワードを指定する、またはパスワードが不要な場合は null 値を指定する java.lang.Object オブジェクト。

このメソッドは、発行可能エンティティのタイプをキーで指定し、また（発行可能エンティティのタイプに対応した *xxx*Info オブジェクトを含む）異機種間の java.util.List オブジェクトを値として指定する、「*key-value*」付きの java.util.Map オブジェクトを返します。有効な発行可能オブジェクトとそれらのタイプのリストについては、18-2 ページの「発行可能オブジェクトおよびタイプ」の表を参照してください。

たとえば、次のコードでは、パスワードの abcd を使用して、shipping.jar ファイル内で指定された発行可能オブジェクトを読み込みます。このコード例では、admin は Admin EJB への EJBObject 参照を表します。

```
byte[ ] pkg = readFile(“shipping.jar”);
Map contents = admin.readPackage(pkg, “abcd”);
```

readPackage() メソッドの詳細については、Javadoc の com.bea.wlpi.server.admin.Admin を参照してください。

Part IV　実行時の管理

19 アクティブなオーガニゼーションの管理

この章では、アクティブ オーガニゼーションの管理方法について説明します。内容は以下のとおりです。

- アクティブ オーガニゼーションについて
- アクティブ オーガニゼーション名の取得
- すべてのオーガニゼーション名の取得
- アクティブ オーガニゼーションの設定
- アクティブ オーガニゼーションの管理例

WebLogic Integration Worklist を使用したアクティブ オーガニゼーションの管理については、『*WebLogic Integration Worklist ユーザーズ ガイド*』の「ワークフローを使った作業」を参照してください。

注意： Worklist クライアント アプリケーションは、このリリースの WebLogic Integration からは非推奨になります。Worklist クライアント アプリケーションに代わる機能については、『*WebLogic Integration リリース ノート*』を参照してください。

アクティブ オーガニゼーションについて

オーガニゼーションは、WebLogic Integration Studio またはカスタム構成クライアントを使用して、異なるビジネス エンティティ、地理的な位置、または会社の特定のビジネスに関連する他のクラスのエンティティを表すものとして定義されます。また、セキュリティのパーミッションを強化するために、オーガニゼーション内のユーザや役割も定義できます。

現在のアクティブ オーガニゼーションは、オーガニゼーションが指定されていないワークリストおよびカスタム実行時の管理クライアント要求の両方のデフォルト オーガニゼーションになっています。アクティブ オーガニゼーションは、まず実行時の管理クライアントを呼び出したユーザのデフォルト オーガニゼーションに設定されます。デフォルトのオーガニゼーションは、ユーザが定義クライアントで最初に作成する際に指定します。

アクティブ オーガニゼーションの取得と設定は、以下に説明するメソッドを使用して行います。

以下の節で説明するそれぞれのメソッドについては、『*BEA WebLogic Integration Javadoc*』を参照してください。Studio クライアント内でのオーガニゼーションの維持については、『*WebLogic Integration Studio ユーザーズ ガイド*』の「データの管理」の「オーガニゼーションの保守」を参照してください。

アクティブ オーガニゼーション名の取得

現在のアクティブ オーガニゼーションの名前を取得するには、次の `com.bea.wlpi.server.worklist.Worklist` メソッドを使用します。

```
public java.lang.String getActiveOrganization(
) throws java.rmi.RemoteException,
  com.bea.wlpi.common.WorkflowException
```

このメソッドは、アクティブ オーガニゼーションの ID を戻します。

たとえば、次のコードでは、アクティブ オーガニゼーションの ID が取得され、またそれが `activeOrgId` 文字列の変数に格納されます。

```
String activeOrgId = worklist.getActiveOrganization();
```

このコード例では、`worklist` は `Worklist` EJB への `EJBObject` 参照を表します。`getActiveOrganization()` メソッドの詳細については、Javadoc の `com.bea.wlpi.server.worklist.Worklist` を参照してください。

すべてのオーガニゼーション名の取得

アクティブ オーガニゼーションを設定するには、アクティブ化するオーガニゼーションの ID を知る必要があります。定義されたすべてのオーガニゼーションの ID のリストを取得するには、9-9 ページの「すべてのオーガニゼーション名を取得する」 を参照してください。

アクティブ オーガニゼーションの設定

アクティブ オーガニゼーションを設定するには、次の `com.bea.wlpi.server.worklist.Worklist` メソッドを使用します。

```
public void setActiveOrganization(
  java.lang.String orgId
) throws java.rmi.RemoteException,
  com.bea.wlpi.common.WorkflowException
```

`setActiveOrganization()` メソッドのパラメータを次の表に示します。パラメータには値を指定する必要があります。

表 19-1 setActiveOrganization() メソッドのパラメータ

パラメータ	説明	有効な値
`orgId`	アクティブ化するオーガニゼーションの ID	有効なオーガニゼーション ID を指定する文字列。 すべてのオーガニゼーション ID のリストの取得方法については、9-9 ページの「すべてのオーガニゼーション名を取得する」 を参照。

たとえば、次のコードでは、ORG1 がアクティブ オーガニゼーションに設定されます。このコード例では、worklist は Worklist EJB への EJBObject 参照を表します。

```
worklist.setActiveOrganization(“ORG1”);
```

setActiveOrganization() メソッドの詳細については、Javadoc の com.bea.wlpi.server.worklist.Worklist を参照してください。

アクティブ オーガニゼーションの管理例

この節では、コマンドライン Worklist サンプルから抜粋して、アクティブ オーガニゼーションの管理方法を示します。

注意： コマンドライン Worklist サンプルの詳細については、1-26 ページの「コマンドライン Worklist サンプル」 を参照してください。

このサンプルでは、ユーザと通信するための入力ストリームが定義されており、ユーザは以下のアクションのいずれかを指定するよう求められます。

- アクティブ オーガニゼーション名を取得する
- すべてのオーガニゼーション名を取得する
- アクティブ オーガニゼーションを設定する

重要なコード行は、**太字**で強調されています。このコード例では、worklist は Worklist EJB への EJBObject 参照を表します。

```
/* ユーザとの通信のための入力ストリームを作成する */
stdIn = new BufferedReader( new InputStreamReader( System.in ) );

/* ツール タイトルを表示する */
System.out.print( "\n---  Command Line Worklist v1.2  ---" );

/* メイン メニューを表示してユーザと交信する */
while( true ) {
   /* メニューを表示する */
   System.out.println( "\n---           Main Menu          ---" );
   System.out.println( "\nEnter choice:" );
   System.out.println( "1) Organizations" );
   System.out.println( "2) Workflows" );
```

```
   System.out.println( "3) Tasks" );
   System.out.println( "Q) Quit" );
   System.out.print( ">> " );
      .
      .
      .
public static void mngOrganization( ) {
   String orgId;

   /* ユーザとの通信のための入力ストリームを作成する */
   BufferedReader stdIn = new BufferedReader(
      new InputStreamReader( System.in ) );

   try {
      /* メイン メニューを表示してユーザと交信する */
      while( true ) {
         /* メニューを表示する */
         System.out.println( "\n\n---         Organizations        ---" );
         System.out.println( "\nEnter choice:" );
         System.out.println( "1) List active organization" );
         System.out.println( "2) List all organizations" );
         System.out.println( "3) Set active organization" );
         System.out.println( "B) Back to previous menu" );
         System.out.println( "Q) Quit" );
         System.out.print( ">> " );

         /* ユーザの選択を取得する */
         String line = stdIn.readLine( );

         /* ユーザが選択を行わないで〔Enter〕を押したか ? */
         if( line.equals( "" ) )
            continue;
         /* ユーザが複数の文字を入力したか ? */
         else if( line.length( ) > 1 ) {
            System.out.println( "*** Invalid choice" );
            continue;
         }

         /* 大文字および文字へのコンバート */
         char choice = line.toUpperCase( ).charAt( 0 );

         /* ユーザの選択を処理する */
         switch( choice ) { ...
```

アクティブ オーガニゼーション名を取得する

アクティブ オーガニゼーションの名前を取得する方法を示します。

```
/* アクティブ オーガニゼーションをリスト アップする */
case '1' :
   /* WLPI 公開 API メソッド */
   /* 注意 : 発生した例外を取り込むコードを
    * 追加するとよい */
   String activeOrgId = worklist.getActiveOrganization( );
   System.out.println( "\nActive organization is " + activeOrgId );
   break; ...
```

すべてのオーガニゼーション名を取得する

すべてのオーガニゼーションの名前を取得する方法を示します。

```
/* すべてのオーガニゼーションをリスト アップする */
case '2' :
   /* WLPI 公開 API メソッド */
   /* 注意 : 発生した例外を取り込むコードを追加するとよい */
   List orgList = principal.getAllOrganizations( false );

   /* 定義されているオーガニゼーションはあるか ? */
   if( orgList.size( ) == 0 )
      System.out.println( "\nNo Organization defined" );
   else
      System.out.println( "\nDefined organizations:" );

   /* リストを処理してオーガニゼーションと属性を表示する */
   for( int i = 0; i < orgList.size( ); i++ ) {
      /* リストから要素を取り出す */
      OrganizationInfo orgInfo = ( OrganizationInfo )orgList.get( i );
      /* オーガニゼーション ID を取り出して表示する */
      System.out.println( "- ID: " + orgInfo.getOrgId( ) );
   }
   break; ...
```

アクティブ オーガニゼーションを設定する

アクティブ オーガニゼーションの設定方法を示します。

```
/* アクティブ オーガニゼーションを設定する */
case '3' :
   /* アクティブ化するオーガニゼーションのオーガニゼーション ID を取得する */
   if( ( orgId = askQuestion( "\nEnter Organization ID" ) ) == null ) {
      /* ユーザによる操作の取り消し */
      System.out.println( "*** Cancelled" );
      break;
   }

   try {
      /* WLPI 公開 API メソッド */
      worklist.setActiveOrganization( orgId );

      /* 正常終了（例外の発生なし）*/
      System.out.println( "- Success" );

      /* WLPI 公開 API メソッド */
      /* 操作が正常に終了したことを確認 */
      activeOrgId = worklist.getActiveOrganization( );
      System.out.println(
         "- The active organization is now " + activeOrgId );
   }
   catch( Exception e ) {
     System.out.println(
         "*** Failed to set the Active Organization (ID: " +
         orgId + ")" );
      System.err.println( e );
   }
   break;
      .
      .
      .
```

20 手動によるワークフローの開始

この章では、ワークフローを手動で開始する方法について説明します。内容は以下のとおりです。

- 開始可能なワークフローの取得
- 手動によるワークフローの開始
- 手動によるワークフロー開始のサンプル

注意： ワークフローの開始は、外部またはタイムアウト イベント、または 1 つのワークフローを別のワークフローから開始する ワークフロー開始アクション `ActionStartWorkflow` のいずれかのメカニズムを使って自動化することもできます。後者のメソッドでは、呼び出したワークフローがこれまでに開始されたかどうかをビジネス ルールを使って判断できるため、制御の幅が広くなります。またこのメカニズムにより、XML の内容を使用する変数の設定、および複数のワークフローの開始が可能になります。ワークフローの開始に使用できるアクションの詳細については、A-53 ページの「テンプレート定義 DTD」、または『*WebLogic Integration Worklist ユーザーズ ガイド*』の「アクションの定義」を参照してください。

WebLogic Integration Worklist を使ってワークフローを手動で開始する場合の説明は、『*WebLogic Integration Worklist ユーザーズ ガイド*』の「ワークフローを使った作業」を参照してください。

注意： Worklist クライアント アプリケーションは、このリリースの WebLogic Integration からは非推奨になります。Worklist クライアント アプリケーションに代わる機能については、『*WebLogic Integration リリース ノート*』を参照してください。

開始可能なワークフローの取得

ワークリストまたはカスタムの実行時管理クライアントから手動で開始可能なすべてのワークリストを取得するには、次の `com.bea.wlpi.server.worklist.Worklist` メソッドを使用します。

```
public java.util.List getStartableWorkflows(
  java.lang.String orgId
) throws java.rmi.RemoteException,
  com.bea.wlpi.common.WorkflowException
```

`getStartableWorkflows()` メソッドのパラメータを次の表に示します。

表 20-1 getStartableWorkflows() メソッドのパラメータ

パラメータ	説明	有効な値
`orgId`	開始可能なワークフローの名前を取得するオーガニゼーションの ID	有効なオーガニゼーション ID を指定する文字列。 すべてのオーガニゼーション ID のリストの取得方法については、19-1 ページの「アクティブなオーガニゼーションの管理」を参照。

このメソッドは、以下の `com.bea.wlpi.common.TemplateInfo` オブジェクトのリストを返します。

- 手動で開始可能なテンプレート定義に応答するオブジェクト。
- 現在アクティブ状態のオブジェクト。

そのテンプレート定義情報にアクセスするには、B-27 ページの「TemplateInfo オブジェクト」に記載の `TemplateInfo` オブジェクトのメソッドを使用します。

たとえば、次のコードでは、オーガニゼーション ID の `ORG1` で指定したオーガニゼーションに対して開始可能なすべてのワークフローのリストが取得されます。このコード例では、`worklist` は `Worklist` EJB への `EJBObject` 参照を表します。

```
List template = worklist.getStartableWorkflows(“ORG1”);
```

`getStartableWorkflows()` メソッドの詳細については、Javadoc の `com.bea.wlpi.server.worklist.Worklist` を参照してください。

手動によるワークフローの開始

ワークフローを手動で開始するには、以下のいずれかの `com.bea.wlpi.server.worklist.Worklist` メソッドを使用します。

メソッド 1

```
public java.lang.String instantiateWorkflow(
  java.lang.String orgId,
  java.lang.String templateId
) throws java.rmi.RemoteException,
  com.bea.wlpi.common.WorkflowException
```

メソッド 2

```
public java.lang.String instantiateWorkflow(
  java.lang.String orgId,
  java.lang.String templateId,
  java.lang.String xml,
  java.util.Map initialValues,
  java.util.Map pluginData
) throws java.rmi.RemoteException,
  com.bea.wlpi.common.WorkflowException
```

メソッド 3

```
public java.lang.String instantiateWorkflow(
  java.lang.String orgId,
  java.lang.String templateId,
  java.lang.Object transactionId
) throws java.rmi.RemoteException,
  com.bea.wlpi.common.WorkflowException
```

メソッド 4

```
public java.lang.String instantiateWorkflow(
  java.lang.String orgId,
  java.lang.String templateId,
  java.lang.String xml,
  java.util.Map initialValues,
  java.util.Map pluginData,
  java.lang.Object transactionId
) throws java.rmi.RemoteException,
  com.bea.wlpi.common.WorkflowException
```

第 1 のメソッドは、非クラスタ化環境で使用できます。クラスタ化環境では第 2 のメソッドを使用することをお勧めします。この場合、トランザクションがコミットされた後、またはサーバ クラッシュやフェイル オーバーが発生した後に、メソッドが再発行されないよう、指定された ID を使用してメソッドの実行状態が追跡されます。

instantiateWorkflow() メソッドのパラメータを次の表に示します。

表 20-2 instantiateWorkflow() メソッドのパラメータ

パラメータ	説明	有効な値
orgId	ユーザが開始するワークフローに関連したオーガニゼーションの ID	有効なオーガニゼーション ID を指定する文字列。 すべてのオーガニゼーション ID のリストの取得方法については、19-1 ページの「アクティブなオーガニゼーションの管理」を参照。
templateId	ワークフローを開始するワークフロー テンプレートの ID	有効なテンプレート ID を指定する文字列。 テンプレート ID を取得するには、次の com.bea.wlpi.common.TemplateInfo メソッドを使用する。 public final String getId() getId() メソッドは、templateId を返す。各 templateId には、複数の定義が含まれる可能性がある。どのテンプレート定義を使用するか判断するために、WebLogic Integration はアクティブ テンプレート定義セットの中から、もっとも有効で適切な日付（現在または過去の日付）と終了日（現在または将来の日付）を持つバージョンを選択する。テンプレート定義に有効日および終了日を定義し、テンプレート定義をアクティブ化する方法については、A-53 ページの「テンプレート定義 DTD」、または『*WebLogic Integration Studio ユーザーズ ガイド*』を参照。 TemplateInfo オブジェクトを取得する方法については、13-4 ページの「オーガニゼーションのテンプレートの取得」を参照。TemplateInfo オブジェクトで選択可能なメソッドの詳細については、B-27 ページの「TemplateInfo オブジェクト」を参照。

表 20-2 instantiateWorkflow() メソッドのパラメータ (続き)

パラメータ	説明	有効な値
`xml`	ワークフロー インスタンスの初期設定値	ワークフロー インスタンスに有意な初期設定値を含む XML ドキュメント。初期設定値が不要の場合は null。
`initialValues`	ワークフロー変数の初期値	キーとして初期設定される変数、および値として必要な値を指定する「*key-value*」ペアを持つマップ オブジェクト。変数の初期設定が不要の場合は null を指定するマップ オブジェクト。
`pluginData`	プラグイン インスタンス データ	キーとしてプラグイン名を指定し、および値としてインスタンス データを指定する「*key-value*」ペアを持つマップ オブジェクト。プラグイン データの定義が不要な場合は、null を指定するマップ オブジェクト。 BPM プラグインのプログラミングの詳細については、『*WebLogic Integration BPM プラグイン プログラミング ガイド*』を参照。
`transactionId`	トランザクションの ID **注意 :** このパラメータは、クラスタ化環境においてのみ必要です。	ユニークなトランザクション ID を指定するオブジェクト。 ユニークなトランザクション ID を生成するには、次のコンストラクタを使用して新しい `com.bea.wlpi.client.common.GUID` オブジェクトを作成する。 `GUID transactionId = new GUID();` GUID クラスの詳細については、Javadoc の `com.bea.wlpi.client.common.GUID` を参照。

`instantiateWorkflow()` メソッドは、ワークフロー テンプレートの呼び出しの他に、テンプレート定義の中でリストアップされているそれぞれの手動による `Start` (開始) ノードに対し、テンプレート定義内で指定されている場合に、以下のタスクを実行します。

- 変数値を、指定された定数値に初期設定します。

- 指定されたアクションを実行します。
- サクセサをアクティブ化します。

テンプレート定義の作成、および呼び出し時に実行するタスクの定義については、14-1 ページの「ワークフロー テンプレート定義の作成および管理」を参照してください。

`instantiateWorkflow()` メソッドは、クライアント要求 DTD に準拠し、また以下のそれぞれのテンプレート定義インスタンスの情報を含む XML ドキュメントを返します。

- テンプレート、テンプレート定義、およびインスタンス ID
- 定義された `Send XML to Client` アクションに関する情報

クライアント要求 DTD の詳細については、A-33 ページの「クライアント要求 DTD」を参照してください。

たとえば、次のコードでは、`activeOrgId` および `templateId` の変数で指定されたオーガニゼーションに対して、テンプレート定義が呼び出されます。このコード例では、`worklist` は `Worklist` EJB への `EJBObject` 参照を表します。

```
String clientReq = worklist.instantiateWorkflow(activeOrgId,
templateId);
```

複数のテンプレート定義が同じ `templateId` で表されている場合、WebLogic Integration はアクティブで、また有効日および終了日がもっとも適切なバージョンを選択します。

`instantiateWorkflows()` メソッドの詳細については、Javadoc の `com.bea.wlpi.server.worklist.Worklist` を参照してください。

手動によるワークフロー開始のサンプル

この節では、コマンドラインおよび JSP Worklist サンプルから抜粋して、ワークフローを手動で開始する方法を示します。

注意： コマンドラインおよび JSP Worklist サンプルの詳細については、1-23 ページの「BPM API のサンプル」を参照してください。

コマンドラインの Worklist サンプル

ワークフローの開始方法を示すコマンドライン Worklist サンプルから抜粋したものを以下に示します。

注意： コマンドライン Worklist サンプルの詳細については、1-26 ページの「コマンドライン Worklist サンプル」を参照してください。

このサンプルでは、ユーザと通信するための入力ストリームが定義されており、ユーザは以下のアクションのいずれかを指定するよう求められます。

- 開始可能なワークフローを取得する
- ワークフローを手動で開始する

重要なコード行は、**太字**で強調されています。このコード例では、`worklist` は Worklist EJB への `EJBObject` 参照を表します。

```
/* ユーザとの通信のための入力ストリームを作成する */
stdIn = new BufferedReader( new InputStreamReader( System.in ) );

/* ツール タイトルを表示する */
System.out.print( "\n---  Command Line Worklist v1.2  ---" );

/* メイン メニューを表示してユーザと交信する */
while( true ) {
   /* メニューを表示する */
   System.out.println( "\n---           Main Menu          ---" );
   System.out.println( "\nEnter choice:" );
   System.out.println( "1) Organizations" );
   System.out.println( "2) Workflows" );
   System.out.println( "3) Tasks" );
   System.out.println( "Q) Quit" );
   System.out.print( ">> " );
      .
      .
      .
public static void mngWorkflows( ) {
   String templateId;

   /* ユーザとの通信のための入力ストリームを作成する */
   BufferedReader stdIn = new BufferedReader(
      new InputStreamReader( System.in ) );

   try {
```

```
/* メイン メニューを表示してユーザと交信する */
while( true ) {
   /* メニューを表示する */
   System.out.println( "\n\n---           Workflows          ---" );
   System.out.println( "\nEnter choice:" );
   System.out.println( "1) List Startable workflow templates" );
   System.out.println( "2) Start a workflow" );
   System.out.println( "B) Back to previous menu" );
   System.out.println( "Q) Quit" );
   System.out.print( ">> " );

   /* ユーザの選択を取得する */
   String line = stdIn.readLine( );

   /* ユーザが選択を行わないで〔Enter〕を押したか ? */
   if( line.equals( "" ) )
      continue;
   /* ユーザが複数の文字を入力したか ? */
   else if( line.length( ) > 1 ) {
      System.out.println( "*** Invalid choice" );
      continue;
   }

   /* 大文字および文字へのコンバート */
   char choice = line.toUpperCase( ).charAt( 0 );

   /* ユーザの選択を処理する */
   switch( choice ) { ...
```

開始可能なワークフローを取得する

開始可能なワークフローのリストを取得する方法を示します。

```
/* 開始可能なワークフロー テンプレートをリストアップする */
case '1' :
   /* WLPI 公開 API メソッド */
   /* 注意 : 発生した例外を取り込むコードを
    * 追加するとよい */
   String activeOrgId = worklist.getActiveOrganization( );
   List templateList;

   /* WLPI 公開 API メソッド */
   /* 注意 : 発生した例外を取り込むコードを
    * 追加するとよい */
   templateList = worklist.getStartableWorkflows( activeOrgId );
```

```
            /* 開始可能なワークフローがあるか ? */
            if( templateList.size( ) == 0 )
               System.out.println( "\nNo Startable Workflows." );
            else
               System.out.println( "\nStartable Workflows:" );

            /* リストを処理して、開始可能なワークフローを表示する */
            for( int i = 0; i < templateList.size( ); i++ ) {
               /* リストから要素を取り出す */
               TemplateInfo templateInfo = (
                  TemplateInfo )templateList.get( i );

               /* テンプレート ID と名前を取り出して表示する */
               System.out.println( "- ID: " + templateInfo.getId( ) +
                  " Name: " + templateInfo.getName( ) );
            }
            break; ...
```

ワークフローを手動で開始する

ワークフローを手動で開始する方法を示します。

```
         /* ワークフロー テンプレートのインスタンス化 */
         case '2' :
            /* インストタンス化するワークフローのテンプレート ID を取得する */
            if( ( templateId = askQuestion(
               "\nEnter Workflow Template ID" ) ) == null ) {
               /* ユーザによる操作の取り消し */
               System.out.println( "*** Cancelled" );
               break;
            }

            try {
               /* WLPI 公開 API メソッド */
               /* アクティブなオーガニゼーション ID を取り出す */
               activeOrgId = worklist.getActiveOrganization( );

               /* WLPI 公開 API メソッド */
               /* ワークフローをインスタンス化する */
               String clientReq = worklist.instantiateWorkflow(
                  activeOrgId, templateId );

               /* 正常終了（例外の発生なし）*/
               System.out.println( "- Success" );
//             System.out.println( "- Client Request:\n" + clientReq );
```

```
                /* クライアントの要求 ( もしあれば ) を処理するために、
                 * サーバからの応答を解析する */
                parser.handleRequest( clientReq );
             }
             catch( Exception e ) {
                System.out.println(
                "*** Failed to instantiate Workflow Template (ID: " +
                   templateId + ")" );
                System.err.println( e );
             }
             break;

          /* 前のメニューに戻る */
          case 'B' :
             return;

          /* ツールを終了 */
          case 'Q' :
             /* サーバから切断 */
             disconnect( );
             System.exit( 1 );

          default:
             System.out.println( "*** Invalid choice" );
          }
       }
    }
    /* 「Unhandled」例外 */
    catch( Exception e ) {
       System.err.println( e );
    }
    return;
}
```

JSP Worklist サンプル

ここでは、JSP Worklist サンプルの抜粋から、ワークフローを手動で開始する方法を示します。

重要なコード行は、**太字**で強調されています。このコード例では、`worklist` は `Worklist` EJB への `EJBObject` 参照を表します。

1-29 ページの「JSP Worklist へのプライマリ インタフェース」の図に示すように、ユーザが JSP Worklist 内の `Start Workflow` リンクをクリックして、開始可能なワークフローを選択すると、ワークフローが開始されます。次のように、`Start Workflow` リンクは `startworkflow.jsp` ファイルを参照します。

```
<a href="startworkflow.jsp"><font color= "white">Start Workflow</font></a>
```

次のサンプルでは、JSP Worklist サンプル内の `startworkflow.jsp` ファイルの抜粋から、ワークフローの開始方法を示します。

```
<%@ page import="
   javax.ejb.*,
   java.rmi.RemoteException,
   java.util.*,
   com.bea.wlpi.common.*
   com.bea.wlpi.server.worklist.*
" session="true"%>

<jsp:usebean id="responseParser" class="jsp_servlet._worklist.ResponseParser"
scope="session"/>

<%
   Worklist worklist = null;
   String name;
   String error = null;
   String currentOrg = (String)session.getValue("currentOrg");
   worklist = (Worklist)session.getValue("worklist");
   name = (String)session.getValue("name");

   if (worklist != null) {
      Hashtable h = new Hashtable();
      h.put(javax.naming.Context.SECURITY_PRINCIPAL, name);
      h.put(javax.naming.Context.SECURITY_CREDENTIALS,
      (String)session.getValue("password"));
      new javax.naming.InitialContext(h);
```

`start` 変数は、ユーザが開始するワークフロー テンプレート定義のテンプレート ID を格納するために使用されます。始めに JSP がロードされると、`start` 変数は `null` に設定され、開始ワークフローの `try` ループはスキップされます。次に、開始可能なワークフローがリストされます。ユーザが開始可能なワークフローを選択すると、`startworkflow.jsp` ファイルが呼び出され、その `start` 変数は、`TemplateInfo` オブジェクトから取得するテンプレート ID の値に設定されます。結果の XML ファイル（これはクライアント要求 DTD に準拠）は、`instantiateWorkflow()` メソッドの呼出しから返され、解析のためにレスポンス パーサーの `ResponseParser` に渡されます。レスポンス パーサーの詳細については、21-53 ページの「タスクを割り当てる」を参照してください。

```
      /* 開始パラメータが設定された場合にワークフローを開始する */
      String start = request.getParameter("start");
      if (start != null)
         try {
            responseParser.parse(worklist.instantiateWorkflow(
               currentOrg, start));
            session.removeValue("error");
            if (responseParser.isQuery())
               pageContext.forward("query.jsp");
            else
               pageContext.forward("worklist.jsp");
            return;
         } catch (Exception e) {
         error = e.getMessage();
      } out.println(error);
%>
<html>
<title>eProcess Integrator</title>
<head>
</head>

<body bgcolor=#ffffff>
<font face="Arial" size="4">


<table dir="ltr" border="0" cellpadding="1" cellspacing="1">
   <tr>
      <td><img src="logo.gif"></td>
      <td>
         <font color="red" size="7"><strong><em>Weblogic</em></strong></font>
         <font color="black" size="7"><strong><em>Process Integrator
         </em></strong></font></td>
   </tr>
   <tr>
      <td bgcolor="navy" valign="top">
      <table dir="ltr" border="0" cellpadding="2" cellspacing="2" >
         <tr>
            <td><font color="yellow"><strong>Go ...</strong></font></td>
         </tr>
         <tr>
            <td><a href="worklist.jsp?worklist">
            <font color="white">Worklist</font></a></td>
         </tr>
         <tr>
            <td><a href="logoff.jsp"><font color="white">Logoff</font></a></td>
         </tr>
      </table>
      </td>
```

```
      <td valign="top">
      <table dir="ltr" border="0" cellpadding="2" cellspacing="2" valign="top">
         <tr>
            <td><font color="navy" size="5" face="Arial">
            Start Workflow</font></td>
         </tr>
```

開始可能なワークフローがリストされます。ユーザが開始可能なワークフローを選択すると、startworkflow.jsp ファイルが呼び出され、その start 変数は、TemplateInfo オブジェクトから取得するテンプレート ID の値に設定されます。

```
/* 開始可能なワークフロー名を取得し、それらをリストアップする /*
<%
   try {
      List templates = worklist.getStartableWorkflows(currentOrg);
      for (int i = 0; i < templates.size(); i++) {
    TemplateInfo info = (TemplateInfo)templates.get(i);

%>
         <tr>
            <td><a href="startworkflow.jsp?start=<%=info.getId()%>">
            <%=info.getName()%></a></td>
         </tr>
<%
   }
   } catch (Exception e) {
   out.print("Error 3: " + e);
   }
   }
%>
      </table>
      </td>
   </tr>
</table>
</font>
</body>
</html>
```

21　実行時タスクの管理

ワークフローを開始すると、20-1 ページの「手動によるワークフローの開始」に記述のように、ワークフローを開くときにインスタンスに関連するタスクの管理が可能になります。この章では、実行時タスクの管理方法について説明します。内容は以下のとおりです。

- タスクの取得
- すべてのタスクの取得
- タスク数の取得
- タスクの実行
- クライアント要求に対する応答
- タスクの割り当て
- タスクへの完了または未完了マークの付与
- タスク プロパティの設定
- インスタンス変数の更新
- 例外ハンドラの呼び出し
- 実行時タスクの管理例

実行時タスク管理の Worklist の使用方法については、『*WebLogic Integration Worklist ユーザーズ ガイド*』の「タスクを使った作業」を参照してください。

タスクの取得

タスクを取得するには、次の com.bea.wlpi.server.worklist.Worklist メソッドを使用します。

```
public com.bea.wlpi.common.TaskInfo getTask(
java.lang.String taskName,
java.lang.String instanceId,
) throws java.rmi.RemoteException,
  com.bea.wlpi.common.WorkflowException
```

getTasks() メソッドのパラメータを次の表に示します。パラメータには値を指定する必要があります。

表 21-1 getTasks() メソッドのパラメータ

パラメータ	説明	有効な値
taskName	取得するタスクの名前	有効なオーガニゼーション ID を指定した文字列。 タスク ID を取得するには、次の com.bea.wlpi.common.TaskInfo メソッドを使用する。 public final java.lang.String getTaskName() TaskInfo オブジェクトの取得方法については、22-10 ページの「ワークフロー インスタンス タスクの取得」を参照。TaskInfo オブジェクトで選択可能なメソッドの詳細については、B-21 ページの「TaskInfo オブジェクト」を参照。

表 21-1 getTasks() メソッドのパラメータ (続き)

パラメータ	説明	有効な値
instanceId	取得するタスクに関連するワークフロー インスタンスの ID	有効なインスタンス ID を指定する文字列。 インスタンス ID を取得するには、次の com.bea.wlpi.common.TaskInfo メソッドを使用する。 public final java.lang.String getInstanceId() TaskInfo オブジェクトの取得方法については、22-10 ページの「ワークフロー インスタンス タスクの取得」を参照。TaskInfo オブジェクトで選択可能なメソッドの詳細については、B-21 ページの「TaskInfo オブジェクト」を参照。

このメソッドは、com.bea.wlpi.common.TaskInfo オブジェクトを戻します。タスクについての情報にアクセスするには、B-21 ページの「TaskInfo オブジェクト」に記載の TaskInfo オブジェクト メソッドを使用します。

たとえば、次のコードでは、特定のワークフロー インスタンスの Task1 というタスクが取得されます。このコード例では、worklist は Worklist EJB への EJBObject 参照を表します。

```
TaskInfo task = worklist.getTask("Task1",instanceID);
```

getTasks() メソッドの詳細については、Javadoc の com.bea.wlpi.server.worklist.Worklist を参照してください。

すべてのタスクの取得

特定の参加コンポーネント（ユーザまたはロール）やオーガニゼーションに関連するタスクのリストを取得する場合、以下のいずれかの `com.bea.wlpi.server.worklist.Worklist` メソッドを使用します。

```
public java.util.List getTasks(
) throws java.rmi.RemoteException,
  com.bea.wlpi.common.WorkflowException
```

```
public java.util.List getTasks(
java.lang.String orgId,
java.lang.String assigneeId,
boolean isRole
) throws java.rmi.RemoteException,
  com.bea.wlpi.common.WorkflowException
```

第 1 のメソッドでは、`Worklist` EJB を参照する `EJBObject` を作成したアクティブなオーガニゼーションに所属するユーザに関連したタスクが取得されます。第 2 のメソッドでは、特定のオーガニゼーションや *assignee* (ロールまたはユーザ) に関連したタスクが取得されます。

`getTasks()` メソッドのパラメータを次の表に示します。パラメータには値を指定する必要があります。

表 21-2 getTasks() メソッドのパラメータ

パラメータ	説明	有効な値
`orgId`	タスクを取得するオーガニゼーションの ID	有効なオーガニゼーション ID を指定する文字列。 すべてのオーガニゼーション ID のリストの取得方法については、19-1 ページの「アクティブなオーガニゼーションの管理」を参照。
`assigneeId`	タスクを取得する *assignee* (ロールまたはユーザ) の ID	有効なユーザまたはロールの ID。 現在のユーザまたはロールのリストの取得方法については、9-1 ページの「セキュリティ レルムのコンフィグレーション」を参照。

表 21-2 getTasks() メソッドのパラメータ (続き)

パラメータ	説明	有効な値
isRole	assigneeId パラメータに指定された割り当て先がロールまたはユーザであるかを指定するブール フラグ	true (ロール) または false (ユーザ)。

各メソッドは、com.bea.wlpi.common.TaskInfo オブジェクトのリストを返します。各タスクについての情報にアクセスするには、B-21 ページの「TaskInfo オブジェクト」に記載の TaskInfo オブジェクト メソッドを使用します。

たとえば、次のコードでは、アクティブなオーガニゼーションに所属する現在のユーザのためのタスクが取得されます。このコード例では、worklist は Worklist EJB への EJBObject 参照を表します。

```
List taskList = worklist.getTasks();
```

次のコードでは、オーガニゼーション ORG1 の ROLE1 というロールのタスクのリストが取得されます（パラメータ isRole は、true）。このコード例では、worklist は Worklist EJB への EJBObject 参照を表します。

```
List taskList = worklist.getTasks(“ORG1”, “ROLE1”, true);
```

getTasks() メソッドの詳細については、Javadoc の com.bea.wlpi.server.worklist.Worklist を参照してください。

タスク数の取得

現在のユーザのために割り当てられているタスク数および期日超過しているタスクの数を取得するには、次の com.bea.wlpi.server.worklist.Worklist メソッドを使用します。

```
public int[ ] getTaskCounts(
) throws java.rmi.RemoteException,
  com.bea.wlpi.common.WorkflowException
```

このメソッドは、2 つの要素からなる配列を返します。最初の要素は、ユーザに割り当てられたタスクの数を示します。次の要素は、その中で期日超過しているタスクの数を示します。

たとえば、次のコードでは、現在のユーザに割り当てられたタスクおよび期日超過したタスクの数が取得され、配列 taskCounts[ ] にその数が保持されます。このコード例では、worklist は Worklist EJB への EJBObject 参照を表します。

```
int taskCounts[ ] = worklist.getTaskCounts();
```

割り当てられたタスク数と期日超過しているタスク数は、それぞれ taskCounts[0] と taskCounts[1] に保持されています。

getTaskCounts() メソッドの詳細については、Javadoc の com.bea.wlpi.server.worklist.Worklist を参照してください。

タスクの実行

タスクを実行するには、以下のいずれかの com.bea.wlpi.server.worklist.Worklist メソッドを使用します。

```
public java.lang.String taskExecute(
java.lang.String taskName,
java.lang.String instanceId
) throws java.rmi.RemoteException,
  com.bea.wlpi.common.WorkflowException

public java.lang.String taskExecute(
java.lang.String templateDefinitionId,
java.lang.String instanceId,
java.lang.String taskId
) throws java.rmi.RemoteException,
  com.bea.wlpi.common.WorkflowException
```

taskExecute() メソッドのパラメータを次の表に示します。パラメータには値を指定する必要があります。

表 21-3 taskExecute() メソッドのパラメータ

パラメータ	説明	有効な値
taskName	実行するタスクの名前	有効なタスク名を指定する文字列。 タスク ID を取得するには、次の com.bea.wlpi.common.TaskInfo メソッドを使用する。 public final java.lang.String getTaskName() TaskInfo オブジェクトの取得方法については、22-10 ページの「ワークフロー インスタンス タスクの取得」を参照。TaskInfo オブジェクトで選択可能なメソッドの詳細については、B-21 ページの「TaskInfo オブジェクト」を参照。
templateDefinitionId	タスクを含むワークフロー インスタンスがインスタンス化されたテンプレート定義の ID	有効なテンプレート定義 ID を指定する文字列。 テンプレート定義 ID を取得するには、次の com.bea.wlpi.common.TaskInfo メソッドを使用する。 public final java.lang.String getTemplateDefinitionId() TaskInfo オブジェクトの取得方法については、22-10 ページの「ワークフロー インスタンス タスクの取得」を参照。TaskInfo オブジェクトで選択可能なメソッドの詳細については、B-21 ページの「TaskInfo オブジェクト」を参照。

表 21-3 taskExecute() メソッドのパラメータ (続き)

パラメータ	説明	有効な値
instanceId	タスクを含むワークフロー インスタンスの ID	有効なインスタンス ID を指定する文字列。 インスタンス ID を取得するには、次の `com.bea.wlpi.common.TaskInfo` メソッドを使用する。 `public final java.lang.String getInstanceId()` `TaskInfo` オブジェクトの取得方法については、22-10 ページの「ワークフロー インスタンス タスクの取得」を参照。`TaskInfo` オブジェクトで選択可能なメソッドの詳細については、B-21 ページの「TaskInfo オブジェクト」を参照。
taskId	実行するタスクの ID	有効なタスク ID を指定する文字列。 タスク ID を取得するには、次の `com.bea.wlpi.common.TaskInfo` メソッドを使用する。 `public final java.lang.String getTaskId()` `TaskInfo` オブジェクトの取得方法については、22-10 ページの「ワークフロー インスタンス タスクの取得」を参照。`TaskInfo` オブジェクトで選択可能なメソッドの詳細については、B-21 ページの「TaskInfo オブジェクト」を参照。

このメソッドの結果として、特定のタスクのための `Executed` イベントに関連付けられたすべてのアクションが連続して実行されます。タスクの `Executed` イベントに関する定義については、A-53 ページの「テンプレート定義 DTD」を参照してください。

このメソッドは、A-33 ページの「クライアント要求 DTD」に記載のとおり、Client Request DTD の `ClientReq.dtd` に準拠する XML ドキュメントを返します。この XML ドキュメントは、実行中のインスタンスの情報を含み、たとえば、SAX (Simple API for XML) パーサーのような XML 構文解釈ツールを使用してドキュメント解析によるアクセスが可能です。SAX パーサーの実装サンプルについては、21-26 ページの「実行時タスクの管理例」を参照してください。

たとえば、次のコードでは、特定のタスクが実行され、文字列 `clientReq` に結果の XML が割り当てられます。このコード例では、`worklist` は `Worklist` EJB への `EJBObject` 参照を表します。

```
String clientReq = worklist.taskExecute(
   task.getTemplateDefinitionId(),
   task.getInstanceId(),
   task.getTaskId()
);
```

テンプレート定義、ワークフロー インスタンス、およびタスク ID は、`com.bea.wlpi.common.TaskInfo` オブジェクト、`task` と関連するメソッドを使用して取得されます。`task` オブジェクトは、21-4 ページの「すべてのタスクの取得」に記載のメソッドを使用して取得されます。

`com.bea.wlpi.common.TaskInfo` メソッドの詳細については、B-21 ページの「TaskInfo オブジェクト」を参照してください。

`taskExecute()` メソッドの詳細については、Javadoc の `com.bea.wlpi.server.worklist.Worklist` を参照してください。

クライアント要求に対する応答

ワークフローを定義するとき、*統合*アクションを作成することで、実行中のインスタンス（プロセス エンジンを介して）とクライアント プログラム間の通信チャネルを確立できます。これらの典型的な XML をクライアントに送信アクションは、標準あるいはカスタム DTD に準拠した XML ドキュメントを介して、要求をクライアントに送ります。たとえば、クライアントへ要求を送るアクションは、ダイアログ ボックスを介して実装できます。

クライアントからの応答が要求されます。最初のアクションとして、一般的に、応答から要求される値を抽出するために `XPath` 関数を使用します。XPath については次のサイトを参照してください。

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

ワークフロー内の 統合アクションの定義については、A-53 ページの「テンプレート定義 DTD」を参照してください。

Send XML to Client アクション（ActionSendXMLToClient）によって要求への応答を生成するには、以下の com.bea.wlpi.server.worklist.Worklist メソッドのいずれかを使用します。

メソッド 1

```
public java.lang.String response(
  java.lang.String templateDefinitionId,
  java.lang.String instanceId,
  java.lang.String nodeId,
  java.lang.Object data
) throws java.rmi.RemoteException,
  com.bea.wlpi.common.WorkflowException
```

メソッド 2

```
public java.lang.String response(
  java.lang.String templateDefinitionId,
  java.lang.String instanceId,
  java.lang.String nodeId,
  java.lang.Object data,
  java.lang.Object transactionId
) throws java.rmi.RemoteException,
  com.bea.wlpi.common.WorkflowException
```

メソッド 3

```
public java.lang.String response(
  java.lang.String templateDefinitionId,
  java.lang.String instanceId,
  java.lang.String nodeId,
  java.lang.String xml
) throws java.rmi.RemoteException,
  com.bea.wlpi.common.WorkflowException
```

response() メソッドのパラメータを次の表に示します。パラメータには値を指定する必要があります。

表 21-4 response() メソッドのパラメータ

パラメータ	説明	有効な値
templateDefinitionId	テンプレート定義の ID	有効なテンプレート ID を指定する文字列。
instanceId	ワークフロー インスタンスの ID	有効なインスタンス ID を指定する文字列。
nodeId	応答呼び出しに応答するクライアント要求を生成するワークフロー オブジェクト（またはノード）	有効なノード ID を指定した文字列。

表 21-4 response() メソッドのパラメータ (続き)

パラメータ	説明	有効な値
data	クライアントの要求に呼応した応答	受信アクションに準拠しているすべてのオブジェクト。 通常、標準 DTD またはカスタム DTD を使用して XML をクライアントに送信アクションによって生成された要求。この場合、応答は xml で記述された XML ドキュメント。
xml	クライアントの要求に呼応した応答	以下の 4 つの事前定義済み DTD のひとつ、またはユーザ定義の DTD に準拠した XML ドキュメント。 ■ クライアント呼び出しアドイン応答 DTD ■ クライアント呼び出しプログラム応答 DTD ■ クライアント メッセージ ボックス応答 DTD ■ クライアント変数設定応答 DTD 上記の DTD に関する詳細については、A-1 ページの「DTD フォーマット」を参照。

表 21-4 response() メソッドのパラメータ (続き)

パラメータ	説明	有効な値
transactionId	トランザクションの ID **注意 :** このパラメータは、クラスタ化環境においてのみ必要です。	ユニークなトランザクション ID を指定するオブジェクト。 ユニークなトランザクション ID を生成するには、次のコンストラクタを使用して新しい com.bea.wlpi.client.common.GUID オブジェクトを作成する。 `GUID transactionId = new GUID();` GUID クラスの詳細については、Javadoc の com.bea.wlpi.client.common.GUID を参照。

response() メソッドは、コールバック変数をセットし、順次すべてのアクションを実行します。コールバック変数とアクションの定義については、A-53 ページの「テンプレート定義 DTD」 を参照してください。

このメソッドは、A-33 ページの「クライアント要求 DTD」 に記載のとおり、Client Request DTD の ClientReq.dtd に準拠する XML ドキュメントを返します。この XML ドキュメントは、実行中のインスタンスの情報を含み、たとえば、SAX (Simple API for XML) パーサーのような XML 構文解釈ツールを使用してドキュメント解析によるアクセスが可能です。SAX パーサーの実装サンプルについては、21-26 ページの「実行時タスクの管理例」 を参照してください。

たとえば、次のコードでは、サーバに応答が送られ、戻り値が clientReq 変数に保存されます。

```
String clientReq = worklist.response( templateDefinitionId,
   instanceId, actionId, response.toString() );
```

サンプルの詳細については、21-48 ページの「クライアント要求に対して応答する」 を参照してください。response() メソッドの詳細については、Javadoc の com.bea.wlpi.server.worklist.Worklist を参照してください。

タスクの割り当て

タスクに割り当てられているユーザまたはロールを変更する、またはタスクの割り当てを解除するには、以下の com.bea.wlpi.server.worklist.Worklist メソッドを使用します。

メソッド 1

```
public java.lang.String taskAssign(
  java.lang.String templateDefinitionId,
  java.lang.String instanceId,
  java.lang.String taskId,
  java.lang.String assigneeId,
  boolean isRole,
  boolean bLoadBalance
) throws java.rmi.RemoteException,
  com.bea.wlpi.common.WorkflowException
```

メソッド 2

```
public java.lang.String taskAssign(
  java.lang.String templateDefinitionId,
  java.lang.String instanceId,
  java.lang.String taskId,
  java.lang.String assigneeId,
  boolean isRole,
  boolean bLoadBalance,
  java.lang.Object transactionId
) throws java.rmi.RemoteException,
  com.bea.wlpi.common.WorkflowException
```

メソッド 3

```
public java.lang.String taskUnassign(
  java.lang.String templateDefinitionId,
  java.lang.String instanceId,
  java.lang.String taskId
) throws RemoteException, WorkflowException
```

taskAssign() および taskUnassign() メソッドのパラメータを次の表に示します。パラメータには値を指定する必要があります。

表 21-5 taskAssign() および taskUnassign() メソッドのパラメータ

パラメータ	説明	有効な値	対象	
			taskAssign()	taskUnassign()
templateDefinitionId	タスクを割り当てるテンプレート定義の ID	有効なテンプレート定義 ID を指定する文字列。 テンプレート定義 ID を取得するには、次の com.bea.wlpi.common.TaskInfo メソッドを使用する。 public final java.lang.String getTemplateDefinitionId() TaskInfo オブジェクトの取得方法については、22-10 ページの「ワークフロー インスタンス タスクの取得」を参照。TaskInfo オブジェクトで選択可能なメソッドの詳細については、B-21 ページの「TaskInfo オブジェクト」を参照。	+	+

表 21-5 taskAssign() および taskUnassign() メソッドのパラメータ (続き)

パラメータ	説明	有効な値	対象	
			taskAssign()	taskUnassign()
instanceId	タスクと対応するワークフロー インスタンスの ID	有効なインスタンス ID を指定する文字列。 インスタンス ID を取得するには、次の com.bea.wlpi.common.TaskInfo メソッドを使用する。 public final java.lang.String getInstanceId() TaskInfo オブジェクトの取得方法については、22-10 ページの「ワークフロー インスタンス タスクの取得」を参照。 TaskInfo オブジェクトで選択可能なメソッドの詳細については、B-21 ページの「TaskInfo オブジェクト」を参照。	+	+

表 21-5 taskAssign() および taskUnassign() メソッドのパラメータ (続き)

パラメータ	説明	有効な値	対象	
			taskAssign()	taskUnassign()
taskId	割り当てるタスクの ID	有効なタスク ID を指定する文字列。 タスク ID を取得するには、次の com.bea.wlpi.common.TaskInfo メソッドを使用する。 public final java.lang.String getTaskId() TaskInfo オブジェクトの取得方法については、22-10 ページの「ワークフロー インスタンス タスクの取得」を参照。TaskInfo オブジェクトで選択可能なメソッドの詳細については、B-21 ページの「TaskInfo オブジェクト」を参照。	+	+
assigneeId	タスクを割り当てる *assignee* (ロールまたはユーザ) の ID	有効なロールまたはユーザ ID を指定する文字列。 現在のユーザまたはロールのリストの取得方法については、9-1 ページの「セキュリティ レルムのコンフィグレーション」を参照。	+	

表 21-5 taskAssign() および taskUnassign() メソッドのパラメータ (続き)

パラメータ	説明	有効な値	対象	
			taskAssign()	taskUnassign()
isRole	assigneeId パラメータに指定された割り当て先がロールまたは ユーザであるかを指定するブール フラグ	true (ロール) または false (ユーザ)。	+	
bLoadBalance	指定されたロール内でロード バランスを行うかどうかを指定するブール フラグ	true (ロード バランスを行う) または false (ロード バランスを行わない)。 この設定は、isRole 値が true に設定されている場合にのみ有効。	+	
transactionId	トランザクションの ID **注意 :** このパラメータは、クラスタ化環境においてのみ必要です。	ユニークなトランザクション ID を指定するオブジェクト。 ユニークなトランザクション ID を生成するには、次のコンストラクタを使用して新しい com.bea.wlpi.client.common.GUID オブジェクトを作成する。 `GUID transactionId = new GUID();` GUID クラスの詳細については、Javadoc の com.bea.wlpi.client.common.GUID を参照。	+	

割り当て先を選択する時は、以下を考慮します。

- 指定された割り当て先に対して現在タスクの再ルーティングが存在するか。この場合、タスクは指定されたとおりのルートを通ります。

- isRole および bLoadBalance 引数は、ロード バランスが有効となっていることを表す true に設定してあるか。この場合、プロセス エンジンでは（現在有効な再ルーティングがない）ロール内に割り当てられたすべてのユーザのタスク数が検討し、割り当てられたタスクが最も少ないユーザを識別し、そのユーザにタスクを割り当てます。

これらのメソッドは、A-33 ページの「クライアント要求 DTD」に記載のとおり、Client Request DTD の ClientReq.dtd に準拠する XML ドキュメントを戻します。この XML ドキュメントは、実行中のインスタンスの情報を含み、たとえば、SAX (Simple API for XML) パーサーのような XML 構文解釈ツールを使用してドキュメント解析によるアクセスが可能です。SAX パーサーの実装サンプルについては、21-26 ページの「実行時タスクの管理例」を参照してください。

たとえば、次のコードでは、ユーザ joe にタスクが割り当てられ、clientReq 文字列に結果の XML が割り当てられます。このコード例では、worklist は Worklist EJB への EJBObject 参照を表します。

```
String clientReq = worklist.taskAssign(
   task.getTemplateDefinitionId(),
   task.getInstanceId(),
   task.getTaskId(),
   “joe”,
   false,
   false
);
```

次のコードでは、同じタスクの割り当てが取り消されます。

```
String clientReq = worklist.taskUnassign(
   task.getTemplateDefinitionId(),
   task.getInstanceId(),
   task.getTaskId(),
   “joe”,
   false,
   false
);
```

テンプレート定義、ワークフロー インスタンス、およびタスク ID は、com.bea.wlpi.common.TaskInfo オブジェクト、task と関連するメソッドを使用して取得されます。task オブジェクトは、21-4 ページの「すべてのタスクの取得」に記載のメソッドを使用して取得されます。

com.bea.wlpi.common.TaskInfo メソッドの詳細については、B-21 ページの「TaskInfo オブジェクト」または、Javadoc の com.bea.wlpi.common.TaskInfo を参照してください。

taskAssign() および taskUnassign() メソッドの詳細については、Javadoc の com.bea.wlpi.server.worklist.Worklist を参照してください。

タスクへの完了または未完了マークの付与

タスクに完了（done）または未完了（undone）としてマークするには、以下の com.bea.wlpi.server.worklist.Worklist メソッドを使用します。

メソッド 1

```
public java.lang.String taskMarkDone(
  java.lang.String templateDefinitionId,
  java.lang.String instanceId,
  java.lang.String taskId
) throws java.rmi.RemoteException,
  com.bea.wlpi.common.WorkflowException
```

メソッド 2

```
public java.lang.String taskMarkDone(
  java.lang.String templateDefinitionId,
  java.lang.String instanceId,
  java.lang.String taskId,
  java.lang.Object transactionId
) throws java.rmi.RemoteException,
  com.bea.wlpi.common.WorkflowException
```

メソッド 3

```
public java.lang.String taskUnmarkDone(
  java.lang.String templateDefinitionId,
  java.lang.String instanceId,
  java.lang.String taskId
) throws java.rmi.RemoteException,
  com.bea.wlpi.common.WorkflowException
```

taskMarkDone() および taskMarkUndone() メソッドのパラメータを次の表に示します。パラメータには値を指定する必要があります。

表 21-6 taskMarkDone() および taskMarkUndone() メソッドのパラメータ

パラメータ	説明	有効な値
templateDefinitionId	タスクを含むワークフロー インスタンスがインスタンス化されたテンプレート定義の ID	有効なテンプレート定義 ID を指定する文字列。 テンプレート定義 ID を取得するには、次の com.bea.wlpi.common.TaskInfo メソッドを使用する。 public final java.lang.String getTemplateDefinitionId() TaskInfo オブジェクトの取得方法については、22-10 ページの「ワークフロー インスタンス タスクの取得」を参照。TaskInfo オブジェクトで選択可能なメソッドの詳細については、B-21 ページの「TaskInfo オブジェクト」を参照。
instanceId	タスクを含むワークフロー インスタンスの ID	有効なインスタンス ID を指定する文字列。 インスタンス ID を取得するには、次の com.bea.wlpi.common.TaskInfo メソッドを使用する。 public final java.lang.String getInstanceId() TaskInfo オブジェクトの取得方法については、22-10 ページの「ワークフロー インスタンス タスクの取得」を参照。TaskInfo オブジェクトで選択可能なメソッドの詳細については、B-21 ページの「TaskInfo オブジェクト」を参照。

表 21-6 taskMarkDone() および taskMarkUndone() メソッドのパラメータ (続き)

パラメータ	説明	有効な値
taskId	完了または未完了マークを付けるタスクの ID	有効なタスク ID を指定する文字列。 タスク ID を取得するには、次の com.bea.wlpi.common.TaskInfo メソッドを使用する。 public final java.lang.String getTaskId() TaskInfo オブジェクトの取得方法については、22-10 ページの「ワークフロー インスタンス タスクの取得」を参照。TaskInfo オブジェクトで選択可能なメソッドの詳細については、B-21 ページの「TaskInfo オブジェクト」を参照。
transactionId	トランザクションの ID **注意 :** このパラメータは、クラスタ化環境においてのみ必要です。	ユニークなトランザクション ID を指定するオブジェクト。 ユニークなトランザクション ID を生成するには、次のコンストラクタを使用して新しい com.bea.wlpi.client.common.GUID オブジェクトを作成する。 GUID transactionId = new GUID(); GUID クラスの詳細については、Javadoc の com.bea.wlpi.client.common.GUID を参照。

taskMarkDone() メソッドは、現在の日付および時間にタスク Completed 値を設定します。さらに、指定されたタスクの Marked Done イベントと関連する、すべてのアクションの連続実行を行います。ただし、taskMarkDone() メソッドは完了マークの付けられたタスクには影響しません。タスクの Marked Done イベントに関する定義については、A-53 ページの「テンプレート定義 DTD」を参照してください。

`taskUnmarkDone()` メソッドは `Completed` 日付を消去します。このメソッドは、指定したタスクの `Activated` イベントに関連したアクションを実行 *しません*。

このメソッドは、A-33 ページの「クライアント要求 DTD」に記載のとおり、Client Request DTD の `ClientReq.dtd` に準拠する XML ドキュメントを返します。この XML ドキュメントは、実行中のインスタンスの情報を含み、たとえば、SAX (Simple API for XML) パーサーのような XML 構文解釈ツールを使用してドキュメント解析によるアクセスが可能です。SAX パーサーの実装サンプルについては、21-26 ページの「実行時タスクの管理例」を参照してください。

たとえば、次のコードでは指定されたタスクに完了マークが付けられ、`Completed` 値が現在の日付および時間に設定されています。また、指定されたタスクの `Marked Done` イベントと関連するアクションが実行され、`clientReq` 文字列に結果の XML が割り当てられます。このコード例では、`worklist` は `Worklist` EJB への `EJBObject` 参照を表します。

```
String clientReq = worklist.taskMarkDone(
   task.getTemplateDefinitionId(),
   task.getInstanceId(),
   task.getTaskId()
);
```

次のコードでは、同じタスクに未完了のマークがつけられ、完了日付が初期化され、`clientReq` 文字列に結果の XML が割り当てられます。

```
String clientReq = worklist.taskMarkUndone(
   task.getTemplateDefinitionId(),
   task.getInstanceId(),
   task.getTaskId()
);
```

テンプレート定義、ワークフロー インスタンス、およびタスク ID は、`com.bea.wlpi.common.TaskInfo` オブジェクト、`task` と関連するメソッドを使用して取得されます。`task` オブジェクトは、21-4 ページの「すべてのタスクの取得」に記載のメソッドを使用して取得されます。

`com.bea.wlpi.common.TaskInfo` メソッドの詳細については、B-21 ページの「TaskInfo オブジェクト」または、Javadoc の `com.bea.wlpi.common.TaskInfo` を参照してください。

`taskMarkDone()` および `taskMarkUndone()` メソッドの詳細については、Javadoc の `com.bea.wlpi.server.worklist.Worklist` を参照してください。

タスク プロパティの設定

タスク プロパティを設定するには、次の com.bea.wlpi.server.worklist.Worklist メソッドを使用します。

```
public java.lang.String taskSetProperties(
  java.lang.String templateDefinitionId,
  java.lang.String instanceId,
  java.lang.String taskId,
  int priority,
  boolean doneWithoutDoit,
  boolean doitIfDone,
  boolean unmarkDone,
  boolean modifiable,
  boolean reassignable
) throws java.rmi.RemoteException,
  com.bea.wlpi.common.WorkflowException
```

taskSetProperties() メソッドのパラメータを次の表に示します。パラメータには値を指定する必要があります

表 21-7 taskSetProperties() メソッドのパラメータ

パラメータ	説明	有効な値
templateDefinitionId	タスクと対応するテンプレート定義の ID	有効なテンプレート定義 ID を指定する文字列。 テンプレート定義 ID を取得するには、次の com.bea.wlpi.common.TaskInfo メソッドを使用する。 public final java.lang.String getTemplateDefinitionId() TaskInfo オブジェクトの取得方法については、22-10 ページの「ワークフロー インスタンス タスクの取得」を参照。TaskInfo オブジェクトで選択可能なメソッドの詳細については、B-21 ページの「TaskInfo オブジェクト」を参照。

表 21-7 taskSetProperties() メソッドのパラメータ (続き)

パラメータ	説明	有効な値
instanceId	タスクと対応するワークフロー インスタンスの ID	有効なインスタンス ID を指定する文字列。 インスタンス ID を取得するには、次の com.bea.wlpi.common.TaskInfo メソッドを使用する。 public final java.lang.String getInstanceId() TaskInfo オブジェクトの取得方法については、22-10 ページの「ワークフロー インスタンス タスクの取得」を参照。TaskInfo オブジェクトで選択可能なメソッドの詳細については、B-21 ページの「TaskInfo オブジェクト」を参照。
taskId	完了または未完了マークを付けるタスクの ID	有効なタスク ID を指定する文字列。 タスク ID を取得するには、次の com.bea.wlpi.common.TaskInfo メソッドを使用する。 public final java.lang.String getTaskId() TaskInfo オブジェクトの取得方法については、22-10 ページの「ワークフロー インスタンス タスクの取得」を参照。TaskInfo オブジェクトで選択可能なメソッドの詳細については、B-21 ページの「TaskInfo オブジェクト」を参照。
priority	タスクの優先順位	0 (低)、1 (中) または 2 (高)。
doneWithoutDoit	21-19 ページの「タスクへの完了または未完了マークの付与」に記載のメソッドを使用してタスクに完了マークを付けるパーミッション	true (有効化) または false (無効化)。

表 21-7 taskSetProperties() メソッドのパラメータ (続き)

パラメータ	説明	有効な値
doitIfDone	21-6 ページの 「タスクの実行」 に記載のメソッドを使用して、完了マークを付けた後にそのタスクを実行するパーミッション	true (有効化) または false (無効化)。
unmarkDone	21-19 ページの 「タスクへの完了または未完了マークの付与」 に記載のメソッドを使用して、タスクに未完了マークを付けるパーミッション。	true (有効化) または false (無効化)。
modifiable	taskSetProperties() メソッドを使用して、実行時プロパティを変更するパーミッション	true (有効化) または false (無効化)。
reassignable	21-13 ページの 「タスクの割り当て」 に記載のメソッドを使用して、タスク インスタンスを別の参加コンポーネントに再び割り当てるパーミッション。	true (有効化) または false (無効化)。

このメソッドは、A-33 ページの 「クライアント要求 DTD」 に記載のとおり、Client Request DTD の ClientReq.dtd に準拠する XML ドキュメントを返します。この XML ドキュメントは、実行中のインスタンスの情報を含み、たとえば、SAX (Simple API for XML) パーサーのような XML 構文解釈ツールを使用してドキュメント解析によるアクセスが可能です。SAX パーサーの実装サンプルについては、21-26 ページの 「実行時タスクの管理例」 を参照してください。

たとえば、次のコードでは、デフォルトのタスクの優先順位が中 (1) に設定され、他のパラメータが有効 (true に設定) 化され、clientReq 変数に結果の XML ドキュメントが保存されます。このコード例では、worklist は Worklist EJB への EJBObject 参照を表します。

```
String clientReq = worklist.taskSetProperties(
   task.getTemplateDefinitionId(),
   task.getInstanceId(),
   task.getTaskId(),
   1,
   true,
   true,
   true,
```

```
    true,
    true
);
```

テンプレート定義、ワークフロー インスタンス、およびタスク ID は、`com.bea.wlpi.common.TaskInfo` オブジェクト、`task` と関連するメソッドを使用して取得されます。`task` オブジェクトは、21-4 ページの 「すべてのタスクの取得」 に記載のメソッドを使用して取得されます。

`com.bea.wlpi.common.TaskInfo` メソッドの詳細については、B-21 ページの「TaskInfo オブジェクト」 または、Javadoc の `com.bea.wlpi.common.TaskInfo` を参照してください。

`taskSetProperties()` メソッドの詳細については、Javadoc の `com.bea.wlpi.server.worklist.Worklist` を参照してください。

インスタンス変数の更新

ワークフロー インスタンス変数の更新については、23-1 ページの 「実行時の変数のモニタリング」 を参照してください。

例外ハンドラの呼び出し

例外ハンドラの呼び出しについては、24-11 ページの 「ワークフロー例外ハンドラの呼び出し」 を参照してください。

実行時タスクの管理例

以下の節では、コマンドラインと JSP Worklist の例から抜粋して、Xerces SAX パーサーの実装を含む実行時タスクの管理方法を示します。

コマンドライン Worklist サンプル

この節では、コマンドライン Worklist サンプルから抜粋して、実行時タスクの管理方法を示します。

注意： コマンドライン Worklist サンプルの詳細については、1-26 ページの「コマンドライン Worklist サンプル」を参照してください。

この例では、ユーザは以下のアクションのいずれかの指定を要求されます。

- タスク数を取得する
- すべてのタスクを取得する
- タスクを割り当てる
- タスクを実行する
- タスクに完了マークを付ける
- タスク プロパティを設定する
- タスクの割り当てを解除する
- タスクに未完了マークを付ける

適用可能なコードのアクションは、以下のように提供されます。また注意が必要な行は**太字**で示されます。このコード例では、`worklist` は `Worklist` EJB への `EJBObject` 参照を表します。

このサンプルでは、ユーザと通信するための入力ストリームが定義されており、ユーザは以下のアクションのいずれかを指定するよう求められます。

```
/* ユーザとの通信のための入力ストリームを作成する */
stdIn = new BufferedReader( new InputStreamReader( System.in ) );

/* ツール タイトルを表示する */
System.out.print( "\n---  Command Line Worklist v1.2  ---" );

/* メイン メニューを表示してユーザと交信する */
while( true ) {
   /* メニューを表示する */
   System.out.println( "\n---           Main Menu          ---" );
   System.out.println( "\nEnter choice:" );
   System.out.println( "1) Organizations" );
   System.out.println( "2) Workflows" );
```

```
   System.out.println( "3) Tasks" );
   System.out.println( "Q) Quit" );
   System.out.print( ">> " );
   .
   .
   .
```

mngTasks() メソッドは、ユーザと交信して必要な情報を取り出しながらユーザを管理する方法を示しています。

```
public static void mngTasks( ) {
   boolean isForCurrentUser;
   boolean isRole, isUserInRole;
   boolean isDoneWithoutDoit, isDoitIfDone, isUnmarkDone, isModifiable,
      isReassignment;
   int priority;
   String answer;
   String assigneeId, instanceId, orgId, taskId, templateDefId;

   /* ユーザとの通信のための入力ストリームを作成する */
   BufferedReader stdIn = new BufferedReader(
      new InputStreamReader( System.in ) );

   try {
      /* メイン メニューを表示してユーザと交信する */
      while( true ) {
         /* メニューを表示する */
         System.out.println( "\n\n---              Tasks            ---" );
         System.out.println( "\nEnter choice:" );
         System.out.println( "1) Get Tasks Count" );
         System.out.println( "2) List Tasks" );
         System.out.println( "3) Assign a Task" );
         System.out.println( "4) Execute a Task" );
         System.out.println( "5) Mark a Task as Done" );
         System.out.println( "6) Set Task Properties" );
         System.out.println( "7) Unassign a Task" );
         System.out.println( "8) Unmark a Task as Done" );
         System.out.println( "B) Back to previous menu" );
         System.out.println( "Q) Quit" );
         System.out.print( ">> " );

         /* ユーザの選択を取得する */
         String line = stdIn.readLine( );

         /* ユーザが選択を行わないで〔Enter〕を押したか ? */
         if( line.equals( "" ) )
            continue;
```

```
/* ユーザが複数の文字を入力したか ? */
else if( line.length( ) > 1 ) {
   System.out.println( "*** Invalid choice" );
   continue;
}

/* 大文字および文字へのコンバート */
char choice = line.toUpperCase( ).charAt( 0 );

/* ユーザの選択を処理する */
switch( choice ) {
.
.
.
```

タスク数を取得する

タスクのルートの取得方法を示します。

```
/* タスク数を取得する */
case '1' :
   /* WLPI 公開 API メソッド */
   /* 注意 : 発生した例外を取り込むコードを
    * 追加するとよい */
   /* 配列より現在のユーザのタスク数を取り出す */
   /* 配列要素 0 = 割り当てられたタスクの総数 */
   /* 配列要素 1 = 期日超過しているタスク数 */
   int taskCounts[ ] = worklist.getTaskCounts();

   /* 割り当て済みのタスクがあるか ? */
   if( taskCounts[0] == 0 )
      System.out.println( "\nNo task assigned" );
   else {
      System.out.println( "\nTasks count:" );

   if( taskCounts[0] == 1 )
      System.out.println( "- 1 task assigned" );
   else
      System.out.println( "- " + taskCounts[0] + " tasks assigned" );

   /* 期日超過しているタスクはあるか ? */
   if( taskCounts[1] == 0 )
      System.out.println( "- No overdue task" );
   else if( taskCounts[1] == 1 )
      System.out.println( "- 1 overdue task" );
```

```
   else
      System.out.println( "- " + taskCounts[1] + " overdue tasks" );
   }
   break;
   .
   .
   .
```

すべてのタスクを取得する

すべてのタスクのリストを取得する方法を示します。

```
/* すべてのタスクをリスト アップする */
case '2' :
   /* 現在のユーザのタスクをリストするか ? */
   if( ( answer = askQuestion(
      "\nList Tasks for the current user (y/n)?" ) ) == null ) {
      /* ユーザによる操作の取り消し */
      System.out.println( "*** Cancelled" );
      break;
   }

   /* 応答を評価する */
   isForCurrentUser = ( answer.equals( "y" ) || answer.equals( "Y" ) );

   List taskList;

   if( isForCurrentUser )
      /* WLPI 公開 API メソッド */
      /* 注意 : 発生した例外を取り込むコードを
       * 追加するとよい */
      /* アクティブなユーザとアクティブな
       * オーガニゼーションのタスク リストを取得する */
      taskList = worklist.getTasks( );
   else {
      /* オーガニゼーション ID を取得する */
      if( ( orgId = askQuestion(
         "\nEnter Organization ID" ) ) == null ) {
         /* ユーザによる操作の取り消し */
         System.out.println( "*** Cancelled" );
         break;
      }

      /* Asignee ID を取得する */
      if( ( assigneeId = askQuestion(
```

```
      "Enter Assignee ID" ) ) == null ) {
   /* ユーザによる操作の取り消し */
   System.out.println( "*** Cancelled" );
   break;
   }

   /* Assignee がロールであるか ? */
   if( ( answer = askQuestion(
      "Is the Assignee a Role (y/n)?" ) ) == null ) {
      /* ユーザによる操作の取り消し */
      System.out.println( "*** Cancelled" );
      break;
   }

   /* 応答を評価する */
   isRole = ( answer.equals( "y" ) || answer.equals( "Y" ) );

   /* WLPI 公開 API メソッド */
   /* 注意 : 発生した例外を取り込むコードを
    * 追加するとよい */
   /* 指定したユーザとオーガニゼーションのタスク リストを取得する */
   taskList = worklist.getTasks( orgId, assigneeId, isRole );
}

/* 割り当て済みのタスクがあるか ? */
if( taskList.size( ) == 0 )
   System.out.println( "\nNo task assigned" );
else
   System.out.print( "\nAssigned Tasks:" );

/* リストを処理して、タスクを表示する */
for( int i = 0; i < taskList.size( ); i++ ) {
   /* リストから要素を取り出す */
   TaskInfo taskInfo = ( TaskInfo )taskList.get( i );

   /* WLPI 公開 API メソッド */
   /* 使用可能な属性のサブセットを取り出して表示する */
   System.out.println( "\n- Name: " + taskInfo.getName( ) );
   System.out.println( "  Template Definition ID: " +
      taskInfo.getTemplateDefinitionId( ) );
   System.out.println( "  Workflow Instance ID: " +
      taskInfo.getInstanceId( ) );
   System.out.println( "  Task ID: " + taskInfo.getTaskId( ) );
   System.out.print( "  Status: " + taskInfo.getStatus( ) );

   /* タスク状態を取り出して表示する */
   if( taskInfo.getStatus( ) == taskInfo.STATUS_PENDING )
```

```
            System.out.println( " (Pending)" );
         else if( taskInfo.getStatus( ) == taskInfo.STATUS_COMPLETE )
            System.out.println( " (Complete)" );
         else if( taskInfo.getStatus( ) == taskInfo.STATUS_OVERDUE )
            System.out.println( " (Overdue)" );
         else if( taskInfo.getStatus( ) == taskInfo.STATUS_INACTIVE )
            System.out.println( " (Inactive)" );


         else
            System.out.println( " (Unknown)" );
      }
      break;
      .
      .
      .
```

タスクを割り当てる

タスクの割り当て方法を示します。

```
/* タスクを割り当てる */
case '3' :
   /* テンプレート定義 ID を取得する */
   if( ( templateDefId = askQuestion(
      "\nEnter Template Definition ID" ) ) == null ) {
      /* ユーザによる操作の取り消し */
      System.out.println( "*** Cancelled" );
      break;
   }

   /* ワークフロー インスタンス ID を取得する */
   if( ( instanceId = askQuestion(
      "Enter Workflow Instance ID" ) ) == null ) {
      /* ユーザによる操作の取り消し */
      System.out.println( "*** Cancelled" );
      break;
   }

   /* タスク ID を取得する */
   if( ( taskId = askQuestion( "Enter Task ID" ) ) == null ) {
      /* ユーザによる操作の取り消し */
      System.out.println( "*** Cancelled" );
      break;
   }
```

```
/* Assignee ID を取得する */
if( ( assigneeId = askQuestion( "Enter Assignee ID" ) ) == null ) {
   /* ユーザによる操作の取り消し */
   System.out.println( "*** Cancelled" );
   break;
}

/* Assignee がロールであるか ? */
if( ( answer = askQuestion( "Assign to a role (y/n)?" ) ) == null ) {
   /* ユーザによる操作の取り消し */
   System.out.println( "*** Cancelled" );
   break;
}

/* 応答を評価する */
isRole = ( answer.equals( "y" ) || answer.equals( "Y" ) );

/* Assignee がロールの場合、ユーザにこのロールを割り当てるか? */
if( isRole ) {
   if( ( answer = askQuestion(
      "Assign to a user in this role (y/n)?" ) ) == null ) {
      /* ユーザによる操作の取り消し */
      System.out.println( "*** Cancelled" );
      break;
   }

   /* 応答を評価する */
   isUserInRole = ( answer.equals( "y" ) || answer.equals( "Y" ) );
}
else
isUserInRole = false;

/* タスクを割り当てる */
try {
   /* WLPI 公開 API メソッド */
   String clientReq = worklist.taskAssign(
      templateDefId, instanceId, taskId, assigneeId,
      isRole, isUserInRole );

   /* 正常終了（例外の発生なし）*/
   System.out.println( "- Success" );

   /* クライアントの要求（もしあれば）を処理するために、
    * サーバからの応答を解析する */
   parser.handleRequest( clientReq );
}
catch( Exception e ) {
```

```
            System.out.println( "*** Failed to assign the task" );
            System.err.println( e );
         }
         break;
.
.
.
```

タスクを実行する

タスクの実行方法を示します。

```
/* タスクを実行する */
case '4' :
   /* テンプレート定義 ID を取得する */
   if( ( templateDefId = askQuestion(
      "\nEnter Template Definition ID" ) ) == null ) {
      /* ユーザによる操作の取り消し */
      System.out.println( "*** Cancelled" );
      break;
   }

   /* ワークフロー インスタンス ID を取得する */
   if( ( instanceId = askQuestion(
      "Enter Workflow Instance ID" ) ) == null ) {
      /* ユーザによる操作の取り消し */
      System.out.println( "*** Cancelled" );
      break;
   }

   /* タスク ID を取得する */
   if( ( taskId = askQuestion( "Enter Task ID" ) ) == null ) {
      /* ユーザによる操作の取り消し */
      System.out.println( "*** Cancelled" );
      break;
   }

   /* タスクを実行する */
   try {
       /* WLPI 公開 API メソッド */
      String clientReq = worklist.taskExecute(
         templateDefId, instanceId, taskId );

      /* 正常終了（例外の発生なし）*/
      System.out.println( "- Success" );
```

```
      /* クライアントの要求（もしあれば）を処理するために、
       * サーバからの応答を解析 */
      parser.handleRequest( clientReq );
   }
   catch( Exception e ) {
      System.out.println( "*** Failed to execute the task" );
      System.err.println( e );
   }
   break;
   .
   .
   .
```

タスクに完了マークを付ける

タスクに完了マークを付ける方法を示します。

```
/* タスクに完了マークを付ける  */
case '5' :
   /* テンプレート定義 ID を取得する */
   if( ( templateDefId = askQuestion(
      "\nEnter Template Definition ID" ) ) == null ) {
      /* ユーザによる操作の取り消し */
      System.out.println( "*** Cancelled" );
      break;
   }

   /* ワークフロー インスタンス ID を取得する */
   if( ( instanceId = askQuestion(
      "Enter Workflow Instance ID" ) ) == null ) {
      /* ユーザによる操作の取り消し */
      System.out.println( "*** Cancelled" );
      break;
   }

   /* タスク ID を取得する */
   if( ( taskId = askQuestion( "Enter Task ID" ) ) == null ) {
      /* ユーザによる操作の取り消し */
      System.out.println( "*** Cancelled" );
      break;
   }

   /* タスクに完了マークを付ける */
   try {
```

```
            /* WLPI 公開 API メソッド */
            String clientReq = worklist.taskMarkDone(
               templateDefId, instanceId, taskId );

            /* 正常終了（例外の発生なし）*/
            System.out.println( "- Success" );

            /* クライアントの要求（もしあれば）を処理するために、
             * サーバからの応答を解析 */
            parser.handleRequest( clientReq );
         }
         catch( Exception e ) {
            System.out.println( "*** Failed to mark the task as done" );
            System.err.println( e );
         }
         break;
         .
         .
         .
```

タスク プロパティを設定する

タスク プロパティの設定方法を示します。

```
/* タスク プロパティを設定する */
case '6' :
   /* テンプレート定義 ID を取得する */
   if( ( templateDefId = askQuestion(
      "\nEnter Template Definition ID" ) ) == null ) {
      /* ユーザによる操作の取り消し */
      System.out.println( "*** Cancelled" );
      break;
   }

   /* ワークフロー インスタンス ID を取得する */
   if( ( instanceId = askQuestion(
      "Enter Workflow Instance ID" ) ) == null ) {
      /* ユーザによる操作の取り消し */
      System.out.println( "*** Cancelled" );
      break;
   }

   /* タスク ID を取得する */
   if( ( taskId = askQuestion( "Enter Task ID" ) ) == null ) {
      /* ユーザによる操作の取り消し */
```

```
            System.out.println( "*** Cancelled" );
            break;
        }

        /* タスク優先順位を取得する */
        if( ( answer = askQuestion(
            "Enter the Priority (0=low, 1, 2=high)" ) ) == null ) {
            /* ユーザによる操作の取り消し */
            System.out.println( "*** Cancelled" );
            break;
        }

        /* 応答を解析する */
        priority = Integer.parseInt( answer );

        /* 「実行せずに完了マークを付ける」プロパティを取得する */
        if( ( answer = askQuestion(
            "Can it be marked done without executing (y/n)?" ) ) == null ) {
        /* ユーザによる操作の取り消し */
            System.out.println( "*** Cancelled" );
            break;
        }

        /* 応答を評価する */
        isDoneWithoutDoit = ( answer.equals( "y" ) || answer.equals( "Y" ) );

        /* 「完了マークが付いていたら、再実行する」プロパティを取得する */
        if( ( answer = askQuestion(
            "Can it be re-executed if marked done (y/n)?" ) ) == null ) {
            /* ユーザによる操作の取り消し */
            System.out.println( "*** Cancelled" );
            break;
        }

        /* 応答を評価する */
        isDoitIfDone = ( answer.equals( "y" ) || answer.equals( "Y" ) );


        /* 「完了マークが付いていたらマークしない」プロパティを取得する */
        if( ( answer = askQuestion(
            "Can it be unmarked if marked done (y/n)?" ) ) == null ) {
            /* ユーザによる操作の取り消し */
            System.out.println( "*** Cancelled" );
            break;
        }

        /* 応答を評価する */
```

```
            isUnmarkDone = ( answer.equals( "y" ) || answer.equals( "Y" ) );

//          System.out.print( "Can the properties be modified
//             at runtime (y/n) >> " );

            /* 応答を取得する */
//          answer = stdIn.readLine( );

            /* ユーザによる操作の取り消しがあるか ? */
//          if( answer.equals( "" ) )
//          {
//             System.out.println( "*** Cancelled" );
//             break;
//          }

            /* 応答を評価する */
//          boolean isModifiable = (


//             answer.equals( "y" ) || answer.equals( "Y" ) );
            isModifiable = true;

            /* 「再割り当て」プロパティを取得する */
            if( ( answer = askQuestion(
               "Can it be reassigned (y/n)?" ) ) == null ) {
               /* ユーザによる操作の取り消し */
               System.out.println( "*** Cancelled" );
               break;
            }

            /* 応答を評価する */
            isReassignment = ( answer.equals( "y" ) || answer.equals( "Y" ) );

            /* タスク プロパティを設定する */
            try {
               /* WLPI 公開 API メソッド */
               String clientReq = worklist.taskSetProperties( templateDefId,
                  instanceId, taskId, priority, isDoneWithoutDoit, isDoitIfDone,
                   isUnmarkDone, isModifiable, isReassignment );

               /* 正常終了（例外の発生なし）*/
               System.out.println( "- Success" );

               /* クライアントの要求（もしあれば）を処理するために、
                * サーバからの応答を解析 */
               parser.handleRequest( clientReq );
            }
            catch( Exception e ) {
```

```
            System.out.println( "*** Failed to set the task properties" );
            System.err.println( e );
        }
        break;
        .
        .
        .
```

タスクの割り当てを解除する

タスクの割り当てを解除する方法を示します。

```
/* タスクの割り当てを解除する */
case '7' :
    /* テンプレート定義 ID を取得する */
    if( ( templateDefId = askQuestion(
        "\nEnter Template Definition ID" ) ) == null ) {
        /* ユーザによる操作の取り消し */
        System.out.println( "*** Cancelled" );
        break;
    }

    /* ワークフロー インスタンス ID を取得する */
    if( ( instanceId = askQuestion(
        "Enter Workflow Instance ID" ) ) == null ) {
        /* ユーザによる操作の取り消し */
        System.out.println( "*** Cancelled" );
        break;
    }

    /* タスク ID を取得する */
    if( ( taskId = askQuestion( "Enter Task ID" ) ) == null ) {
        /* ユーザによる操作の取り消し */
        System.out.println( "*** Cancelled" );
        break;
    }

    /* タスクの割り当てを解除する */
    try {
        /* WLPI 公開 API メソッド */
        worklist.taskUnassign( templateDefId, instanceId, taskId );

        /* 正常終了（例外の発生なし）*/
        System.out.println( "- Success" );
    }
```

```
   catch( Exception e ) {
      System.out.println( "*** Failed to unassign the task" );
      System.err.println( e );
   }
   break;
   .
   .
   .
```

タスクに未完了マークを付ける

タスクに未完了マークを付ける方法を示します。

```
/* タスクの完了マークを外す */
case '8' :
   /* テンプレート定義 ID を取得する */
   if( ( templateDefId = askQuestion(
      "\nEnter Template Definition ID" ) ) == null ) {
      /* ユーザによる操作の取り消し */
      System.out.println( "*** Cancelled" );
      break;
   }

   /* ワークフロー インスタンス ID を取得する */
   if( ( instanceId = askQuestion(
      "Enter Workflow Instance ID" ) ) == null ) {
      /* ユーザによる操作の取り消し */
      System.out.println( "*** Cancelled" );
      break;
   }

   /* タスク ID を取得する */
   if( ( taskId = askQuestion( "Enter Task ID" ) ) == null ) {
      /* ユーザによる操作の取り消し */
      System.out.println( "*** Cancelled" );
      break;
   }

   /* タスクの完了マークを外す */
   try {
      /* WLPI 公開 API メソッド */
      String clientReq = worklist.taskUnmarkDone(
         templateDefId, instanceId, taskId );

      /* 正常終了（例外の発生なし）*/
```

```
                System.out.println( "- Success" );

                /* クライアントの要求（もしあれば）を処理するために、
                 * サーバからの応答を解析 */
                parser.handleRequest( clientReq );
             }
             catch( Exception e ) {
                System.out.println( "*** Failed to unmark the task as done" );
                System.err.println( e );
             }
             break;

          /* 前のメニューに戻る */
          case 'B' :
             return;

          /* ツールを終了 */
          case 'Q' :
             /* サーバから切断 */
             disconnect( );
             System.exit( 1 );

          default:
             System.out.println( "*** Invalid choice" );


          }
       }
    }
    /* 「Unhandled」例外 */
    catch( Exception e ) {
       System.err.println( e );
    }
    return;
}
```

コマンドライン SAX パーサー サンプル

この節では、コマンドライン SAX パーサーから、抜粋して、SAX パーサーの設定方法と set-variables クライアント要求への応答の方法を示します。

注意： コマンドライン SAX パーサーのコード例の詳細については、1-26 ページの「コマンドライン SAX パーサ サンプル」 を参照してください。

message-box クライアント要求への応答方法の例は、21-47 ページの「クライアント要求を解析する」を参照してください。

SAX パーサーは、クライアント要求を解析するため、コマンドライン Worklist により使用されます。この節での抜粋例は、以下のとおりです。

- クライアント要求を解析する
- クライアント要求に対して応答する

注意が必要な行は**太字**で示されます。

クライアント要求を解析する

次のコード抜粋では、カスタム実行時管理クライアントにおける Xerces SAX パーサーの使用方法を示します。handleRequest() メソッドは、BPM サーバから受信したクライアント要求を解析するために、Xerces SAX パーサーを使用します。request 文字列は、A-33 ページの「クライアント要求 DTD」に記載のクライアント要求 DTD に適合した XML ドキュメントです。

```
private static final String DEFAULT_PARSER =
   "weblogic.apache.xerces.parsers.SAXParser";
   public void handleRequest( String request ) {
      if( request == null )
         return;

      /* ハンドラのインスタンスを取得する */
      ContentHandler contentHandler = new CLContentHandler();
      ErrorHandler errorHandler = new CLErrorHandler();

      try {
         /* パーサーをインスタンス化する */
         XMLReader parser = XMLReaderFactory.createXMLReader(
            DEFAULT_PARSER );

         /*  コンテンツ ハンドラを登録する */
         parser.setContentHandler( contentHandler );

         /* エラー ハンドラを登録する */
         parser.setErrorHandler( errorHandler );

         /* 文字列ストリームを作成する */
         Reader inReader = new StringReader( request );
         /* SAX パーサーのストリームをカプセル化する */
         InputSource inputSource = new InputSource( inReader );

         /* 要求を解析する */
         parser.parse( inputSource );
      }
```

```
            catch( Exception e ) {
            System.err.println( e );
         }
      }
   }
```

クライアント要求に対して応答する

以下のメソッドは、set-variable クライアント要求を取得するために、ユーザとやり取りを行います。これは、ActionSendXMLToClient イベントよりトリガされ、set-variable クライアント要求に応答します。

次のメソッドは、set-variable クライアント要求を取得します。

```
public void setVariables( List listVarInfo ) {
   String varValue;

   /* プロンプトを表示し値を取り出すためのリストを
    * 処理する */
   for( int i = 0; i < listVarInfo.size( ); i++ ) {
   /* リストから要素を取り出す */
      VariableInfo varInfo = ( VariableInfo )listVarInfo.get( i );

      /* 変数値をユーザに要求する */
      if( ( varValue = askQuestion(
         "- " + varInfo.getVarPrompt() ) ) != null ) {
         /* ユーザによる値の入力；その場合、保持された値を上書きする */
         varInfo.setVarValue( varValue );
         listVarInfo.set( i, ( VariableInfo )varInfo );
      }
   }
   return;
}
   .
   .
   .
```

次のメソッドは、set-variable クライアント要求に応答します。

```
private static String sendSetVariablesResponse(
   String templateDefinitionId, String instanceId,
   String actionId, List listVarInfo  ) {

   String clientReq = null;

   /* XML クライアント応答を保存する変数を作成し
    * ドキュメントのプロローグを追加する */
   StringBuffer response = new StringBuffer(
      "<?xml version=\"1.0\"?>\r\n" );
```

```
        /* XML ドキュメントのルートを追加する */
        response.append( "<set-variables>\r\n" );

        /* 変数を取り出すためのリストを処理する
         * NAME と値 */
        for( int i = 0; i < listVarInfo.size( ); i++ ) {
           /* VARIABLE ( 要素 ) と NAME ( 属性 ) を追加する */
           response.append( "  <variable name=\"" );

           /* リストから要素を取り出す */
           VariableInfo varInfo = ( VariableInfo )listVarInfo.get( i );

           /* NAME 属性の値を追加する */
           response.append( varInfo.getVarName() + "\">" );

           /* VARIABLE 要素の値を追加する */
           response.append( XMLToString( varInfo.getVarValue() ) );

           /* エンド タグを追加する */
           response.append( "</variable>\r\n" );
        }

        /* XML ドキュメントにルート エンド タグを追加する */
        response.append( "</set-variables>" );

        try {
           /* WLPI 公開 API メソッド */
           /* set-variables クライアント要求にクライアントの
            * 応答を送る */
           clientReq = clWorklist.getWorklist().response(
              templateDefinitionId, instanceId, actionId,
              response.toString() );

           /* 正常終了（例外の発生なし）*/
           System.out.println( "\nResponse sent to server" );
        }
        catch( Exception e ) {
           System.out.println( "\nFailed sending response to server" );
           System.err.println( e );
        }
        return( clientReq );
     }
     .
     .
     .
```

JSP Worklist サンプル

この節では、JSP Worklist サンプルから抜粋して、実行時タスクの管理方法を示します。

注意： JSP Worklist サンプルの詳細については、1-27 ページの「JSP Worklist サンプル」を参照してください。

抜粋例は、以下の機能を示します。

- タスクを取得する
- タスクを実行する
- クライアント要求を解析する
- クライアント要求に対して応答する
- タスクを割り当てる
- タスクに完了または未完了マークを付ける
- タスク プロパティを設定する

次のコード例では、注意が必要なコードを含むすべての行は**太字**で示されます。`worklist` は、`Worklist` EJB に対する `EJBObject` 参照を表します。

タスクを取得する

JSP Worklist (`worklist.jsp`) から抜粋して、タスクリストの取得方法とタスク情報を取得するための `com.bea.wlpi.common.TaskInfo` メソッドの使用方法を示します。

タスクリストを取得するためには、1-29 ページの「JSP Worklist へのプライマリインタフェース」の図に示すように、`Display Options` を設定します。

```
try {
   List v = worklist.getTasks(currentOrg, currentRole, isRole);
   session.putValue("tasklist", v);

   // Get item count
   int numItems = 0;
   for (int i = 0, n = v.size(); i < n; i++) {
      TaskInfo task = (TaskInfo)v.get(i);
      if (task.getStarted() != null) {
         if (task.getCompleted() != null) {
```

```
                    if (!done)
                       continue;
                 }
                 else {
                    if (!pending)
                       continue;
                    }
                 }
                 else if (!inactive)
                    continue;
                 numItems++;
              }
           }
           .
           .
           .
```

com.bea.wlpi.common.TaskInfo メソッドの詳細については、B-21 ページの「TaskInfo オブジェクト」または、Javadoc の com.bea.wlpi.common.TaskInfo を参照してください。

タスクを実行する

タスクの実行方法を JSP Worklist (worklist.jsp) から抜粋して示します。

ユーザは、1-29 ページの「JSP Worklist へのプライマリ インタフェース」の図に示すように、Perform Task タスクのリンクの検索や実行すべきタスクの検索をすることで、タスクを実行できます。

```
try {
   List vTaskList = (List)session.getValue("tasklist");
   TaskInfo task = (TaskInfo)vTaskList.get(Integer.parseInt(cmd));
   session.putValue("action", ACTION_DOIT);
   if (action.equals(ACTION_DOIT)) {
      System.out.println(worklist.taskExecute(task.getTemplateDefinitionId(),
         task.getInstanceId(), task.getTaskId()));
      responseParser.parse( worklist.taskExecute(
                              task.getTemplateDefinitionId(),
                              task.getInstanceId(),
                              task.getTaskId() )
      );
      if (responseParser.isQuery()) {
         pageContext.forward("query.jsp");
         return;
      }
      if (responseParser.isMessage()) {
         pageContext.forward("message.jsp");
         return;
```

```
    }
}
.
.
.
```

テンプレート定義、ワークフロー インスタンス、およびタスク ID は、`com.bea.wlpi.common.TaskInfo` オブジェクト、`task` と関連するメソッドを使用して取得されます。結果の XML ドキュメント（A-33 ページの「クライアント要求 DTD」に記述のように、クライアント要求 DTD に準拠した）は、`taskExecute()` メソッドから返されます。また、レスポンス パーサーである `ResponseParser` に渡され解析されます。レスポンス パーサーの詳細については、21-47 ページの「クライアント要求を解析する」を参照してください。

`com.bea.wlpi.common.TaskInfo` メソッドの詳細については、B-21 ページの「TaskInfo オブジェクト」または、Javadoc の `com.bea.wlpi.common.TaskInfo` を参照してください。

クライアント要求を解析する

次の JSP SAX パーサー（`ResponseParser`）からの抜粋は、SAX パーサーの設定方法と `message-box` クライアント要求への応答方法を示しています。`set-variables` クライアント要求への応答方法に関するサンプルは、コマンドライン SAX パーサーの 21-42 ページの「クライアント要求を解析する」を参照してください。

```
package jsp_servlet._worklist;

import java.io.*;
import java.util.*;
import javax.servlet.http.*;
import com.sun.xml.parser.Parser;
import org.xml.sax.*;

public class ResponseParser implements DocumentHandler {
      .
      .
      .
   public boolean isMessage() {
      return message;
   }
      .
      .
```

```
      .
   public String getMessageOption() {
      return ((String)messageBox.getOption());
   }
   public String getMessageStyle() {
      return ((String)messageBox.getStyle());
   }  .
      .
      .
   public String generateMessageResponse(HttpServletRequest request) {
      StringBuffer response = new StringBuffer();
      response.append(
         "<?xml version=\"1.0\"?>\r\n<message-box
         option=\""+request.getParameter("option")+"\"/>");
      System.out.println((String) response.toString());
      return((String) response.toString());
   }
      .
      .
      .
   public void parse(String xml) {
      try {
         parser = new Parser();
         parser.setDocumentHandler(this);
         parser.parse(new InputSource(new ByteArrayInputStream(
           xml.getBytes())));
      } catch (SAXParseException spe) {
      } catch (Exception e) {
      }
   }
.
.
.
```

クライアント要求に対して応答する

この節では、JSP Worklist から抜粋したクライアント要求に対する応答方法を示す 2 つのコード例を紹介します。

第 1 の抜粋は、`worklist.jsp` ファイルより選択されています。`responseParser` は、XML ドキュメントを解析します。`set-variables` 要求に出会うと、コンテキストはファイル `query.jsp` に渡されます。`message-box` 要求に出会うと、コンテキストはファイル `message.jsp` に渡されます。レスポンス パーサーの詳細については、21-47 ページの「クライアント要求を解析する」を参照してください。

```
if (action.equals(ACTION_DOIT)) {
   responseParser.parse( worklist.taskExecute(
                            task.getTemplateDefinitionId(),
                            task.getInstanceId(),
                            task.getTaskId() )
   );
   if (responseParser.isQuery()) {
      pageContext.forward("query.jsp");
      return;
   }
   if (responseParser.isMessage()) {
      pageContext.forward("message.jsp");
     return;
   }
}
.
.
.
```

第 2 の抜粋では、response() メソッド呼び出しが示され、message.jsp ファイルの内容が示されます。このメソッドでは、XML パーサー（responseParser）を利用して、XML (その際、テンプレート定義、インスタンス ID およびワークフロー オブジェクト ID が取得される) および要求に基づいた応答が生成されます。レスポンス パーサーの詳細については、21-47 ページの「クライアント要求を解析する」を参照してください。

```
<%@ page import="
   javax.ejb.*,
   java.rmi.RemoteException,
   java.util.*,
   java.text.*,
   com.bea.wlpi.common.*
   com.bea.wlpi.server.worklist.*
"%>

<jsp:usebean id="responseParser" class="jsp_servlet._worklist.ResponseParser"
scope="session"/>

<%
   Worklist worklist = null;
   TaskInfo task = null;
   String error = null;
   String resp = null;
   String messageOption = null;
   worklist = (Worklist)session.getValue("worklist");

   if (worklist != null) {
      Hashtable h = new Hashtable();
```

```
      h.put(javax.naming.Context.SECURITY_PRINCIPAL, session.getValue("name"));
      h.put(javax.naming.Context.SECURITY_CREDENTIALS,
         session.getValue("password"));
      new javax.naming.InitialContext(h);

   if (request.getParameter("option") != null) {
      try {
         System.out.println(responseParser.generateMessageResponse(request));
         System.out.println(responseParser.getTemplateDefinitionId());
         System.out.println(responseParser.getInstanceId());
         System.out.println(responseParser.getActionId());

         resp = worklist.response(
                   responseParser.getTemplateDefinitionId(),
                   responseParser.getInstanceId(),
                   responseParser.getActionId(),
                   responseParser.generateMessageResponse(request)
         );
         pageContext.forward("worklist.jsp");
         return;
      } catch (Exception e) {
         e.printStackTrace();
      }
      }
   }
%>
<html>
<head>
<title>Message</title>
</head>
<body bgcolor=#ffffff>
<font face="Arial" size="4">
<form action="message.jsp" method="POST">
<table dir="ltr" border="0" cellpadding="1" cellspacing="1">
   <tr>
      <td><img src="logo.gif"></td>
      <td><font color="red" size="7">
         <strong><em>Weblogic</em></strong></font>
         <font color="black" size="7">
         <strong><em>Process Integrator</em></strong></font>
      </td>
   </tr>
   <tr>
      <td bgcolor="navy" valign="top">
      <table dir="ltr" border="0" cellpadding="2" cellspacing="2">
         <tr>
            <td><font color="yellow">
               <strong>Go ...</strong></font></td>
         </tr>
```

```
         <tr>
            <td><a href="worklist.jsp">
               <font color="white">Worklist</font></a></td>
         </tr>
         <tr>
            <td><a href="startworkflow.jsp">
               <font color="white">Start Workflow</font></a></td>
         </tr>
         <tr>
            <td><a href="logoff.jsp">
               <font color="white">Logoff</font></a></td>
         </tr>
      </table>
      </td>
      <td valign="top">
      <table dir="ltr" border="0" cellpadding="2" cellspacing="2" valign="top">
<%
    if (error != null) {
%>


         <tr>
            <td><font color="red size="4" face="Arial"><%=error%></td>
         </tr>
<%
       }
%>
         <tr>
            <td><font color="navy" size="4" face="Arial">
<%=responseParser.getHeader()%></font></td>
         </tr>
         <tr>
            <td>
            <table border="0" cellpadding="2" cellspacing="2">
               <tr>
                  <td></td>
<%
   messageOption = responseParser.getMessageOption();
   if (messageOption.equals("ok")) {
%>
                  <td width="10%">
                     <input type="submit" name="option" value="ok"></td>
                  <td width="90%"> </td>
<%
   }
%>
<%
   if (messageOption.equals("ok_cancel")) {
%>
```

```
                  <td width="10%">
                     <input type="submit" name="option" value="ok"></td>
                  <td width="10%">
                     <input type="submit" name="option" value="cancel"></td>
                  <td width="80%"> </td>

<%
   }
%>
<%
   if (messageOption.equals("yes_no")) {
%>
                  <td width="10%">
                     <input type="submit" name="option" value="yes"></td>
                  <td width="10%">
                     <input type="submit" name="option" value="no"></td>
                  <td width="80%"> </td>
<%
   }
%>
<%


   if (messageOption.equals("yes_no_cancel")) {
%>
                  <td width="10%">
                     <input type="submit" name="option" value="yes"></td>
                  <td width="10%">
                     <input type="submit" name="option" value="no"></td>
                  <td width="10%">
                     <input type="submit" name="option" value="cancel"></td>
                  <td width="70%"> </td>
<%
   }
%>
               </tr>
            </table>
         </td>
      </tr>
   </table>
   </td>
</tr>
</table>
</form>
</font>
</body>
</html>
```

タスクを割り当てる

この節では、JSP Worklist から抜粋した、異なる参加者（ユーザまたはロール）へのタスクを割り当てる方法を示す 4 つのコード例を紹介します。

ユーザは、1-29 ページの「JSP Worklist へのプライマリ インタフェース」の図に示すように Reassign リンクを検索し、異なる参加者へのタスクの割り当てを変更できます。また、再割り当てをすべきタスクを検索したり、ユーザ、ロールまたはタスクの再割り当てを要求されたときのロール内のユーザを特定することによっても、タスクの割り当てを変更できます。

次のコード例では、worklist.jsp ファイルを参照する Reassign リンクの SetAction パラメータが 4 にセットされ、チェックマークをビジュアルな手がかりとして、次のアクションの左側に追加されます。

```
<tr>
   <td><%=action.equals("4") ? "<img src=\"check.gif\">" : ""%></td>
   <td><a href="worklist.jsp?SetAction=4">
      <font color="white">Reassign</font></a></td>
</tr>
```

次に示すように、SetAction 値は action 変数から抽出され保存されます。

```
if (worklist != null) {
   String cmd = request.getParameter("SetFilter");
   if (cmd != null) {
      .
      .
      .
   }
   else if (request.getParameter("SetAction") != null)
   {
      action = request.getParameter("SetAction");
      session.putValue("action", action);
   }
.
.
.
```

action 変数の値に基づき、コンテキストは、次のように reassign.jsp ファイルに送られます。

```
static final String ACTION_DOIT = "0";
static final String ACTION_MARKDONE = "1";
static final String ACTION_UNMARKDONE = "2";
static final String ACTION_TASKPROPERTIES = "3";
static final String ACTION_REASSIGN = "4";
```

```
    .
    .
    .
try {
   List vTaskList = (List)session.getValue("tasklist");
   TaskInfo task = (TaskInfo)vTaskList.get(Integer.parseInt(cmd));
   session.putValue("action", ACTION_DOIT);
   if (action.equals(ACTION_DOIT)) {
      .
      .
      .
   }
   else if (action.equals(ACTION_REASSIGN)) {
      session.removeValue(“error”);
      pageContext.forward(“reassign.jsp?task= “+cmd);
      return;
   }
.
.
.
```

次のコード例は、reassign.jsp ファイルの内容から抜き出した、タスクの再割り当てを行うための taskAssign() メソッドの使用方法を示しています。テンプレート定義、ワークフロー インスタンス、およびタスク ID は、com.bea.wlpi.common.TaskInfo オブジェクト、task と関連するメソッドを使用して取得されます。task オブジェクトは、21-4 ページの「すべてのタスクの取得」に記載のメソッドを使用して取得されます。

```
<%@ page import="
   javax.ejb.*,
   java.rmi.RemoteException,
   java.util.*,
   java.text.*,
   com.bea.wlpi.common.*
   com.bea.wlpi.server.worklist.*
   com.bea.wlpi.server.principal.*
   javax.naming.*
"%>
<%!
   static final String TYPE_USER       = "0";
   static final String TYPE_USERINROLE = "1";
   static final String TYPE_ROLE       = "2";
%>
<%
   Worklist worklist = null;
   TaskInfo task = null;
   String error = null;
   String assignType = TYPE_USER;
```

```
worklist = (Worklist)session.getValue("worklist");
if (worklist != null) {
   Hashtable h = new Hashtable();
   h.put(javax.naming.Context.SECURITY_PRINCIPAL, session.getValue("name"));
    h.put(javax.naming.Context.SECURITY_CREDENTIALS,
        session.getValue("password"));
   new javax.naming.InitialContext(h);
   String cmd = request.getParameter("task");
   if (cmd != null) {
      try {
         List vTaskList = (List)session.getValue("tasklist");
         task = (TaskInfo)vTaskList.get(Integer.parseInt(cmd));
         session.putValue("task", task);
      } catch (Exception e) {
      }
   }
   else
      task = (TaskInfo)session.getValue("task");
}
if (request.getParameter("Set") != null)
   assignType = request.getParameter("Type");
if (request.getParameter("Reassign") != null) {
   // Set properties
   try {
      assignType = request.getParameter("Type");
      boolean bRole       = false;
      boolean bUserInRole = false;
      if (!assignType.equals(TYPE_USER)) {
         bRole = true;
         if (assignType.equals(TYPE_USERINROLE))
            bUserInRole = true;
      }
      worklist.taskAssign(task.getTemplateDefinitionId(),
                          task.getInstanceId(),
                          task.getTaskId(),
                          request.getParameter("Assignee"),
                          bRole,
                          bUserInRole
      );
      session.removeValue("task");
      session.removeValue("error");
      pageContext.forward("worklist.jsp");
      return;
   } catch (Exception e) {
      error = e.getMessage();
   }
}
if (task != null) {
```

```
%>
<html>
<head>
<title>Reassign Task</title>
</head>
<body bgcolor=#ffffff>
<font face="Arial" size="4">
<form action="reassign.jsp" method="POST">
<table dir="ltr" border="0" cellpadding="1" cellspacing="1">
   <tr>
      <td><img src="logo.gif"></td>
      <td><font color="red" size="7"><strong><em>Weblogic</em></strong></font>
         <font color="black" size="7"><strong><em>
         Process Integrator</em></strong></font></td>
   </tr>
   <tr>
      <td bgcolor="navy" valign="top">
      <table dir="ltr" border="0" cellpadding="2" cellspacing="2" >
         <tr>
            <td><font color="yellow">
            <strong>Go ...</strong></font></td>
         </tr>


         <tr>
            <td><a href="worklist.jsp">
            <font color="white">Worklist</font></a></td>
         </tr>
         <tr>
            <td><a href="startworkflow.jsp">
            <font color="white">Start Workflow</font></a></td>
         </tr>
         <tr>
            <td><a href="logoff.jsp">
            <font color="white">Logoff</font></a></td>
         </tr>
      </table>
      </td>
      <td valign="top">
      <table dir="ltr" border="0" cellpadding="2" cellspacing="2" valign="top">
<%
   if (error != null) {
%>
         <tr>
            <td><font color="red size="4" face="Arial"><%=error%></td>
         </tr>
<%      }
%>
         <tr>
```

```
            <td><font color="navy" size="5" face="Arial">Reassign Task:
               <strong><%=task.getName()%></strong></font></td>
         </tr>
         <tr>
            <td>
            <table border="0" cellpadding="2" cellspacing="2">
               <tr>
                  <td><p align="right">Type:</p></td>
                  <td>
                     <select name="Type" size="1">
<option <%=assignType.equals(TYPE_USER) ? "selected" : ""%> value="0">
   User
</option>
<option <%=assignType.equals(TYPE_USERINROLE) ? "selected" : ""%> value="1">
   User in Role|
</option>
<option <%=assignType.equals(TYPE_ROLE) ? "selected" : ""%> value="2">
   Role
</option>
                     </select>
                     <input type="submit" name="Set" value="Set "></td>
               </tr>
               <tr>
                  <td valign="top"><p align="right">Assign To:</p></td>


                  <td><select name="Assignee" size="10">
<%
   if (assignType.equals(TYPE_USER)) {
      List vUsers = (List)session.getValue("users");
      if (vUsers == null)
      try {
         Context ctx = (Context)session.getValue("ctx");
         WLPIPrincipalHome pHome =
            (WLPIPrincipalHome)ctx.lookup("com.bea.wlpi.WLPIPrincipal");
         WLPIPrincipal principal = (WLPIPrincipal)pHome.create();
         vUsers = principal.getUsersInOrganization("ORG1", false);
         session.putValue("users", vUsers);
         principal.remove();
      } catch (Exception e) {
      }
      for (int i = 0, n = vUsers.size(); i < n; i++) {
         String name = ((UserInfo)vUsers.get(i)).getUserId();
         out.print("<option value=\"" + name + "\">" + name + "</option>");
      }
   }
   else {
      List vRoles = (List)session.getValue("roles");
      if (vRoles == null)
```

```
         try {
            Context ctx = (Context)session.getValue("ctx");
            WLPIPrincipalHome pHome =
               (WLPIPrincipalHome)ctx.lookup("com.bea.wlpi.WLPIPrincipal");
            WLPIPrincipal principal = (WLPIPrincipal)pHome.create();
            vRoles = principal.getRolesInOrganization("ORG1", false);
            session.putValue("roles", vRoles);
            principal.remove();
         } catch (Exception e) {}
      for (int i = 0, n = vRoles.size(); i < n; i++) {
         String name = ((RoleInfo)vRoles.get(i)).getRoleId();
         out.print("<option value=\"" + name + "\">" + name + "</option>");
      }
   }
%>
                     </select>
                  </td>
               </tr>
               <tr>
                  <td></td>
                  <td><input type="submit" name="Reassign" value="Reassign ">
                  </td>
               </tr>
            </table>
         </td>
      </tr>


   </table>
   </td>
</tr>
</table>
</form>
<%
}
%>
</font>
</body>
</html>
```

タスクに完了または未完了マークを付ける

次のコード例は、JSP Worklist からの抜粋で、タスクへの完了マークまたは未完了マークを付ける方法を示します。

ユーザは、1-29 ページの「JSP Worklist へのプライマリ インタフェース」の図に示すように Mark Done または Mark Undone リンクを検索することで、タスクに完了、未完了のマークを付けることができます。

次のコード例では、worklist.jsp ファイルを参照する Mark Done および Mark Undone リンクの SetAction パラメータが 1 および 2 にセットされ、チェックマークをビジュアルな手がかりとして、次のアクションの左側に追加されます。

```
<tr>
   <td><%=action.equals("1") ? "<img src=\"check.gif\">" : ""%></td>
   <td><a href="worklist.jsp?SetAction=1"><font color="white">
      Mark Done</font></a>
   </td>
</tr>
<tr>
   <td><%=action.equals("2") ? "<img src=\"check.gif\">" : ""%></td>
   <td><a href="worklist.jsp?SetAction=2"><font color="white">
      Unmark Done</font></a>
   </td>
</tr>
```

次に示すように、SetAction 値は action 変数から抽出され保存されます。

```
if (worklist != null) {
   String cmd = request.getParameter("SetFilter");
   if (cmd != null) {
   .
   .
   .
   else if (request.getParameter("SetAction") != null) {
      action = request.getParameter("SetAction");
      session.putValue("action", action);
   }
```

次の worklist.jsp ファイルから取り出したコードは、それぞれのタスクに完了または未完了マークを付けるために、taskMarkDone() および taskUnmarkDone() を使用する方法を示します。

```
try {
   List vTaskList = (List)session.getValue("tasklist");
   TaskInfo task = (TaskInfo)vTaskList.get(Integer.parseInt(cmd));
   session.putValue("action", ACTION_DOIT);
   if (action.equals(ACTION_DOIT)) {
      .
      .
      .
   }
   else if (action.equals(ACTION_MARKDONE))
```

```
      worklist.taskMarkDone(task.getTemplateDefinitionId(),
                            task.getInstanceId(),
                            task.getTaskId());

   else if (action.equals(ACTION_UNMARKDONE))
    worklist.taskUnmarkDone(task.getTemplateDefinitionId(),
                            task.getInstanceId(),
                            task.getTaskId());

.
.
.
```

タスク プロパティを設定する

この節では、JSP Worklist から抜粋した、タスク プロパティの設定を示す多くのコード例を紹介します。

ユーザは、1-29 ページの「JSP Worklist へのプライマリ インタフェース」の図に示すように、Task Properties リンクあるいはタスクの検索または必要なときに要求されるプロパティを指定することでタスクプロパティの設定を行うことができます。

次のコード例では、worklist.jsp ファイルを参照する Task Properties リンクの SetAction パラメータを 3 にセットし、チェックマークをビジュアルな手がかりとして、次のアクションの左側に追加します。

```
<tr>
   <td><%=action.equals("4") ? "<img src=\"check.gif\">" : ""%></td>
   <td><a href="worklist.jsp?SetAction=3"><font color="white">
      Task Properties</font></a></td>
</tr>
```

次に示すように SetAction 値は action 変数から抽出され保存されます。

```
if (worklist != null) {
   String cmd = request.getParameter("SetFilter");
   if (cmd != null) {
      .
      .
      .
   else if (request.getParameter("SetAction") != null) {
      action = request.getParameter("SetAction");
      session.putValue("action", action);
```

```
   }
.
.
.
```

action 変数の値に基づき、コンテキストは、次の例のように taskproperties.jsp ファイルに送られます。

```
static final String ACTION_DOIT = "0";
static final String ACTION_MARKDONE = "1";
static final String ACTION_UNMARKDONE = "2";
static final String ACTION_TASKPROPERTIES = "3";
static final String ACTION_REASSIGN = "4";
   .
   .
   .
try {
   List vTaskList = (List)session.getValue("tasklist");
   TaskInfo task = (TaskInfo)vTaskList.get(Integer.parseInt(cmd));
   session.putValue("action", ACTION_DOIT);
   if (action.equals(ACTION_DOIT)) {
      .
      .
      .
   else if (action.equals(ACTION_TASKPROPERTIES)) {
      session.removeValue(“error”);
      pageContext.forward(“taskproperties.jsp?task= “+cmd);;
```

最後に、タスク プロパティが設定されます。taskproperties.jsp ファイルから抜粋された次のコード例は、タスクのプロパティを設定するために taskSetProperties() メソッドを使用する方法を示します。

```
<%@ page import="
   javax.ejb.*,
   java.rmi.RemoteException,
   java.util.*,
   java.text.*,
   com.bea.wlpi.common.*
   com.bea.wlpi.server.worklist.*
"%>

<%!
   DateFormat df = new SimpleDateFormat("MMM d, yyyy h:mm a");
%>

<%
   Worklist worklist = null;
   TaskInfo task = null;
```

```
   String error = null;

   worklist = (Worklist)session.getValue("worklist");

   if (worklist != null) {
      Hashtable h = new Hashtable();
      h.put(javax.naming.Context.SECURITY_PRINCIPAL, session.getValue("name"));
      h.put(javax.naming.Context.SECURITY_CREDENTIALS,
session.getValue("password"));
      new javax.naming.InitialContext(h);

      String cmd = request.getParameter("task");
      if (cmd != null) {
         try {
            List vTaskList = (List)session.getValue("tasklist");
            task = (TaskInfo)vTaskList.get(Integer.parseInt(cmd));
         } catch (Exception e) {
         }
      }
   }

   if (request.getParameter("Set") != null) {
   // Set properties
   task = (TaskInfo)session.getValue("task");
   try {
      worklist.taskSetProperties( task.getTemplateDefinitionId(),
         task.getInstanceId(), task.getTaskId(),
         Integer.parseInt(request.getParameter("Priority")),
         request.getParameter("P0") != null,
         request.getParameter("P1") != null,
         request.getParameter("P2") != null,
         request.getParameter("P3") != null,
         request.getParameter("P4") != null);
      session.removeValue("task");


      session.removeValue("error");
      pageContext.forward("worklist.jsp");
      return;
   } catch (Exception e) {
      error = e.getMessage();
   }
   }
   if (task != null) {
      session.putValue("task", task);
%>
<html>
<head>
<title>Task Properties</title>
```

```
</head>

<body bgcolor=#ffffff>
<font face="Arial" size="4">

<form action="taskproperties.jsp" method="POST">
<table dir="ltr" border="0" cellpadding="1" cellspacing="1">
   <tr>
      <td><img src="logo.gif"></td>
      <td>
         <font color="red" size="7">
         <strong><em>Weblogic</em></strong></font>
         <font color="black" size="7">
         <strong><em>Process Integrator</em></strong></font>
      </td>
   </tr>
   <tr>
      <td bgcolor="navy" valign="top">
         <table dir="ltr" border="0" cellpadding="2" cellspacing="2" >
            <tr>
               <td><font color="yellow">
            <strong>Go ...</strong></font>
         </td>
      </tr>
      <tr>
         <td><a href="worklist.jsp">
            <font color="white">Worklist</font></a>
         </td>
      </tr>
      <tr>
         <td><a href="startworkflow.jsp">
            <font color="white">Start Workflow</font></a>
         </td>
      </tr>
      <tr>
         <td><a href="logoff.jsp">


            <font color="white">Logoff</font></a></td>
      </tr>
   </table>
   </td>
   <td valign="top">
   <table dir="ltr" border="0" cellpadding="2" cellspacing="2" valign="top">
<%
    if (error != null) {
%>
   <tr>
      <td><font color="red size="4" face="Arial"><%=error%></td>
```

```
   </tr>
<%      }
%>
   <tr>
      <td><font color="navy" size="5" face="Arial">
         Task Properties for:
         <strong><%=task.getName()%></strong></font>
      </td>
   </tr>
   <tr>
      <td>
      <table border="0" cellpadding="2" cellspacing="2">
         <tr>
            <td><p align="right">Workflow:</p></td>
            <td><%=task.getWorkflow()%> <%=task.getWorkflowId()%></td>
         </tr>
         <tr>
            <td><p align="right">Comment:</p></td>
            <td><%=task.getComment()%></td>
         </tr>
         <tr>
            <td><p align="right">Started:</p></td>
           <td><%=task.getStarted() == null ? "" : df.format(task.getStarted())%>
            </td>
         </tr>
         <tr>
            <td><p align="right">Completed:</p></td>
             <td><%=task.getCompleted() == null ? "" :
df.format(task.getCompleted())%>
            </td>
         </tr>
         <tr>
            <td><p align="right">Due:</p></td>
            <td><%=task.getDue() == null ? "" : df.format(task.getDue())%></td>
         </tr>
         <tr>
            <td><p align="right">Priority:</p></td>
            <td><select name="Priority" size="1">
               <option


                 <%=task.getPriority()==0 ? "selected" : ""%> value="0">Low
               </option>
               <option
                  <%=task.getPriority()==1 ? "selected" : ""%> value="1">Medium
               </option>
               <option
                  <%=task.getPriority()==2 ? "selected" : ""%> value="2">High
               </option>
```

```
                </select>
             </td>
          </tr>
          <tr>
             <td valign="top"><p align="right">Permissions:</p></td>
             <td>
             <table border="0" cellpadding="0" cellspacing="0">
             <tr>
                <td>
                   <input type="checkbox" name="P0"
                   <%=task.getDoneWithoutDoit() ? "checked" : ""%> value="ON">
                   Mark done without executing
                </td>
             </tr>
             <tr>
                <td>
                   <input type="checkbox" name="P1"
                      <%=task.getDoitIfDone() ? "checked" : ""%> value="ON">
                      Re-execute if marked done
                </td>
             </tr>
             <tr>
                <td><input type="checkbox" name="P2"
                      <%=task.getUnmarkDone() ? "checked" : ""%> value="ON">
                      Unmark if marked done
                </td>
             </tr>
   <tr>
      <td><input type="checkbox" name="P3"
         <%=task.getModifiable() ? "checked" : ""%> value="ON">
         Allow modification
      </td>
   </tr>
   <tr>
      <td><input type="checkbox" name="P4"
         <%=task.getReassignment() ? "checked" : ""%> value="ON">
         Allow reassignment
      </td>
   </tr>
</table>
</td>


</tr>
<tr>
   <td></td>
   <td><input type="submit" name="Set" value="Set Properties "></td>
</tr>
</table>
```

```
</td>
</tr>
</table>
</td>
</tr>
</table>
</form>
<%
    }
%>
</font>
</body>
</html>
```

テンプレート定義、インスタンス ID、タスク ID および現在のタスクのプロパティは、`com.bea.wlpi.common.TaskInfo` オブジェクト、`task` と関連するメソッドを使用して取得されます。`task` オブジェクトは、21-4 ページの「すべてのタスクの取得」に記載のメソッドを使用して取得されます。

Part V　モニタ

22 実行時ワークフロー インスタンスのモニタ

この章では、実行時ワークフロー インスタンスをモニタする方法について説明します。内容は以下のとおりです。

- ワークフロー インスタンスの取得
- ワークフロー インスタンスの確認
- ワークフロー インスタンス タスクの取得
- ワークフロー インスタンス情報の取得
- ワークフロー インスタンスのアカウントの取得
- ワークフロー インスタンスの削除
- 実行時ワークロードのクエリ
- 実行時統計のクエリ

WebLogic Integration Studio を使用して実行時ワークフロー インスタンスをモニタする方法の詳細については、『*WebLogic Integration Studio ユーザーズ ガイド*』の「ワークフローのモニタリング」 を参照してください。

ワークフロー インスタンスの取得

オーガニゼーション内のテンプレート インスタンスおよびテンプレート定義インスタンスのリストを取得するには、以下の com.bea.wlpi.server.admin.Admin メソッドを使用してください。

メソッド 1

```
public java.util.List getTemplateInstances(
  java.lang.String templateId,
  java.lang.String orgId,
  boolean bStarted,
  java.util.Date from,
  java.util.Date to,
  int state,
  int max
) throws java.rmi.RemoteException,
  com.bea.wlpi.common.WorkflowException
```

メソッド 2

```
public java.util.List getTemplateDefinitionInstances(
  java.lang.String templateDefinitionId,
  java.lang.String orgId,
  boolean bStarted,
  java.util.Date from,
  java.util.Date to,
  int state,
  int max
) throws java.rmi.RemoteException,
  com.bea.wlpi.common.WorkflowException
```

getTemplateInstances() メソッドおよび getTemplateDefinitionInstances() メソッドのパラメータを次の表に示します。パラメータには値を指定する必要があります。

表 22-1 getTemplateInstances() および getTemplateDefinitionInstances() メソッドのパラメータ

パラメータ	説明	有効な値
templateId	インスタンスを取得するテンプレートの ID	有効なテンプレート ID。 テンプレート ID を取得するには、13-4 ページの「オーガニゼーションのテンプレートの取得」で説明されているように、インスタンスを取得したいテンプレートに関連する TemplateInfo オブジェクトを取得する必要がある。テンプレート ID を取得するには、次の com.bea.wlpi.common.TemplateInfo メソッドを使用する。 `public final java.lang.String getId()` **注意：** TemplateInfo オブジェクトで選択可能なメソッドの詳細については、B-27 ページの「TemplateInfo オブジェクト」を参照してください。 getId() メソッドは、templateId を返す。各 templateId には、複数の定義が含まれる可能性がある。どのテンプレート定義を使用するか判断するために、WebLogic Integration はアクティブ テンプレート定義セットの中から、もっとも有効で適切な日付（現在または過去の日付）と終了日（現在または将来の日付）を持つバージョンを選択する。テンプレートの有効な日付と終了日の定義およびそれらをアクティブにする方法については、『*WebLogic Integration Studio ユーザーズ ガイド*』の「ワークフロー テンプレートの定義」を参照。

表 22-1 getTemplateInstances() および getTemplateDefinitionInstances() メソッドのパラメータ (続き)

パラメータ	説明	有効な値
templateDefinitionId	インスタンスを取得するテンプレート定義の ID	有効なテンプレート定義 ID。 テンプレート定義 ID を取得するには、14-2 ページの「テンプレート定義の作成」で説明されているように、インスタンスを取得するテンプレート定義に対応する TemplateDefinitionInfo オブジェクトを取得する必要がある。テンプレート定義 ID を取得するには、次の com.bea.wlpi.common.TemplateDefinitionInfo メソッドを使用する。 public final java.lang.String getId() **注意 :** TemplateDefinitionInfo オブジェクトで選択可能なメソッドの詳細については、B-24 ページの「TemplateDefinitionInfo オブジェクト」を参照してください。
orgId	テンプレートまたはテンプレート定義インスタンスを取得するためのオーガニゼーションの ID	有効なオーガニゼーション ID。 オーガニゼーション ID の取得については、19-1 ページの「アクティブなオーガニゼーションの管理」を参照。
bStarted	開始日付または終了日付のどちらでクエリを行うかを指定するブール フラグ	true (クエリにインスタンスの開始日付を使用) または false (クエリにインスタンスの終了日付を使用)。
from	日付範囲の下限値	java.util.Date オブジェクト
to	日付範囲の上限値	java.util.Date オブジェクト

表 22-1 getTemplateInstances() および getTemplateDefinitionInstances() メソッドのパラメータ（続き）

パラメータ	説明	有効な値
start	戻り値のリストを開始するオフセット。クライアントが長いリストを一定量づつ表示できる	整数値
max	クライアントが一度に表示する項目数を制限するための戻り値の最大項目数。	整数値

これらのメソッドは、それぞれテンプレート インスタンスおよびテンプレート定義インスタンスに関連した com.bea.wlpi.common.InstanceInfo オブジェクトのリストを返します。各タスクについての情報にアクセスするには、B-6 ページの「InstanceInfo オブジェクト」に記載の InstanceInfo オブジェクト メソッドを使用します。

たとえば、次のコードでは、activeOrgId 変数の値で指定されたオーガニゼーションのための特定の ID に関連する、すべてのテンプレートおよびテンプレート定義インスタンスがそれぞれ取得されます。このコード例では、admin は Admin EJB への EJBObject 参照を表します。

```
List tempinst = admin.getTemplateInstances(templateId,
   activeOrgId, true, dateFrom, dateTo, 20, 20);

List tempdefinst = admin.getTemplateDefinitionInstances(
   templatedefId, activeOrgId, true, dateFrom,
   dateTo, 20, 20);
```

この例では、以下のパラメータを設定します。

- 情報を開始日付でクエリするように bStarted を true に設定します。
- 日付範囲の下限および上限値は、java.util.Date オブジェクト変数の dateFrom および dateTo を使用して設定します。
- オフセット値は、20 に設定します。
- 一度に表示する値の最大数は、20 に設定します。

getTemplateInstances() および getTemplateDefinitionInstances() メソッドの詳細については、Javadoc の com.bea.wlpi.server.admin.Admin を参照してください。

ワークフロー インスタンスの確認

この節では、ワークフロー テンプレートまたはテンプレート定義インスタンスの確認を行うために使用するメソッドについて説明します。

ワークフロー テンプレート インスタンスの確認

ワークフロー テンプレート インスタンスが現在実行中かどうかを確認するには、次の com.bea.wlpi.server.admin.Admin メソッドを使用します。

```
public boolean checkForTemplateInstances(
  java.lang.String templateId
) throws java.rmi.RemoteException,
  com.bea.wlpi.common.WorkflowException
```

checkForTemplateInstances() メソッドのパラメータを次の表に示します。パラメータには値を指定する必要があります。

表 22-2 checkForTemplateInstances() メソッドのパラメータ

パラメータ	説明	有効な値
templateId	実行中のインスタンスを確認するテンプレートの ID	有効なテンプレート ID。 テンプレート ID を取得するには、13-4 ページの「オーガニゼーションのテンプレートの取得」で説明されているようにインスタンスを確認したいテンプレートに関連する TemplateInfo オブジェクトを取得する必要がある。テンプレート ID を取得するには、次の com.bea.wlpi.common.TemplateInfo メソッドを使用する。 public final java.lang.String getId() **注意：** TemplateInfo オブジェクトで選択可能なメソッドの詳細については、B-27 ページの「TemplateInfo オブジェクト」を参照してください。 getId() メソッドは、templateId を返す。各 templateId には、複数の定義が含まれる可能性がある。どのテンプレート定義を使用するか判断するために、WebLogic Integration はアクティブ テンプレート定義セットの中から、もっとも有効で適切な日付（現在または過去の日付）と終了日（現在または将来の日付）を持つバージョンを選択する。テンプレートの有効な日付と終了日の定義およびそれらをアクティブにする方法については、『*WebLogic Integration Studio ユーザーズ ガイド*』の「ワークフロー テンプレートの定義」を参照。

このメソッドは、テンプレート インスタンスが存在するかどうかを示すブール値を返します。

たとえば、次のコードでは、指定した ID の値に関連するテンプレート インスタンスが現在実行中かどうかが確認されます。このコード例では、admin は Admin EJB への EJBObject 参照を表します。

```
boolean tempexists = admin.checkForTemplateInstances(templateId);
```

checkForTemplateInstances() メソッドの詳細については、Javadoc の com.bea.wlpi.server.admin.Admin を参照してください。

ワークフロー テンプレート定義の確認

ワークフロー テンプレート定義インスタンスを取得するには、次の com.bea.wlpi.server.admin.Admin メソッドを使用します。

```
public java.util.List checkForTemplateDefinitionInstances(
  java.lang.String templateDefinitionId
) throws java.rmi.RemoteException,
  com.bea.wlpi.common.WorkflowException
```

checkForTemplateDefinitionInstances() メソッドのパラメータを次の表に示します。パラメータには値を指定する必要があります。

表 22-3 checkForTemplateInstances() メソッドのパラメータ

パラメータ	説明	有効な値
templateDefinitionId	インスタンスを確認するテンプレート定義の ID	有効なテンプレート定義 ID。 テンプレート定義 ID を取得するには、14-2 ページの「テンプレート定義の作成」で説明されているように、インスタンスを取得するテンプレート定義に対応する TemplateDefinitionInfo オブジェクトを取得する必要がある。テンプレート定義 ID を取得するには、次の com.bea.wlpi.common.TemplateDefinitionInfo メソッドを使用する。 public final java.lang.String getId() **注意：** TemplateDefinitionInfo オブジェクトで選択可能なメソッドの詳細については、B-24 ページの「TemplateDefinitionInfo オブジェクト」を参照してください。

このメソッドは、テンプレートまたはテンプレート定義インスタンスが存在するかどうかを示すブール値を返します。

たとえば、次のコードでは、指定した ID の値に関連するテンプレート定義インスタンスが存在するかどうかが確認されます。このコード例では、admin は Admin EJB への EJBObject 参照を表します。

```
boolean tempexists = admin.checkForTemplateInstances(templateId);

boolean tempdefexists = admin.checkForTemplateDefinitionInstances(
   templateDefinitionId);
```

checkForTemplateDefinitionInstances() メソッドの詳細については、Javadoc の com.bea.wlpi.server.admin.Admin を参照してください。

ワークフロー インスタンス タスクの取得

ワークフロー インスタンスに関連したタスクのリストを取得するには、次の com.bea.wlpi.server.admin.Admin メソッドを使用してください。

```
public java.util.List getInstanceTasks(
  java.lang.String instanceId
) throws java.rmi.RemoteException,
  com.bea.wlpi.common.WorkflowException
```

getInstanceTasks() メソッドのパラメータを次の表に示します。パラメータには値を指定する必要があります。

表 22-4 getInstanceTasks() メソッドのパラメータ

パラメータ	説明	有効な値
instanceId	タスクを取得するワークフロー インスタンスの ID	有効なワークフロー インスタンス ID。 ワークフロー インスタンス ID を取得するには、22-2 ページの「ワークフロー インスタンスの取得」で説明するように、タスクを取得するインスタンスに対応する InstanceInfo オブジェクトを取得する必要がある。インスタンス ID を取得するには、次の com.bea.wlpi.common.InstanceInfo メソッドを使用する。 public final java.lang.String getId() **注意：** InstanceInfo オブジェクトで利用可能なメソッドの詳細については、B-6 ページの「InstanceInfo オブジェクト」を参照してください。

このメソッドは、com.bea.wlpi.common.TaskInfo オブジェクトのリストを返します。各タスクについての情報にアクセスするには、B-21 ページの「TaskInfo オブジェクト」に記載の TaskInfo オブジェクト メソッドを使用します。

たとえば、次のコードでは、instanceId 変数の値で指定されたワークフロー インスタンスのタスクが取得されます。このコード例では、admin は、Admin EJB への EJBObject 参照を表します。

```
List tasks = admin.getInstanceTasks(instanceId);
```

getInstanceTasks() メソッドの詳細については、Javadoc の com.bea.wlpi.server.admin.Admin を参照してください。

ワークフロー インスタンス情報の取得

ワークフロー インスタンス情報を取得するには、次の com.bea.wlpi.server.admin.Admin メソッドを使用します。

```
public com.bea.wlpi.common.InstanceInfo getInstanceInfo(
  java.lang.String instanceId
) throws java.rmi.RemoteException,
  com.bea.wlpi.common.WorkflowException
```

getInstanceInfo() メソッドのパラメータを次の表に示します。パラメータには値を指定する必要があります。

表 22-5 getInstanceTasks() メソッドのパラメータ

パラメータ	説明	有効な値
instanceId	情報を取得するワークフロー インスタンスの ID	有効なワークフロー インスタンス ID。 ワークフロー インスタンス ID を取得するには、15-1 ページの「タスクの取得」で説明するように、ワークフロー インスタンス情報を取得するタスクに対応する TaskInfo オブジェクトを取得する必要がある。タスク ID を取得するには、次の com.bea.wlpi.common.TaskInfo メソッドを使用する。 public final java.lang.String getInstanceId() **注意：** TaskInfo オブジェクトで選択可能なメソッドの詳細については、B-21 ページの「TaskInfo オブジェクト」を参照してください。

このメソッドは、com.bea.wlpi.common.InstanceInfo オブジェクトを返します。ワークフロー インスタンスについての情報にアクセスするには、B-6 ページの「InstanceInfo オブジェクト」に記載の InstanceInfo オブジェクト メソッドを使用します。

たとえば、次のコードでは、instanceId 変数の値で指定されたワークフロー インスタンスの情報が取得されます。このコード例では、admin は、Admin EJB への EJBObject 参照を表します。

```
InstanceInfo = admin.getInstanceInfo(instanceId);
```

getInstanceInfo() メソッドの詳細については、Javadoc の com.bea.wlpi.server.admin.Admin を参照してください。

ワークフロー インスタンスのアカウントの取得

ワークフロー インスタンスの数や完了したワークフロー インスタンスの数を取得するには、それぞれ、以下の com.bea.wlpi.server.admin.Admin メソッドを使用してください。

メソッド 1

```
public int getInstanceCount(
  java.lang.String templateId
) throws java.rmi.RemoteException,
  com.bea.wlpi.common.WorkflowException
```

メソッド 2

```
public int getInstanceCount(
  java.lang.String templateI,
  boolean completed
) throws java.rmi.RemoteException,
  com.bea.wlpi.common.WorkflowException
```

getInstanceCount() メソッドのパラメータを次の表に示します。パラメータには値を指定する必要があります。

表 22-6 getInstanceCount() メソッドのパラメータ

パラメータ	説明	有効な値
templateId	インスタンスを取得するテンプレートの ID	有効なテンプレート ID。 テンプレート ID を取得するには、13-4 ページの「オーガニゼーションのテンプレートの取得」で説明されているように、インスタンスを取得したいテンプレートに関連する TemplateInfo オブジェクトを取得する必要がある。テンプレート ID を取得するには、次の com.bea.wlpi.common.TemplateInfo メソッドを使用する。 `public final java.lang.String getId()` **注意：** TemplateInfo オブジェクトで選択可能なメソッドの詳細については、B-27 ページの「TemplateInfo オブジェクト」を参照してください。 getId() メソッドは、templateId を返す。各 templateId には、複数の定義が含まれる可能性がある。どのテンプレート定義を使用するか判断するために、WebLogic Integration はアクティブ テンプレート定義セットの中から、もっとも有効で適切な日付（現在または過去の日付）と終了日（現在または将来の日付）を持つバージョンを選択する。テンプレートの有効な日付と終了日の定義およびそれらをアクティブにする方法については、『*WebLogic Integration Studio ユーザーズ ガイド*』の「ワークフロー テンプレートの定義」を参照してください。

表 22-6 getInstanceCount() メソッドのパラメータ (続き)

パラメータ	説明	有効な値
completed	完了したワークフロー インスタンスだけをカウントしたかどうかを指定するブール フラグ	true (完了したワークフロー インスタンスのみをカウント) または false (すべてのワークフロー インスタンスをカウント)

各メソッドは、指定された基準に基づいてカウントしたワークフロー インスタンスの数に対応する整数値を戻します。

たとえば、次のコードは、すべてのワークフロー インスタンスの数と完了したすべてのワークフロー インスタンスの数を、指定された ID 値に対応してそれぞれ取得します。このコード例では、admin は Admin EJB への EJBObject 参照を表します。

```
int count = admin.getInstanceCount(templateId);
int completeCount = admin.getInstanceCount(templateId, true);
```

getInstanceCount() メソッドの詳細については、Javadoc の com.bea.wlpi.server.admin.Admin を参照してください。

ワークフロー インスタンスの削除

この節で説明しているメソッドを使用して、ワークフロー インスタンスを削除できます。

- 特定のワークフロー インスタンス
- オーガニゼーションのすべてのワークフロー インスタンス

特定のワークフロー インスタンスの削除

特定のワークフロー インスタンスを削除するには、次の com.bea.wlpi.server.admin.Admin メソッドを使用します。

```
public void deleteInstance(
  java.lang.String instanceId
) throws java.rmi.RemoteException,
  com.bea.wlpi.common.WorkflowException
```

deleteInstance() メソッドのパラメータを次の表に示します。パラメータには値を指定する必要があります。

表 22-7 deleteInstance() メソッドのパラメータ

パラメータ	説明	有効な値
instanceId	タスクを取得するワークフロー インスタンスの ID	有効なワークフロー インスタンス ID。 ワークフロー インスタンス ID を取得するには、22-2 ページの「ワークフロー インスタンスの取得」で説明するように、タスクを取得するインスタンスに対応する InstanceInfo オブジェクトを取得する必要がある。インスタンス ID を取得するには、次の com.bea.wlpi.common.InstanceInfo メソッドを使用する。 public final java.lang.String getId() **注意：** InstanceInfo オブジェクトで利用可能なメソッドの詳細については、B-6 ページの「InstanceInfo オブジェクト」を参照してください。

たとえば、次のコードでは、instanceId で指定されたインスタンスが削除されます。このコード例では、admin は Admin EJB への EJBObject 参照を表します。

```
admin.deleteInstance(instanceId);
```

ワークフロー テンプレートまたはテンプレート定義のすべてのインスタンスの削除

オーガニゼーション内のテンプレート インスタンスおよびテンプレート定義のインスタンスを削除するには、次の com.bea.wlpi.server.admin.Admin メソッドを使用してください。

メソッド 1

```
public void deleteTemplateInstances(
  java.lang.String templateId,
  java.lang.String orgId,
  boolean bStarted,
```

```
            java.util.Date from,
            java.util.Date to
          ) throws java.rmi.RemoteException,
            com.bea.wlpi.common.WorkflowException
```

メソッド 2

```
public void deleteTemplateDefinitionInstances(
  java.lang.String templateDefinitionId,
  java.lang.String orgId,
  boolean bStarted,
  java.util.Date from,
  java.util.Date to
) throws java.rmi.RemoteException,
  com.bea.wlpi.common.WorkflowException
```

deleteTemplateInstances() および deleteTemplateDefinitionInstances() メソッドのパラメータを次の表に示します。パラメータには値を指定する必要があります。

表 22-8 deleteTemplateInstances() および deleteTemplateDefinitionInstances() メソッドのパラメータ

パラメータ	説明	有効な値
templateId	インスタンスを削除するテンプレートの ID	有効なテンプレート ID。 テンプレート ID を取得するには、13-4 ページの「オーガニゼーションのテンプレートの取得」で説明されているようにインスタンスを削除したいテンプレートに関連する TemplateInfo オブジェクトを取得する必要がある。テンプレート ID を取得するには、次の com.bea.wlpi.common.TemplateInfo メソッドを使用する。 `public final java.lang.String getId()` **注意：** TemplateInfo オブジェクトで選択可能なメソッドの詳細については、B-27 ページの「TemplateInfo オブジェクト」を参照してください。 getId() メソッドは、templateId を返す。各 templateId には、複数の定義が含まれる可能性がある。どのテンプレート定義を使用するか判断するために、WebLogic Integration はアクティブ テンプレート定義セットの中から、もっとも有効で適切な日付（現在または過去の日付）と終了日（現在または将来の日付）を持つバージョンを選択する。テンプレートの有効な日付と終了日の定義およびそれらをアクティブにする方法については、『*WebLogic Integration Studio ユーザーズ ガイド*』の「ワークフロー テンプレートの定義」を参照してください。

表 22-8 deleteTemplateInstances() および deleteTemplateDefinitionInstances() メソッドのパラメータ (続き)

パラメータ	説明	有効な値
templateDefinitionId	インスタンスを削除するテンプレート定義の ID	有効なテンプレート定義 ID。 テンプレート定義 ID を取得するには、14-2 ページの「テンプレート定義の作成」で説明されているように、インスタンスを削除するテンプレート定義に対応する TemplateDefinitionInfo オブジェクトを取得する必要がある。テンプレート定義 ID を取得するには、次の com.bea.wlpi.common.TemplateDefinitionInfo メソッドを使用する。 `public final java.lang.String getId()` **注意 :** TemplateDefinitionInfo オブジェクトで選択可能なメソッドの詳細については、B-24 ページの「TemplateDefinitionInfo オブジェクト」を参照してください。
orgId	テンプレート インスタンスまたはテンプレート定義インスタンスを削除するためのオーガニゼーションの ID	有効なオーガニゼーション ID。 オーガニゼーション ID の取得については、19-1 ページの「アクティブなオーガニゼーションの管理」を参照。
bStarted	開始または終了のどちらで削除を行うかを指定するブール フラグ	true (削除にインスタンスの開始日付を使用) または false (削除にインスタンスの終了日付を使用)
from	日付範囲の下限値	java.util.Date オブジェクト
to	日付範囲の上限値	java.util.Date オブジェクト

たとえば、次のコードでは、activeOrgId 変数の値で指定されたオーガニゼーションのための特定の ID に関連する、すべてのテンプレート インスタンスまたはテンプレート定義インスタンスを削除します。この例では、以下のように設定します。

- 情報を開始日付でクエリするように bStarted を true に設定します。
- 日付範囲の下限および上限値は、java.util.Date オブジェクト変数の dateFrom および dateTo を使用して設定します。

このコード例では、admin は Admin EJB への EJBObject 参照を表します。

```
admin.deleteTemplateInstances(templateId, activeOrgId, true,
  dateFrom, dateTo);

admin.deleteTemplateDefinitionInstances(
   templatedefId, activeOrgId, true, dateFrom, dateTo);
```

deleteInstance()、deleteTemplateInstances() および deleteTemplateDefinitionInstances() メソッドの詳細については、Javadoc の com.bea.wlpi.server.admin.Admin を参照してください。

実行時ワークロードのクエリ

実行時のワークロードをクエリするために、次の com.bea.wlpi.server.admin.Admin メソッドを使用します。

```
public java.util.List workloadQuery(
  java.lang.String xml
) throws java.rmi.RemoteException,
  com.bea.wlpi.common.WorkflowException
```

workloadQuery() メソッドのパラメータを次の表に示します。パラメータには値を指定する必要があります。

表 22-9 workloadQuery() メソッドのパラメータ

パラメータ	説明	有効な値
xml	実行時ワークロードのクエリ要求	A-121 ページの「ワークロード要求 DTD」に記載の Workload Request DTD に準拠する XML ドキュメント。

このメソッドは、A-124 ページの「ワークロード応答 DTD」で説明されているように、Workload Response DTD に準拠する XML フォーマットでワークロードレポートを返します。

たとえば、次のコードでは、Workload Request DTD に準拠する `workloadReq` XML ドキュメントに基づいて実行時ワークロードに対するクエリが開始されます。

```
String workloadresp = workloadQuery(workloadReq);
```

このメソッドは、Workload Response DTD に準拠するフォーマットで `workloadresp` ファイルにその応答を書き込みます。`workloadQuery()` メソッドの詳細については、Javadoc の `com.bea.wlpi.server.admin.Admin` を参照してください。

実行時統計のクエリ

実行時統計をクエリするために、次の `com.bea.wlpi.server.admin.Admin` メソッドを使用します。

```
public java.util.List statisticsQuery(
  java.lang.String xml
) throws java.rmi.RemoteException,
  com.bea.wlpi.common.WorkflowException
```

`statisticsQuery()` メソッドのパラメータを次の表に示します。パラメータには値を指定する必要があります。

表 22-10 statisticsQuery() メソッドのパラメータ

パラメータ	説明	有効な値
`xml`	実行時統計のクエリ要求	A-44 ページの「統計要求 DTD」に記載の Workload Request DTD に準拠する XML ドキュメント。

このメソッドは、A-47 ページの「統計応答 DTD」で説明されているように、Statistics Response DTD に準拠する XML フォーマットで統計レポートを返します。

たとえば、次のコードでは、Statistics Request DTD に準拠する statisticsReq XML ドキュメントに基づいて実行時統計に対するクエリが開始されます。

```
java.lang.String statisticsresp = statisticsQuery(statisticsReq);
```

このメソッドは、Statistics Response DTD に準拠するフォーマットで statisticsresp ファイルにその応答を書き込みます。statisticsQuery() メソッドの詳細については、Javadoc の com.bea.wlpi.server.admin.Admin を参照してください。

23　実行時の変数のモニタリング

この章では、実行時の変数をモニタする方法について説明します。内容は以下のとおりです。

- ワークフロー インスタンスの変数の取得
- ワークフロー インスタンスの変数の設定

WebLogic Integration Studio を使用して実行時の変数をモニタする方法の詳細については、『*WebLogic Integration Studio ユーザーズ ガイド*』の「ワークフローのモニタリング」を参照してください。

ワークフロー インスタンスの変数の取得

ワークフロー インスタンスに関連した変数のリストを取得するには、次の `com.bea.wlpi.server.admin.Admin` メソッドを使用します。

```
public java.util.List getInstanceVariables(
  java.lang.String instanceId
) throws java.rmi.RemoteException,
  com.bea.wlpi.common.WorkflowException
```

getInstanceVariables() メソッドのパラメータを次の表に示します。パラメータには値を指定する必要があります。

表 23-1 getInstanceVariables() メソッドのパラメータ

パラメータ	説明	有効な値
instanceId	変数を取得するワークフロー インスタンスの ID	有効なワークフロー インスタンス ID。 ワークフロー インスタンス ID を取得するには、22-2 ページの「ワークフロー インスタンスの取得」で説明するように、タスクを取得するインスタンスに対応する InstanceInfo オブジェクトを取得する必要がある。インスタンス ID を取得するには、次の com.bea.wlpi.common.InstanceInfo メソッドを使用する。 `public final java.lang.String getId()` **注意：** InstanceInfo オブジェクトで利用可能なメソッドの詳細については、B-6 ページの「InstanceInfo オブジェクト」を参照してください。

このメソッドは、com.bea.wlpi.common.VariableInfo オブジェクトのリストを返します。各変数に関する情報にアクセスするには、B-30 ページの「VariableInfo オブジェクト」で説明する VariableInfo オブジェクトを使用します。

注意： XML の型の変数の場合、返される値は byte[ ] 型になります。戻り値をプリントまたは表示するには、それを String の値にコンバートする必要があります。たとえば、次のコードでは、byte[ ] 型の variable の値が String の値にコンバートされます。

```
String value=new String((byte[ ])variable.getValue())
```

たとえば、次のコードでは、指定されたインスタンス ID に対応するワークフロー インスタンスに対する変数が取得されます。このコード例では、admin は Admin EJB への EJBObject 参照を表します。

```
List list = admin.getInstanceVariables(instance.getId());
```

インスタンス ID は、com.bea.wlpi.common.InstanceInfo オブジェクトの instance に関連した getInstanceId() メソッドを使用して取得されます。instance オブジェクトは、22-2 ページの 「ワークフロー インスタンスの取得」に記載のメソッドを使用して取得されます。

getInstanceVariables() メソッドの詳細については、Javadoc の com.bea.wlpi.server.admin.Admin を参照してください。

ワークフロー インスタンスの変数の設定

ワークフロー インスタンスに関連する変数を設定するには、次の com.bea.wlpi.server.admin.Admin メソッドのいずれかを使用します。

```
public void setInstanceVariable(
  java.lang.String templateDefinitionId,
  java.lang.String instanceId,
  java.lang.String variable,
  java.lang.Object value
) throws java.rmi.RemoteException,
  com.bea.wlpi.common.WorkflowException

public void setInstanceVariables(
  java.lang.String templateDefinitionId,
  java.lang.String instanceId,
  java.lang.String variables
) throws java.rmi.RemoteException,
  com.bea.wlpi.common.WorkflowException
```

`setInstanceVariable()` メソッドのパラメータを次の表に示します。パラメータには値を指定する必要があります。

表 23-2 setInstanceVariable() メソッドのパラメータ

パラメータ	説明	説明
`templateDefinitionId`	ユーザがワークフロー インスタンスの変数を設定するテンプレート定義の ID	有効なテンプレート定義 ID。 テンプレート定義のインスタンス ID を取得するには、22-2 ページの「ワークフロー インスタンスの取得」で説明するように、インスタンス変数を設定するインスタンスに対応する `InstanceInfo` オブジェクトを取得する必要がある。インスタンス ID を取得するには、次の `com.bea.wlpi.common.InstanceInfo` メソッドを使用する。 `public final java.lang.String getTemplateDefinitionId()` **注意：** `InstanceInfo` オブジェクトで利用可能なメソッドの詳細については、B-6 ページの「InstanceInfo オブジェクト」を参照してください。

表 23-2 setInstanceVariable() メソッドのパラメータ (続き)

パラメータ	説明	説明
instanceId	タスクと対応するワークフロー インスタンスの ID	有効なワークフロー インスタンス ID。 テンプレート定義のインスタンス ID を取得するには、22-2 ページの「ワークフロー インスタンスの取得」で説明するように、インスタンス変数を設定するインスタンスに対応する InstanceInfo オブジェクトを取得する必要がある。インスタンス ID を取得するには、次の com.bea.wlpi.common.InstanceInfo メソッドを使用する。 `public final java.lang.String getInstanceId()` **注意 :** InstanceInfo オブジェクトで利用可能なメソッドの詳細については、B-6 ページの「InstanceInfo オブジェクト」を参照してください。

表 23-2 setInstanceVariable() メソッドのパラメータ (続き)

パラメータ	説明	説明
variable	ユーザが設定する変数の名前	有効な変数名。 変数名を取得するには、次の com.bea.wlpi.common.VariableInfo メソッドを使用する。 public final String getName() VariableInfo オブジェクトの取得に関する詳細については、23-1 ページの「ワークフロー インスタンスの変数の取得」を参照。VariableInfo オブジェクトで利用可能なメソッドの詳細については、B-30 ページの「VariableInfo オブジェクト」を参照。
variables	変数名と値	キーとして設定される変数、および値として必要な値を指定する「*key-value*」ペアを持つマップ オブジェクト。
value	変数用に該当する値	指定された変数に対して有効な値。 value パラメータ用にサポートされているコンバートのリストについては、Javadoc の com.bea.wlpi.server.admin.Admin を参照してください。

たとえば、次のコードでは、指定された変数が指定された value に設定され、更新されます。このコード例では、admin は Admin EJB への EJBObject 参照を表します。

```
admin.setInstanceVariable(
   instance.getTemplateDefinitionId(),
   instance.getInstanceId(),
   variable.getName(),
   value
);
```

変数名は、com.bea.wlpi.common.VariableInfo オブジェクトの variable に関連した getName() メソッドを使用して取得されます。変数のオブジェクトは、23-1 ページの「ワークフロー インスタンスの変数の取得」に記載のメソッドを使用して取得されます。テンプレート定義とワークフロー インスタンス ID は、com.bea.wlpi.common.InstanceInfo オブジェクトである instance と関連付けられたメソッドを使用して取得されます。instance オブジェクトは、22-2 ページの「ワークフロー インスタンスの取得」に記載のメソッドを使用して取得されます。

com.bea.wlpi.common.VariableInfo と com.bea.wlpi.common.TaskInfo メソッドの詳細については、B-1 ページの「値オブジェクトのまとめ」を参照してください。

setInstanceVariable() および setInstanceVariables() メソッドの詳細については、Javadoc の com.bea.wlpi.server.admin.Admin を参照してください。

24 ワークフロー例外のモニタリング

この章では、ワークフロー例外をモニタする方法について説明します。内容は以下のとおりです。

- 例外処理の概要
- ワークフロー例外の作成
- ワークフロー例外情報の取得
- ワークフロー例外ハンドラの呼び出し

WebLogic Integration Studio を使用したワークフロー例外のモニタリングについては、『*WebLogic Integration Studio ユーザーズ ガイド*』の「ワークフロー例外の処理」を参照してください。

例外処理の概要

BEA WebLogic Integration の例外処理機能により、実行時における例外条件に対し、生成、トラップ、および対処できます。例外処理は、タスク レベルではなく、ワークフロー レベルで行われます。

以下の節では、ワークフロー例外と例外ハンドラについて説明します。例外処理の詳細については、『*WebLogic Integration Studio ユーザーズ ガイド*』の「ワークフローのモニタリング」を参照してください。

ワークフロー例外

*ワークフロー例外*とは、ユーザがワークフロー内で定義する異常な条件、またはユーザが適宜にトラップと処理を行う実行時の代表的なサーバ例外のことです。ワークフロー例外は、それぞれが適切な重大度レベルに対応する多数の条件の 1 つに対して送出されます。

ワークフロー例外に対して定義された重大度の各レベルを、その重大度コードの値を含め次の表に示します。

表 24-1 ワークフロー例外の重大度レベル

重大度レベル	重大度コード	説明
システム エラー	ERROR_SYSTEM (1)	ユーザ要求を処理している時に致命的な例外が発生した。
ワークフロー エラー	ERROR_WORKFLOW (2)	ワークフロー ステータスの不一致など、致命的で不正な条件が発生した。
ワークフローの警告	WARNING_WORKFLOW (3)	致命的でないワークフロー条件が発生したが、これはユーザが手動で修正可能である。
不明なエラー	ERROR_UNKNOWN (0)	システム固有のエラーが発生した。この情報は内部使用でのみ利用可能である。
カスタム エラー	ERROR_CUSTOM (4)	com.bea.wlpi.server.worklist.Worklist.invokeWorkflowExceptionHandler() メソッドを呼び出すアプリケーション、または Studio 内の Invoke Error Handler アクションにより、カスタム エラーが発生した。

com.bea.wlpi.common.WorkflowException オブジェクトは、EJB のリモート呼び出しの実行時に発生したワークフロー例外をカプセル化します。

例外は wlpiError の JMS トピックに送られます。JMS への接続の詳細については、6-1 ページの「JMS コネクタの確立」を参照してください。

ワークフロー例外ハンドラ

ワークフローのすべてのテンプレート定義では、システム例外ハンドラという少なくとも 1 つの*例外ハンドラ*が指定されます。システム例外ハンドラは、デフォルトで設定される初期例外ハンドラで、例外が発生した時に常に呼び出されます。カスタム例外ハンドラを定義することもできます。ワークフロー テンプレート定義用のアクティブ例外ハンドラは、ワークフローの実行過程で繰り返し変更できます。

ワークフロー例外の作成

新しい WorkflowException オブジェクトを作成するコンストラクタを次の表にリストします。

表 24-2 WorkflowException オブジェクトのコンストラクタ

コンストラクタ	作成オブジェクト
`public WorkflowException (java.lang.Exception e)`	ネストされた例外を含むワークフロー例外。指定された例外には、ERROR_SYSTEM の重大度レベルが割り当てられる。ただし、新たな例外がその重大度をネストされた例外から継承する WorkflowException クラスに属している場合を除く。
`public WorkflowException (java.lang.Exception e, int severity)`	指定された重大度レベルのネストされた例外を含むワークフロー例外。 有効な重大度レベルのリストについては、24-2 ページの「ワークフロー例外の重大度レベル」の表を参照。
`public WorkflowException( java.lang.Exception e, int msgNum, int severity)`	指定されたメッセージ コードおよび重大度レベルのネストされた例外を含むワークフロー例外。 有効なメッセージ コードのリストについては、Javadoc の com.bea.wlpi.common.Messages を参照。有効な重大度レベルのリストについては、24-2 ページの「ワークフロー例外の重大度レベル」の表を参照。

表 24-2 WorkflowException オブジェクトのコンストラクタ

コンストラクタ	作成オブジェクト
`public WorkflowException(java.lang.Exception e, int msgNum, java.lang.Object[ ] args)`	指定されたメッセージ コードと、ローカライズしたメッセージ文字列に置き換える引数のリストを持つ、ネストされた例外を含むワークフロー例外。 有効なメッセージ コードのリストについては、Javadoc の `com.bea.wlpi.common.Messages` を参照。
`public WorkflowException(java.lang.Exception e, int msgNum, java.lang.Object[ ] args, int severity)`	指定されたメッセージ番号、ローカライズしたメッセージ文字列に置き換える引数のリスト、およびメッセージの重大度レベルを持つ、ネストされた例外を含むワークフロー例外。 有効なメッセージ コードのリストについては、Javadoc の `com.bea.wlpi.common.Messages` を参照。有効な重大度レベルのリストについては、24-2 ページの「ワークフロー例外の重大度レベル」 の表を参照。
`public WorkflowException(int msgNum)`	指定されたエラー コードを持つワークフロー例外。指定された例外には、デフォルトで、`ERROR_SYSTEM` の重大度レベルが割り当てられる。 有効なメッセージ コードのリストについては、Javadoc の `com.bea.wlpi.common.Messages` を参照。

表 24-2 WorkflowException オブジェクトのコンストラクタ

コンストラクタ	作成オブジェクト
`public WorkflowException(int msgNum, java.lang.Object[ ] args)`	指定されたエラー コードと、ローカライズしたメッセージ文字列に置き換える引数のリストを持つワークフロー例外。指定された例外には、デフォルトで、ERROR_SYSTEM の重大度レベルが割り当てられる。 有効なメッセージ コードのリストについては、Javadoc の com.bea.wlpi.common.Messages を参照。
`public WorkflowException(int msgNum, int severity)`	指定されたエラー コードと重大度レベルを持つワークフロー例外。 有効なメッセージ コードのリストについては、Javadoc の com.bea.wlpi.common.Messages を参照。有効な重大度レベルのリストについては、24-2 ページの「ワークフロー例外の重大度レベル」 の表を参照。
`public WorkflowException(int msgNum, java.lang.Object[ ] args, int severity)`	指定されたエラー コード、ローカライズしたメッセージ文字列に置き換える引数のリスト、およびメッセージの重大度レベルを持つワークフロー例外。 有効なメッセージ コードのリストについては、Javadoc の com.bea.wlpi.common.Messages を参照。有効な重大度レベルのリストについては、24-2 ページの「ワークフロー例外の重大度レベル」 の表を参照。

表 24-2 WorkflowException オブジェクトのコンストラクタ

コンストラクタ	作成オブジェクト
`public WorkflowException(int msgNum, java.lang.Object[ ] args, int severity, java.lang.String origin)`	指定されたエラー コード、ローカライズしたメッセージ文字列に置き換える引数のリスト、メッセージの重大度レベル、およびメッセージの発生源を持つワークフロー例外。 有効なメッセージ コードのリストについては、Javadoc の `com.bea.wlpi.common.Messages` を参照。有効な重大度レベルのリストについては、24-2 ページの「ワークフロー例外の重大度レベル」 の表を参照。

ワークフロー例外情報の取得

以下の節では、ワークフロー例外（重大度レベル、メッセージ テキスト、メッセージ番号、メッセージの発生源、ワークフロー例外がデッドロックに起因するかどうか、など）の情報を収集する方法と、スタック トレースのプリント方法を説明します。

ワークフロー例外の取得

オリジナルまたはネストされたワークフロー例外を取得するには、以下のそれぞれの `com.bea.wlpi.common.WorkflowException` メソッドを使用します。

```
public java.lang.Exception getOriginalException()
```

```
public java.lang.Exception getNestedException()
```

第 1 のメソッドでは、オリジナルのワークフロー例外が返されます。第 2 のメソッドでは、ネストされたワークフロー例外が返されますが、ネストされたワークフロー例外が存在しない場合は null が返されます。

たとえば、次のコードでは、所定のワークフロー例外に対してネストされたワークフロー例外が取得されます。

```
try {
   // some methods;
}
catch(WorkflowException e) {
   Exception nested e.getNestedException();
}
```

getOriginalException() と getNestedException() メソッドの詳細については、Javadoc の com.bea.wlpi.common.WorkflowException を参照してください。

重大度レベルの情報の取得

ワークフロー例外のメッセージの重大度レベルを取得するには、以下のいずれかの com.bea.wlpi.common.WorkflowException メソッドを使用します。

メソッド 1 `public java.lang.String getSeverity()`

メソッド 2 `public java.lang.String getSeverityDescription()`

メソッド 3 `public java.lang.String getLocalizedSeverityDescription()`

第 1 のメソッドでは、メッセージの重大度コードが整数値で返されます。第 2 のメソッドでは、メッセージの重大度の値が文字列の値で返されます。第 3 のメソッドでは、ローカライズされたメッセージの重大度コードが文字列の値で返されます。重大度レベルの値とコードについては、24-2 ページの「ワークフロー例外の重大度レベル」 の表を参照してください。

たとえば、次のコードでは、所定のワークフロー例外に対するメッセージの重大度コードを取得します。

```
try {
   // some methods;
}
catch(WorkflowException e) {
   String severity e.getSeverity();
}
```

getSeverity() メソッドの詳細については、Javadoc の com.bea.wlpi.common.WorkflowException を参照してください。

メッセージ テキストの取得

ワークフロー例外に関連したメッセージ テキストを取得するには、以下のいずれかの com.bea.wlpi.common.WorkflowException メソッドを使用します。

メソッド 1 `public java.lang.String getMessage()`

メソッド 2 `public java.lang.String getLocalizedMessage()`

第 1 のメソッドでは、ネストされたメッセージに関連したメッセージ テキスト (可能な場合)、例外を送出するシステム ロケールの言語によるメッセージ テキスト、およびメッセージ テキストが作成されなかった場合は null のいずれか 1 つが返されます。

第 2 のメソッドでは、ローカライズされたメッセージ テキスト (可能な場合)、例外を送出するシステム ロケールの言語によるメッセージ テキスト、およびメッセージ テキストが作成されなかった場合は null のいずれか 1 つが返されます。

たとえば、次のコードでは、所定のワークフロー例外に対するメッセージ テキストを取得します。

```
try {
   // some methods;
}
catch(WorkflowException e) {
   String message e.getMessage();
}
```

`getMessage()` または `getLocalizedMessage()` メソッドの詳細については、Javadoc の `com.bea.wlpi.common.WorkflowException` を参照してください。

メッセージ番号の取得

ワークフロー例外の場合のメッセージ番号を取得するには、次の com.bea.wlpi.common.WorkflowException メソッドを使用します。

`public java.lang.String getMessageNumber()`

たとえば、次のコードでは、所定のワークフロー例外に対するメッセージ テキストを取得します。

```
try {
   // some methods;
}
catch(WorkflowException e) {
   String number e.getMessageNumber();
}
```

getMessageNumber() メソッドの詳細については、Javadoc の com.bea.wlpi.common.WorkflowException を参照してください。

発生源の名前の取得

ワークフロー例外が発生した発生源（ワークフロー コンポーネント）の名前を取得するには、次の com.bea.wlpi.common.WorkflowException メソッドを使用します。

```
public java.lang.String getOrigin()
```

たとえば、次のコードでは、所定のワークフロー例外に対する発生源のテキストを取得します。

```
try {
   // some methods;
}
catch(WorkflowException e) {
   String origin e.getOrigin();
}
```

getOrigin() メソッドの詳細については、Javadoc の com.bea.wlpi.common.WorkflowException を参照してください。

ワークフロー例外がデータベースのデッドロックに起因するかの確認

ワークフロー例外がデッドロックに起因するかどうか確認するには、以下のいずれかの com.bea.wlpi.common.WorkflowException メソッドを使用します。

メソッド 1 `public boolean isDeadlock()`

メソッド 2 `public static boolean isDeadlock(java.lang.Exception e)`

各メソッドは、ワークフロー例外がデータベースのデッドロックに起因する場合は true を返し、それ以外の場合は false を返します。

第 2 のメソッドでは、オリジナルの例外を取り出すために、`getOrginalException()` メソッドが呼び出されます。オリジナルの例外が `java.sql.SQLException` のインスタンスの場合、このメソッドはオリジナルの問題がデータベースのデッドロックに起因するかどうかを判断するために、`getSQLState()` メソッドが呼び出されます。

たとえば、次のコードでは、メソッドが所定のワークフロー例外に対してデッドロックするかどうか判断されます。

```
try {
   // some methods;
}
catch(WorkflowException e) {
   String boolean e.isDeadlock();
}
```

`isDeadlock()` メソッドの詳細については、Javadoc の `com.bea.wlpi.common.WorkflowException` を参照してください。

スタック トレースのプリント

スタック トレースをプリントするには、以下のいずれかの `com.bea.wlpi.common.WorkflowException` メソッドを使用します。

メソッド 1 `public void printStackTrace()`

メソッド 2 `public void printStackTrace(java.io.PrintStream stream)`

メソッド 3 `public void printStackTrace(java.io.PrintWriter writer)`

各メソッドは、ワークフロー例外とそのバック トレースをプリントします。第 1 のメソッドでは、情報が標準エラー ストリーム（`System.err`）にプリントされ、第 2 のメソッドでは、指定されたプリント ストリームにプリントされ、第 3 のメソッドでは、指定されたプリント ライタにプリントされます。

出力の一行目には、このオブジェクトに対する `toString()` メソッドの呼び出しの結果が含まれます。残りの内容は、前回 `fillInStackTrace()` メソッドで記録されたデータを表します。ワークフロー例外にネストされたワークフロー例外が含まれる場合、このメソッドはそのスタック トレースもプリントされます。

たとえば、次のコードでは、所定のワークフロー例外に対するスタック トレースがプリントされます。

```
try {
   // some methods;
}
catch(WorkflowException e) {
   e.printStackTrace();
}
```

printStackTrace() メソッドの詳細については、Javadoc の com.bea.wlpi.common.WorkflowException を参照してください。

ワークフロー例外ハンドラの呼び出し

ワークフロー インスタンスに対してワークフロー例外ハンドラを呼び出す _ には、以下の com.bea.wlpi.server.worklist.Worklist メソッドのいずれかを使用します。

メソッド 1

```
public java.lang.String invokeWorkflowExceptionHandler(
  java.lang.String templateDefinitionId,
  java.lang.String instanceId,
  java.lang.String handlerName,
  java.lang.String xml
) throws java.rmi.RemoteException,
  com.bea.wlpi.common.WorkflowException
```

メソッド 2

```
public java.lang.String invokeWorkflowExceptionHandler(
  java.lang.String templateDefinitionId,
  java.lang.String instanceId,
  java.lang.String handlerName,
  java.lang.String xml,
  java.lang.Object transactionId
) throws java.rmi.RemoteException,
  com.bea.wlpi.common.WorkflowException
```

`invokeWorkflowExceptionHandler()` メソッドの、値を指定するパラメータを次に表に示します。

表 24-3 invokeWorkflowExceptionHandler() メソッドのパラメータ

パラメータ	説明	有効な値
`templateDefinitionId`	ユーザがエラー ハンドラを呼び出しているワークフロー テンプレート定義の ID	有効なテンプレート定義 ID。 テンプレート定義 ID を取得するには、次の `com.bea.wlpi.common.TemplateDefinitionInfo` メソッドを使用する。 `public final String getId()` `TemplateDefinitionInfo` オブジェクトで選択可能なメソッドの詳細については、B-24 ページの「TemplateDefinitionInfo オブジェクト」を参照。
`instanceId`	ワークフロー インスタンスの ID	テンプレート定義に関連するワークフロー インスタンスの ID。 インスタンス ID を取得するには、次の `com.bea.wlpi.common.TemplateDefinitionInfo` メソッドを使用する。 `public final String getInstanceId()` `TemplateDefinitionInfo` オブジェクトを取得する方法については、22-2 ページの「ワークフロー インスタンスの取得」を参照。 `TemplateDefinitionInfo` オブジェクトで選択可能なメソッドの詳細については、B-24 ページの「TemplateDefinitionInfo オブジェクト」を参照。
`handlerName`	呼び出しするエラー ハンドラの名前	有効なエラー ハンドラ。

表 24-3 invokeWorkflowExceptionHandler() メソッドのパラメータ (続き)

パラメータ	説明	有効な値
xml	ユーザ定義のサブツリー	サブツリーを定義する XML ドキュメント。
transactionId	トランザクションの ID **注意 :** このパラメータは、クラスタ化環境においてのみ必要です。	ユニークなトランザクション ID を指定するオブジェクト。 ユニークなトランザクション ID を生成するには、次のコンストラクタを使用して新しい com.bea.wlpi.client.common.GUID オブジェクトを作成する。 `GUID transactionId = new GUID();` GUID クラスの詳細については、Javadoc の com.bea.wlpi.client.common.GUID を参照。

このメソッドは、A-33 ページの「クライアント要求 DTD」に記載のとおり、Client Request DTD の ClientReq.dtd に準拠する XML ドキュメントを返します。この XML ドキュメントは、実行中のインスタンスの情報を含み、たとえば、SAX (Simple API for XML) パーサーのような XML 構文解釈ツールを使用してドキュメント解析によるアクセスが可能です。SAX パーサーの実装サンプルについては、21-26 ページの「実行時タスクの管理例」を参照してください。

たとえば、次のコード例はエラー ハンドラの MyErrorHandler を呼び出し、ユーザ定義のツリーを myXml 文字列の変数を介して渡し、結果の XML ドキュメントを clientReq 変数に格納します。このコード例では、worklist は Worklist EJB への EJBObject 参照を表します。

```
String clientReq = worklist.invokeWorkflowExceptionHandler(
  definition.getId(), instanceID, “MyErrorHandler”, myXml);
```

テンプレート定義 ID は、com.bea.wlpi.common.TemplateDefinitionInfo オブジェクトである definition と関連付けられたメソッドを使用して取得されます。InstanceInfo オブジェクトは、22-2 ページの「ワークフロー インスタンスの取得」に記載のメソッドを使用して取得されます。instanceID は、InstanceInfo オブジェクトのメソッドを使用して取得されます。

`com.bea.wlpi.common.TemplateDefinitionInfo` メソッドの詳細については、B-24 ページの 「TemplateDefinitionInfo オブジェクト」 を参照してください。`com.bea.wlpi.common.InstanceInfo` メソッドの詳細については、B-6 ページの 「InstanceInfo オブジェクト」 を参照してください。

`invokeWorkflowExceptionHandler()` メソッドの詳細については、Javadoc の `com.bea.wlpi.server.worklist.Worklist` を参照してください。

A　DTD フォーマット

この付録では、WebLogic Integration DTD フォーマットについて説明します。内容は以下のとおりです。

- 監査 DTD
- ビジネス カレンダー DTD
- クライアント呼び出しアドイン要求 DTD
- クライアント呼び出しアドイン応答 DTD
- クライアント呼び出しプログラム要求 DTD
- クライアント呼び出しプログラム応答 DTD
- クライアント メッセージ ボックス要求 DTD
- クライアント メッセージ ボックス応答 DTD
- クライアント要求 DTD
- クライアント変数設定要求 DTD
- クライアント変数設定応答 DTD
- インポート応答 DTD
- 統計要求 DTD
- 統計応答 DTD
- テンプレート DTD
- テンプレート定義 DTD
- ワークロード要求 DTD
- ワークロード応答 DTD

監査 DTD

監査 DTD は、監査統計作成時に監査機能によって用いられる XML ドキュメントのフォーマットを説明します。

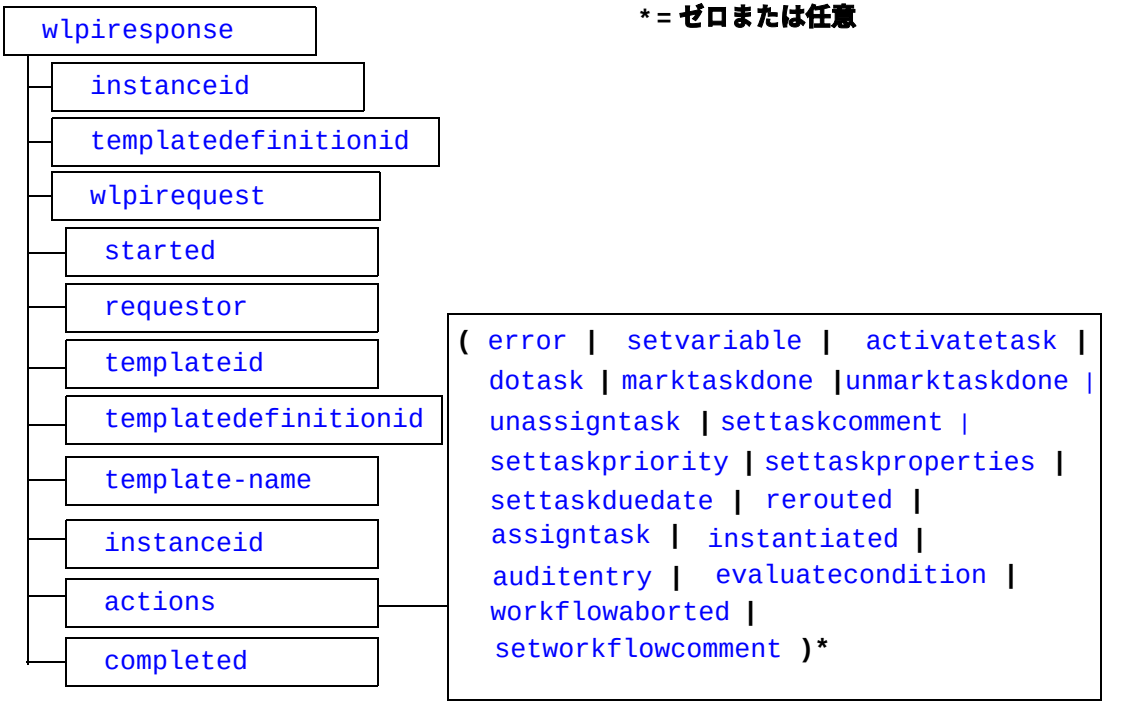
この節では、監査 DTD について説明します。内容は以下のとおりです。

- 階層構造ダイアグラム
- DTD フォーマット
- 要素説明
- 監査 DTD のサンプル

階層構造ダイアグラム

監査 DTD 階層構造を次のダイアグラムに示します。

図 A-1　監査 DTD 階層構造ダイアグラム



DTD フォーマット

監査 DTD のフォーマット Audit.dtd を次のリストに示します。

```
<!ELEMENT wlpiresponse (instanceid, templatedefinitionid, wlpirequest)>
<!ELEMENT wlpirequest (started, requestor, templateid, template-name,
                       templatedefinitionid, instanceid, actions, completed)>
<!ELEMENT actions ((error | setvariable | activatetask | dotask | marktaskdone |
                    unmarktaskdone | unassigntask | settaskcomment |
                    settaskpriority | settaskproperties | settaskduedate |
                    rerouted | assigntask | instantiated | auditentry |
                    evaluatecondition | workflowaborted | setworkflowcomment)*)>
<!ELEMENT completed (#PCDATA)>
<!ELEMENT instanceid (#PCDATA)>
<!ELEMENT requestor (#PCDATA)>
<!ELEMENT started (#PCDATA)>
```

```
<!ELEMENT templatedefinitionid (#PCDATA)>
<!ELEMENT templateid (#PCDATA)>
<!ELEMENT template-name (#PCDATA)>
<!ELEMENT error (#PCDATA)>
<!ATTLIST error time CDATA #REQUIRED id CDATA #REQUIRED>
<!ELEMENT setvariable (#PCDATA)>
<!ATTLIST setvariable time CDATA #REQUIRED variable NMTOKEN #REQUIRED)>
<!ELEMENT activatetask (#PCDATA)>
<!ATTLIST activatetask time CDATA #REQUIRED taskid CDATA #REQUIRED name CDATA>
<!ELEMENT dotask (#PCDATA)>
<!ATTLIST dotask time CDATA #REQUIRED taskid CDATA #REQUIRED name CDATA>
<!ELEMENT marktaskdone (#PCDATA)>
<!ATTLIST marktaskdone time CDATA #REQUIRED taskid CDATA #REQUIRED name CDATA>
<!ELEMENT unmarktaskdone (#PCDATA)>
<!ATTLIST unmarktaskdone time CDATA #REQUIRED taskid CDATA #REQUIRED name CDATA>
<!ELEMENT unassigntask (#PCDATA)>
<!ATTLIST unassigntask time CDATA #REQUIRED taskid CDATA #REQUIRED name CDATA>
<!ELEMENT settaskcomment (#PCDATA)>
<!ATTLIST settaskcomment time CDATA #REQUIRED taskid CDATA #REQUIRED name CDATA>
<!ELEMENT settaskpriority (#PCDATA)>
<!ATTLIST settaskpriority time CDATA #REQUIRED taskid CDATA #REQUIRED name CDATA>
<!ELEMENT settaskproperties (#PCDATA)>
<!ATTLIST settaskproperties time CDATA #REQUIRED taskid CDATA #REQUIRED
  name CDATA>
<!ELEMENT settaskduedate (#PCDATA)>
<!ATTLIST settaskduedate time CDATA #REQUIRED taskid CDATA #REQUIRED name CDATA>
<!ELEMENT rerouted (#PCDATA)>
<!ATTLIST rerouted time CDATA #REQUIRED taskid CDATA #REQUIRED name CDATA>
<!ELEMENT assigntask (#PCDATA)>
<!ATTLIST assigntask time CDATA #REQUIRED taskid CDATA #REQUIRED name CDATA>
<!ELEMENT instantiated (#PCDATA)>
<!ATTLIST instantiated time CDATA #REQUIRED taskid CDATA #REQUIRED name CDATA>
<!ELEMENT auditentry ANY>
<!ATTLIST auditentry time CDATA #REQUIRED taskid CDATA #REQUIRED name CDATA>
<!ELEMENT evaluatecondition (#PCDATA)>
<!ATTLIST evaluatecondition time CDATA #REQUIRED taskid CDATA #REQUIRED
  name CDATA>
<!ELEMENT workflowaborted (#PCDATA)>
<!ATTLIST workflowaborted time CDATA #REQUIRED taskid CDATA #REQUIRED name CDATA>
<!ELEMENT setworkflowcomment (#PCDATA)>
<!ATTLIST setworkflowcomment time CDATA #REQUIRED taskid CDATA #REQUIRED
  name CDATA>
```

要素説明

監査 DTD の要素を次の表に示します。

表 A-1 監査 DTD 要素

要素	説明	サンプル値
actions	実行されたアクションを定義する。 以下の下位要素のゼロまたは任意の数の発生を定義する。 ■ error ■ setvariable ■ activatetask ■ dotask ■ marktaskdone ■ unmarktaskdone ■ unassigntask ■ settaskcomment ■ settaskpriority ■ settaskproperties ■ settaskduedate ■ rerouted ■ assigntask ■ instantiated ■ auditentry ■ evaluatecondition ■ workflowaborted ■ setworkflowcomment	A-11 ページの「監査 DTD のサンプル」を参照。
activatetask	以下の属性を定義する。 ■ time －タスクがアクティブになった時間 ■ taskid －アクティブになったタスクの ID ■ name －アクティブになったタスクの名前	<activatetask time="2001-06-12 12:45:44.824" taskid="2" name="Task 1"/>

表 A-1 監査 DTD 要素 (続き)

要素	説明	サンプル値
assigntask	以下の属性を定義する。 ■ time －タスクが割り当てられた時間 ■ taskid －割り当てられたタスクの ID ■ name －割り当てられたタスクの名前	<assigntask time="2001-06-12 12:45:44.824" taskid="2" name="Task 1"/>
auditentry	以下の属性を定義する。 ■ time －監査エントリがロギングされた時間 ■ taskid －監査エントリがロギングされたタスクの ID ■ name －監査エントリがロギングされたタスクの名前	<auditentry time="2001-06-12 12:45:44.824" taskid="2" name="Task 1"/>
completed	タスクの実行が完了した時間	<completed>2001-06-12 12:45:45.115 </completed>
dotask	以下の属性を定義する。 ■ time －タスクが実行された時間 ■ taskid －実行されたタスクの ID ■ name －実行されたタスクの名前	<dotask time="2001-06-12 12:45:44.824" taskid="2" name="Task 1"/>
error	以下の属性を定義する。 ■ time －エラーが発生した時間 ■ id －エラー ID	<error time="2001-06-12 12:45:44.824" id="2"/>
evaluatecondition	以下の属性を定義する。 ■ time －条件が評価された時間 ■ taskid －条件が評価されたタスクの ID ■ name －条件が評価されたタスクの名前	<evaluatecondition time="2001-06-12 12:45:44.824" taskid="2" name="Task 1"/>

表 A-1 監査 DTD 要素 (続き)

要素	説明	サンプル値
instanceid	応答ドキュメントの対象となるワークフロー インスタンスの ID	<instanceid> 2 </instanceid>
instantiated	以下の属性を定義する。 ■ time －タスクがインスタンス化された時間 ■ name －インスタンス化されたタスクの ID	<instantiated time="2001-06-12 12:45:44.824" name="Start Test"/>
marktaskdone	以下の属性を定義する。 ■ time －タスクに完了マークが付けられた時間 ■ taskid －完了マークが付けられたタスクの ID ■ name －完了マークが付けられたタスクの名前	<marktaskdone time="2001-06-12 12:45:44.824" taskid="2" name="Task 1"/>
requestor	ID リクエスタ	<requestor> wlisystem </requestor>
rerouted	以下の属性を定義する。 ■ time －タスクが再ルーティングされた時間 ■ taskid －再ルーティングされたタスクの ID ■ name －再ルーティングされたタスクの名前	<rerouted time="2001-06-12 12:45:44.824" taskid="2" name="Task 1"/>
settaskcomment	以下の属性を定義する。 ■ time －タスク コメントが設定された時間 ■ taskid －コメントが設定されたタスクの ID ■ name －コメントが設定されたタスクの名前	<settaskcomment time="2001-06-12 12:45:44.824" taskid="2" name="Task 1"> This is a comment. </settaskcomment>

表 A-1 監査 DTD 要素 (続き)

要素	説明	サンプル値
settaskduedate	以下の属性を定義する。 ■ time －タスクの期限日付が設定された時間 ■ taskid －期限日付が設定されたタスクの ID ■ name －期限日付が設定されたタスクの名前	<settaskduedate time="2001-06-12 12:45:44.824" taskid="2" name="Task 1"> </settaskduedate>
settaskpriority	以下の属性を定義する。 ■ time －タスクの優先順位が設定された時間 ■ taskid －優先順位が設定されたタスクの ID ■ name －優先順位が設定されたタスクの名前	<settaskpriority time="2001-06-12 12:45:44.824" taskid="2" name="Task 1"> </settaskpriority>
settaskproperties	以下の属性を定義する。 ■ time －タスクのプロパティが設定された時間 ■ taskid －プロパティが設定されたタスクの ID ■ name －プロパティが設定されたタスクの名前	<settaskproperties time="2001-06-12 12:45:44.824" taskid="2" name="Task 1"> </settaskproperties>
setvariable	以下の属性を定義する。 ■ time －変数が設定された時間 ■ variable － XML 名前トークン（NMTOKEN）フォーマットの変数の名前	<setvariable time="2001-06-12 12:45:44.824" variable="test1"> value1 </setvariable>
setworkflowcomment	以下の属性を定義する。 ■ time －ワークフロー コメントが設定された時間 ■ taskid －コメントが設定されたタスクの ID ■ name －コメントが設定されたタスクの名前	<setworkflowcomment time="2001-06-12 12:45:44.824" taskid="2" name="Task 1"> </setworkflowcomment>

表 A-1 監査 DTD 要素 (続き)

要素	説明	サンプル値
started	タスクの実行が開始された時間	<started> 2001-06-12 12:45:44.824 </started
templatedefinitionid	応答ドキュメントの対象となるワークフロー テンプレート定義の ID	<templatedefinitionid> 2 </templatedefinitionid>
templateid	応答ドキュメントの対象となるワークフロー テンプレートの ID	<templateid> 2 </templateid>
template-name	ワークフロー テンプレートの名前	<template-name> Start Test </template-name>
unassigntask	以下の属性を定義する。 ■ time －タスクの割り当てが解除された時間 ■ taskid －割り当てが取り消されたタスクの ID ■ name －割り当てが取り消されたタスクの名前	<unassigntask time="2001-06-12 12:45:44.824" taskid="2" name="Task 1"/>
unmarktaskdone	以下の属性を定義する。 ■ time －タスクに未完了マークが付けられた時間 ■ taskid －未完了マークが付けられたタスクの ID ■ name －未完了マークが付けられたタスクの名前	<unmarktaskdone time="2001-06-12 12:45:44.824" taskid="2" name="Task 1"/>

表 A-1 監査 DTD 要素 (続き)

要素	説明	サンプル値
wlpirequest	以下の下位要素を含む要求を定義する。 ■ started ■ requestor ■ templateid ■ templatedefinitionid ■ instanceid ■ actions ■ completed	A-11 ページの「監査 DTD のサンプル」を参照。
wlpiresponse	ルート要素 以下の下位要素を定義する。 ■ instanceid ■ templatedefinitionid ■ wlpirequest	A-11 ページの「監査 DTD のサンプル」を参照。
workflowaborted	以下の属性を定義する。 ■ time －ワークフローが中断された時間 ■ taskid －ワークフローが中断されたタスクの ID ■ name －未完了マークが付けられたタスクの名前	<workflowaborted time="2001-06-12 12:45:44.824" taskid="2" name="Task 1"/>

監査 DTD のサンプル

監査 DTD の有効な適用サンプルを次に示します。

```
<wlpirequest>
   <started>2001-06-12 12:45:44.824</started
   <requestor>wlisystem</requestor>
   <templateid>2</templateid>
   <template-name>Start Test</template-name>
   <templatedefinitionid>2</templatedefinitionid>
   <instanceid>8010</instanceid>
   <actions>
      <instantiated time="2001-06-12 12:45:44.824"
         name="Start Test"/>
      <setvariable time="2001-06-12 12:45:44.824"
         variable="test1">value1</setvariable>
      <setvariable time="2001-06-12 12:45:44.824"
         variable="test2">value2</setvariable>
      <activatetask time="2001-06-12 12:45:44.824" taskid="2"
         name="Task 1"/>
      <dotask time="2001-06-12 12:45:44.824" taskid="2"
         name="Task 1"/>
      <marktaskdone time="2001-06-12 12:45:44.824"
         taskid="2" name="Task 1"/>
      <workflowdone time="2001-06-12 12:45:45.115"
         name="Start Test"/>
   </actions>
   <completed>2001-06-12 12:45:45.115</completed>
</wlpirequest>
```

ビジネス カレンダー DTD

ビジネス カレンダー DTD は、ビジネス カレンダー作成に用いられる XML ドキュメントのフォーマットを説明します。ビジネス カレンダーは、オーガニゼーションの稼動時間を定義するために用いられます。ビジネス カレンダーのコンフィグレーションに関する詳細については、12-1 ページの「ビジネス カレンダーのコンフィグレーション」を参照してください。

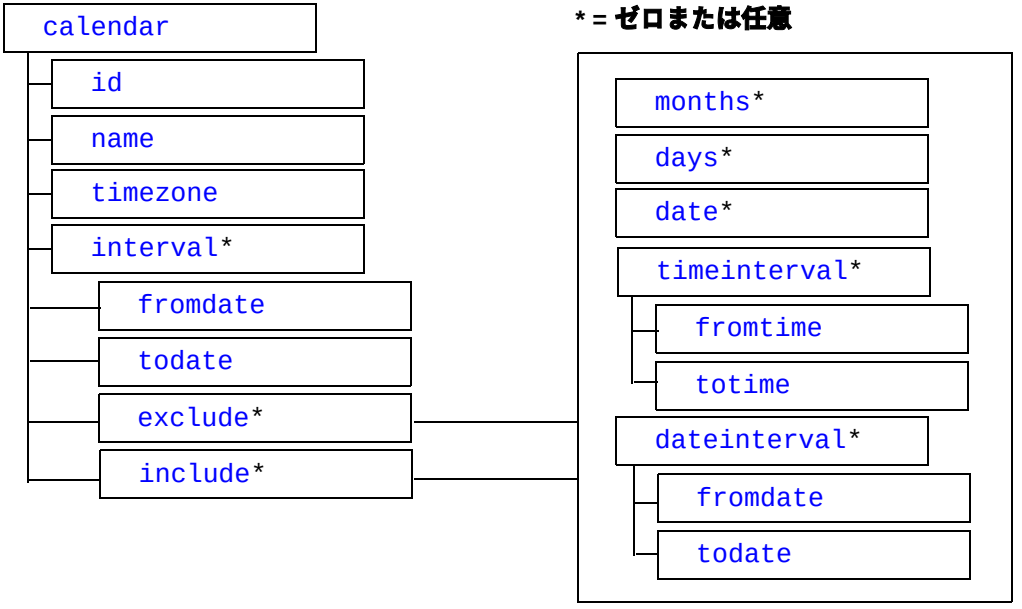
この節では、ビジネス カレンダー DTD について説明します。内容は以下のとおりです。

- 階層構造ダイアグラム
- DTD フォーマット
- 要素説明
- ビジネス カレンダー DTD のサンプル

階層構造ダイアグラム

ビジネス カレンダー DTD の階層構造を次のダイアグラムで示します。

図 A-2　ビジネス カレンダー DTD 階層構造ダイアグラム



DTD フォーマット

ビジネス カレンダー DTD のフォーマット BusinessCalender.dtd を次のリストに示します。

```
<!ELEMENT calendar (id, name, timezone, interval*)>
<!ELEMENT date (#PCDATA)>
<!ELEMENT dateinterval (fromdate, todate)>
<!ELEMENT days (#PCDATA)>
<!ELEMENT exclude (months*, days*, date*, timeinterval*,
dateinterval*)>
<!ELEMENT fromdate (#PCDATA)>
<!ELEMENT fromtime (#PCDATA)>
<!ELEMENT id (#PCDATA)>
<!ELEMENT include (months*, days*, date*, timeinterval*,
dateinterval*)>
<!ELEMENT interval (fromdate, todate, exclude*, include*)>
<!ELEMENT months (#PCDATA)>
<!ELEMENT name (#PCDATA)>
<!ELEMENT timeinterval (fromtime, totime)>
<!ELEMENT timezone (#PCDATA)>
<!ELEMENT todate (#PCDATA)>
<!ELEMENT totime (#PCDATA)>
```

要素説明

ビジネス カレンダー DTD の要素を次の表に示します。

表 A-2 ビジネス カレンダー DTD 要素

要素	説明	サンプル値
calendar	ルート要素 以下の下位要素を定義する。 ■ id ■ name ■ timezone ■ interval (ゼロまたは任意の数)	A-19 ページの「ビジネス カレンダー DTD のサンプル」を参照。

表 A-2 ビジネス カレンダー DTD 要素 (続き)

要素	説明	サンプル値
date	含める、または除外する日付 この日付は、次のフォーマットを用いて指定する。month dd, yyyy ― ここで month は 1 月などの月名、dd は日付、yyyy は年を表す。	<date> December 25, 2001 </date>
dateinterval	カレンダーに含める、あるいはカレンダーから除外する時間のインターバル 以下の下位要素を定義する。 ■ fromdate ■ todate	<dateinterval> <fromdate> January 1, 2001 </fromdate> <todate> January 1, 2002 </todate> </dateinterval>
days	含める、または除外する曜日 この要素の値は、日曜日などの曜日名をテキスト文字列で指定する。	<days>Saturday</days>
exclude	メインの interval から除外する時間のインターバルを定義するルール 以下の下位要素を定義する。 ■ months (ゼロまたは任意の数) ■ days (ゼロまたは任意の数) ■ date (ゼロまたは任意の数) ■ timeinterval (ゼロまたは任意の数) ■ dateinterval (ゼロまたは任意の数)	<exclude> <days> Saturday, Sunday </days> <date> January 1, 2001 </date> </exclude>

表 A-2 ビジネス カレンダー DTD 要素 (続き)

要素	説明	サンプル値
fromdate	interval または dateinterval の下位要素 interval の下位要素として用いる場合、インターバルの開始日を指定する。dateinterval の下位要素として用いる場合、include または exclude を指定する。 この日付は、次のフォーマットを用いて指定する。month dd, yyyy ― ここで month は 1 月などの月名、dd は日付、yyyy は年を表す。	<fromdate> January 1, 2001 </fromdate>
fromtime	ビジネス カレンダーが有効となる開始時間、あるいは timeinterval の下位要素 下位要素の場合、含めるあるいは除外する時間の範囲を指定する。 この時間は、次のフォーマットを用いて指定する。*hh:mm* ― ここで *hh* は時間、mm は分を表す。	08:00
id	固有 ID	<id>10234</id>
include	メインの interval に含める時間のインターバルを定義するルール 以下の下位要素を定義する。 ■ months (ゼロまたは任意の数) ■ days (ゼロまたは任意の数) ■ date (ゼロまたは任意の数) ■ timeinterval (ゼロまたは任意の数) ■ dateinterval (ゼロまたは任意の数)	<include> <timeinterval> 08:00, 17:00 </timeinterval> </include>

表 A-2 ビジネス カレンダー DTD 要素 (続き)

要素	説明	サンプル値
interval	カレンダーが有効である時間のインターバル 以下の下位要素を定義する。 ■ fromdate ■ todate ■ exclude (ゼロまたは任意の数) ■ include (ゼロまたは任意の数)	<interval> <fromdate> 20010101 </fromdate> <todate> 20020101 </todate> <exclude> <date> 20011225 </date> <days> Saturday, Sunday </days> </exclude> <include> <timeinterval> 08:00. 17:00 </timeinterval> </include> </interval>
months	含める、または除外する月 この要素の値は、1 月などの月名を指定するテキスト文字列。	<months> August </months>
name	カレンダー名	<name> MyCalendar </name>
timeinterval	カレンダーに含める、あるいはカレンダーから除外する時間のインターバル 以下の下位要素を定義する。 ■ fromtime ■ totime	<timeinterval> <fromtime> 17:00:00 </fromtime> <totime> 08:00:00 </totime> </timeinterval>
timezone	カレンダーのタイム ゾーン (有効値については、表の最後の注意を参照)	<timezone> EST </timezone>

表 A-2 ビジネス カレンダー DTD 要素 (続き)

要素	説明	サンプル値
todate	ビジネス カレンダーが有効である最終日 この日付は、次のフォーマットを用いて指定する。month dd, yyyy ― ここで month は 1 月などの月名、dd は日付、yyyy は年を表す。	<todate> January 1, 2002 </todate>
totime	ビジネス カレンダーが有効である最終時間 この時間は、次のフォーマットを用いて指定する。*hh:mm* ― ここで *hh* は時間、mm は分を表す。	<totime>17:00</totime>

注意 : 有効なタイムゾーンの値は、次のとおりです。Africa/Lome, GMT, UTC, Atlantic/Faeroe, Atlantic/Canary, Europe/Dublin, Europe/Lisbon, Europe/London, Africa/Luanda, Africa/Porto-Novo, Africa/Bangui, Africa/Kinshasa, Africa/Douala, Africa/Libreville, Africa/Malabo, Africa/Niamey, Africa/Lagos, Africa/Ndjamena, Africa/Tunis, Africa/Algiers, Europe/Andorra, Europe/Tirane, Europe/Vienna, Europe/Brussels, Europe/Zurich, Europe/Prague, Europe/Berlin, Europe/Copenhagen, Europe/Madrid, Europe/Gibraltar, Europe/Budapest, Europe/Rome, Europe/Vaduz, Europe/Luxembourg, Africa/Tripoli, Europe/Monaco, Europe/Malta, Africa/Windhoek, Europe/Amsterdam, Europe/Oslo, Europe/Warsaw, Europe/Stockholm, Europe/Belgrade, Europe/Paris, ECT, Africa/Bujumbura, Africa/Gaborone, Africa/Lubumbashi, Africa/Maseru, Africa/Blantyre, Africa/Maputo, Africa/Kigali, Africa/Khartoum, Africa/Mbabane, Africa/Lusaka, Africa/Harare, CAT, Africa/Johannesburg, Europe/Sofia, Europe/Minsk, Asia/Nicosia, Europe/Tallinn, Africa/Cairo, ART, Europe/Helsinki, Europe/Athens, Asia/Jerusalem, Asia/Amman, Asia/Beirut, Europe/Vilnius, Europe/Riga, Europe/Chisinau, Europe/Bucharest, Europe/Kaliningrad, Asia/Damascus, Europe/Kiev, Europe/Istanbul, EET, Asia/Bahrain, Africa/Djibouti, Africa/Asmera, Africa/Addis_Ababa, EAT, Africa/Nairobi, Indian/Comoro, Asia/Kuwait,

注意： Indian/Antananarivo, Asia/Qatar, Africa/Mogadishu, Africa/Dar_es_Salaam, Africa/Kampala, Asia/Aden, Indian/Mayotte, Asia/Riyadh, Asia/Baghdad, Europe/Simferopol, Europe/Moscow, Asia/Tehran, MET, Asia/Dubai, Indian/Mauritius, Asia/Muscat, Indian/Reunion, Indian/Mahe, Asia/Yerevan, NET, Asia/Baku, Asia/Aqtau, Europe/Samara, Asia/Kabul, Indian/Kerguelen, Asia/Tbilisi, Indian/Chagos, Indian/Maldives, Asia/Dushanbe, Asia/Ashkhabad, Asia/Tashkent, Asia/Karachi, PLT, Asia/Bishkek, Asia/Aqtobe, Asia/Yekaterinburg, Asia/Calcutta, IST, Asia/Katmandu, Antarctica/Mawson, Asia/Thimbu, Asia/Colombo, Asia/Dacca, BST, Asia/Alma-Ata, Asia/Novosibirsk, Indian/Cocos, Asia/Rangoon, Indian/Christmas, Asia/Jakarta, Asia/Phnom_Penh, Asia/Vientiane, Asia/Saigon, VST, Asia/Bangkok, Asia/Krasnoyarsk, Antarctica/Casey, Australia/Perth, Asia/Brunei, Asia/Hong_Kong, Asia/Ujung_Pandang, Asia/Ishigaki, Asia/Macao, Asia/Kuala_Lumpur, Asia/Manila, Asia/Singapore, Asia/Taipei, Asia/Shanghai, CTT, Asia/Ulan_Bator, Asia/Irkutsk, Asia/Jayapura, Asia/Pyongyang, Asia/Seoul, Pacific/Palau, Asia/Tokyo, JST, Asia/Yakutsk, Australia/Darwin, ACT, Australia/Adelaide, Antarctica/DumontDUrville, Pacific/Truk, Pacific/Guam, Pacific/Saipan, Pacific/Port_Moresby, Australia/Brisbane, Asia/Vladivostok, Australia/Sydney, AET, Australia/Lord_Howe, Pacific/Ponape, Pacific/Efate, Pacific/Guadalcanal, SST, Pacific/Noumea, Asia/Magadan, Pacific/Norfolk, Pacific/Kosrae, Pacific/Tarawa, Pacific/Majuro, Pacific/Nauru, Pacific/Funafuti, Pacific/Wake, Pacific/Wallis, Pacific/Fiji, Antarctica/McMurdo, Asia/Kamchatka, Pacific/Auckland, NST, Pacific/Chatham, Pacific/Enderbury, Pacific/Tongatapu, Asia/Anadyr, Pacific/Kiritimati

ビジネス カレンダー DTD のサンプル

ビジネス カレンダー DTD の有効アプリケーションを次のサンプルで示します。

```
<calendar>
<id>acme2000</id>
<name>Acme, Inc. Year 2000 Business Calendar</name>
<timezone>EST</timezone>
<interval>
<fromdate>January 3, 2000</fromdate>
<todate>December 31, 2000</todate>
<exclude>
<date>January 3, 2000</date>
<date>December 25, 2000</date>
<days>Saturday, Sunday</days>
<dateinterval>
<fromdate>
   December 22, 2000 12:00:00
</fromdate>
<todate>
   January 1, 2001 00:00:00
</todate>
</dateinterval>
<timeinterval>
<fromtime>17:00:00</fromtime>
<totime>08:00:00</totime>
</timeinterval>
</exclude>
</interval>
</calendar>
```

クライアント呼び出しアドイン要求 DTD

クライアント呼び出しアドイン要求 DTD は、アドインを呼び出す場合に用いられる XML ドキュメントのフォーマットを説明します（戻り値のフォーマットについては、「クライアント呼び出しアドイン応答 DTD」で説明）。クライアント呼び出しアドイン要求 DTD に準拠した XML ドキュメントは、`ActionSendXMLToClient` アクションの一部としてテンプレート定義を作成する際に渡すことができます。詳細については、A-53 ページの「テンプレート定義 DTD」を参照してください。

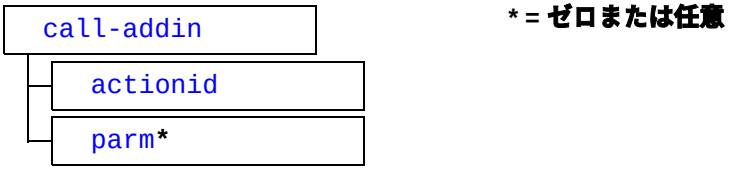
クライアント呼び出しアドイン要求 DTD について説明します。内容は以下のとおりです。

- 階層構造ダイアグラム
- DTD フォーマット
- 要素説明
- クライアント呼び出しアドイン要求 DTD のサンプル

階層構造ダイアグラム

クライアント呼び出しアドイン要求 DTD の階層構造を次のダイアグラムで示します。

図 A-3　クライアント呼び出しアドイン要求 DTD 階層構造ダイアグラム



DTD フォーマット

クライアント呼び出しアドイン要求 DTD のフォーマット ClientCallAddInReq.dtd を次のリストに示します。

```
<!ELEMENT call-addin (actionid, parm*)>
<!ATTLIST call-addin name CDATA #REQUIRED
                     mode (sync|async) "async">
<!ELEMENT actionid (#PCDATA)>
<!ELEMENT parm (#PCDATA)>
```

要素説明

クライアント呼び出しアドイン要求 DTD の要素を次の表に示します。

表 A-3 クライアント呼び出しアドイン要求 DTD 要素

要素	説明	サンプル値
actionid	アクション ID	<actionid> 959395846210 </actionid>
call-addin	ルート要素 以下の下位要素を定義する。 ■ actionid ■ parm（ゼロまたは任意の数） 以下の属性を定義する。 ■ name －呼び出す Worklist アドインの完全修飾 Java クラス名。com.bea.wlpi.client.worklist.WorklistAddIn インタフェースを実装する。この属性は必須 ■ mode －クライアント要求の実行中に、実行時クライアントがブロックするかどうかを指定するプログラム モード。この値は、要求の実行完了まで実行時クライアントをブロックする場合は sync（同期モード）に、バックグラウンド タスクとしてクライアント要求を実行する場合は async（非同期モード）に設定。この属性のデフォルトは、async	A-22 ページの「クライアント呼び出しアドイン要求 DTD のサンプル」を参照。
parm	パラメータ この値は、どのような文字列にも設定できる。	<parm> itemNumber </parm>

クライアント呼び出しアドイン要求 DTD のサンプル

クライアント呼び出しアドイン要求 DTD の有効アプリケーションを次のサンプルで示します。

```
<call-addin name="com.somedomain.someproduct.WorklistAddInImpl"
mode="async">
<actionid>959395846210</actionid>
<parm>itemNumber</parm>
</call-addin>
```

クライアント呼び出しアドイン応答 DTD

クライアント呼び出しアドイン応答 DTD は、アドインを呼び出した場合に戻される XML ドキュメントのフォーマットを説明します（アドイン呼び出しに用いられる XML ドキュメントのフォーマットについては、「クライアント呼び出しアドイン要求 DTD」で説明）。

XPath 関数を用いて、応答ファイルからの戻り値を抽出できます。たとえば、

```
XPath("/call-addin/child::text()")
XPath("/call-addin/text()")
```

戻り値に XML ノードなどのオブジェクトが含まれている場合、XPath 式はその構造体に従って構築される必要があります。

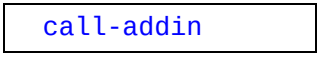
クライアント呼び出しアドイン応答 DTD を以下の項目で説明します。内容は以下のとおりです。

- 階層構造ダイアグラム
- DTD フォーマット
- 要素説明

階層構造ダイアグラム

クライアント呼び出しアドイン応答 DTD の階層構造を次のダイアグラムに示します。

図 A-4　クライアント呼び出しアドイン応答 DTD 階層構造ダイアグラム



DTD フォーマット

クライアント呼び出しアドイン応答 DTD のフォーマット `ClientCallAddInResp.dtd` を次のリストに示します。

```
<!ELEMENT call-addin ANY>
```

要素説明

クライアント呼び出しアドイン応答 DTD の要素を次の表に示します。

表 A-4　クライアント呼び出しアドイン応答 DTD 要素

要素	説明
`call-addin`	ルート要素 この要素の値は、要素とテキストの組み合わせから成る。

クライアント呼び出しプログラム要求 DTD

クライアント呼び出しプログラム要求 DTD は、外部プログラムを呼び出す場合に用いられる XML ドキュメントのフォーマットを説明します（「クライアント呼び出しプログラム応答 DTD」は、戻り値のフォーマットを説明）。クライアント呼び出しプログラム要求 DTD に準拠した XML ドキュメントは、`ActionSendXMLToClient` アクションの一部としてテンプレート定義を作成する際に渡すことができます。詳細については、A-53 ページの「テンプレート定義 DTD」を参照してください。

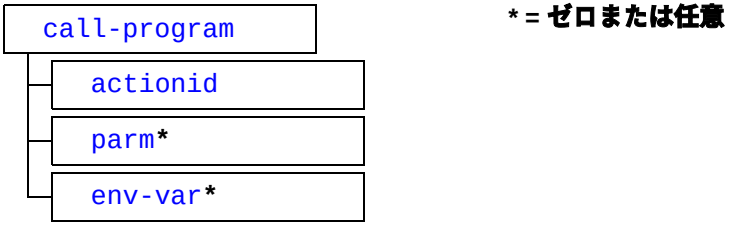
この節では、クライアント呼び出しプログラム要求 DTD について説明します。内容は以下のとおりです。

- 階層構造ダイアグラム
- DTD フォーマット
- 要素説明
- クライアント呼び出しプログラム応答 DTD のサンプル

階層構造ダイアグラム

クライアント呼び出しプログラム要求 DTD の階層構造を次のダイアグラムに示します。

図 A-5　クライアント呼び出しプログラム要求 DTD 階層構造ダイアグラム



DTD フォーマット

クライアント呼び出しプログラム要求 DTD のフォーマット ClientCallPgmReq.dtd を次のリストに示します。

```
<!ELEMENT call-program (actionid, parm*, env-var*)>
<!ATTLIST call-program name CDATA #REQUIRED
                       mode (sync|async) "async">
<!ELEMENT actionid (#PCDATA)>
<!ELEMENT parm (#PCDATA)>
<!ELEMENT env-var (#PCDATA)>
<!ATTLIST env-var name NMTOKEN #REQUIRED>
```

要素説明

クライアント呼び出しプログラム要求 DTD の要素を次の表に示します。

表 A-5 クライアント呼び出しプログラム要求 DTD 要素

要素	説明	サンプル値
call-program	ルート要素 以下の下位要素を定義する。 ■ actionid ■ parm（ゼロまたは任意の数） ■ env-var（ゼロまたは任意の数） 以下の属性を定義する。 ■ mode －クライアント要求の実行中に、実行時クライアントがブロックするかどうかを指定するプログラム モード。この値は、要求の実行完了まで実行時クライアントをブロックする場合は sync（同期モード）に、バックグラウンド タスクとしてクライアント要求を実行する場合は async（非同期モード）に設定。この属性のデフォルトは、async ■ name －プログラム名。この属性は必須	A-26 ページの「クライアント呼び出しプログラム要求 DTD のサンプル」を参照。

表 A-5 クライアント呼び出しプログラム要求 DTD 要素 (続き)

要素	説明	サンプル値
actionid	アクション ID	<actionid> 992456131534 </actionid>
parm	パラメータ定義この値は、どのような文字列にも設定できる。	<parm> C:\WLPI\readme.txt </parm>
env-var	環境変数定義 XML 名前トークン（NMTOKEN）フォーマットで環境変数名を指定する name 属性を定義する。	<env-var name="TEMP"> C:\TEMP </env-var>

クライアント呼び出しプログラム要求 DTD のサンプル

クライアント呼び出しプログラム要求 DTD の有効アプリケーションを次のサンプルで示します。

```
<call-program mode="async" name="notepad">
  <actionid>992456131534</actionid>
  <parm>C:\WLPI\readme.txt</parm>
  <env-var name="TEMP">C:\TEMP</env-var>
  <env-var name="TMP">C:\TEMP</env-var>
</call-program>
```

クライアント呼び出しプログラム応答 DTD

クライアント呼び出しプログラム応答 DTD は、外部プログラムを呼び出した場合に戻される XML ドキュメントのフォーマットを説明します（外部プログラム呼び出しに用いられる XML ドキュメントのフォーマットについては、「クライアント呼び出しプログラム要求 DTD」 で説明）。

XPath 関数を用いて、戻り値を抽出できます。たとえば、

```
XPath("/call-program/attribute::exit-value")
XPath("/call-program/@exit-value")
```

この節では、クライアント呼び出しプログラム応答 DTD について説明します。内容は以下のとおりです。

- 階層構造ダイアグラム
- DTD フォーマット
- 要素説明
- クライアント呼び出しプログラム応答 DTD のサンプル

階層構造ダイアグラム

クライアント呼び出しプログラム応答 DTD の階層構造を次のダイアグラムに示します。

図 A-6　クライアント呼び出しプログラム応答 DTD 階層構造ダイアグラム

DTD フォーマット

クライアント呼び出しプログラム応答 DTD のフォーマット ClientCallProgramResp.dtd を次のリストに示します。

```
<!ELEMENT call-program EMPTY>
<!ATTLIST call-program exit-value NUMBER #REQUIRED>
```

要素説明

クライアント呼び出しプログラム応答 DTD の要素を次の表に示します。

表 A-6　クライアント呼び出しプログラム応答 DTD 要素

要素	説明	サンプル値
call-program	ルート要素この要素は空になっている。 オペレーティング システムによって検索され、呼び出し元のワークフローのコンテキストにおいて有意な数値終了コードを指定するための exit-value を定義する。有効な終了コードに関する詳細については、該当するプログラム ドキュメントを参照。この属性は必須	「A-28 ページの「クライアント呼び出しプログラム応答 DTD のサンプル」」を参照。

クライアント呼び出しプログラム応答 DTD のサンプル

クライアント呼び出しプログラム応答 DTD の有効アプリケーションを次のサンプルで示します。

```
<call-program exit-value=0>
</call-program>
```

クライアント メッセージ ボックス要求 DTD

クライアント メッセージ ボックス要求 DTD は、以下に用いられる XML ドキュメントのフォーマットを説明するためのものです。(a) ダイアログ ボックスでメッセージに応答するようユーザにプロンプトし、(b) 応答を検索します（「クライアント メッセージ ボックス応答 DTD」は、戻り値のフォーマットを説明）。クライアント メッセージ ボックス要求 DTD に準拠した XML ドキュメントは、ActionSendXMLToClient アクションの一部としてテンプレート定義を作成する際に渡すことができます。詳細については、A-53 ページの「テンプレート定義 DTD」を参照してください。

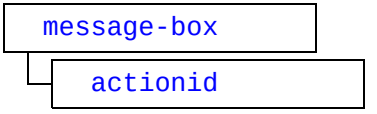
この節では、クライアント メッセージ ボックス要求 DTD について説明します。内容は以下のとおりです。

- 階層構造ダイアグラム
- DTD フォーマット
- 要素説明
- クライアント メッセージ ボックス要求 DTD のサンプル

階層構造ダイアグラム

クライアント メッセージ ボックス要求 DTD の階層構造を次のダイアグラムに示します。

図 A-7　クライアント メッセージ ボックス要求 DTD 階層構造ダイアグラム



DTD フォーマット

クライアント メッセージ ボックス要求 DTD のフォーマット ClientMsgBoxReq.dtd を次のリストに示します。

```
<!ELEMENT message-box (#PCDATA, actionid)>
<!ATTLIST message-box title CDATA #IMPLIED
                      style (plain|information|question|warning|
                      error) "plain"
                      options (ok|ok_cancel|yes_no|
                      yes_no_cancel) "ok">
<!ELEMENT actionid (#PCDATA)>
```

要素説明

クライアント メッセージ ボックス要求 DTD の要素を次の表に示します。

表 A-7 クライアント メッセージ ボックス要求 DTD 要素

要素	説明	サンプル値
actionid	アクション ID	<actionid> 990705990915 </actionid>
message-box	ルート要素 actionid の下位要素を定義する。 以下の属性を定義する。 ■ title －メッセージ ボックスのタイトル フィールドに表示されるテキスト。この属性の入力は省略可能 ■ style －メッセージ ボックスのスタイル。この要素の値は、error、information、plain、question、あるいは warning のいずれかに設定可能。デフォルトは plain ■ options －メッセージ ボックスのボタン オプション。表 A-8 に、割り当てた各スタイル値で使用できるメッセージ ボックス オプションを示す。	A-31 ページの「クライアント メッセージ ボックス要求 DTD のサンプル」 を参照。

次の表に、割り当てた各 style 値で使用できるメッセージ ボックス オプションを示します。

表 A-8 各 style 値で使用できるメッセージ ボックス オプション

メッセージ ボックスの style 値	メッセージ ボックス オプション
error	ok、ok_cancel

表 A-8 各 style 値で使用できるメッセージ ボックス オプション (続き)

メッセージ ボックスの style 値	メッセージ ボックス オプション
information	ok
plain	ok、ok_cancel、yes_no、yes_no_cancel
question	yes_no、yes_no_cancel
warning	ok、ok_cancel

クライアント メッセージ ボックス要求 DTD のサンプル

クライアント メッセージ ボックス要求 DTD の有効アプリケーションを次のサンプルで示します。

```
<message-box options="yes_no" style="question"
  title="Credit Check">Does customer 6831 pass credit check?|
  <actionid>990705990915</actionid>
</message-box>
```

クライアント メッセージ ボックス応答 DTD

クライアント メッセージ ボックス応答 DTD は、メッセージ ダイアログ ボックスでユーザが追加情報を求められた場合に戻される XML ドキュメントのフォーマットを説明します（ユーザへのプロンプトに用いられる XML ドキュメントのフォーマットについては、「クライアント メッセージ ボックス要求 DTD」で説明）。

XPath 関数を用いて、戻り値を抽出できます。たとえば、

```
XPath("/message-box/attribute::option")
XPath("/message-box/@option")
```

この節では、クライアント メッセージ ボックス応答 DTD について説明します。内容は以下のとおりです。

- 階層構造ダイアグラム
- DTD フォーマット
- 要素説明
- テンプレート定義 DTD サンプル

階層構造ダイアグラム

クライアント メッセージ ボックス応答 DTD の階層構造を次のダイアグラムに示します。

図 A-8　クライアント メッセージ ボックス応答 DTD 階層構造ダイアグラム

DTD フォーマット

クライアント メッセージ ボックス応答 DTD のフォーマット `ClientMsgBoxResp.dtd` を次のリストに示します。

```
<!ELEMENT message-box EMPTY>
<!ATTLIST message-box option (ok|yes|no|cancel) #REQUIRED>
```

要素説明

クライアント メッセージ ボックス応答 DTD の要素を次の表に示します。

表 A-9 クライアント メッセージ ボックス応答 DTD 要素

要素	説明	サンプル値
message-box	ルート要素この要素は空になっている。 ユーザによって入力されるオプションを指定するためには、option 属性を定義する。この値は、ok、yes、no、あるいは cancel のいずれかに設定可能。この属性は必須	A-33 ページの「クライアント メッセージ ボックス応答 DTD のサンプル」を参照。

クライアント メッセージ ボックス応答 DTD のサンプル

クライアント メッセージ ボックス応答 DTD の有効アプリケーションを次のサンプルで示します。

```
<message-box option=yes>
</message-box>
```

クライアント要求 DTD

クライアント要求 DTD は、Part IV「実行時の管理」に記載の実行時管理メソッドのサブセットによって戻される XML ドキュメントのフォーマットを説明します。

戻された XML ファイルには、実行時メソッドの呼び出しの結果発生したすべてのワークフローおよびタスクの更新情報が含まれています。また、クライアントに対し処理すべき要求が含まれている場合もあります。このような要求は、A-53 ページの「テンプレート定義 DTD」に記載の ActionSendXMLToClient アクションの一部として生成されます。

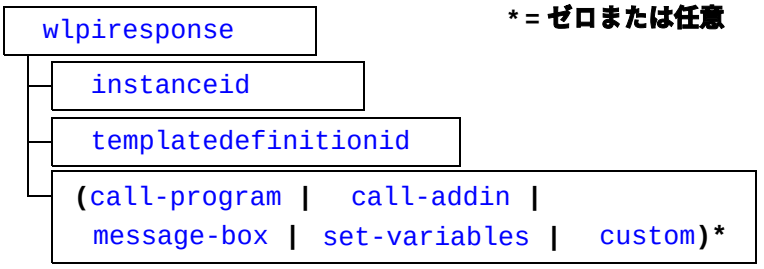
この節では、クライアント要求 DTD について説明します。内容は以下のとおりです。

- 階層構造ダイアグラム
- DTD フォーマット
- 要素説明
- エンティティ説明

階層構造ダイアグラム

クライアント要求 DTD の階層構造を次のダイアグラムに示します。

図 A-9　クライアント要求 DTD 階層構造ダイアグラム



DTD フォーマット

クライアント要求 DTD のフォーマット ClientReq.dtd を次のリストで示します。

```
<!ENTITY callpgm   SYSTEM "ClientCallPgmReq.dtd">
<!ENTITY calladdin SYSTEM "ClientCallAddInReq.dtd">
<!ENTITY msgbox    SYSTEM "ClientMsgBoxReq.dtd">
<!ENTITY setvars   SYSTEM "ClientSetVarsReq.dtd">

<!ELEMENT wlpiresponse (instanceid, templatedefinitionid,
(call-program | call-addin | set-variables |  message-box |
custom)*)>
<!ELEMENT instanceid (#PCDATA)>
<!ELEMENT templatedefinitionid (#PCDATA)>
&callpgm;
&calladdin;
```

```
&msgbox;
&setvars;
<!ELEMENT custom ANY>
```

要素説明

クライアント要求 DTD の要素を次の表に示します。

表 A-10 クライアント要求 DTD 要素

要素	説明	サンプル値
call-addin	クライアントに対する、A-19 ページの「クライアント呼び出しアドイン要求 DTD」に準拠した java アドイン呼び出し要求	A-22 ページの「クライアント呼び出しアドイン要求 DTD のサンプル」を参照。
call-program	クライアントに対する、A-24 ページの「クライアント呼び出しプログラム要求 DTD」に準拠した外部プログラム呼び出し要求	A-26 ページの「クライアント呼び出しプログラム要求 DTD のサンプル」を参照。
custom	要素とテキストの組み合わせから成るカスタム要求	
instanceid	応答ドキュメントの対象となるワークフロー インスタンスの ID	<instanceid> 3 </instanceid>
message-box	クライアントに対する、A-28 ページの「クライアント メッセージ ボックス要求 DTD」に準拠したメッセージへの応答要求	A-31 ページの「クライアント メッセージ ボックス要求 DTD のサンプル」を参照。
set-variables	クライアントに対する、A-37 ページの「クライアント変数設定要求 DTD」に準拠した変数設定要求	A-39 ページの「クライアント変数設定要求 DTD のサンプル」を参照。
templatedefinitionid	応答ドキュメントの対象となるワークフロー テンプレート定義の ID	<templatedefinitionid> 2 </templatedefinitionid>

表 A-10　クライアント要求 DTD 要素 (続き)

要素	説明	サンプル値
wlpiresponse	ルート要素 以下の下位要素を定義する。 ■ wlpiresponse ■ instanceid ■ templatedefinitionid ■ call-program | call-addin | set-variables | message-box | custom (ゼロまたは任意の数)	

エンティティ説明

クライアント要求 DTD のエンティティの説明を次の表に示します。

表 A-11　クライアント要求 DTD エンティティ

要素	意味
calladdin	クライアントアドイン呼び出し要求 DTD、ClientCallAddInReq.dtd
callpgm	クライアント呼び出しプログラム要求 DTD、ClientCallPgmReq.dtd
msgbox	クライアント メッセージ ボックス要求 DTD、ClientMsgBoxReq.dtd
setvars	クライアント変数設定要求 DTD、ClientSetVarsReq.dtd

クライアント変数設定要求 DTD

クライアント変数設定要求 DTD は、以下に用いられる XML ドキュメントのフォーマットを説明するためのものです。(a) ダイアログ ボックスで変数を設定するようユーザにプロンプトし、(b) 応答を検索します（「クライアント変数設定応答 DTD」は、戻り値のフォーマットを説明）。クライアント変数設定要求 DTD に準拠した XML ドキュメントは、`ActionSendXMLToClient` アクションの一部としてテンプレート定義を作成する際に渡すことができます。詳細については、A-53 ページの「テンプレート定義 DTD」を参照してください。

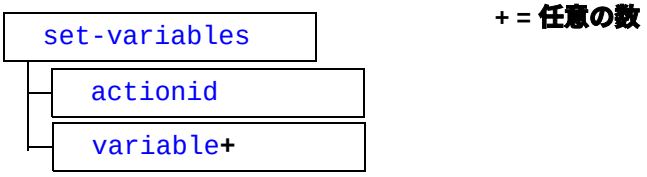
この節では、クライアント変数設定要求 DTD について説明します。内容は以下のとおりです。

- 階層構造ダイアグラム
- DTD フォーマット
- 要素説明
- クライアント変数設定要求 DTD のサンプル

階層構造ダイアグラム

クライアント変数設定要求 DTD の階層構造を次のダイアグラムに示します。

図 A-10　クライアント変数設定要求 DTD 階層構造ダイアグラム



DTD フォーマット

クライアント変数設定要求 DTD のフォーマット `ClientSetVarsReq.dtd` を次に示します。

```
<!ELEMENT set-variables (actionid, variable+)>
<!ATTLIST set-variables title CDATA #IMPLIED>
<!ELEMENT actionid (#PCDATA)>
<!ELEMENT variable (#PCDATA)>
<!ATTLIST variable name NMTOKEN #REQUIRED
                   prompt NMTOKEN #IMPLIED>
```

要素説明

クライアント変数設定要求 DTD の要素を次の表に示します。

表 A-12　クライアント変数設定要求 DTD 要素

要素	説明	サンプル値

`actionid`	アクション ID	`<actionid>`
		`991408825931`
		`</actionid>`

`set-variables`	ルート要素 以下の下位要素を定義する。 ■ `actionid` ■ `variable`　(任意の数) また、`title` 属性を定義して変数設定ボックスのタイトル フィールドに表示されるテキストを指定することも可能。この属性の入力は省略可能	A-39 ページの「クライアント変数設定要求 DTD のサンプル」を参照。

表 A-12 クライアント変数設定要求 DTD 要素 (続き)

要素	説明	サンプル値
variable	設定する変数 以下の属性を定義する。 ■ name — XML 名前トークン（NMTOKEN）フォーマットの、設定する変数の名前。この属性は必須 ■ prompt — XML 名前トークン（NMTOKEN）フォーマットで指定する、ユーザへの @ としてダイアログ ボックスに表示されるテキスト。この属性の入力は省略可能	<variable name="ShipToState" prompt="An error has occurred in processing this order. Please enter the valid US state abbreviation for shipping:"> </variable>

クライアント変数設定要求 DTD のサンプル

クライアント変数設定要求 DTD の有効アプリケーションを次のサンプルで示します。

```
<set-variables title="Error Warning">
  <actionid>991408825931</actionid>
  <variable name="ShipToState" prompt="An error has occurred in
  processing this order. Please enter the valid US state
  abbreviation for shipping:"></variable>
</set-variables>
```

クライアント変数設定応答 DTD

クライアント変数設定応答 DTD は、変数設定ダイアログ ボックスでユーザが追加情報を求められた場合に戻される XML ドキュメントのフォーマットを説明します（ユーザへのプロンプトに用いられる XML ドキュメントのフォーマットについては、「クライアント変数設定要求 DTD」 で説明）。

XPath 関数を用いて、戻り値を抽出できます。たとえば、

```
XPath("/set-variables/child::variable[attribute::name=
      "<var-name>"]/child::text()")
XPath("/set-variables/variable[@name="<var-name>"]/text()")
```

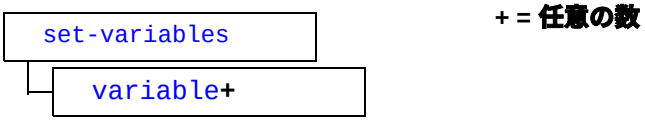
この節では、クライアント変数設定応答 DTD について説明します。内容は以下のとおりです。

- 階層構造ダイアグラム
- DTD フォーマット
- 要素説明
- クライアント変数設定応答 DTD のサンプル

階層構造ダイアグラム

クライアント変数設定応答 DTD 階層構造を次のダイアグラムに示します。

図 A-11　クライアント変数設定応答 DTD 階層構造ダイアグラム



DTD フォーマット

クライアント変数設定応答 DTD のフォーマット ClientSetVarResp.dtd を次のリストに示します。

```
<!ELEMENT set-variables (variable+)>
<!ELEMENT variable (#PCDATA)>
<!ATTLIST variable name NMTOKEN #REQUIRED>
```

要素説明

統計要求 DTD の要素を次の表に示します。

表 A-13 クライアント変数設定応答 DTD 要素

要素	説明	サンプル値
set-variables	ルート要素 任意の数の variable 下位要素を定義する。	A-41 ページの「クライアント変数設定応答 DTD のサンプル」を参照。
variable	変数の値 XML 名前トークン（NMTOKEN）フォーマットで設定された変数の名前を指定する name 属性を定義する。この属性は必須	<variable name="ShipToState"> CA </variable>

クライアント変数設定応答 DTD のサンプル

クライアント変数設定応答 DTD の有効アプリケーションを次のサンプルで示します。

```
<set-variables>
  <variable name="ShipToState">CA</variable>
</set-variables>
```

インポート応答 DTD

インポート応答 DTD は、`com.bea.wlpi.server.admin.Admin` インタフェースへの `importPackage()` メソッドが呼び出された場合に戻される XML ドキュメントのフォーマットを説明します。`importPackage()` メソッドに関する詳細については、18-6 ページの「発行可能オブジェクトのパッケージのインポート」を参照してください。

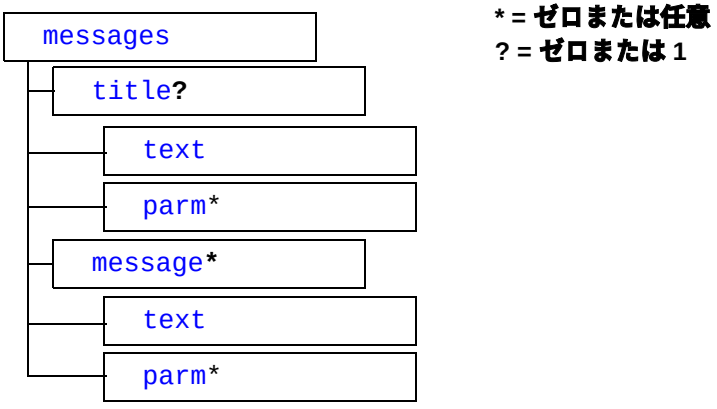
この節では、インポート応答 DTD について説明します。内容は以下のとおりです。

- 階層構造ダイアグラム
- DTD フォーマット
- 要素説明

階層構造ダイアグラム

インポート応答 DTD の階層構造を次のダイアグラムに示します。

図 A-12　インポート応答 DTD 階層構造ダイアグラム



DTD フォーマット

インポート応答 DTD のフォーマット ImportResp.dtd を次のリストに示します。

```
<!ELEMENT messages (title?, message*>
<!ELEMENT title (text, parm*)>
<!-- msg-num は Messages.properties の中のインターナショナライズされた
メッセージ番号を参照する -->
<!ATTLIST text msg-num NUMBER #REQUIRED>
<!-- text 値はサーバのロケールでフォーマットされた、そのままのメッセージ -->
<!ELEMENT text (#PCDATA)>
<!ELEMENT message (text, parm*)>
<!-- 各 parm 要素は置き換え可能なパラメータ値で、
ローカライズされたメッセージの中の該当するパラメータ マーカに挿入する -->
<!ELEMENT parm (#PCDATA)>
<!-- type は Messages.properties の中のインターナショナライズされたオブ
ジェクト タイプを参照する -->
<!ATTLIST parm type NUMBER #IMPLIED>
```

要素説明

インポート応答 DTD の要素を次の表に示します。

表 A-14 インポート応答 DTD 要素

要素	説明
message	単一メッセージ 以下の下位要素を定義する。 ■ text ■ parm (ゼロまたは任意の数)
messages	メッセージのリスト 以下の下位要素を定義する。 ■ title (ゼロまたは 1) ■ message (ゼロまたは任意の数)

表 A-14　インポート応答 DTD 要素 (続き)

要素	説明
parm	ローカライズされたメッセージの対応パラメータ マーカに代わるパラメータ値 次の属性を定義する。 type ―パラメータのタイプ
text	サーバのロケールでフォーマットされたメッセージ 次の属性を定義する。 msg-num ―メッセージ番号
title	メッセージ リストのタイトル 以下の下位要素を定義する。 ■ text ■ parm (ゼロまたは任意の数)

統計要求 DTD

統計要求 DTD は、実行時統計レポートが生成される際に用いられる XML ドキュメントのフォーマットを説明します（戻される統計レポートのフォーマットについては、「統計応答 DTD」で説明）。実行時統計の生成に用いられるメソッドに関する詳細については、22-20 ページの「実行時統計のクエリ」を参照してください。

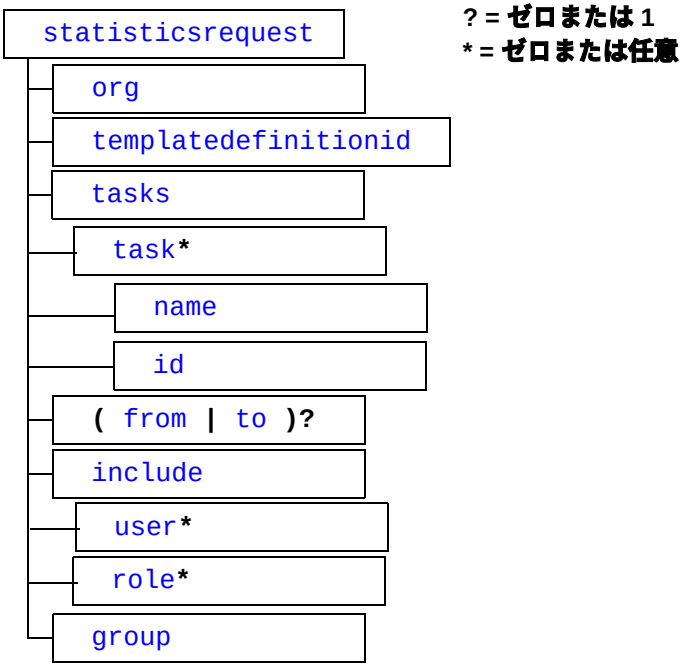
この節では、統計応答 DTD ついて説明します。内容は以下のとおりです。

- 階層構造ダイアグラム
- DTD フォーマット
- 要素説明

階層構造ダイアグラム

統計要求 DTD の階層構造を次のダイアグラムに示します。

図 A-13　統計要求 DTD 階層構造ダイアグラム



DTD フォーマット

統計要求 DTD のフォーマット StatisticsReq.dtd を次に示します。

```
!ELEMENT statisticsrequest (org, templatedefinition,
                            tasks, (from, to)?, include, group)>
<!ELEMENT tasks (task*)>
<!ELEMENT task (name, id)>
<!ELEMENT include (#PCDATA, user* | role*)>
<!ELEMENT org (#PCDATA)>
<!ELEMENT templatedefinitionid (#PCDATA)>
<!ELEMENT name (#PCDATA)>
```

```
<!ELEMENT id (#PCDATA)>
<!ELEMENT from (#PCDATA)>
<!ELEMENT to (#PCDATA)>
<!ELEMENT user (#PCDATA)>
<!ELEMENT role (#PCDATA)>
<!ELEMENT group (#PCDATA)>
```

要素説明

統計要求 DTD の要素を次の表に示します。

表 A-15　統計要求 DTD 要素

要素	説明
from	統計レポートを生成する日付この日付は、次のフォーマットを用いて指定する。yyyyMMddHHmm ―ここで yyyy は年、MM は月、dd は日付、HH は時間、mm は分を表す。
group	合計をグループ化するかどうかを指定するブール フラグ
id	レポートに含めるタスクの ID
include	レポートに含めるユーザやロールのリスト 以下の下位要素を定義する。 ■ user（ゼロまたは任意の数） ■ role（ゼロまたは任意の数）
name	レポートに含めるタスクの名前
org	生成するレポートの対象となるオーガニゼーション
role	レポートに含めるロールの名前
statisticsrequest	ルート要素 以下の下位要素を定義する。 ■ org ■ templatedefinitionid ■ tasks ■ (from, to)（ゼロまたは 1） ■ include ■ group

表 A-15 統計要求 DTD 要素 (続き)

要素	説明
task	レポートに含めるタスク 以下の下位要素を定義する。 ■ name ■ id
tasks	レポートに含めるタスクのリスト 以下の下位要素のゼロまたは任意の数の発生を定義する。task
templatedefinitionid	生成するレポートのテンプレート定義の ID
to	生成する統計レポートの対象となる期間のエンティティ日付この日付は、次のフォーマットを用いて指定する。yyyyMMddHHmm ― ここで yyyy は年、MM は月、dd は日付、HH は時間、mm は分を表す。
user	レポートに含めるユーザの名前

統計応答 DTD

統計応答 DTD は、実行時統計レポートがクエリされた場合に戻される XML ドキュメントのフォーマットを説明します（クエリの要求に用いられる XML ドキュメントのフォーマットについては、「統計要求 DTD」で、説明）。実行時統計の集計に関する詳細については、22-20 ページの「実行時統計のクエリ」を参照してください。

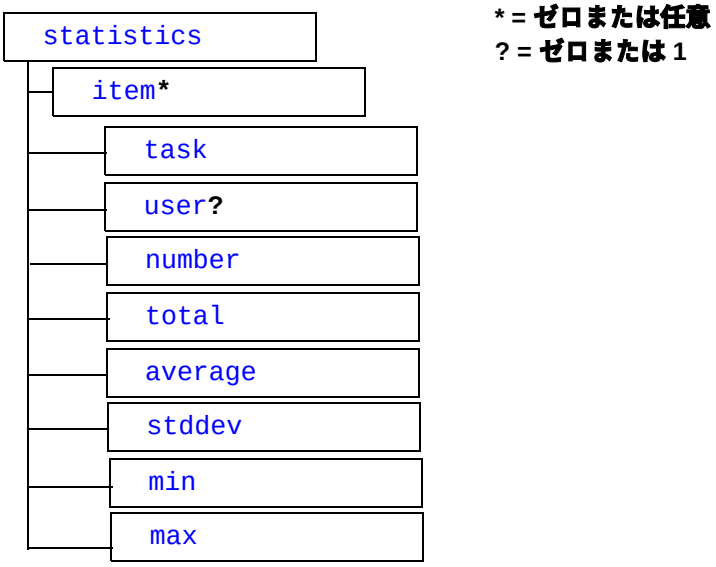
この節では、統計応答 DTD ついて説明します。内容は以下のとおりです。

- 階層構造ダイアグラム
- DTD フォーマット
- 要素説明

階層構造ダイアグラム

統計応答 DTD の階層構造を次のダイアグラムに示します。

図 A-14　統計応答 DTD 階層構造ダイアグラム



DTD フォーマット

統計応答 DTD のフォーマット StatisticsResp.dtd を次に示します。

```
<!ELEMENT statistics (item*)>
<!ELEMENT item (task, user?, number, total, average, stddev,
          min, max)>
<!ELEMENT task (#PCDATA)>
<!ELEMENT user (#PCDATA)>
<!ELEMENT number (#PCDATA)>
<!ELEMENT total (#PCDATA)>
<!ELEMENT average (#PCDATA)>
<!ELEMENT stddev (#PCDATA)>
<!ELEMENT min (#PCDATA)>
<!ELEMENT max (#PCDATA)>
```

要素説明

統計応答 DTD の要素を次の表に示します。

表 A-16 統計応答 DTD 要素

要素	説明
average	タスクの実行にかかった平均時間
item	生成するワークロード レポートの対象となる項目 以下の下位要素を定義する。 ■ task ■ user (ゼロまたは 1) ■ number ■ total ■ average ■ stddev ■ min ■ max
max	タスクの実行にかかった最大時間
min	タスクの実行にかかった最低時間
number	実行されたタスクの数
statistics	ルート要素 下位要素のゼロまたは任意の数の発生を定義する。item
stddev	タスクの実行にかかった時間の標準偏差 対象となる AVERAGE の数値の正確性を示す。
task	次の算出対象となるタスク
total	タスクの実行にかかった合計時間
user	タスクを実行したユーザ

テンプレート DTD

テンプレート DTD は、ワークフロー テンプレート作成に用いられる XML ドキュメントのフォーマットを説明します。ワークフロー テンプレートの作成に関する詳細については、13-1 ページの「ワークフロー テンプレートの作成および管理」を参照してください。

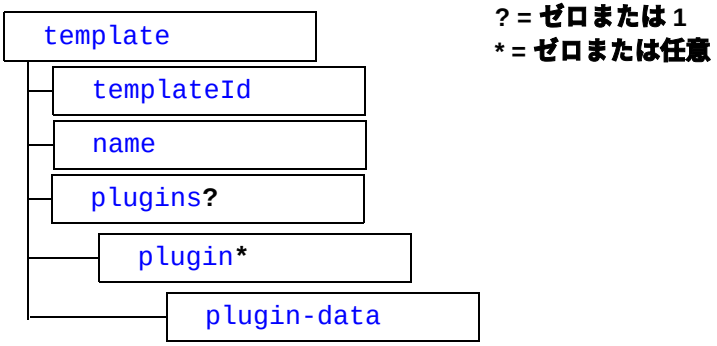
この節では、テンプレート DTD について説明します。内容は以下のとおりです。

- 階層構造ダイアグラム
- DTD フォーマット
- 要素説明

階層構造ダイアグラム

テンプレート DTD の階層構造を次のダイアグラムに示します。

図 A-15　テンプレート DTD 階層構造ダイアグラム



DTD フォーマット

テンプレート DTD のフォーマット Template.dtd を次のリストに示します。

```
<!ELEMENT template (templateId, name, plugins?)>
<!ELEMENT plugins (plugin*)>
<!ELEMENT plugin (plugin-data)>
<!ATTLIST plugin
  name CDATA #REQUIRED
  major-version CDATA #REQUIRED
  minor-version CDATA #REQUIRED
  vendor CDATA #REQUIRED
  url CDATA>
<!ELEMENT plugin-data ANY>
<!ATTLIST plugin-data
    name CDATA #REQUIRED
    id CDATA #REQUIRED
>
<!ELEMENT templateId #PCDATA>
<!ELEMENT name #PCDATA>
```

要素説明

テンプレート DTD の要素を次の表に示します。

表 A-17 テンプレート DTD 要素

要素	説明
name	テンプレートの名前

表 A-17 テンプレート DTD 要素 (続き)

要素	説明
plugin	プラグイン エントリテンプレートが参照する各プラグインおよび独自のテンプレート プロパティを定義する各プラグインに対し、1 つの要素が表示される。 〔省略可能〕 下位要素を定義する。plugin-data. 以下の属性を定義する。 ■ name —プラグインの名前 ■ major-version —プラグイン ソフトウェアのメジャーバージョン ■ minor-version —プラグイン ソフトウェアのマイナーバージョン ■ vendor —ベンダ名 ■ url — URL (デフォルトは空の文字列) プラグインのプログラミングの詳細は、『*WebLogic Integration BPM プラグイン プログラミング ガイド*』 を参照。
plugin-data	要素とテキストの組み合わせから成るプラグイン データ 以下の属性を定義する。 ■ name —プラグインの名前。この属性は必須 ■ ID —割り当てられた値オブジェクトに対する、プラグインによって与えられる固有 ID。この属性は必須 プラグインのプログラミングの詳細は、『*WebLogic Integration BPM プラグイン プログラミング ガイド*』 を参照。
plugins	プラグインが提供される場合にのみ用いられるプラグインの定義 以下の下位要素の、ゼロまたは任意の数の発生を定義する。plugin プラグインのプログラミングの詳細は、『*WebLogic Integration BPM プラグイン プログラミング ガイド*』 を参照。

表 A-17 テンプレート DTD 要素 (続き)

要素	説明
template	ルート要素 以下の下位要素を定義する。 ■ templateId ■ name ■ plugins (ゼロまたは 1)
templateId	テンプレートの ID

テンプレート定義 DTD

テンプレート定義 DTD は、ワークフロー テンプレート定義作成に用いられる XML ドキュメントのフォーマットを説明します。ワークフロー テンプレート定義の作成に関する詳細については、14-1 ページの「ワークフロー テンプレート定義の作成および管理」を参照してください。

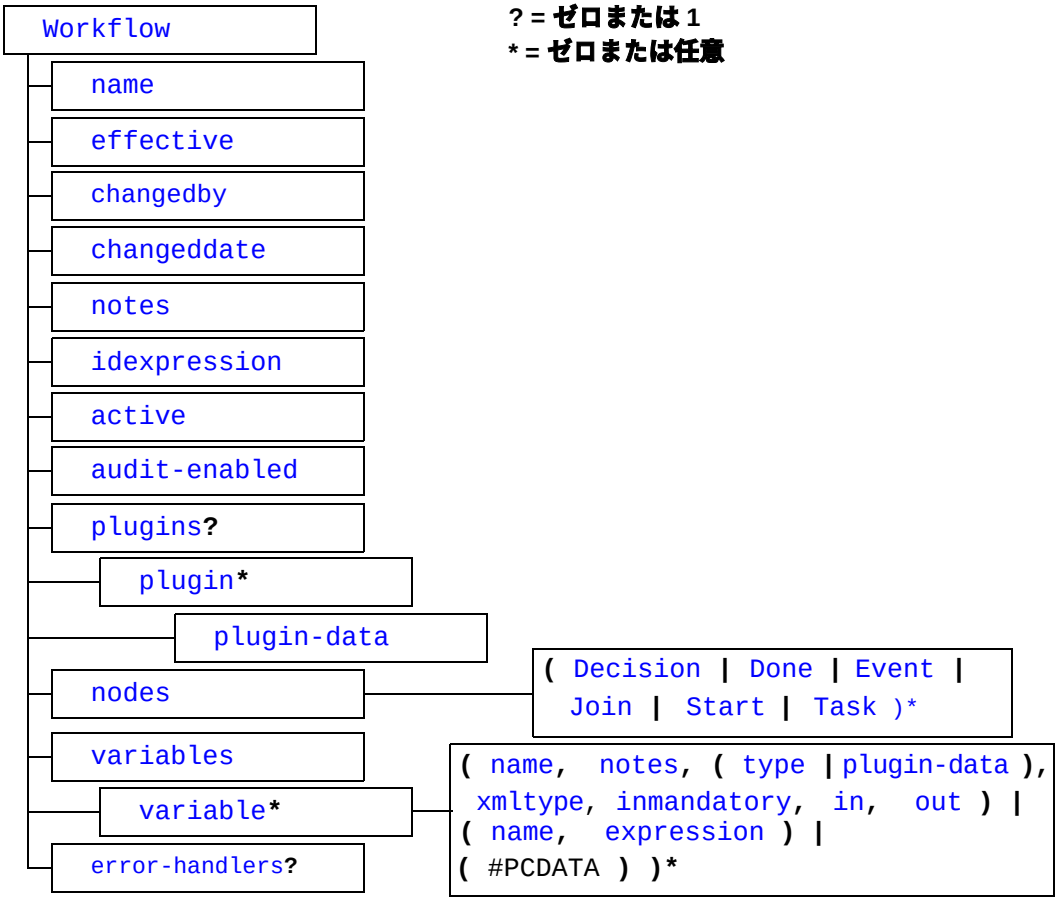
この節では、テンプレート定義 DTD について説明します。内容は以下のとおりです。

- 階層構造ダイアグラム
- DTD フォーマット
- 要素説明
- エンティティ説明
- テンプレート定義 DTD サンプル

階層構造ダイアグラム

テンプレート定義 DTD の階層構造を次のダイアグラムに示します。

図 A-16 テンプレート定義 DTD 階層構造ダイアグラム



DTD フォーマット

テンプレート定義 DTD のフォーマット TemplateDefinition.dtd を次のリストに示します。

注意： DTD 要素の説明は、A-62 ページの「テンプレート定義 DTD 要素」の表に示します。DTD エンティティの説明は、A-113 ページの「テンプレート定義 DTD エンティティ」の表に示します。

```
<!ELEMENT Workflow (name, effective, changedby, changeddate, notes,
                    idexpression, active, audit-enabled, plugins?, nodes,
                    variables, error-handlers?)>
<!ATTLIST Workflow
    major-version   CDATA  "1"
    minor-version   CDATA  "0"
    build-number    CDATA  "0"
>

<!ENTITY % node "id, x, y, notes">

<!ENTITY % std-action-types "
    ActionAuditEntry |
    ActionBusinessOperation |
    ActionCallProgram |
    ActionCancelEvent |
    ActionCondition |
    ActionExpression |
    ActionNoOp |
    ActionPostXMLEvent |
    ActionPlugin |
    ActionSendEMail |
    ActionSendXMLToClient |
    ActionTaskAssignRole |
    ActionTaskAssignRoutingTable |
    ActionTaskAssignUser |
    ActionTaskDoit |
    ActionTaskDone |
    ActionTaskDueDate |
    ActionTaskSetComment |
    ActionTaskSetPriority |
    ActionTaskUnassign |
    ActionTaskUndo |
    ActionTimedEvent |
    ActionWorkflowAbort |
    ActionWorkflowDone |
    ActionWorkflowSetComment |
```

```
    ActionWorkflowStart |
    ActionXSLTransform
">

<!ENTITY % action-types "%std-action-types; | ActionInvokeErrorHandler |
                         ActionSetErrorHandler">

<!ENTITY % commit-action-types "%std-action-types; | ActionExitErrorHandler |
                                ActionSetErrorHandler">

<!ENTITY % rollback-action-types "ActionAuditEntry | ActionBusinessOperation |
                                  ActionCallProgram | ActionCondition |
                                  ActionExitErrorHandler |
                                  ActionExpression | ActionNoOp | ActionPlugin |
                                  ActionPostXMLEvent | ActionSendEMail |
                                  ActionSendXMLToClient">

<!-- エレメント 'actions' のあいまいな定義 -->
<!-- コンテキスト 'Task' では、要素 actions は次のとおり :
     <!ELEMENT actions (created, activated, executed, markeddone)> -->
<!-- コンテキスト 'Decision'、'ActionCondition' では、要素 actions は次のとおり :
     <!ELEMENT actions (false, true)> -->
<!-- コンテキスト 'Done'、'Event'、'Start'、'ActionTaskDueDate'、
'ActionTimedEvent'、'ActionWorkflowStart' では、
     要素 actions は次のとおり :
     <!ELEMENT actions (%action-types;)*> -->

<!ELEMENT actions ((created, activated, executed, markeddone)
                 | (false, true)
                 | (%action-types;)*)>
<!ELEMENT ActionAuditEntry (notes, audittext)>
<!ELEMENT ActionBusinessOperation (notes, descriptorid, description,
                                   result, result-type, instance-variable,
                                   instance-variable-type, parms,
                                   iviewx?, iviewy?)>
<!ELEMENT ActionCallProgram (notes, program, arguments)>
<!ELEMENT ActionCancelEvent (notes, target)>
<!ELEMENT ActionCondition (notes, condition, actions)>

<!ELEMENT ActionExitErrorHandler (notes)>
<!ATTLIST ActionExitErrorHandler exit-type (rollback|stop|retry|continue)
                                           "rollback">

<!ELEMENT ActionExpression (notes, variable, (expression | xml)>

<!ELEMENT ActionInvokeErrorHandler (notes, xml?)>
<!ATTLIST ActionInvokeErrorHandler error-handler-name CDATA #REQUIRED>
```

```
<!ELEMENT ActionNoOp (notes, description)>


<!ELEMENT ActionPlugin (id, notes, plugin-data, actions*)>
<!-- プラグインが利用できない場合は、アクション リストに説明が表示される -->
<!ATTLIST ActionPlugin description CDATA #REQUIRED>

<!ELEMENT ActionPostXMLEvent (notes, (topic | queue), transaction, addressees?,
                              message-properties?, delivery-mode,
                              priority, time-to-live, orderkey?
                             (variable | xml)),
                             iviewx?, iviewy?>
<!ELEMENT ActionSendEmail (notes, subject, message, to, cc, bcc)>
<!ELEMENT ActionSendXMLToClient (id, notes, xml, variables. iviewx?, iviewy?)>

<!ELEMENT ActionSetErrorHandler (notes)>
<!ATTLIST ActionSetErrorHandler error-handler-name CDATA #REQUIRED>

<!ELEMENT ActionTaskAssignRole (notes, target, role, expression)>
<!ELEMENT ActionTaskAssignRoutingTable (notes, target, routeconditions)>
<!ELEMENT ActionTaskAssignUser (notes, target, user, expression, role)>
<!ELEMENT ActionTaskDoIt (notes, target)>
<!ELEMENT ActionTaskDone (notes, target)>
<!ELEMENT ActionTaskDueDate (id, notes, target, expression, calendartype,
                             calendarname, actions)>
<!ELEMENT ActionTaskSetComment (notes, target, comment)>
<!ELEMENT ActionTaskSetPriority (notes, target, priority)>
<!ELEMENT ActionTaskUnassign (notes, target)>
<!ELEMENT ActionTaskUndo (notes, target)>
<!ELEMENT ActionTimedEvent (id, notes, expression, calendartype, executionunits,
                         scheduleunits, executiontime, scheduletime, recoverable?
                            stopwhentaskdone, usetimeexpression, actions)>
<!ELEMENT ActionWorkflowAbort (notes)>
<!ELEMENT ActionWorkflowDone (notes)>
<!ELEMENT ActionWorkflowSetComment (notes, comment)>
<!ELEMENT ActionWorkflowStart (id, notes, templateid, template-name, refvariable,
                               orgexpression, parameters, results, actions,
                               iviewx?, iviewy?)>
<!ELEMENT ActionXSLTransform (notes, input-expression, transform-document,
                              transform-source, output-variable,
                              xsl-parameters?)>
<!ELEMENT xsl-parameters (xsl-parameter+)>
<!ELEMENT xsl-parameter (name, expression)>
<!ELEMENT activated ((%action-types;)*)>
<!ELEMENT addressee (name, expression, type)>
<!ELEMENT addressees (instanceid)
<!ELEMENT bcc (addressee*)>
<!ELEMENT cc (addressee*)>
```

```
<!ELEMENT commit-actions ((%commit-action-types;)*)>
<!ELEMENT created ((%action-types;)*)>
<!ELEMENT Decision (%node;, condition, truenexts, falsenexts, actions)>
<!ELEMENT Done (%node;, actions, plugin-data?)>


<!ELEMENT Event (%node;, ((root, description, key, condition) | (plugin-data)),
                 actions, variables, ivewx?, iviewy?)>
<!ELEMENT error-handlers (error-handler*)>
<!ELEMENT error-handler (notes, variables, commit-actions, rollback-actions)>
<!-- エラーハンドラは 1 つだけ初期設定で true となっている -->
<!ATTLIST error-handler
    name    CDATA        #REQUIRED
    initial (true|false) "false"
>

<!ELEMENT executed ((%action-types;)*)>
<!ELEMENT false ((%action-types;)*)>
<!ELEMENT falsenexts (next*)>
<!ELEMENT Join (%node;, and)>
<!ELEMENT markeddone ((%action-types;)*)>
<!ELEMENT message-properties (property+)>
<!ELEMENT nexts (next*)>
<!ELEMENT nodes (Decision | Done | Event | Join | Start | Task)*>
<!ELEMENT parameters (parm*)>

<!-- エレメント 'parm' のあいまいな定義 -->
<!-- コンテキスト 'ActionBusinessOperation' では、要素 parm は次のとおり <!ELEMENT parm
(#PCDATA)> -->
<!-- コンテキスト 'ActionWorkflowStart' では、要素 parm は次のとおり <!ELEMENT parm
(name, value)> -->
<!-- N.B. コンテンツ モデルでは、繰り返し可能な要素は
     完全な DTD 構文準拠であることを示す -->
<!-- <parm> 要素は、上で例示した 2 つの形式でのみ表示される -->
<!ELEMENT parm ((#PCDATA | (name, value))*)>
<!ELEMENT parms (parm*)>

<!-- plungins エレメントは、テンプレート定義がプラグイン機能を使用する場合、
     または独自のテンプレート定義プロパティを定義するプラグインがある場合に
     表示される -->
<!-- Studio はこの定義を保存するときに、
     この要素を更新する -->
<!ELEMENT plugins (plugin*)>
<!-- テンプレート定義より参照される各プラグインおよび独自のテンプレートを
     定義する各プラグインに対して 1 つの
    plugin 要素が表示される -->
<!ELEMENT plugin (plugin-data?)>
```

```
<!-- name はプラグイン名で、
     referenced はこのテンプレート定義がこのプラグイン固有の機能を使用しているかどうかを示し、
     Version はプラグインのバージョン番号である -->
<!ATTLIST plugin
    name CDATA #REQUIRED
    referenced (true|false) #REQUIRED
    major-version CDATA #REQUIRED
    minor-version CDATA #REQUIRED
    vendor CDATA #REQUIRED
    url CDATA ""


>
<!ELEMENT property (name, value)>
<!-- 要素 result のあいまいな定義 -->
<!-- コンテキスト 'ActionBusinessOperation' では、要素 result は次のとおり <!ELEMENT
result (#PCDATA)> -->
<!-- コンテキスト 'ActionWorkflowStart' では､要素 result は次のとおり <!ELEMENT result
(name, value)> -->
<!ELEMENT result ((#PCDATA | (name, value))*)>
<!ELEMENT results (result*)>
<!ELEMENT rollback-actions ((%rollback-action-types;)*)>
<!ELEMENT routecondition (to, type, condition)>
<!ELEMENT routeconditions (routecondition*)>
<!ELEMENT Start (%node;, description, (manual
                                      | called
                                      | (timed, dateexpression,
                                         reschedamount, reschedunits,
                                         recoverable?, calendarname, startorg)
                                      | (event, root, key, condition, startorg)
                                      | (plugin-data, startorg)),
                 nexts, actions, variables,iviewx?, iviewy?)>
<!ELEMENT plugin-data ANY>
<!-- name はプラグイン名、
     id はプラグインが xxxInfo オブジェクトに与えたユニークな ID -->
<!ATTLIST plugin-data
    name CDATA #REQUIRED
    ID CDATA #REQUIRED
>
<!ELEMENT Task (%node;, name, priority, donewithoutdoit, doitifdone,
                unmarkdone, modifiable, reassignment, nexts, actions)>
<!ELEMENT to (addressee*)>
<!ELEMENT true ((%action-types;)*)>
<!ELEMENT truenexts (next*)>

<!-- 要素 variable のあいまいな定義 -->
<!-- コンテキスト 'Workflow' では、要素 variable は次のとおり :
     <!ELEMENT variable (name, notes, (type|plugin-data), xmltype,
```

```
                          inmandatory, in, out)> -->
<!-- コンテキスト 'Start'、'ActionSendXMLToClient' では、要素 variable は次のとおり :
     <!ELEMENT variable (name, expression)> -->
<!-- コンテキスト 'ActionExpression' では、要素 variable は次のとおり :
     <!ELEMENT variable (#PCDATA)> -->
<!-- N.B. コンテンツ モデルでは、繰り返し可能な要素は
     完全な DTD 構文準拠であることを示す -->
<!-- <variable> 要素は、上で例示した 3 つの形式でのみ表示される -->
<!ELEMENT variable (((name, notes, (type|plugin-data), xmltype, inmandatory,
                       in, out)
                   |  (name, expression)
                   |  (#PCDATA))*)>
<!--ELEMENT variable (#PCDATA)-->
<!ELEMENT variables (variable*)>


<!ELEMENT active (#PCDATA)>
<!ELEMENT and (#PCDATA)>
<!ELEMENT arguments (#PCDATA)>
<!ELEMENT audit-enabled (#PCDATA)>
<!ELEMENT audittext (#PCDATA)>
<!ELEMENT calendarname (#PCDATA)>
<!ELEMENT calendartype (#PCDATA)>
<!ELEMENT called (#PCDATA)>
<!ELEMENT changedby (#PCDATA)>
<!ELEMENT changeddate (#PCDATA)>
<!ELEMENT comment (#PCDATA)>
<!ELEMENT condition (#PCDATA)>
<!ELEMENT correlationid (#PCDATA)>
<!ELEMENT dateexpression (#PCDATA)>
<!ElEMENT delivery-mode (#PC-DATA)
<!ELEMENT description (#PCDATA)>
<!ELEMENT descriptorid (#PCDATA)>
<!ELEMENT doitifdone (#PCDATA)>
<!ELEMENT donewithoutdoit (#PCDATA)>
<!ELEMENT effective (#PCDATA)>
<!ELEMENT event (#PCDATA)>
<!ELEMENT executionunits (#PCDATA)>
<!ELEMENT executiontime (#PCDATA)>
<!ELEMENT expression (#PCDATA)>
<!ELEMENT id (#PCDATA)>
<!ELEMENT idexpression (#PCDATA)>
<!ELEMENT in (#PCDATA)>
<!ELEMENT inmandatory (#PCDATA)>
<!ELEMENT input-expression (#PCDATA)
<!ELEMENT instanceid (#PCDATA)>
<!ELEMENT instance-variable (#PCDATA)>
```

```
<!ELEMENT instance-variable-type (#PCDATA)>
<!ELEMENT iviewx (#PCDATA)
<!ELEMENT iviewy (#PCDATA)
<!ELEMENT jmstype (#PCDATA)
<!ELEMENT key (#PCDATA)>
<!ELEMENT manual (#PCDATA)>
<!ELEMENT message (#PCDATA)>
<!ELEMENT modifiable (#PCDATA)>
<!ELEMENT name (#PCDATA)>
<!ELEMENT next (#PCDATA)>
<!ELEMENT notes (#PCDATA)>
<!ELEMENT orderkey (#PCDATA)>
<!ELEMENT orgexpresion (#PCDATA)>
<!ELEMENT out (#PCDATA)>
<!ELEMENT output-variable (#PCDATA)
<!ELEMENT priority (#PCDATA)>
<!ELEMENT program (#PCDATA)>
<!ELEMENT queue (#PCDATA)>


<!ATTLIST queue expression (true | false) "false";
<!ELEMENT reassignment (#PCDATA)>
<!ELEMENT recoverable (#PCDATA)>
<!ELEMENT refvariable (#PCDATA)>
<!ELEMENT replyto (#PCDATA)
<!ELEMENT reschedamount (#PCDATA)>
<!ELEMENT reschedunits (#PCDATA)>
<!ELEMENT result-type (#PCDATA)>
<!ELEMENT role (#PCDATA)>
<!ELEMENT root (#PCDATA)>
<!ELEMENT scheduletime (#PCDATA)>
<!ELEMENT scheduleunits (#PCDATA)>
<!ELEMENT startorg (#PCDATA)>
<!ATTLIST startorg expression (true | false) "true"
<!ELEMENT stopwhentaskdone (#PCDATA)>
<!ELEMENT subject (#PCDATA)>
<!ELEMENT target (#PCDATA)>
<!ELEMENT templateid (#PCDATA)>
<!ELEMENT template-name (#PCDATA)>
<!ELEMENT time-to-live (#PCDATA)>
<!ELEMENT timed (#PCDATA)>
<!ELEMENT topic (#PCDATA)>
<!ATTLIST topic expression (true | false) "false")
<!ELEMENT type (#PCDATA)>
<!-- 'transaction' 要素は 'true' または 'false' -->
<!ELEMENT transaction (#PCDATA)
<!ELEMENT transform-document (#PCDATA)
<!ELEMENT transform-source (#PCDATA)
<!ELEMENT unmarkdone (#PCDATA)>
```

```
<!ELEMENT user (#PCDATA)>
<!ELEMENT usetimeexpression (#PCDATA)>
<!ELEMENT value (#PCDATA)>
<!ELEMENT x (#PCDATA)>
<!ELEMENT xml ANY>
<!ELEMENT xmltype (#PCDATA)>
<!ELEMENT y (#PCDATA)>
```

要素説明

テンプレート定義 DTD の要素を次の表に示します。

表 A-18 テンプレート定義 DTD 要素

要素	説明	サンプル値
ActionAuditEntry	主要なインタラクションや変更を含むデフォルトの監査エントリに加え、JMS wlpiAudit トピックに発行される監査エントリを定義するためのアクション Studio では、[アクションを追加] ダイアログ ボックスの [その他のアクション] カテゴリ内で [監査エントリを作成] を選択することでこのアクションを定義する。 以下の下位要素を定義する。 ■ notes ■ audittext	<ActionAuditEntry> <notes></notes> <audittext> "Workflow cancelled." </audittext> </ActionAuditEntry>

表 A-18 テンプレート定義 DTD 要素 (続き)

要素	説明	サンプル値
ActionBusinessOperation	EJB または Java クラスで定義済みのビジネス オペレーションを呼び出すためのアクションビジネス オペレーション呼び出しの結果はワークフロー変数に割り当てることが可能。 Studio では、[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] を選択することでビジネス オペレーションをコンフィグレーションする。ビジネス オペレーションは、アクション定義時に [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] カテゴリ内で [ビジネス オペレーションを実行] を選択することで呼び出す。 以下の下位要素を定義する。 ■ notes ■ descriptorid ■ description ■ result ■ result-type ■ instance-variable ■ instance-variable-type ■ parms ■ iviewx (ゼロまたは 1) ■ iviewy (ゼロまたは 1)	<ActionBusinessOperation> <notes> This is a note. </notes> <descriptorid> 1 </descriptorid> <description> Check Inventory </description> <result> Inventory </result> <result-type> integer </result-type> <instance-variable> PobeanHandle </instance-variable> <instance-variable-type> session </instance-variable-type> <parms> <parm> ItemNumber </parm> </parms> <iviewx>150</iviewx> <iviewy>70</iviewy> </ActionBusinessOperation>

表 A-18 テンプレート定義 DTD 要素 (続き)

要素	説明	サンプル値
ActionCallProgram	定義済みの実行プログラムを呼び出すためのアクションプログラム呼び出しの結果はワークフロー変数に割り当てることが可能。 Studio では、[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] カテゴリ内で [プログラムの呼び出し] を選択することでこのアクションを定義する。 以下の下位要素を定義する。 ■ notes ■ program ■ arguments	<ActionCallProgram> <notes> This is a note. </notes> <program> ProgramA </program> <arguments> CurrentUser() </arguments> </ActionCallProgram>
ActionCancelEvent	ワークフロー内で定義されたワークフロー イベントをキャンセルするためのアクション Studio では、[アクションを追加] ダイアログ ボックスの [その他のアクション] カテゴリ内で [ワークフロー イベントを取消し] を選択することでこのアクションを定義する。 以下の下位要素を定義する。 ■ notes ■ target	<ActionCancelEvent> <notes> This is a note. </notes> <target> 987445252339 </target> </ActionCancelEvent>

表 A-18 テンプレート定義 DTD 要素 (続き)

要素	説明	サンプル値
ActionCondition	実行時に条件式を評価し、その結果により別のサブアクション シーケンスを実行するためのアクション Studio では、[アクションを追加] ダイアログ ボックスの [その他のアクション] カテゴリ内で [条件を評価] を選択することでこのアクションを定義する。 以下の下位要素を定義する。 ■ notes ■ condition ■ actions	<ActionCondition> <notes> This is a note. </notes> <condition> CurrentUser()= "joe" </condition> <actions> <ActionTaskUnassign> <notes> This is a note. </notes> <target> 963511923442 </target> </ActionTaskUnassign> </false> <true> <ActionTaskDone> <notes> This is a note. </notes> <target> 963511923442 </target> </ActionTaskDone> </actions> </ActionCondition>

表 A-18 テンプレート定義 DTD 要素 (続き)

要素	説明	サンプル値
ActionExitErrorHandler	例外ハンドラを終了するためのアクション Studio では、[例外ハンドラを編集] ダイアログ ボックスを用いてこのアクションを定義する。このダイアログ ボックスには、[例外ハンドラのプロパティ] ダイアログ ボックスで [コミット時アクション] または [ロールバック時アクション] タブが選択されている状態で、[**追加**] ボタンをクリックしてアクセスできる。 以下の下位要素を定義する。notes exit-type 属性に対して以下のいずれかの値を指定し、エラー ハンドラ終了時に用いられるメソッドを指定する。 ■ rollback －ユーザ トランザクションをロールバックのみとしてマークし、ワークフローを、トランザクション開始前の状態に戻す。例外は、Workflow Exception としてクライアントに伝播される。ロールバック パスでは、これが唯一の選択可能オプションである。一部の EJB は、UserTransaction メソッドを呼び出してロールバックのみのトランザクションにマークを付けることができ、またコンテナ境界で非検査の例外を返すことができる。どちらの方法でも、WebLogic Server によってトランザクションがロールバックされます。 ■ stop －エラー ハンドラを終了し、後続のアクションの実行を取り止める。 ■ retry －エラー ハンドラを終了し、失敗したオペレーションの再試行を試みる（失敗したオペレーションとは、アクションや変数の設定を指す）。 ■ continue －例外ハンドラの実行を停止し、次のオペレーションでワークフローの実行の継続を試みる（次のオペレーションとは、実行される次のアクションまたは変数設定を指す）。	<ActionExitErrorHandler exit-type="rollback"> <notes> This is a note. </notes> </ActinoExitErrorHandler>

表 A-18 テンプレート定義 DTD 要素 (続き)

要素	説明	サンプル値
ActionExpression	ワークフロー変数の設定のためのアクション Studio では、[アクションを追加] ダイアログ ボックスの [ワークフロー アクション] カテゴリ内で [ワークフロー変数を設定] を選択することでこのアクションを定義する。 以下の下位要素を定義する。 ■ notes ■ variable ■ (expression | xml)	<ActionExpression> <notes> This is a note. </notes> <variable> CustomerState </variable> <expression> "NJ" </expression> </ActionExpression>
ActionInvokeErrorHandler	指定した例外ハンドラの実施を有効にするためのアクションこのアクションを実行すると、WebLogic Process Integrator はユーザ定義 XML ドキュメントを例外プロセッサに送り、指定したハンドラを呼び出す。 Studio では、[アクションを追加] ダイアログ ボックスの [例外処理アクション] カテゴリ内で [例外ハンドラの呼び出し] を選択することでこのアクションを定義する。 以下の下位要素を定義する。 ■ notes ■ xml (ゼロまたは 1) error-handler-name 属性に対し、有効な例外ハンドラの名前を指定するテキスト文字列を指定する。 この属性は必須	<ActionInvokeErrorHandler error-handler-name= "EH Wrong data to Pobean"> <notes> This is a note. </notes> </ActionInvokeErrorHandler>

表 A-18 テンプレート定義 DTD 要素 (続き)

要素	説明	サンプル値
ActionNoOp	将来のアクションのプレースホルダとして用いられるアクションこのアクションは、機能を実行するものではない。 Studio では、[アクションを追加] ダイアログ ボックスの [その他のアクション] カテゴリ内で [処理なし] を選択することでこのアクションを定義する。 以下の下位要素を定義する。 ■ notes ■ description	<ActionNoOp> <notes> This is a note. </notes> <description> Replace with task assignment. </description> </ActionNoOp>
ActionPlugin	すべてのプラグイン アクションに対する汎用的なコンテナ 以下の下位要素を定義する。 ■ id ■ plugin-data ■ notes ■ actions (ゼロまたは任意の数) label 属性に対し、プラグイン名の説明を示すテキスト文字列を指定する。この属性は必須 **注意 :** プラグインが選択可の場合、Studio ではアクション リストにその説明が表示されます。 プラグインのプログラミングの詳細は、『*WebLogic Integration BPM プラグイン プログラミング ガイド*』を参照。	<ActionPlugin label='Send Confirm Order Event'> <id>1000936287350 </id> <plugin-data name="com.bea.wlpi.Sam plePlugin" ID="2"> <statusvariable> OrderStatus </statusvariable> <totalpricevariable> OrderTotalPrice </totalpricevariable> </plugin-data> <notes></notes> </ActionPlugin>

表 A-18 テンプレート定義 DTD 要素 (続き)

要素	説明	サンプル値
ActionPostXMLEvent	XML イベントのポスティングに用いられるアクション Studio では、[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] カテゴリ内で [XML イベントをポスト] を選択することでこのアクションを定義する。 以下の下位要素を定義する。 ■ notes ■ topic | queue ■ transaction ■ addressees (ゼロまたは 1) ■ message-properties (ゼロまたは 1) ■ delivery-mode ■ priority ■ time-to-live ■ orderkey (ゼロまたは 1) ■ variable | xml ■ iviewx (ゼロまたは 1) ■ iviewy (ゼロまたは 1)	<ActionPostXMLEvent> <notes> This is a note. </notes> <transaction> false </transaction> <delivery-mode> PERSISTENT </delivery-mode> <priority>4 </priority> <expression>false </expression> <time-to-live>0 </time-to-live> <xml> <order> "new" </order> </xml> <iviewx>240</iviewx> <iviewy>20</iviewy> </ActionPostXMLEvent>

表 A-18　テンプレート定義 DTD 要素 (続き)

要素	説明	サンプル値

actions	アクションの定義
	Task ノードに対し、以下の下位要素を定義する。
	■ created
	■ activated
	■ executed
	■ markeddone
	Decision ノードまたは ActionCondition アクションに対し、以下の下位要素を定義する。
	■ false
	■ true
	Done、Event あるいは Start ノード、あるいは ActionTaskDueDate、ActionTimedEvent あるいは ActionWorkflowStart イベントに対し、下位要素の、ゼロまたは任意の数の発生を定義することが可能。
	%action-types
		<actions>
		  <created>
		  </created>
		  <activated>
		<ActionTaskAssignUser>
		  <notes>
		  This is a note.
		  </notes>
		  <target>
		  2
		  </target>
		  <user>joe</user>
		  <expression>
		  false
		  </expression>
		  <role>
		  false
		  </role>
		</ActionTaskAssignUser>
		  </activated>
		  <executed>
		  </executed>
		  <markeddone>
		  </markeddone>
		</actions>

表 A-18 テンプレート定義 DTD 要素 (続き)

要素	説明	サンプル値
ActionSendEMail	WebLogic Process Integrator ユーザまたはその他の個人に電子メール メッセージを送信するためのアクション メッセージの送信には、内部標準 SMTP プロトコルが用いられる。 Studio では、[アクションを追加] ダイアログ ボックスの [その他のアクション] カテゴリ内で [電子メール メッセージを送信] を選択することでこのアクションを定義する。 以下の下位要素を定義する。 ■ notes ■ subject ■ message ■ to ■ cc ■ bcc	<ActionSendEMail> <notes> This is a note. </notes> <subject> "Order " </subject> <message> "This is the message. " </message> <to> <addressee> <name> joe@work </name> <expression> false </expression> <type> address </type> </addressee> </to> <cc> </cc> <bcc> </bcc> </ActionSendEMail>

表 A-18　テンプレート定義 DTD 要素 (続き)

要素	説明	サンプル値
ActionSendXMLToClient	クライアントに XML ドキュメントを送信するためのアクション。このアクションにより、WebLogic Process Integrator とクライアント プログラム間での XML メッセージを介した通信が可能となる。 Studio では、[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] カテゴリ内で [XML をクライアントに送信] を選択することでこのアクションを定義する。 XML ドキュメントは、送信される要求に応じて、以下の DTD フォーマットのいずれかに準拠する必要がある。 ■ A-19 ページの「クライアント呼び出しアドイン要求 DTD」 ■ A-24 ページの「クライアント呼び出しプログラム要求 DTD」 ■ A-28 ページの「クライアント メッセージ ボックス要求 DTD」 ■ A-37 ページの「クライアント変数設定要求 DTD」 カスタム クライアントは、独自のサービスおよび XML フォーマットを定義できる。 以下の下位要素を定義する。 ■ id ■ notes ■ xml ■ variables ■ iviewx (ゼロまたは 1) ■ iviewy (ゼロまたは 1)	<ActionSendXMLToClient> <id>959395846210</id> <notes>This is a note. </notes> <xml> <message-box title=""Order Confirmation"" style=""question "" options=""yes_no""> "Do you Confirm this order?" <actionid>"959395846210 "</actionid> </message-box> </xml> <variables> <variable> <name>Confirm</name> <expression>XPath("/message-box/attribute::option")</expression> </variable> </variables> <actions> <ActionTaskDone> <notes>This is a note.</notes> <target>2</target> </ActionTaskDone> </actions> </ActionSendXMLToClient>

表 A-18 テンプレート定義 DTD 要素 (続き)

要素	説明	サンプル値
ActionSetErrorHandler	特定のワークフロー テンプレート定義に対しアクティブであるとして指定した例外ハンドラの設定に用いられるアクションこの例外ハンドラは、後続の ActionSetErrorHandler アクションが同じワークフロー内で実行されるまで、このワークフロー インスタンスに対して存続する。デフォルトでは、システム例外ハンドラが用いられる。 Studio では、[アクションを追加] ダイアログ ボックスの [例外処理アクション] カテゴリ内で [ワークフロー例外ハンドラを設定] を選択することでこのアクションを定義する。 以下の下位要素を定義する。notes. error-handler-name 属性に対し、有効な例外ハンドラの名前を指定するテキスト文字列を指定する。この属性は必須	<ActionSetErrorHandler error-handler-name= "EH Wrong data to Pobean"> <notes> This is a note. </notes> </ActionSetErrorHandler>
ActionTaskAssignRole	ロールにタスクを割り当てるためのアクション Studio では、[アクションを追加] ダイアログ ボックスの [タスク アクション] カテゴリ内で [ロールにタスクを割り当て] を選択することでこのアクションを定義する。 以下の下位要素を定義する。 ■ notes ■ target ■ role ■ expression	<ActionTaskAssignRole> <notes> This is a note. </notes> <target> 963511923442 </target> <role> Role1 </role> <expression> false </expression> </ActionTaskAssignRole>

表 A-18 テンプレート定義 DTD 要素 (続き)

要素	説明	サンプル値
ActionTaskAssignRoutingTable	一連の条件に従ってユーザまたはロールにタスクを割り当てるためのアクション条件は、タスクが特定の割り当て先（ユーザまたはロール）に割り当てられなければならない状況を指定する。実行時、システムは true と評価される条件に対応する割り当て先を選択する。 Studio では、[アクションを追加] ダイアログ ボックスの [タスク アクション] カテゴリ内で [ロールにタスクを割り当て] を選択することでこのアクションを定義する。 以下の下位要素を定義する。 ■ notes ■ target ■ routeconditions	<ActionTaskAssignRoutingTable> <notes> This is a note. </notes> <target>2</target> <routeconditions> <routecondition> <to>Role1</to> <type> userinrole </type> <condition> a+b </condition> </routecondition> </routeconditions> </ActionTaskAssignRoutingTable>

表 A-18 テンプレート定義 DTD 要素 (続き)

要素	説明	サンプル値
ActionTaskAssignUser	タスクをユーザまたはロール内のユーザに割り当てるためのアクション ロール内のユーザにタスクを割り当てる場合、プロセス エンジンは (現在再ルーティングが行われていない) ロール内のすべてのユーザに割り当てられたタスク数をレビューし、割り当てられたタスクが最も少ないユーザを識別し、このユーザにタスクを割り当てることでロード バランスを行う。 Studio では、[アクションを追加] ダイアログ ボックスの [タスク アクション] カテゴリ内で [ユーザにタスクを割り当て] を選択することでこのアクションを定義する。 以下の下位要素を定義する。 ■ notes ■ target ■ user ■ expression ■ role	<ActionTaskAssignUser> <notes> This is a note. </notes> <target> 963511775400 </target> <user>joe</user> <expression> false </expression> <role> false </role> </ActionTaskAssignUser>
ActionTaskDoit	タスクを実行するためのアクション Studio では、[アクションを追加] ダイアログ ボックスの [タスク アクション] カテゴリ内で [タスクを実行] を選択することでこのアクションを定義する。 以下の下位要素を定義する。 ■ notes ■ target	<ActionTaskDoIt> <notes> This is a note. </notes> <target> 963511923442 </target> </ActionTaskDoIt>

表 A-18 テンプレート定義 DTD 要素 (続き)

要素	説明	サンプル値
ActionTaskDone	タスクに完了マークを付けるためのアクションこのアクションにより、タスクの完了値は現在の日時に設定され、指定タスクの Marked Done イベントに割り当てられたすべてのアクションが実行される。 Studio では、[アクションを追加] ダイアログ ボックスの [タスク アクション] カテゴリ内で [タスクに完了マークを付ける] を選択することでこのアクションを定義する。 以下の下位要素を定義する。 ■ notes ■ target	<ActionTaskDone> <notes> This is a note. </notes> <target> 963511923442 </target> </ActionTaskDone>

表 A-18 テンプレート定義 DTD 要素 (続き)

要素	説明	サンプル値
ActionTaskDueDate	タスクの期限をアクションが実行された時点からのインターバル（分、時間、日、週あるいは月で表す）として指定するためのアクションこのアクションにより、Marked Done イベントに割り当てられたすべてのアクションが実行される。また、タスクが期日超過となった場合に実行される一連のアクションを定義することもできる。 Studio では、[アクションを追加] ダイアログ ボックスの [タスク アクション] カテゴリ内で [タスク期日を設定] を選択することでこのアクションを定義する。 以下の下位要素を定義する。 ■ id ■ notes ■ target ■ expression ■ calendartype ■ calendarname ■ actions	<ActionTaskDueDate> <id>987445484073</id> <notes> This is a note. </notes> <target>2</target> <expression> DateAdd(Date(), "m",1) </expression> <calendartype> 0 </calendartype> <calendarname> </calendarname> <actions> <ActionNoOp> <notes> This is a note. </notes> <description> Replace with task assignment. </description> </ActionNoOp> </actions> </ActionTaskDueDate>

表 A-18 テンプレート定義 DTD 要素 (続き)

要素	説明	サンプル値
ActionTaskSetComment	アクション実行時にユーザに対し表示されるコメントを指定するためのアクションコメントには通常、説明文が入る。 Worklist では、コメントはコメントカラムに自由な形式のテキスト メッセージとして表示される。 Studio では、[アクションを追加] ダイアログ ボックスの [タスク アクション] カテゴリ内で [タスクコメントを設定] を選択することでこのアクションを定義する。 以下の下位要素を定義する。 ■ notes ■ target ■ comment	<ActionTaskSetComment> <notes> This is a note. </notes> <target> 963511923442 </target> <comment> This is a comment. </comment> </ActionTaskSetComment>
ActionTaskSetPriority	タスクの優先順位を設定するためのアクション Worklist では、コメントはコメントカラムに自由な形式のテキスト メッセージとして表示される。 Studio では、[アクションを追加] ダイアログ ボックスの [タスク アクション] カテゴリ内で [タスク優先度を設定] を選択することでこのアクションを定義する。 以下の下位要素を定義する。 ■ notes ■ target ■ priority	<ActionTaskSetPriority> <notes> This is a note. </notes> <target> 963511923442 </target> <priority> 1 </priority> </ActionTaskSetPriority> [Sets priority to medium (1).]

表 A-18 テンプレート定義 DTD 要素 (続き)

要素	説明	サンプル値
ActionTaskUnassign	タスクの割り当てを解除するためのアクション Studio では、[アクションを追加] ダイアログ ボックスの [タスク アクション] カテゴリ内で [タスク優先度を設定] を選択することでこのアクションを定義する。 以下の下位要素を定義する。 ■ notes ■ target	<ActionTaskUnassign> <notes> This is a note. </notes> <target> 963511923442 </target> </ActionTaskUnassign>
ActionTaskUndo	タスクに未完了マークを付けるためのアクションこのアクションを実行しても、指定タスクの Activated イベントに割り当てられたアクションは実行*されない*。 Studio では、[アクションを追加] ダイアログ ボックスの [タスク アクション] カテゴリ内で [タスクの完了マークを外す] を選択することでこのアクションを定義する。 以下の下位要素を定義する。 ■ notes ■ target	<ActionTaskUndo> <notes> This is a note. </notes> <target> 963511923442 </target> </ActionTaskUndo>

表 A-18 テンプレート定義 DTD 要素 (続き)

要素	説明	サンプル値
ActionTimedEvent	正確な日時にトリガされ、特定のアクションを実行する時限イベントを作成するためのアクション Studio では、[アクションを追加] ダイアログ ボックスの [タスク アクション] カテゴリ内で [時限イベント] を選択することでこのアクションを定義する。 以下の下位要素を定義する。 ■ id ■ notes ■ expression ■ calendartype ■ executionunits ■ scheduleunits ■ executiontime ■ scheduletime ■ recoverable (ゼロまたは 1) ■ stopwhentaskdone ■ usetimeexpression ■ actions	<ActionTimedEvent> <id>987100768489</id> <notes> This is a note. </notes> <expression> DateAdd(Date(), "m", 1) </expression> <calendartype>0 </calendartype> <executionunits>3 </executionunits> <scheduleunits>0 </scheduleunits> <executiontime>5 </executiontime> <scheduletime> </scheduletime> <recoverable> true </recoverable> <stopwhentaskdone> true </stopwhentaskdone> <usetimeexpression> true </usetimeexpression> <actions> <ActionTaskAssignUser> <notes> This is a note. </notes> <target>2 </target> <user>joe</user> <expression> false </expression> <role>false </role> </ActionTaskAssignUser </actions> </ActionTimedEvent>

表 A-18 テンプレート定義 DTD 要素 (続き)

要素	説明	サンプル値
active	ワークフロー テンプレート定義がアクティブか非アクティブかを指定するフラグ（true または false）	<active> true </active>
activated	タスクの Activated イベントに関連付けられたアクション 以下の下位要素の、ゼロまたは任意の数の発生を定義することが可能。 %action-types	<activated> <ActionTaskAssignUser> <notes> This is a note. </notes> <target> 2 </target> <user>joe</user> <expression> false </expression> <role> false </role> </ActionTaskAssignUser> </activated>
ActionWorkflowAbort	ワークフローを中止するためのアクションこのアクションを実行すると、ワークフローは永久的に停止され、サーバから削除される。 Studio では、[アクションを追加] ダイアログ ボックスの [ワークフロー アクション] カテゴリ内で [ワークフローを中断] を選択することでこのアクションを定義する。 以下の下位要素を定義する。notes	<ActionWorkflowAbort> <notes> This is a note. </notes> </ActionWorkflowAbort>

表 A-18 テンプレート定義 DTD 要素 (続き)

要素	説明	サンプル値
ActionWorkflowDone	ワークフローに完了マークを付けるためのアクション Studio では、[アクションを追加] ダイアログ ボックスの [ワークフロー アクション] カテゴリ内で [ワークフローに完了マークを付ける] を選択することでこのアクションを定義する。 以下の下位要素を定義する。notes	<ActionWorkflowDone> <notes> This is a note. </notes> </ActionWorkflowDone>
ActionWorkflowSetComment	アクション実行時にユーザに対し表示されるコメントを指定するためのアクションコメントには通常、説明文が入る。 Studio では、[アクションを追加] ダイアログ ボックスの [ワークフロー アクション] カテゴリ内で [ワークフローに完了マークを付ける] を選択することでこのアクションを定義する。 以下の下位要素を定義する。 ■ notes ■ comment	<ActionWorkflowSetComment> <notes> This is a note. </notes> <comment> "Order Cancelled" </comment> </ActionWorkflowSetComment>

表 A-18 テンプレート定義 DTD 要素 (続き)

要素	説明	サンプル値
ActionWorkflowStart	ワークフローを開始するための、サブワークフローをサポートするアクション Studio では、[アクションを追加] ダイアログ ボックスの [ワークフロー アクション] カテゴリ内で [ワークフローを開始] を選択することでこのアクションを定義する。 以下の下位要素を定義する。 ■ id ■ notes ■ templateid ■ template-name ■ refvariable ■ orgexpression ■ parameters ■ results ■ actions ■ iviewx (ゼロまたは 1) ■ iviewy (ゼロまたは 1)	<ActionWorkflowStart> <id> 987445540073 </id> <notes> This is a note. </notes> <templateid> 3 </templateid> <template-name> ShipBill </template-name> <refvariable> Name </refvariable> <parameters> </parameters> <results> </results> <actions> </actions> </ActionWorkflowStart>
ActionXSLTransform	XML トランスフォーメーション実行のためのアクション 以下の下位要素を定義する。 ■ notes ■ input-expression ■ transform-document ■ transform-source ■ output-variable ■ xsl-parameters (任意の数)	

表 A-18 テンプレート定義 DTD 要素 (続き)

要素	説明	サンプル値
addressee	受取人のアドレス 以下の下位要素を定義する。 ■ name ■ expression ■ type	<addressee> <name> joe@work </name> <expression> false </expression> <type> address </type> </addressee>
addressees	任意の数のアドレス以下の下位要素を定義する。 ■ instanceid	
and	Join ノードが AND ノード（true）か OR ノード（false）かを指定するフラグ	<and>true</and>
arguments	実行時に評価され、program 要素で定義し呼び出されたプログラムに渡される、有効なワークフロー式	<arguments> CurrentUser() </arguments>
audit-enabled	ワークフロー テンプレート定義に対して監査が有効か無効かを指定するフラグ（true または false）	<audit-enabled> false </audit-enabled>
audittext	将来の検索時のために監査リスナに送られる監査エントリ式監査エントリは、関数とワークフロー変数、あるいはテキスト文字列で構築された式の場合もある。	<audittext> "Workflow cancelled." </audittext>

表 A-18 テンプレート定義 DTD 要素 (続き)

要素	説明	サンプル値
bcc	ActionSendEMail アクションで定義される、ブラインド コピーとして電子メール メッセージを送信する個々の電子メール アドレス 以下の下位要素の、ゼロまたは任意の数の発生を定義することが可能。 addressee	<bcc> <addressee> <name> joe@work </name> <expression> false </expression> <type> address </type> </addressee> </bcc>
calendartype	ビジネス カレンダー タイプ	<calendartype> Type1 </calendartype>
calendarname	ビジネス カレンダー名	<calendarname> MyCalendar </calendarname>
called	Start ノードを別のワークフローによって呼び出すことが可能かどうかを指定するフラグ（true または false）	<called> true </called>
cc	ActionSendEMail アクションで定義される、カーボン コピーとして電子メール メッセージを送信する個々の電子メール アドレス 以下の下位要素の、ゼロまたは任意の数の発生を定義することが可能。 addressee	<cc> <addressee> <name> joe@work </name> <expression> false </expression> <type> address </type> </addressee> </cc>

表 A-18　テンプレート定義 DTD 要素 (続き)

要素	説明	サンプル値
changedby	ワークフロー テンプレート定義を最後に修正した有効ユーザ	<changedby> joe </changedby>
changeddate	次のフォーマットで指定された、ワークフロー テンプレートが変更された日付 *yyyyMMddHHmm* ― ここで *yyyy* は年、*MM* は月、*dd* は日付、HH は時間（00 ～ 23）、mm は分を表す。	<changeddate> 200103220930 </changeddate> [March 22, 2001 9:30AM]
comment	コメントを定義し、実効時に評価される有効なワークフロー式またはテキスト文字列コメントには通常、説明文が入る。	<comment> "Order Cancelled" </comment>
commit-actions	トランザクションのコミット時に実行されるアクション 以下の下位要素の、ゼロまたは任意の数の発生を定義することが可能。 %commit-action-types	<commit-actions> <ActionTaskDone> <notes> This is a note. </notes> <target> 963511775400 </target> </ActionTaskDone> </commit-actions>
condition	実行時に評価され、どのサブアクションを実行するかを決定する有効なワークフロー条件	<condition> status="complete" </condition>
correlationid	相関 ID	

表 A-18 テンプレート定義 DTD 要素 (続き)

要素	説明	サンプル値
created	タスクの Created イベントに関連付けられたアクション 以下の下位要素の、ゼロまたは任意の数の発生を定義することが可能。 %action-types	<created> <ActionTaskAssignUser> <notes> This is a note. </notes> <target> 2 </target> <user>joe</user> <expression> false </expression> <role> false </role> </ActionTaskAssignUser> </created>
dateexpression	時限トリガされるワークフローを開始する日付この値は、ワークフロー式またはテキスト文字列で指定でき、結果は次のフォーマットとなる。*yyyyMMddHHmm* ―ここで *yyyy* は年、*MM* は月、*dd* は日付、HH は時間（00 ～ 23）、mm は分を表す。	<dateexpression> 200007120000 </dateexpression> [July 12, 2000 12:00 am]
Decision	Decision (分岐) ノード 以下の下位要素を定義する。 ■ %node ■ condition ■ truenexts ■ falsenexts ■ actions	A-116 ページの「テンプレート定義 DTD サンプル」を参照。
delivery-mode	ポスティングされた XML イベントを配信するためのモード	<delivery-mode> PERSISTENT </delivery-mode>

表 A-18　テンプレート定義 DTD 要素 (続き)

要素	説明	サンプル値
description	テキストによる要素の説明	<description> Start </description>
descriptorid	ActionBusinessOperation アクションに対して定義されるビジネスオペレーションの記述子 ID	<descriptorid> 31 </descriptorid>
doitifdone	21-6 ページの「タスクの実行」に記載のメソッドを用いて、完了マークの付いたタスクを実行することができるかどうかを指定するパーミッション フラグ（true または false）	<doitifdone> true </doitifdone>
Done	Done (完了) ノード 手動で開始されたワークフローに対し、以下の下位要素を定義する。 ■ %node ■ description ■ actions ■ plugin-data (ゼロまたは 1)	A-116 ページの「テンプレート定義 DTD サンプル」を参照。
donewithoutdoit	21-19 ページの「タスクへの完了または未完了マークの付与」に記載のメソッドを用いて、実行されていないタスクに完了マークを付けられるかどうかを指定するパーミッション フラグ（true または false）	<donewithoutdoit> true </donewithoutdoit>
effective	次のフォーマットの有効日付。*yyyyMMddHHmm* ― ここで *yyyy* は年、*MM* は月、*dd* は日付、HH は時間（00 ～ 23）、mm は分を表す。	<effective> 200007120000 </effective> [July 12, 2000 12:00 am]

表 A-18 テンプレート定義 DTD 要素 (続き)

要素	説明	サンプル値
error-handler	エラー ハンドラ 以下の下位要素を定義する。 ■ notes ■ variables ■ commit-actions ■ rollback-actions 以下の属性を定義する。 ■ name －エラー ハンドラ名 ■ initial －これが使用するデフォルトのエラー ハンドラかどうかを指定するフラグ（true または false）。この属性は、いずれか 1 つのエラー ハンドラに対してのみ true に設定できる。この属性のデフォルトは、false。	<error-handler name="EH Wrong data to Pobean" initial="false"> <notes> This is a note. </notes> <variables> </variables> <commit-actions> <ActionExitErrorHandler exit-type="continue"> <notes> This is a note. </notes> </ActionExitErrorHandler> </commit-actions> <rollback-actions> <ActionPostXMLEvent> <notes>Notes. </notes> <xml> <order> "new" </order> </xml> </ActionPostXMLEvent> <ActionExitErrorHandler exit-type="rollback"> <notes>Notes. </notes> </ActionExitErrorHandler> </rollback-actions> </error-handler>

表 A-18　テンプレート定義 DTD 要素 (続き)

要素	説明	サンプル値
error-handlers	ワークフロー テンプレート定義に対して定義されたエラー ハンドラ 以下の下位要素を定義する。 error-handler	<error-handlers> <error-handler> ... </error-handler> </error-handlers>
Event	Event (イベント) ノード *標準の* Event (イベント) ノードに対し、以下の下位要素を定義する。 ■ %node ■ root ■ description ■ key ■ condition ■ actions ■ variables ■ iviewx (ゼロまたは 1) ■ iviewy (ゼロまたは 1) *プラグイン* Event (イベント) ノードに対しては、以下の下位要素を定義する。 ■ %node ■ plugin-data ■ actions ■ variables ■ iviewx (ゼロまたは 1) ■ iviewy (ゼロまたは 1)	A-116 ページの「テンプレート定義 DTD サンプル」を参照。
event	Start ノードがイベント トリガ型かどうかを指定するフラグ（true または false）	<event> true </event>

表 A-18 テンプレート定義 DTD 要素 (続き)

要素	説明	サンプル値
executed	タスクの Executed イベントに関連付けられたアクション 以下の下位要素の、ゼロまたは任意の数の発生を定義することが可能。 %action-types	<executed> <ActionTaskDone> <notes> This is a note. </notes> <target> 963511923442 </target> </ActionTaskDone> </executed>
executiontime	(executionunits アクションによって定義された) 時限トリガ イベントが実行されるまでの時間を定義する、現在時間からの ActionTimedEvent ユニットの数を指定する数値	<executiontime> 5 </executiontime>
executionunits	実行時間を表すユニット以下の値に設定可能。 ■ 0 ―ミリ秒 ■ 1 ―秒 ■ 2 ―分 ■ 3 ―時間 ■ 4 ―日 ■ 5 ―週 ■ 6 ―月 ■ 7 ―ビジネス時間 ■ 8 ―ビジネス日	<executionunits> 3 </executionunits>

表 A-18　テンプレート定義 DTD 要素 (続き)

要素	説明	サンプル値
expression	ActionExpression、ActionTaskDueDate、ActionTimedEvent あるいは variable に対し、親要素のコンテキストにおいて有意なワークフロー式を指定する。 ActionTaskAssignRole、ActionTaskAssignUser あるいは addressee に対し、関連付けられた値がワークフロー式として表されるかどうかを定義するフラグ（true または false）を指定する。	<expression> XPath(" /order/text()") </expression>
false	Decision (分岐) ノードの false ブランチに関連付けられたアクション 以下の下位要素の、ゼロまたは任意の数の発生を定義することが可能。 %action-types	<false> <ActionWorkflowAbort> <notes> This is a note. </notes> </ActionWorkflowAbort> </false>
falsenexts	Decision (分岐) ノードの false ブランチに関連付けられた次のノードを指定する文字列 以下の下位要素の、ゼロまたは任意の数の発生を定義することが可能。 next	<falsenexts> 963410128518 </falsenexts>
id	親要素のコンテキストにおいて有意なユニークな ID	<id>3</id>
idexpression	ワークフロー テンプレート定義を識別するための式	<idexpression> "Order" +"" +:OrderNumber </idexpression>

表 A-18 テンプレート定義 DTD 要素 (続き)

要素	説明	サンプル値
in	呼び出し可能な Start ノードによって定義される変数が入力パラメータかどうかを指定するフラグ（true または false）	<in>true</in>
inmandatory	呼び出し可能な Start ノードによって定義される変数が必須入力パラメータかどうかを指定するフラグ（true または false）	<inmandatory> true </inmandatory>
input-expression	XSL トランスフォーメーションの実行の対象となる XML ドキュメントを識別するワークフロー式	
instanceid	ActionPostXMLEvent アクションによって定義された XML イベントの対象を定義するワークフロー インスタンス ID	<instanceid> 10 </instanceid>
instance-variable	WebLogic Process Integrator が ActionBusinessOperation アクションによって定義されたビジネスオペレーションを呼び出す対象となるインスタンス変数（Java オブジェクトまたは EJB を含む変数名）	<instance-variable> PobeanHandle </instance-variable>
instance-variable-type	instance-variable 要素によって定義されたインスタンス変数のインスタンス変数型	<instance-variable-type> session </instance-variable-type>
iviewx	Studio インタフェース表示シェイプの X 座標	<iviewx>150</iviewx>
iviewy	Studio インタフェース表示シェイプの Y 座標	<iviewx>150</iviewx>
jmstype	JMS 送り先タイプ	

表 A-18 テンプレート定義 DTD 要素 (続き)

要素	説明	サンプル値
Join	Join (結合) ノード 以下の下位要素を定義する。 ■ %node ■ and	A-116 ページの「テンプレート定義 DTD サンプル」を参照。
key	実行時にキーとして評価されるワークフロー式システムは、ワークフロー式を（指定したタイプの）着信 XML ドキュメントのルート要素と比較して評価し、ドキュメント インスタンスを一意に識別するキー値を導き出す。	<key> $orderid </key>
manual	Start ノードが手動で開始されるかどうかを指定するフラグ（true または false）	<manual> true </manual>
markeddone	タスクの Marked Done イベントに関連付けられたアクション 以下の下位要素の、ゼロまたは任意の数の発生を定義することが可能。 %action-types	<markeddone> <ActionWorkflowSetComment> <notes> This is a note. </notes> <comment> "Order Cancelled" </comment> </ActionWorkflowSetComment>
message	ActionSendEMail アクションによって定義された、送信される電子メール メッセージ テキスト実行時に評価される有効なワークフロー式または（ダブル クォーテーションで括られた）リテラル文字列。	<message> The order has been processed. </message>

表 A-18 テンプレート定義 DTD 要素 (続き)

要素	説明	サンプル値
message-properties	メッセージのプロパティ 以下の下位要素の、任意の数の発生を定義することが可能。property	
modifiable	21-23 ページの「タスク プロパティの設定」に記載のメソッドを用いて、実行時にタスク プロパティを修正できるかどうかを指定するパーミッション フラグ（true または false）	<modifiable> true </modifiable>
name	親要素のコンテキストにおいて有意な要素名を指定するユニークな文字列	<name> Order Processing </name>
next	次にアクティブにするノードを指定する文字列	<next> 963410128518 </next>
nexts	ゼロまたは任意の数の next アクションを含むリスト 以下の下位要素の、ゼロまたは任意の数の発生を定義することが可能。next	<nexts> <next> 963410128518 </next> </nexts>
nodes	ワークフロー テンプレート定義に対して定義されたノード 以下の下位要素の、ゼロまたは任意の数の発生を定義することが可能。 ■ Decision ■ Done ■ Event ■ Join ■ Start ■ Task	A-116 ページの「テンプレート定義 DTD サンプル」を参照。

表 A-18 テンプレート定義 DTD 要素 (続き)

要素	説明	サンプル値
notes	ユーザ定義のメモ	<notes> This workflow is used for order processing. </notes>
orderkey	6-1 ページの「JMS コネクタの確立」に記載の順序付けられたメッセージングに用いる順序付けキー	
orgexpression	ActionWorkflowStart アクションを用いて開始を試みているワークフローに関連付けられたオーガニゼーションを指定するワークフロー式	
out	呼び出し可能な Start ノードによって定義される変数が出力パラメータかどうかを指定するフラグ（true または false）	<out>false</out>
output-variable	XML トランスフォーメーションの結果が格納される変数	
parameters	呼び出し元のワークフローで、明示的に定義された変数を用いて設定するパラメータの初期値 以下の下位要素の、ゼロまたは任意の数の発生を定義することが可能。 parm	<parameters> <parm> <name> CustomerId </name> <value> $CustomerId </value> </parm> </parameters>

表 A-18 テンプレート定義 DTD 要素 (続き)

要素	説明	サンプル値
parm	パラメータ定義 ActionBusinessOperation アクションの場合、この要素はどのような文字列でも可。 ActionWorkflowStart アクションの場合、ゼロまたは任意の数の、以下の下位要素のペアを定義できる。 ■ name ■ value	<parm> :ItemNumber </parm>
parms	パラメータのリスト（必要な場合） 以下の下位要素の、ゼロまたは任意の数の発生を指定する必要がある。 parm	<parms> <parm> :ItemNumber </parm> <parm> :ItemQuantity </parm> <parm> :CustomerState </parm> </parms>

表 A-18 テンプレート定義 DTD 要素 (続き)

要素	説明	サンプル値
plugin	プラグイン エントリテンプレート定義で参照される各プラグインおよび独自のテンプレート定義プロパティを定義する各プラグインに対し、1 つの要素が表示される。 (省略可能) 下位要素を定義する。 plugin-data 以下の属性を定義する。 ■ name ―プラグインの名前 ■ referenced ―テンプレート定義がプラグインを用いるかどうかを指定するフラグ (true または false) ■ major-version ―プラグイン ソフトウェアのメジャー バージョン ■ minor-version ―プラグイン ソフトウェアのマイナー バージョン ■ vendor ―ベンダ名 ■ url ― URL (デフォルトは空の文字列) プラグインのプログラミングの詳細は、『*WebLogic Integration BPM プラグイン プログラミング ガイド*』を参照。	<plugin name="com.bea.wlpi.SamplePlugin" referenced="true" major-version="1" minor-version="0"> </plugin>

表 A-18 テンプレート定義 DTD 要素 (続き)

要素	説明	サンプル値
plugin-data	要素とテキストの組み合わせから成るプラグイン データ 以下の属性を定義する。 ■ name －プラグインの名前。この属性は必須 ■ ID －関連付けられた値オブジェクトに対してプラグインに与えられたユニークな ID。この属性は必須 プラグインのプログラミングの詳細は、『*WebLogic Integration BPM プラグイン プログラミング ガイド*』を参照。	<plugin-data name="com.bea.wlpi.SamplePlugin" ID="2"> <statusvariable> OrderStatus </statusvariable> <totalpricevariable> OrderTotalPrice </totalpricevariable> </plugin-data>
plugins	プラグインが提供される場合にのみ用いられるプラグインの定義 以下の下位要素の、ゼロまたは任意の数の発生を定義する。plugin プラグインのプログラミングの詳細は、『*WebLogic Integration BPM プラグイン プログラミング ガイド*』を参照。	<plugins> <plugin name="com.bea.wlpi.SamplePlugin" referenced="true" major-version="1" minor-version="0"> </plugin> </plugins>
priority	タスクの優先順位以下の値に設定可能。0 (低)、1 (中) および 2 (高)。	<priority> 1 </priority>
program	実行可能プログラムへの完全修飾パス	<program> MyProgram </program>
property	メッセージのプロパティ 以下の下位要素を定義する。 ■ name ■ value	

表 A-18　テンプレート定義 DTD 要素 (続き)

要素	説明	サンプル値
queue	XML ドキュメントが送信または検索される JMS キュー 指定された値が式かどうかを指定する expression 属性を定義する。この属性のデフォルトは、false。	<queue> Queue1 </queue>
reassignment	21-13 ページの「タスクの割り当て」に記載のメソッドを用いて、タスクを別の参加コンポーネントに再割り当てすることが可能かどうかを指定するパーミッション フラグ（true または false）	<reassignment> true </reassignment>
recoverable	システム障害が発生した場合に消失するすべてのアクションを復旧するかどうかを指定するフラグ	
refvariable	ワークフローが ActionWorkflowStart アクションによって開始された場合にワークフロー ID を格納する変数この変数は、呼び出し元のワークフロー内からの式で変数検索のために用いることができる。たとえば、NewWF.Name は Name 変数の値を検索する。	<refvariable> Name </refvariable>
replyto	返信先アドレス	
reschedamount	数値この値は、時限トリガ型 Start ノードが再スケジューリングされる dateexpression からの reschedunits ユニット数を指定する。	<reschedamount> 30 </reschedamount>

表 A-18 テンプレート定義 DTD 要素 (続き)

要素	説明	サンプル値
reschedunits	時限トリガ型ワークフローの再スケジューリング時間を表すユニット以下の値に設定可能。 ■ 0 ーミリ秒 ■ 1 ー秒 ■ 2 ー分 ■ 3 ー時間 ■ 4 ー日 ■ 5 ー週 ■ 6 ー月 ■ 7 ービジネス時間 ■ 8 ービジネス日	<reschedunits> 0 </reschedunits>
result	結果 ActionBusinessOperation アクションの場合、この要素はどのような文字列でも可。 ActionWorkflowStart アクションの場合、以下の下位要素を定義する。 ■ name ■ value	<result> PobeanHandle </result>
result-type	result-type 変数のタイプ	<result-type> Session </result-type>
results	結果 以下の下位要素の、ゼロまたは任意の数の発生を定義することが可能。 result	<results> <result> PobeanHandle </result> </results>

表 A-18　テンプレート定義 DTD 要素 (続き)

要素	説明	サンプル値
role	ActionTaskAssignRole の場合、既にあるロールまたは既にあるロールとして評価されるワークフロー式を指定する。ワークフロー式の場合、expression フラグは true に設定される。 ActionTaskAssignUser の場合、user 要素がユーザ（false）を指定するか、あるいはロール（true）を指定するかを示すフラグ（true または false）を指定する。	<role> "Role1" </role>
rollback-actions	トランザクションのロールバック時に実行されるアクション 以下の下位要素の、ゼロまたは任意の数の発生を定義することが可能。 %rollback-action-types	<rollback-actions> <ActionTaskDone> <notes> This is a note. </notes> <target> 963511775400 </target> </ActionTaskDone> </rollback-actions>
root	関連付けられたイベントをトリガする XML ドキュメントのルート要素	<root>order</root>
routecondition	ルーティング条件 以下の下位要素を定義する。 ■ to ■ type ■ condition	<routecondition> <to>Role1</to> <type> userinrole </type> <condition> a+b </condition> </routecondition>

表 A-18 テンプレート定義 DTD 要素 (続き)

要素	説明	サンプル値
routeconditions	ルーティング条件 以下の下位要素の、ゼロまたは任意の数の発生を定義することが可能。 routecondition	<routeconditions> <routecondition> <to>Role1</to> <type> userinrole </type> <condition> a+b </condition> </routecondition> </routeconditions>
scheduletime	数値この値は、（scheduleunits アクションによって定義された）時限トリガ型イベントが再スケジューリングされるまでの時間を定義する、現在時間からの ActionTimedEvent ユニットの数を指定する。	<scheduletime> 5 </scheduletime>
scheduleunits	再スケジューリング時間を表すユニット以下の値に設定可能。 ■ 0 ―ミリ秒 ■ 1 ―秒 ■ 2 ―分 ■ 3 ―時間 ■ 4 ―日 ■ 5 ―週 ■ 6 ―月 ■ 7 ―ビジネス時間 ■ 8 ―ビジネス日	<scheduleunits> 0 </scheduleunits>
subject	ActionSendEMail アクションによって定義された、送信される電子メール メッセージ テキストの件名行実行時に評価される有効なワークフロー式または（ダブル クォーテーションで括られた）リテラル文字列。	<subject> Urgent </subject>

表 A-18 テンプレート定義 DTD 要素 (続き)

要素	説明	サンプル値
Start	Start (開始) ノード *手動での開始が可能な*ワークフローに対し、以下の下位要素を定義する。 ■ %node ■ description ■ manual ■ nexts ■ actions ■ variables ■ iviewx (ゼロまたは 1) ■ iviewy (ゼロまたは 1) *呼び出し可能な*ワークフローに対しては、以下の下位要素を定義する。 ■ %node ■ description ■ called ■ nexts ■ actions ■ variables ■ iviewx (ゼロまたは 1) ■ iviewy (ゼロまたは 1)	A-116 ページの「テンプレート定義 DTD サンプル」を参照。

表 A-18 テンプレート定義 DTD 要素 (続き)

要素	説明	サンプル値
Start (続き)	*時限トリガ型*ワークフローに対しては、以下の下位要素を定義する。 ■ %node ■ description ■ timed ■ dateexpression ■ reschedamount ■ reschedunits ■ recoverable (ゼロまたは 1) ■ calendarname ■ startorg ■ nexts ■ actions ■ variables ■ iviewx (ゼロまたは 1) ■ iviewy (ゼロまたは 1) *イベントトリガ型*ワークフローに対しては、以下の下位要素を定義する。 ■ %node ■ description ■ event ■ root ■ key ■ condition ■ startorg ■ nexts ■ actions ■ variables	

表 A-18　テンプレート定義 DTD 要素 (続き)

要素	説明	サンプル値
Start (続き)	*プラグイントリガ型*ワークフローに対しては、以下の下位要素を定義する。 ■ %node ■ description ■ plugin-data ■ startorg ■ nexts ■ actions ■ variables	
startorg	Start ノードに関連付けられたオーガニゼーション	<startorg> "ORG1" </startorg>
stopwhentaskdone	タスク（true）完了時、あるいはワークフロー（false）完了時のどちらに実行を停止するかを指定するフラグ	<stopwhentaskdone> true </stopwhentaskdone>
target	操作している要素を識別する、ユニークかつ親要素のコンテキストにおいて有意な数値	<target> 963511923442 </target>
Task	Task (タスク) ノード 以下の下位要素を定義する。 ■ %node ■ name ■ priority ■ donewithoutdoit ■ doitifdone ■ unmarkdone ■ modifiable ■ reassignment ■ nexts ■ actions	A-116 ページの「テンプレート定義 DTD サンプル」を参照。

表 A-18 テンプレート定義 DTD 要素 (続き)

要素	説明	サンプル値
template-entry	アドレッシングされた JMS メッセージの対象	
template-name	ワークフロー テンプレートの名前	<template-name> ShipBill </template-name>
templateid	ActionWorkflowStart アクションによって開始するワークフローのテンプレート ID	<templateId> 3 </templateId>
time-to-live	存続時間値	<time-to-live> 0 </time-to-live>
timed	Start ノードが時限トリガ型かどうかを指定するフラグ（true または false）	<timed> true </timed>
to	ActionSendEMail アクションで定義される、主要受信者として電子メール メッセージを送信する個々の電子メール アドレス 以下の下位要素の、ゼロまたは任意の数の発生を定義することが可能。 addressee	<to> <addressee> <name> joe@work </name> <expression> false </expression> <type> address </type> </addressee> </to>
topic	XML ドキュメントが送信または検索される JMS トピック 指定された値が式かどうかを指定する expression 属性を定義する。この属性のデフォルトは、false。	<topic> Topic1 </topic

表 A-18 テンプレート定義 DTD 要素 (続き)

要素	説明	サンプル値
transaction	ワークフローがトランザクションの一部となり得るかどうかを指定するブール フラグ	<transaction> false </transaction>
transform-document	input-expression のトランスフォームに用いる XSLT ドキュメント	
transform-source	transform-document がワークフロー式から取得したもの（E）か、あるいは XML リポジトリ ファイルへのパスから取得したもの（R）かを示すフラグ	<transform-source> E </transform-source
true	Decision (分岐) ノードの true ブランチに関連付けられたアクション 以下の下位要素の、ゼロまたは任意の数の発生を定義することが可能。 %action-types	<true> <ActionWorkflowAbort> <notes> This is a note. </notes> </ActionWorkflowAbort> </true>
truenexts	Decision (分岐) ノードの true ブランチに関連付けられた次のノードを指定する文字列 以下の下位要素の、ゼロまたは任意の数の発生を定義することが可能。 next	<truenexts> 963410128518 </truenexts>

表 A-18 テンプレート定義 DTD 要素 (続き)

要素	説明	サンプル値
type	variable の場合、以下のいずれかの変数タイプを指定する。boolean、date、double、entity (エンティティ EJB)、integer、object、session (セッション EJB)、string、または xml。 routecondition の場合、to 要素に対して指定されたアドレッシング先 (user、userinrole あるいは role) のタイプを指定する。 addressee の場合、name 要素に対して指定されたアドレッシング先 (address、user あるいは role) のタイプを指定する。	<type>string</type>
unmarkdone	21-19 ページの「タスクへの完了または未完了マークの付与」に記載のメソッドを用いて、完了マークの付いたタスクを実行することができるかどうかを指定するパーミッション フラグ (true または false)	<unmarkdone> true </unmarkdone>
user	ユーザまたはロール内のユーザこの値は、既にあるユーザやロール、あるいは ActionTaskAssignUser アクションによってタスクが割り当てられるユーザやロールとして評価されるワークフロー式を指定することが可能。ワークフロー式の場合、関連付けられた expression フラグは true に設定される。	<user>true</user>
usetimeexpression	ActionTimedEvent アクションをトリガするために、指定した expression (true) を使用するか、あるいは executiontime (false) を使用するかを指定するフラグ	<usetimeexpression> true </usetimeexpression>

表 A-18　テンプレート定義 DTD 要素 (続き)

要素	説明	サンプル値
value	親要素のコンテキストにおいて有意なパラメータ値	<value> 5 </value>
variable	ワークフロー テンプレート定義に対して定義された変数 Workflow の場合、以下の下位要素を定義する。 ■ name ■ notes ■ type または plugin-data ■ xmltype ■ inmandatory ■ in ■ out Start ノードまたは ActionSendXMLToClient アクションの場合、以下の下位要素を定義する。 ■ name ■ expression ActionExpression アクションの場合、以下の下位要素を定義する。 ■ name ■ expression	<variable> <name> TotalPrice </name> <notes> This is a note. </notes> <type> double </type> <xmltype> </xmltype> <inmandatory> false </inmandatory> <in>false</in> <out>false</out> </variable>

表 A-18 テンプレート定義 DTD 要素 (続き)

要素	説明	サンプル値
variables	ワークフロー テンプレート定義に対して定義された変数 以下の下位要素の、ゼロまたは任意の数の発生を定義することが可能。 variable	<variables> <variable> <name> TotalPrice </name> <notes> This is a note. </notes> <type> double </type> <xmltype> </xmltype> <inmandatory> false </inmandatory> <in>false</in> <out>false</out> </variable> <variables>

表 A-18　テンプレート定義 DTD 要素 (続き)

要素	説明	サンプル値
Workflow	ルート要素 以下の下位要素を定義する。 ■ name ■ effective ■ changedby ■ changeddate ■ notes ■ idexpression ■ active ■ audit-enabled ■ plugins (ゼロまたは 1) ■ nodes ■ variables ■ error-handlers (ゼロまたは 1) 以下の属性を定義する。 ■ major-version －ソフトウェアのメジャー バージョン。デフォルトは 1 ■ minor-version －ソフトウェアのマイナー バージョン。デフォルトは 0 ■ build-number －ソフトウェアのビルド番号。デフォルトは 0。	A-116 ページの「テンプレート定義 DTD サンプル」を参照。
x	Studio シェイプの X 座標	<x>20</x>
xml	要素とテキストの組み合わせから成る XML ドキュメント構造体	<xml> <order> "new" </order> </xml>
xmltype	variable タイプが xml に設定されている場合、XML ドキュメント タイプ	

表 A-18 テンプレート定義 DTD 要素 (続き)

要素	説明	サンプル値
xsl-parameters	XML トランスフォーメーションに用いられるパラメータ 任意の数の xsl-parameter 下位要素を定義する。	
xsl-parameter	XML トランスフォーメーションに用いられるパラメータ 以下の下位要素を定義する。 ■ name ■ expression	
y	Studio シェイプの Y 座標	<y>40</y>

エンティティ説明

テンプレート定義のエンティティを次の表に示します。

表 A-19 テンプレート定義 DTD エンティティ

エンティティ	説明
%action-types	すべてのアクション タイプの定義 %std-action-types および以下のタイプを含む。 ■ ActionInvokeErrorHandler ■ ActionSetErrorHandler
%commit-action-types	トランザクション アクション タイプのコミット %std-action-types および以下のタイプを含む。 ■ ActionExitErrorHandler ■ ActionSetErrorHandler
%node	以下の特性を含むノードの特性 ■ id ■ x ■ y ■ notes

表 A-19 テンプレート定義 DTD エンティティ (続き)

エンティティ	説明
%rollback-action-types	トランザクション アクション タイプのロールバック以下のタイプを含む。 ■ ActionAuditEntry ■ ActionBusinessOperation ■ ActionCallProgram ■ ActionCondition ■ ActionExitErrorHandler ■ ActionExpression ■ ActionNoOp ■ ActionPlugin ■ ActionPostXMLEvent ■ ActionSendEMail ■ ActionSendXMLToClient

表 A-19 テンプレート定義 DTD エンティティ (続き)

エンティティ	説明
%std-action-types	以下のタイプを含む標準アクション タイプ ■ ActionAuditEntry ■ ActionBusinessOperation ■ ActionCallProgram ■ ActionCancelEvent ■ ActionCondition ■ ActionExpression ■ ActionNoOp ■ ActionPlugin ■ ActionPostXMLEvent ■ ActionSendEMail ■ ActionSendXMLToClient ■ ActionTaskAssignRole ■ ActionTaskAssignRoutingTable ■ ActionTaskAssignUser ■ ActionTaskDoit ■ ActionTaskDone ■ ActionTaskDueDate ■ ActionTaskSetComment ■ ActionTaskSetPriority ■ ActionTaskUnassign ■ ActionTaskUndo ■ ActionTimedEvent ■ ActionWorkflowAbort ■ ActionWorkflowDone ■ ActionWorkflowSetComment ■ ActionWorkflowStart ■ ActionXSLTransform

テンプレート定義 DTD サンプル

次のサンプルは、examples ディレクトリにある Order Processing テンプレート定義から抜粋したものです。このサンプルは、テンプレート定義 DTD の有効アプリケーションを示します。

```
<?xml version="1.0" encoding="UTF-8"?>
<Workflow major-version="1" minor-version="2" build-number="0">
  <name>Order Processing</name>
  <effective>200007120000</effective>
  <changedby>joe</changedby>
  <changeddate>20011281424</changeddate>
  <notes>This is a note.</notes>
  <idexpression>&amp;quot;Order&amp;quot;+&amp;quot; &amp;quot;+:OrderNumber</idexpression>
  <active>true</active>
  <audit-enabled>false</audit-enabled>
  <nodes>
.
.
.
  </nodes>
  <variables>
.
.
.
  </variables>
  <error-handlers>
.
.
.
  </error-handlers>
</Workflow>
```

Decision、Done、Event、Join、Start、および Task の各ノード タイプの詳細なサンプルを以下に示します。各要素のサンプルについては、A-62 ページの「要素説明」を参照してください。

分岐ノードのサンプル

次の抜粋は Decision (分岐) ノードのサンプルです。

```
    Decision>
      <id>963410131712</id>
      <x>100</x>
      <y>140</y>
```

```
  <notes></notes>
  <condition>:Confirm=&amp;quot;yes&amp;quot;</condition>
  <truenexts>
    <next>963410119665</next>
  </truenexts>
  <falsenexts>
  </falsenexts>
  <actions>
    <false>
      <ActionPostXMLEvent>
        <notes></notes>
        <xml>
          <order>
            <status>&amp;quot;cancelled&amp;quot;</status>
            <ordernumber>:OrderNumber</ordernumber>
          </order>
        </xml>
      </ActionPostXMLEvent>
    </false>
    <true>
    </true>
  </actions>
</Decision>
```

完了ノードのサンプル

次の抜粋は Done (完了) ノードのサンプルです。

```
<Done>
  <id>3</id>
  <x>140</x>
  <y>490</y>
  <notes></notes>
  <actions>
  </actions>
</Done>
```

イベント ノードのサンプル

次の抜粋は Event (イベント) ノードのサンプルです。

```
<Event>
  <id>991155339988</id>
  <x>370</x>
  <y>10</y>
  <notes></notes>
```

```
      <iviewx>290</iviewx>
      <iviewy>0</iviewy>
      <description>Watch for Cancellation</description>
      <root>cancelledorder</root>
      <key>$OrderID</key>
      <condition></condition>
      <nexts>
        <next>3</next>
      </nexts>
      <actions>
      </actions>
      <variables>
      </variables>
    </Event>
```

結合ノードのサンプル

次の抜粋は Join (結合) ノードのサンプルです。

```
    <Join>
      <id>963511805733</id>
      <x>110</x>
      <y>230</y>
      <notes></notes>
      <and>true</and>
      <nexts>
        <next>963511923442</next>
      </nexts>
    </Join>
```

開始ノードのサンプル

次の抜粋は Start (開始) ノードのサンプルです。

```
<Start>
      <id>1</id>
      <x>20</x>
      <y>40</y>
      <notes></notes>
      <iviewx>150</iviewx>
      <iviewy>50</iviewy>
      <description>Start</description>
      <called>true</called>
      <nexts>
        <next>2</next>
        <next>963410125634</next>
```

```
  </nexts>
  <actions>
  </actions>
  <variables>
    <variable>
      <name>CustomerName</name>
     <expression>XPath(&amp;quot;/order/customer/name/text()&amp;quot;)</expression>
    </variable>
    <variable>
      <name>CustomerId</name>
      <expression>XPath(&amp;quot;/order/customer/id/text()&amp;quot;)</expression>
    </variable>
    <variable>
      <name>orderstatus</name>
      <expression>XPath(&amp;quot;/order/status/text()&amp;quot;)</expression>
    </variable>
    <variable>
      <name>ordernumber</name>
      <expression>XPath(&amp;quot;/order/number/text()&amp;quot;)</expression>
    </variable>
    <variable>
      <name>CustomerMail</name>
    <expression>XPath(&amp;quot;/order/customer/email/text()&amp;quot;)</expression>
    </variable>
    <variable>
      <name>ItemName</name>
      <expression>XPath(&amp;quot;/order/item/name/text()&amp;quot;)</expression>
    </variable>
    <variable>
      <name>ItemNumber</name>
      <expression>XPath(&amp;quot;/order/item/id/text()&amp;quot;)</expression>
    </variable>
    <variable>
      <name>ItemQuantity</name>
     <expression>XPath(&amp;quot;/order/item/quantity/text()&amp;quot;)</expression>
    </variable>
    <variable>
      <name>ItemPrice</name>
      <expression>XPath(&amp;quot;/order/item/price/text()&amp;quot;)</expression>
    </variable>
    <variable>
      <name>CustomerState</name>
    <expression>XPath(&amp;quot;/order/customer/state/text()&amp;quot;)</expression>
    </variable>
  </variables>
</Start>
```

タスク ノードのサンプル

次の抜粋は Task (タスク) ノードのサンプルです。

```
<Task>
  <id>2</id>
  <x>100</x>
  <y>20</y>
  <notes></notes>
  <name>Confirm Order</name>
  <priority>1</priority>
  <donewithoutdoit>true</donewithoutdoit>
  <doitifdone>true</doitifdone>
  <unmarkdone>true</unmarkdone>
  <modifiable>true</modifiable>
  <reassignment>true</reassignment>
  <nexts>
    <next>963410131712</next>
  </nexts>
  <actions>
    <created>
    </created>
    <activated>
      <ActionTaskAssignUser>
        <notes></notes>
        <target>2</target>
        <user>joe</user>
        <expression>false</expression>
        <role>false</role>
      </ActionTaskAssignUser>
    </activated>
    <executed>
      <ActionSendXMLToClient>
        <id>959395846210</id>
        <notes></notes>
        <xml>
          <message-box title="&quot;Order Confirmation&quot;"
          style="&quot;question&quot;" options="&quot;yes_no&quot;">
          &quot;Do you Confirm this order?&quot;
          <actionid>&quot;959395846210&quot;</actionid>
          </message-box>
        </xml>
        <variables>
          <variable>
            <name>Confirm</name>
            <expression>XPath(&quot;/message-box/attribute::option&quot;)
            </expression>
          </variable>
```

```
          </variables>
          <actions>
            <ActionTaskDone>
              <notes></notes>
              <target>2</target>


            </ActionTaskDone>
          </actions>
        </ActionSendXMLToClient>
      </executed>
      <markeddone>
      </markeddone>
    </actions>
  </Task>
```

ワークロード要求 DTD

ワークロード要求 DTD は、実行時ワークロード統計を生成するために用いられる XML ドキュメントのフォーマットを説明します（戻される統計レポートのフォーマットについては、「ワークロード応答 DTD」で説明）。実行時ワークロードのクエリに用いられるメソッドに関する詳細については、22-19 ページの「実行時ワークロードのクエリ」を参照してください。

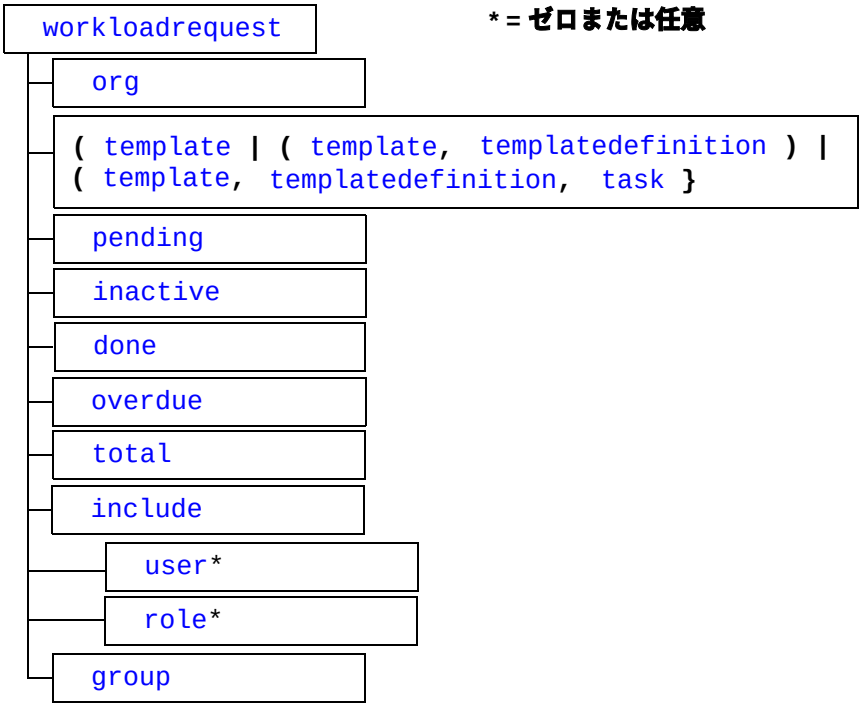
この節では、ワークロード要求 DTD について説明します。内容は以下のとおりです。

- 階層構造ダイアグラム
- DTD フォーマット
- 要素説明

階層構造ダイアグラム

ワークロード要求 DTD の階層構造を次のダイアグラムに示します。

図 A-17　ワークロード要求 DTD 階層構造ダイアグラム



DTD フォーマット

ワークロード要求 DTD のフォーマット WorkloadReq.dtd を次のリストに示します。

```
<!ELEMENT workloadrequest (org, template
                           | (template, templatedefinition)
                           | (template, templatedefinition, task),
                           pending, inactive, done, overdue,
                           total, include, group)>
<!ELEMENT include (#PCDATA, user* | role*)>
<!ELEMENT org (#PCDATA)>
<!ELEMENT template (#PCDATA)>
<!ELEMENT templatedefinition (#PCDATA)>
<!ELEMENT task (#PCDATA)>
<!ELEMENT pending (#PCDATA)>
```

```
<!ELEMENT inactive (#PCDATA)>
<!ELEMENT done (#PCDATA)>
<!ELEMENT overdue (#PCDATA)>
<!ELEMENT total (#PCDATA)>
<!ELEMENT user (#PCDATA)>
<!ELEMENT role (#PCDATA)>
<!ELEMENT group (#PCDATA)>
```

要素説明

ワークロード要求 DTD の要素を次の表に示します。

表 A-20 ワークロード要求 DTD 要素

要素	説明
done	完了マークの付いたワークフローが含まれるかどうかを指定するブール フラグ
group	レポートをグループ化し、合計のみを表示するかどうかを指定するブール フラグ
inactive	非アクティブなワークフローが含まれるかどうかを指定するブール フラグ
include	レポートに含めるユーザやロールのリスト 以下の下位要素を定義する。 ■ user (ゼロまたは任意の数) ■ role (ゼロまたは任意の数)
org	生成しているワークロード レポートの対象となるオーガニゼーション
overdue	期日超過のワークフローが含まれるかどうかを指定するブール フラグ
pending	ペンディングのワークフローが含まれるかどうかを指定するブール フラグ
role	レポートに含めるロールの名前
task	生成しているワークロード レポートの対象となるタスクの ID

表 A-20 ワークロード要求 DTD 要素 (続き)

要素	説明
template	生成しているワークロード レポートの対象となるテンプレートの ID
templatedefinition	生成しているワークロード レポートの対象となるテンプレート定義の ID
total	タスクの状態にかかわらず、すべてのタスクを含めるかどうかを指定するブール フラグ
user	レポートに含めるユーザの名前
workloadrequest	ルート要素 以下の下位要素を定義する。 ■ org ■ template | (template, templatedefinition) | (template, templatedefinition, task) ■ pending ■ inactive ■ done ■ overdue ■ total ■ include ■ group

ワークロード応答 DTD

ワークロード 応答 DTD は、実行時ワークロードをクエリした場合に戻される XML ドキュメントのフォーマットを説明します（クエリの要求に用いられる XML ドキュメントのフォーマットについては、「ワークロード要求 DTD」 で説明）。実行時ワークロードのクエリに関する詳細については、22-19 ページの「実行時ワークロードのクエリ」 を参照してください。

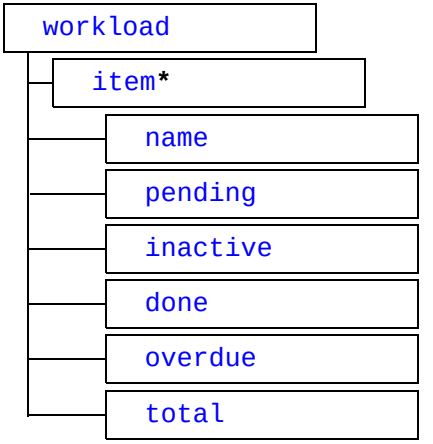
この節では、ワークロード応答 DTD について説明します。内容は以下のとおりです。

- 階層構造ダイアグラム
- DTD フォーマット
- 要素説明

階層構造ダイアグラム

ワークロード応答 DTD の階層構造を次のダイアグラムに示します。

図 A-18　ワークロード応答 DTD 階層構造ダイアグラム



DTD フォーマット

ワークロード応答 DTD のフォーマット `WorkloadResp.dtd` を次のリストに示します。

```
<!ELEMENT workload (item*)>
<!ELEMENT item (name, pending, inactive, done, overdue, total)>
<!ELEMENT name (#PCDATA)>
<!ELEMENT pending (#PCDATA)>
<!ELEMENT inactive (#PCDATA)>
<!ELEMENT done (#PCDATA)>
```

```
<!ELEMENT overdue (#PCDATA)>
<!ELEMENT total (#PCDATA)>
```

要素説明

ワークロード応答 DTD の要素を次の表に示します。

表 A-21 ワークロード応答 DTD 要素

要素	説明
done	項目ごとの完了マークの付いたタスクの数
inactive	項目ごとの非アクティブなタスクの数
item	生成するワークロード レポートの対象となる項目 以下の下位要素を定義する。 ■ name ■ pending ■ inactive ■ done ■ overdue ■ total
name	生成するワークロード レポートの対象となる項目の名前
overdue	項目ごとの期日超過のタスクの数
pending	項目ごとのペンディングのタスクの数
total	項目ごとのタスクの総数
workload	ルート要素 下位要素のゼロまたは任意の数の発生を定義する。item

B　値オブジェクトのまとめ

この付録では、BPM 値オブジェクトおよびそのメソッドを説明します。内容は以下のとおりです。

- BusinessCalendarInfo オブジェクト
- EventKeyInfo オブジェクト
- InstanceInfo オブジェクト
- OrganizationInfo オブジェクト
- PermissionInfo オブジェクト
- RepositoryFolderInfo オブジェクト
- RepositoryFolderInfoHelper オブジェクト
- RerouteInfo オブジェクト
- RoleInfo オブジェクト
- RolePermissionInfo オブジェクト
- TaskInfo オブジェクト
- TemplateDefinitionInfo オブジェクト
- TemplateInfo オブジェクト
- UserInfo オブジェクト
- UserPermissionInfo オブジェクト
- VariableInfo オブジェクト
- VersionInfo オブジェクト
- XMLEntityInfo オブジェクト
- XMLEntityInfoHelper オブジェクト

オブジェクト データへのアクセスの詳細については、5-1 ページの「値オブジェクトの使い方」を参照してください。

BusinessCalendarInfo オブジェクト

com.bea.wlpi.common.BusinessCalendarInfo オブジェクトにより、ビジネス カレンダーに関するオブジェクト データが管理されます。

新しい BusinessCalendarInfo オブジェクトは以下のコンストラクタを使用して作成できます。

```
public BusinessCalendarInfo(
  java.lang.String id,
  java.lang.String name,
  java.lang.String timezone,
  java.lang.String xml
)

public BusinessCalendarInfo(
  java.lang.String id,
  java.lang.String name,
  java.lang.String timezone,
  java.lang.String xml,
  boolean published
)
```

BusinessCalendarInfo オブジェクト データ、そのデータを定義する際に使用するコンストラクタ パラメータ、オブジェクト定義後のデータにアクセスする際に使用可能な取得および設定メソッドを次の表に示します。

表 B-1 BusinessCalendarInfo オブジェクト データ

オブジェクト データ	コンストラクタ パラメータ	取得メソッド	設定メソッド
ビジネス カレンダー ID	id	public java.lang.String getId()	public void setId(java.lang.String id)
ビジネス カレンダー名	name	public java.lang.String getName()	public void setName(java.lang.String name)

表 B-1 BusinessCalendarInfo オブジェクト データ (続き)

オブジェクト データ	コンストラクタ パラメータ	取得メソッド	設定メソッド
ビジネス カレンダーが作動するタイムゾーン	timezone	public java.lang.String getTimeZone()	public void setTimeZone(java.lang.String id)
ビジネス カレンダーの定義（XML） Business Calendar DTD フォーマットの説明については、A-11 ページの「ビジネス カレンダー DTD」を参照。	xml	public java.lang.String getXML()	public void setXML(java.lang.String xml)
ビジネス カレンダーを発行するかどうかを指定するブール フラグ	パブリッシュ	public boolean isPublished()	なし

詳細については、Javadoc の com.bea.wlpi.common.BusinessCalendarInfo を参照してください。

EventKeyInfo オブジェクト

com.bea.wlpi.common.EventKeyInfo オブジェクトにより、イベント キーに関するオブジェクト データが管理されます。

イベント キーは、イベント データの一部としてコード化された一意な値を指定するのに用いられます。この値は、着信 XML 文書に関連するワークフローの定義、またはインスタンスを識別するために使用されます。プロセス エンジンは、イベント コンテンツ型および記述子に基づいてワークフローの式とイベントを関連付ける EventKey 表を生成します。ワークフローの式は着信イベントに対するイベント キーを判断するために使用されます。

新しい EventKeyInfo オブジェクトは以下のコンストラクタを使用して作成できます。

```
public EventKeyInfo(
  java.lang.String contentType,
  java.lang.String eventDescriptor,
  java.lang.String expr,
  java.lang.String plugin,
  int fieldId
)
```

EventKeyInfo オブジェクト データ、そのデータを定義する際に使用するコンストラクタ パラメータ、オブジェクト定義後のデータにアクセスする際に使用可能な取得および設定メソッドを次の表に示します。

注意： プラグインのプログラミングの詳細は、『*WebLogic Integration BPM プラグイン プログラミング ガイド*』を参照してください。

表 B-2 EventKeyInfo オブジェクト データ

オブジェクト データ	パラメータ	取得メソッド	設定メソッド
MIME Content 型（例：text/xml）	contentType	public final java.lang.String getContentType()	なし
MIME Content 型が text/xml に設定される場合、以下のいずれかを指定。 ■ XML 文書 DTD のパブリック ID ■ パブリック ID が呼び出されない場合はシステム ID ■ DOCTYPE が指定されない場合はルート要素名 その他の場合は、プラグインが定義されたフォーマットでイベント記述子を指定する。	eventDescriptor	public final java.lang.String getEventDescriptor( )	なし

表 B-2 EventKeyInfo オブジェクト データ (続き)

オブジェクト データ	パラメータ	取得メソッド	設定メソッド
特定のコンテンツ型およびフォーマット説明を持つ文書に対して一意のイベント キーを提供する式	expr	public final java.lang.String getExpr()	public final void setExpr(java.lang.String expr)
式の評価に必要なフィールド型を提供するプラグイン。NULL およびコンテンツ型が text/xml の場合は、デフォルトの XML フィールド型が使用される。	plugin	public final java.lang.String getPlugin()	なし
プラグインにより割り当てられたフィールド ID	fieldId	public final int getFieldId()	なし

詳細については、Javadoc の com.bea.wlpi.common.EventKeyInfo を参照してください。

InstanceInfo オブジェクト

com.bea.wlpi.common.InstanceInfo オブジェクトにより、ワークフロー インスタンスに関するオブジェクト データが管理されます。

新しい InstanceInfo オブジェクトは以下のコンストラクタを使用して作成できます。

```
public InstanceInfo(
  java.lang.String id,
  java.lang.String templateId,
  java.lang.String templateDefinitionId,
  java.lang.String name,
  java.lang.String initiator,
  java.lang.String parentId,
  java.sql.Timestamp started,
  java.sql.Timestamp completed,
  java.lang.String idString,
  int state,
  java.lang.String comment
)

public InstanceInfo(
  java.lang.String id,
  java.lang.String templateId,
  java.lang.String templateDefinitionId,
  java.lang.String name,
  java.lang.String initiator,
  java.lang.String parentId,
  java.sql.Timestamp started,
  java.sql.Timestamp completed,
  java.lang.String idString,
  int state,
  java.lang.String comment
  java.util.Map pluginData
)
```

InstanceInfo オブジェクト データ、そのデータを定義する際に使用するコンストラクタ パラメータ、オブジェクト定義後のデータにアクセスする際に使用可能な取得メソッドを次の表に示します。

注意： プラグインのプログラミングの詳細は、『*WebLogic Integration BPM プラグイン プログラミング ガイド*』を参照してください。

表 B-3 InstanceInfo オブジェクト データ

オブジェクト データ	コンストラクタ パラメータ	取得メソッド
インスタンス ID	id	public final java.lang.String getId()
テンプレート ID	templateId	public final java.lang.String getTemplateId()
テンプレート定義 ID	templateDefinitionId	public final java.lang.String getTemplateDefinitionId()
テンプレート定義名	name	public final java.lang.String getName()
開始者 ID	initiator	public final java.lang.String getInitiator()
親 ID	parentId	public final java.lang.String getParentId()
ワークフローが開始された日付と時間	started	public final java.sql.Timestamp getStarted()
ワークフローが完了した日付と時間	completed	public final java.sql.Timestamp getCompleted()
インスタンス ラベル	idString	public final java.lang.String getIdString()

表 B-3 InstanceInfo オブジェクト データ (続き)

オブジェクト データ	コンストラクタ パラメータ	取得メソッド
以下の静的整数値のいずれかで表すことが可能なワークフロー インスタンス状態。 ■ WORKFLOW_STATE_ACTIVE ーインスタンス正常実行中 ■ WORKFLOW_STATE_SUSPENDED ーインスタンス サスペンド中	state	public final int getState()
Comment	Comment	public final java.lang.String getComment()
プラグイン データ	pluginData	public java.lang.Object getPluginInstanceData(java.lang.String pluginName)

詳細については、Javadoc の com.bea.wlpi.common.InstanceInfo を参照してください。

OrganizationInfo オブジェクト

com.bea.wlpi.common.OrganizationInfo オブジェクトにより、オーガニゼーションに関するオブジェクト データが管理されます。

新しい OrganizationInfo オブジェクトは以下のコンストラクタを使用して作成できます。

```
public OrganizationInfo(
  java.lang.String orgId
)

public OrganizationInfo(
  java.lang.String orgId,
  java.lang.String calendarId
)
```

OrganizationInfo オブジェクト データ、そのデータを定義する際に使用するコンストラクタ パラメータ、オブジェクト定義後のデータにアクセスする際に使用可能な取得および設定メソッドを次の表に示します。

表 B-4 OrganizationInfo オブジェクト データ

オブジェクト データ	コンストラクタ パラメータ	取得メソッド	設定メソッド
オーガニゼーション ID	orgId	public final java.lang.String getOrgId()	なし
使用するビジネス カレンダーの ID	calendarId	public final java.lang.String getCalendarId()	public final void setCalendarId(java.lang.String calendarId)

詳細については、Javadoc の com.bea.wlpi.common.OrganizationInfo を参照してください。

PermissionInfo オブジェクト

com.bea.wlpi.common.security.PermissionInfo オブジェクトにより、関連 com.bea.wlpi.common.security.EnumPermission オブジェクトで定義されたとおりに*プリンシパル*（ロールまたはユーザ）に対するパーミッション情報が管理されます。

新しい PermissionInfo オブジェクトは以下のコンストラクタを使用して作成できます。

```
public PermissionInfo(
  java.lang.String principalId
)
```

PermissionInfo オブジェクト データ、そのデータを定義する際に使用するコンストラクタ パラメータ、オブジェクト定義後のデータにアクセスする際に使用可能な取得および設定メソッドを次の表に示します。

表 B-5 PermissionInfo オブジェクト データ

オブジェクト データ	コンストラクタ パラメータ	取得メソッド	設定メソッド
プリンシパル ID	principalId	public java.lang.String getPrincipalId()	なし

表 B-5 PermissionInfo オブジェクト データ (続き)

オブジェクト データ	コンストラクタ パラメータ	取得メソッド	設定メソッド
パーミッション レベルおよび値 パーミッション レベルは、表で定義されている com.bea.wlpi.common.security.EnumPermission 静的整数値のうちのいずれかに設定可能 9-77 ページの「パーミッションの概要」。 また、パーミッション値は true (有効化) または false (無効化) に設定可能。	なし	なし	public void setPermission(com.bea.wlpi.common.security.EnumPermission permission, boolean value)
パーミッション レベルを有効化するかどうかを指定するブール フラグ 設定可能なパーミッション レベルの詳細については、9-77 ページの「パーミッションの概要」を参照。	なし	public boolean hasPermission(com.bea.wlpi.common.security.EnumPermission permission)	なし

詳細については、Javadoc の com.bea.wlpi.common.security.PermissionInfo を参照してください。また B-19 ページの「RolePermissionInfo オブジェクト」および B-29 ページの「UserPermissionInfo オブジェクト」も参照してください。

RepositoryFolderInfo オブジェクト

注意： com.bea.eci.repository.helper.RepositoryFolderInfo オブジェクトは、com.bea.wlpi.common.RepositoryFolderInfoHelper オブジェクトにより補助クラスが与えられることで、オブジェクト データのインポートおよびエクスポートが可能となります。詳細については、B-14 ページの「RepositoryFolderInfoHelper オブジェクト」を参照してください。

com.bea.eci.repository.helper.RepositoryFolderInfo オブジェクトにより XML リポジトリ内のフォルダに関するオブジェクト データが管理されます。

新しい OrganizationInfo オブジェクトは以下のコンストラクタを使用して作成できます。

```
public RepositoryFolderInfo(
  java.lang.String type,
  java.lang.String name,
  java.lang.String desc,
  java.lang.String notes
)
```

RepositoryFolderInfo オブジェクト データ、そのデータを定義する際に使用するコンストラクタ パラメータ、オブジェクト定義後のデータにアクセスする際に使用可能な取得メソッドを次の表に示します。

表 B-6 RepositoryFolderInfo オブジェクト データ

オブジェクト データ	コンストラクタ パラメータ	取得メソッド
フォルダ型	type	public java.lang.String getType()
フォルダ名	name	public java.lang.String getName()
フォルダ説明	desc	public java.lang.String getDescription()

表 B-6 RepositoryFolderInfo オブジェクト データ (続き)

オブジェクト データ	コンストラクタ パラメータ	取得メソッド
フォルダ メモ	notes	public java.lang.String getNotes()
フォルダ作成日	なし	public java.sql.Timestamp getCreatedOn()
フォルダ最終更新日付	なし	public java.sql.Timestamp getLastModifiedOn()

詳細については、Javadoc の

com.bea.eci.repository.helper.RepositoryFolderInfo を参照してください。

RepositoryFolderInfoHelper オブジェクト

com.bea.eci.repository.helper.RepositoryFolderInfo オブジェクトは、com.bea.wlpi.common.RepositoryFolderInfoHelper オブジェクトにより補助クラスが与えられることで、オブジェクト データのインポートおよびエクスポートが可能となります。

新しい RepositoryFolderInfoHelper オブジェクトは以下のコンストラクタを使用して作成できます。

```
public RepositoryFolderHelperInfo(
  java.lang.String type,
  java.lang.String name,
  java.lang.String desc,
  java.lang.String notes
)

public RepositoryFolderInfoHelper(
  java.lang.String type,
  java.lang.String name,
  java.lang.String desc,
  java.lang.String notes,
  com.bea.wlpi.common.RepositoryFolderInfoHelper parent
)
```

RepositoryFolderInfoHelper オブジェクト データ、そのデータを定義する際に使用するコンストラクタ パラメータ、オブジェクト定義後のデータにアクセスする際に使用可能な取得メソッドを次の表に示します。

表 B-7 RepositoryFolderInfo オブジェクト データ

オブジェクト データ	コンストラクタ パラメータ	取得メソッド	設定メソッド
フォルダ型	type	public java.lang.String getFolderType()	なし
フォルダ名	name	public java.lang.String getName()	なし
フォルダ説明	desc	public java.lang.String getDescription()	なし

表 B-7 RepositoryFolderInfo オブジェクト データ (続き)

オブジェクト データ	コンストラクタ パラメータ	取得メソッド	設定メソッド
フォルダ メモ	notes	public java.lang.String getNotes()	なし
親	parent	public com.bea.wlpi.common.RepositoryFolderInfoHelper getParent()	public void setParent(com.bea.wlpi.common.RepositoryFolderInfoHelper p)
XML リポジトリ発行可能オブジェクト コンテンツ	なし	public java.lang.Object getContents()	なし
XML リポジトリ発行可能エントリ名	なし	public java.lang.String getEntryName()	なし
XML リポジトリ発行可能オーナ名	なし	public java.lang.String getOwnerName()	なし
特定の発行可能オブジェクトに参照される XML リポジトリ発行可能物	なし	public java.util.List getReferencedPublishables(java.util.Map publishables)	なし
XML リポジトリ発行可能オブジェクト型	なし	public int getType()	なし

詳細については、Javadoc の com.bea.wlpi.common.RepositoryFolderInfoHelper を参照してください。

RerouteInfo オブジェクト

com.bea.wlpi.common.RerouteInfo オブジェクトにより、タスクの再ルーティングに関するオブジェクト データが管理されます。

新しい RerouteInfo オブジェクトは以下のコンストラクタを使用して作成できます。

```
public RerouteInfo(
  java.lang.String id,
  java.lang.String from,
  java.lang.String to,
  int type,
  java.sql.Timestamp effective,
  java.sql.Timestamp expiry
)
```

RerouteInfo オブジェクト データ、そのデータを定義する際に使用するコンストラクタ パラメータ、オブジェクト定義後のデータにアクセスする際に使用する取得および設定メソッドを次の表に示します。

表 B-8 RerouteInfo オブジェクト データ

オブジェクト データ	コンストラクタ パラメータ	取得メソッド	設定メソッド
タスク再ルーティング ID	id	public final java.lang.String getId()	なし
タスクが再ルーティングされるユーザまたはロール	from	public final java.lang.String getFrom()	public void setFrom(java.lang.String from)
再ルーティングされたタスクが割り当てられるユーザまたはロール	to	public final java.lang.String getTo()	public void setTo(java.lang.String to)

表 B-8 RerouteInfo オブジェクト データ (続き)

オブジェクト データ	コンストラクタ パラメータ	取得メソッド	設定メソッド
タスク再ルーティング型。ReRouteInfo クラスの最終静的メンバーにより定義された、以下のいずれかの静的整数値で表すことが可能。 ■ TYPE_USER ー他のユーザにタスクを再ルーティング ■ TYPE_USERINROLE ーロード バランスを使用してロールを持つユーザにタスクを再ルーティング ■ TYPE_ROLE ーロールにタスクを再ルーティング	type	public final int getType()	public void setType(int type)
タスク再ルーティングが有効化される日付および時間	effective	public final java.sql.Timestamp getEffective()	public void setEffective(java.sql.Timestamp effective)
タスク再ルーティングが終了となる日付および時間	expiry	public final java.sql.Timestamp getExpiry()	public void setExpiry(java.sql.Timestamp expiry)

詳細については、Javadoc の com.bea.wlpi.common.RerouteInfo を参照してください。

RoleInfo オブジェクト

com.bea.wlpi.common.RoleInfo オブジェクトにより、ロールに関するオブジェクト データが管理されます。

新しい RoleInfo オブジェクトは以下のコンストラクタを使用して作成できます。

```
public RoleInfo(
  java.lang.String roleId,
  java.lang.String orgId
)

public RoleInfo(
  java.lang.String roleId,
  java.lang.String orgId,
  java.lang.String calendarId
)

public RoleInfo(
  java.lang.String roleId,
  java.lang.String orgId,
  java.lang.String calendarId,
  java.lang.String groupId
)
```

RoleInfo オブジェクト データ、そのデータを定義する際に使用するコンストラクタ パラメータ、オブジェクト定義後のデータにアクセスする際に使用可能な取得および設定メソッドを次の表に示します。

表 B-9 RoleInfo オブジェクト データ

オブジェクト データ	コンストラクタ パラメータ	取得メソッド	設定メソッド
ロール ID	roleId	public final java.lang.String getRoleId()	なし
ロールが定義されたオーガニゼーションの ID	orgId	public final java.lang.String getOrgId()	なし

表 B-9 RoleInfo オブジェクト データ (続き)

オブジェクト データ	コンストラクタ パラメータ	取得メソッド	設定メソッド
使用するビジネス カレンダーの ID	calendarId	public final java.lang.String getCalendarId()	public void setCalendarId(java.lang.String calendarId)
このロールがマップする先のセキュリティ レルム グループの ID	groupId	public final java.lang.String getGroupId()	public final java.lang.String setGroupId(java.lang.String groupId)

詳細については、Javadoc の com.bea.wlpi.common.RoleInfo を参照してください。

RolePermissionInfo オブジェクト

com.bea.wlpi.common.security.RolePermissionInfo オブジェクトにより、関連 com.bea.wlpi.common.security.EnumPermission オブジェクトで定義されたとおりにオブジェクト データが管理されます。

新しい RolePermissionInfo オブジェクトは以下のコンストラクタを使用して作成できます。

```
public RolePermissionInfo(
  java.lang.String roleId,
  java.lang.String orgId
)

public RolePermissionInfo(
  java.lang.String roleId,
  java.lang.String orgId,
  java.lang.String groupId
)
```

RolePermissionInfo オブジェクト データ、そのデータを定義する際に使用するコンストラクタ パラメータ、オブジェクト定義後のデータにアクセスする際に使用可能な取得および設定メソッドを次の表に示します。

注意： RolePermissionInfo クラスは PermissionInfo クラスの拡張版です。特定のロールに関するパーミッションの設定方法を含む、取得および設定メソッドの詳細については、B-10 ページの「PermissionInfo オブジェクト」を参照してください。

表 B-10 RolePermissionInfo オブジェクト データ

オブジェクト データ	コンストラクタ パラメータ	取得メソッド	設定メソッド
ロール ID	roleId	なし	なし
ロールが定義されたオーガニゼーションの ID	orgId	public java.lang.String getOrgId()	public void setOrgId(java.lang.String orgId)
ロールが定義されたセキュリティ レルム グループの ID	groupId	public java.lang.String getGroupId()	public void setGroupId(java.lang.String groupId)

詳細については、Javadoc の com.bea.wlpi.common.security.RolePermissionInfo を参照してください。B-10 ページの「PermissionInfo オブジェクト」も参照してください。

TaskInfo オブジェクト

com.bea.wlpi.common.TaskInfo オブジェクトにより、ワークフロー タスクに関するオブジェクト データが管理されます。

新しい TaskInfo オブジェクトは以下のコンストラクタを使用して作成できます。

```
public TaskInfo(
  java.lang.String templateId,
  java.lang.String templateDefinitionId,
  java.lang.String instanceId,
  java.lang.String taskId,
  java.lang.String name,
  java.lang.String assignee,
  boolean assigneeIsRole,
  java.lang.String workflow,
  java.lang.String workflowId,
  int priority,
  java.sql.Timestamp started,
  java.sql.Timestamp completed,
  java.sql.Timestamp due,
  java.lang.String comment,
  boolean doneWithoutDoit,
  boolean doitIfDone,
  boolean unmarkDone,
  boolean modifiable,
  boolean reassignment
)
```

TaskInfo オブジェクト データ、そのデータを定義する際に使用するコンストラクタ パラメータ、オブジェクト定義後のデータにアクセスする際に使用可能な取得メソッドを次の表に示します。

表 B-11 TaskInfo オブジェクト データ

オブジェクト データ	コンストラクタ パラメータ	取得メソッド
テンプレート ID	templateId	public final java.lang.String getTemplateId()

表 B-11 TaskInfo オブジェクト データ (続き)

オブジェクト データ	コンストラクタ パラメータ	取得メソッド
テンプレート定義 ID	templateDefinitionId	public final java.lang.String getTemplateDefinitionId()
Instance ID	instanceId	public final java.lang.String getInstanceId()
タスク ID	taskId	public final java.lang.String getTaskId()
タスク名	name	public final java.lang.String getTaskName()
タスクの割り当て先（ユーザまたはロール）	assignee	public final java.lang.String getAssignee()
割り当て先がロールかどうかを指定するブール フラグ	assigneeIsRole	public final boolean isAssignedToRole()
ワークフロー名	workflow	public final java.lang.String getWorkflowName()
ワークフロー ID	workflowId	public final java.lang.String getWorkflowId()
タスク優先順位	priority	public final int getPriority()
タスクが開始された日付と時間	started	public final java.sql.Timestamp getStarted()
タスクが完了した日付と時間	completed	public final java.sql.Timestamp getCompleted()

表 B-11 TaskInfo オブジェクト データ (続き)

オブジェクト データ	コンストラクタ パラメータ	取得メソッド
タスクの完了期日および時間	due	public final Timestamp getDue()
コメント	Comment	public final java.lang.String getComment()
タスクに完了マークを付けるパーミッション	doneWithoutDoIt	public final boolean getDoneWithoutDoIt()
完了マークを付けた後タスクを実行するパーミッション	doitIfDone	public final boolean getDoitIfDone()
タスクに未完了マークを付けるパーミッション	unmarkDone	public final boolean getUnmarkDone()
タスク実行時プロパティを変更するパーミッション	modifiable	public final boolean getModifiable()
別の参加コンポーネントにタスク インスタンスを再割り当てするパーミッション	reassignment	public final boolean getReassignment()

表 B-11 TaskInfo オブジェクト データ (続き)

オブジェクト データ	コンストラクタ パラメータ	取得メソッド
TaskInfo クラスの最終静的メンバーにより定義された、以下の静的整数値いずれかで表すことが可能なタスク状態 ■ STATUS_COMPLETE ― タスク完了 ■ STATUS_INACTIVE ― タスク未開始 ■ STATUS_OVERDUE ― タスクは開始しているが、未完了で完了期日が過ぎている ■ STATUS_PENDING ― タスクは開始しているが未完了	なし	public final int getStatus()

詳細については、Javadoc の com.bea.wlpi.common.TaskInfo を参照してください。

TemplateDefinitionInfo オブジェクト

com.bea.wlpi.common.TemplateDefinitionInfo オブジェクトにより、テンプレート定義に関するオブジェクト データが管理されます。

新しい TemplateDefinitionInfo オブジェクトは以下のコンストラクタを使用して作成できます。

```
public TemplateDefinitionInfo(
  java.lang.String id,
  java.sql.Timestamp effective,
  java.sql.Timestamp expiry,
```

```
  boolean active
)

public TemplateDefinitionInfo(
  java.lang.String id,
  java.sql.Timestamp effective,
  java.sql.Timestamp expiry,
  boolean active,
  java.lang.String templateId,
  java.lang.String templateName
)

public TemplateDefinitionInfo(
  java.lang.String id,
  java.sql.Timestamp effective,
  java.sql.Timestamp expiry,
  boolean active,
  java.lang.String templateId,
  java.lang.String templateName,
  java.lang.String xml,
  java.utilMap pluginData,
  boolean published
)
```

TemplateDefinitionInfo オブジェクト データ、そのデータを定義する際に使用するコンストラクタ パラメータ、オブジェクト定義後のデータにアクセスする際に使用可能な取得メソッドを次の表に示します。

注意： プラグインのプログラミングの詳細は、『*WebLogic Integration BPM プラグイン プログラミング ガイド*』を参照してください。

表 B-12 TemplateDefinitionInfo オブジェクト データ

オブジェクト データ	コンストラクタ パラメータ	取得メソッド
テンプレート定義 ID	id	public final java.lang.String getId()
テンプレート定義の有効日付	effective	public final java.sql.Timestamp getEffective()
テンプレート定義の終了日付	expiry	public final java.sql.Timestamp getTemplateDefinition( )

表 B-12 TemplateDefinitionInfo オブジェクト データ (続き)

オブジェクト データ	コンストラクタ パラメータ	取得メソッド
テンプレート定義にアクティブのマークをつけるかどうかを指定するブール フラグ	active	public final boolean getActive()
テンプレート ID	templateId	public final java.lang.String getTemplateId()
テンプレート名	templateName	public final java.lang.String getTemplateName()
XML フォーマットのワークフロー テンプレート定義	xml	public java.lang.String getContents()
プラグイン データ	pluginData	public java.util.Map getPluginData() public com.bea.wlpi.common.plugin.PluginObject getPluginData(java.lang.String pluginName)
テンプレート定義を発行するかどうかを指定するブール フラグ	published	public boolean isPublished()

詳細については、Javadoc の com.bea.wlpi.common.TemplateDefinitionInfo を参照してください。

TemplateInfo オブジェクト

com.bea.wlpi.common.TemplateInfo オブジェクトにより、テンプレートに関するオブジェクト データが管理されます。

新しい TemplateInfo オブジェクトは以下のコンストラクタを使用して作成できます。

```
public TemplateInfo(
  java.lang.String id,
  java.lang.String name
)

public TemplateInfo(
  java.lang.String id,
  java.lang.String name,
  java.lang.String xml
)

public TemplateInfo(
  java.lang.String id,
  java.lang.String name,
  java.lang.String xml,
  java.util.Map pluginData,
  boolean published
)
```

TemplateInfo オブジェクト データ、そのデータを定義する際に使用するコンストラクタ パラメータ、オブジェクト定義後のデータにアクセスする際に使用可能な取得メソッドを次の表に示します。

表 B-13 TemplateInfo オブジェクト データ

オブジェクト データ	コンストラクタ パラメータ	取得メソッド
テンプレート ID	id	public final java.lang.String getId()
テンプレート名	name	public final java.lang.String getName()

表 B-13 TemplateInfo オブジェクト データ (続き)

オブジェクト データ	コンストラクタ パラメータ	取得メソッド
プラグイン定義された XML フォーマットのテンプレート プロパティ	xml	なし
プラグイン データ	pluginData	public java.util.Map getPluginData() public com.bea.wlpi.common.plugin.PluginObject getPluginData(java.lang.String pluginName)
テンプレートを発行するかどうかを指定するブール フラグ	published	public boolean isPublished()

詳細については、Javadoc の com.bea.wlpi.common.TemplateInfo を参照してください。

UserInfo オブジェクト

com.bea.wlpi.common.UserInfo オブジェクトにより、ユーザに関するオブジェクト データが管理されます。

新しい UserInfo オブジェクトは以下のコンストラクタを使用して作成できます。

```
public UserInfo(
  java.lang.String userId
)

public UserInfo(
  java.lang.String userId,
  java.lang.String eMailAddress,
  java.lang.String defaultOrgId,
  java.lang.String calendarId
)
```

UserInfo オブジェクト データ、そのデータを定義する際に使用するコンストラクタ パラメータ、オブジェクト定義後のデータにアクセスする際に使用可能な取得および設定メソッドを次の表に示します。

表 B-14 UserInfo オブジェクト データ

オブジェクト データ	コンストラクタ パラメータ	取得メソッド	設定メソッド
ユーザ ID	userId	public final java.lang.String getUserId()	なし
ユーザ電子メール アドレス	eMailAddress	public final java.lang.String getEMailAddress()	public void setEMailAddress(java.lang.String eMailAddress)
ユーザが定義されたデフォルト オーガニゼーションの ID	defaultOrgId	public final java.lang.String getDefaultOrgId()	public void setDefaultOrgId(java.lang.String defaultOrgId)
使用するビジネス カレンダーの ID	calendarId	public final java.lang.String getCalendarId()	public void setCalendarId(java.lang.String calendarId)

詳細については、Javadoc の com.bea.wlpi.common.UserInfo を参照してください。

UserPermissionInfo オブジェクト

com.bea.wlpi.common.security.UserPermissionInfo オブジェクトにより、ユーザ パーミッションに関するオブジェクト データが管理されます。

新しい UserPermissionInfo オブジェクトは以下のコンストラクタを使用して作成できます。

```
public UserPermissionInfo(
  java.lang.String userId
)

public UserPermissionInfo(
  java.lang.String userId,
  java.util.List roles
)
```

UserPermissionInfo オブジェクト データ、そのデータを定義する際に使用するコンストラクタ パラメータ、オブジェクト定義後のデータにアクセスする際に使用可能な取得および設定メソッドを次の表に示します。

表 B-15 RolePermissionInfo オブジェクト データ

オブジェクト データ	コンストラクタ パラメータ	取得メソッド	設定メソッド
ユーザ ID	userId	なし	なし
ユーザが属するロールの ID	roleId	public java.util.List getRoles()	なし

詳細については、Javadoc の com.bea.wlpi.common.security.UserPermissionInfo を参照してください。

VariableInfo オブジェクト

com.bea.wlpi.common.VariableInfo オブジェクトにより、ワークフロー変数に関するオブジェクト データが管理されます。

新しい VariableInfo オブジェクトは以下のコンストラクタを使用して作成できます。

```
public VariableInfo(
  java.lang.String name,
  java.lang.Object value
)

public VariableInfo(
  java.lang.String name,
```

```
  java.lang.String type
)
```

VariableInfo オブジェクト データ、そのデータを定義する際に使用するコンストラクタ パラメータ、オブジェクト定義後のデータにアクセスする際に使用可能な取得メソッドを次の表に示します。

表 B-16 VariableInfo オブジェクト データ

オブジェクト データ	コンストラクタ パラメータ	取得メソッド
変数名	name	public final java.lang.String getName()
変数値	value	public final java.lang.String getValue()
変数型	type	public final java.lang.String getType() 以下のメソッドを使用して型を有効化することも可能。 static void validateType(java.lang.String type) static void validateTypes(java.lang.String[ ] types)

詳細については、Javadoc の com.bea.wlpi.common.VariableInfo を参照してください。

VersionInfo オブジェクト

com.bea.wlpi.common.VersionInfo オブジェクトにより、バージョン番号に関するオブジェクト データが管理されます。

新しい VersionInfo オブジェクトは以下のコンストラクタを使用して作成できます。

```
public VersionInfo(
  int majorVersion,
  int minorVersion,
  int volume,
  java.lang.String build,
  java.lang.String name
)

public VersionInfo(
  int majorVersion,
  int minorVersion,
  java.lang.String build,
  java.lang.String name
)

public VersionInfo(
  java.lang.String version
)
```

VersionInfo オブジェクト データ、そのデータを定義する際に使用するコンストラクタ パラメータ、オブジェクト定義後のデータにアクセスする際に使用可能な取得メソッドを次の表に示します。

表 B-17 VersionInfo オブジェクト データ

オブジェクト データ	コンストラクタ パラメータ	取得メソッド
メジャー バージョン番号	majorVersion	public final int getMajorVersion()
マイナー バージョン番号	minorVersion	public final int getMinorVersion()
ボリューム番号	volume	public final int getVolume()
ビルド番号またはビルド名	build	public final java.lang.String getBuild()
バージョン名	name	public final java.lang.String getName()

表 B-17 VersionInfo オブジェクト データ (続き)

オブジェクト データ	コンストラクタ パラメータ	取得メソッド
デフォルト バージョン	なし	`public static final VersionInfo getDefaultVersion()`
以下のデータを指定する (リスト順)、ドットで区切られた 2 ～ 4 つの整数コンポーネントがある 10 進フォーマットのバージョン番号 `majorVersion`、`minorVersion`, `volume` および `build` 値	`version`	なし

詳細については、Javadoc の `com.bea.wlpi.common.VersionInfo` を参照してください。

XMLEntityInfo オブジェクト

注意: `com.bea.eci.repository.helper.XMLEntityInfo` オブジェクトは、`com.bea.wlpi.common.XMLEntityInfoHelper` オブジェクトにより補助クラスが与えられることで、オブジェクト データのインポートおよびエクスポートが可能となります。詳細については、B-35 ページの「XMLEntityInfoHelper オブジェクト」を参照してください。

`com.bea.eci.repository.helper.XMLEntityInfo` オブジェクトにより XML リポジトリ内のエンティティに関するオブジェクト データが管理されます。

新しい `XMLEntityInfo` オブジェクトは以下のコンストラクタを使用して作成できます。

```
public XMLEntityInfo(
  int type,
  java.lang.String name,
  java.lang.String desc,
```

```
  java.lang.String notes
)

public XMLEntityInfo(
  int type,
  java.lang.String name,
  java.lang.String desc,
  java.lang.String notes,
  byte[ ] content
)

public XMLEntityInfo(
  int type,
  java.lang.String name,
  java.lang.String desc,
  java.lang.String notes,
  java.sql.Timestamp createdOn,
  java.sql.Timestamp lastModifiedOn
)

public XMLEntityInfo(
  int type,
  java.lang.String name,
  java.lang.String desc,
  java.lang.String notes,
  java.sql.Timestamp createdOn,
  java.sql.Timestamp lastModifiedOn,
  java.lang.String content
)
```

XMLEntityInfoHelper オブジェクト データ、そのデータを定義する際に使用するコンストラクタ パラメータ、オブジェクト定義後のデータにアクセスする際に使用可能な取得メソッドを次の表に示します。

表 B-18 XMLEntityInfo オブジェクト データ

オブジェクト データ	コンストラクタ パラメータ	取得メソッド
エンティティ型	type	public java.lang.String getType()
エンティティ名	name	public java.lang.String getName()
エンティティ説明	desc	public java.lang.String getDescription()

表 B-18 XMLEntityInfo オブジェクト データ (続き)

オブジェクト データ	コンストラクタ パラメータ	取得メソッド
エンティティ メモ	notes	public java.lang.String getNotes()
エンティティ コンテンツ (バイト配列)	content	public byte[ ] getContent()
エンティティ作成日付	createdOn	public java.sql.Timestamp getCreatedOn()
エンティティ最終更新日	lastModifiedOn	public java.sql.Timestamp getLastModifiedOn()
エンティティ コンテンツ (文字列)	content	public java.lang.String getContentAsString()

詳細については、Javadoc の com.bea.eci.repository.helper.XMLEntityInfo を参照してください。

XMLEntityInfoHelper オブジェクト

com.bea.eci.repository.helper.XMLEntityInfo オブジェクトは、com.bea.wlpi.common.XMLEntityInfoHelper オブジェクトにより補助クラスが与えられることで、オブジェクト データのインポートおよびエクスポートが可能となります。

新しい XMLEntityInfoHelper オブジェクトは以下のコンストラクタを使用して作成できます。

```
public XMLEntityInfo(
  int type,
  java.lang.String name,
  java.lang.String desc,
```

```
  java.lang.String notes
)
```

XMLEntityInfoHelper オブジェクト データ、そのデータを定義する際に使用するコンストラクタ パラメータ、オブジェクト定義後のデータにアクセスする際に使用可能な取得メソッドを次の表に示します。

表 B-19 XMLEntityInfoHelper オブジェクト データ

オブジェクト データ	コンストラクタ パラメータ	取得メソッド	設定メソッド
エンティティ型	type	public java.lang.String getEntityType()	なし
エンティティ名	name	public java.lang.String getName()	なし
エンティティ説明	desc	public java.lang.String getDescription()	なし
エンティティ メモ	notes	public java.lang.String getNotes()	なし
親フォルダ	なし	public java.util.List getParentFolders()	public void addParentFolder(com.bea.wlpi.common.RepositoryFolderInfoHelper rfih)
XML リポジトリ発行可能オブジェクト コンテンツ	なし	public java.lang.Object getContents()	なし
XML リポジトリ発行可能エントリ名	なし	public java.lang.String getEntryName()	なし
XML リポジトリ発行可能オーナー名	なし	public java.lang.String getOwnerName()	なし

表 B-19 XMLEntityInfoHelper オブジェクト データ (続き)

オブジェクト データ	コンストラクタ パラメータ	取得メソッド	設定メソッド
特定の発行可能オブジェクトに参照される XML リポジトリ発行可能物	なし	`public java.util.List getReferencedPublishables(java.util.Map publishables)`	なし
XML リポジトリ発行可能オブジェクト型	なし	`public int getType()`	なし

詳細については、Javadoc の `com.bea.wlpi.common.XMLEntityInfoHelper` を参照してください。

C EJB 記述子および Java クラス記述子

記述子とはサーバ側の各 EJB および Java クラスを説明するもので、クライアントに情報を渡すために使用されます。この付録では、EJB および Java クラスの記述子オブジェクトおよびそのメソッドを説明します。内容は以下のとおりです。

- ClassDescriptor オブジェクト
- ClassInvocationDescriptor オブジェクト
- EJBDescriptor オブジェクト
- EJBInvocationDescriptor オブジェクト
- MethodDescriptor オブジェクト

EJB および Java クラスの記述子オブジェクトの詳細については、10-1 ページの「ビジネス オペレーションのコンフィグレーション」 を参照してください。

ClassDescriptor オブジェクト

com.bea.wlpi.common.ClassDescriptor オブジェクトでは、サーバ側 Java クラスが説明されます。ClassDescriptor は、サーバにデプロイされた Java クラスに関する情報をクライアントに渡すために使用されます。

新しい ClassDescriptor オブジェクトは以下のコンストラクタを使用して作成できます。

```
public ClassDescriptor(java.lang.Class javaClass)
```

ClassDescriptor オブジェクト データおよびオブジェクト定義後のデータにアクセスする際に使用可能な取得メソッドを次の表に示します。

表 C-1 ClassDescriptor オブジェクト データ

説明	取得メソッド
Java クラス メソッド記述子	`public final java.util.List getMethodDescriptors()`
Java クラス コンストラクタ記述子	`public final java.lang.Util getConstructorDescriptors()`
Java クラス修飾子	`public final int getModifiers()`
Java クラスをシリアライズ可能にするかどうかを指定するブール フラグ	`public final boolean isSerializable`

詳細については、Javadoc の com.bea.wlpi.common.ClassDescriptor を参照してください。

ClassInvocationDescriptor オブジェクト

com.bea.wlpi.common.ClassInvocationDescriptor オブジェクトにより、Java クラス メソッドの呼び出しの説明および実装が行われます。

ClassInvocationDescriptor は、Java クラスの関連メソッドのインスタンス化および呼び出しのために使用します。

新しい ClassInvocationDescriptor オブジェクトは以下のコンストラクタを使用して作成できます。

```
public ClassInvocationDescriptor(
  java.lang.String description,
  java.lang.String className,
  com.bea.wlpi.common.MethodDescriptor constructorDescriptor,
  java.lang.String[ ] constructorParmDescriptions,
  com.bea.wlpi.common.MethodDescriptor methodDescriptor,
  java.lang.String[ ] methodParmDescriptions
)

public ClassInvocationDescriptor(
  java.lang.String description,
  java.lang.String className,
  com.bea.wlpi.common.MethodDescriptor constructorDescriptor,
  java.lang.String[ ] constructorParmDescriptions,
  com.bea.wlpi.common.MethodDescriptor methodDescriptor,
  java.lang.String[ ] methodParmDescriptions,
  boolean published
)
```

ClassInvocationDescriptor オブジェクト データ、データを定義する際に使用するコンストラクタ パラメータ、オブジェクト定義後のデータにアクセスする際に使用可能な取得および設定メソッドを次の表に示します。

表 C-2 ClassInvocationDescriptor オブジェクト データ

オブジェクト データ	パラメータ	取得メソッド	設定メソッド
クラス起動記述子 ID	なし	public java.lang.String getId()	public void setId()
読み取り可能なフォーマットのクラス記述子	description	public java.lang.String getDescription()	public void set(...) (表の下の「注意」を参照)

表 C-2 ClassInvocationDescriptor オブジェクト データ (続き)

オブジェクト データ	パラメータ	取得メソッド	設定メソッド
完全修飾クラス名	className	public java.lang.String getClassName()	public void set(...) (表の下の「注意」を参照)
クラス コンストラクタ	constructorDescriptor	public com.bea.wlpi.common.MethodDescriptor getConstructor()	public void set(...) (表の下の「注意」を参照)
クラス コンストラクタ パラメータ記述子	constructorParmDescriptions	public java.lang.String[ ] getConstructorParmsDescriptions()	public void set(...) (表の下の「注意」を参照)
クラス メソッド記述子	methodDescriptor	public com.bea.wlpi.common.MethodDescriptor getMethod()	public void set(...) (表の下の「注意」を参照)
メソッド パラメータ記述子	methodParmDescriptions	public java.lang.String[ ] getMethodParmsDescriptions()	public void set(...) (表の下の「注意」を参照)
情報をロックするかどうかを指定するブール フラグ	published	public boolean isPublished()	public void set(...) (表の下の「注意」を参照)

注意 : 上の表に指定されているように、以下のメソッドは特定の Java クラス記述子情報の設定に使用できます。

```
public void set(
  java.lang.String description,
  java.lang.String className,
  com.bea.wlpi.common.MethodDescriptor
    constructorDescriptor,
  java.lang.String[ ] constructorParmDescriptions,
  com.bea.wlpi.common.MethodDescriptor methodDescriptor,
```

```
    java.lang.String[ ] methodParmDescriptions
)

public void set(
  java.lang.String description,
  java.lang.String className,
  com.bea.wlpi.common.MethodDescriptor
    constructorDescriptor,
  java.lang.String[ ] constructorParmDescriptions,
  com.bea.wlpi.common.MethodDescriptor methodDescriptor,
  java.lang.String[ ] methodParmDescriptions,
  boolean published
)
```

詳細については、Javadoc の com.bea.wlpi.client.common.WLPI を参照してください。

EJBDescriptor オブジェクト

com.bea.wlpi.common.EJBDescriptor オブジェクトでは、EJB が説明されます。EJBDescriptor は、サーバにデプロイされた各 EJB に関する情報をクライアントに渡すために使用されます。

新しい EJBDescriptor オブジェクトは以下のコンストラクタを使用して作成できます。

```
public EJBDescriptor()
```

EJBDescriptor オブジェクト データおよびオブジェクト定義後のデータにアクセスする際に使用可能な取得および設定メソッドを次の表に示します。

表 C-3 EJBDescriptor オブジェクト データ

説明	取得メソッド	設定メソッド
EJB デプロイメント名	public java.lang.String getEJBDeploymentName()	public void setEJBDeploymentName(java.lang.String ejbDeploymentName)
EJB ホーム名	public java.lang.String getEJBHomeName()	public void getEJBHomeName(java.lang.String ejbHomeName)

表 C-3 EJBDescriptor オブジェクト データ (続き)

説明	取得メソッド	設定メソッド
EJB ホーム インタフェース メソッド記述子	`public java.util.List getEJBHomeMethodDescriptors()`	`public void setEJBHomeMethodDescriptors(java.util.List methodDescriptors)`
EJB 主キー名	`public java.lang.String getEJBPrimaryKeyName()`	`public void setEJBPrimaryKeyName()`
EJB リモート インタフェース メソッド記述子	`public java.lang.Util getEJBRemoteMethodDescriptors()`	`public void setEJBRemoteMethodDescriptors(java.util.List methodDescriptors)`
EJB リモート名	`public java.lang.String getEJBRemoteName()`	`public void setEJBRemoteName(java.lang.String ejbRemoteName)`
セッションまたはエンティティ EJB	`public boolean isSessionEJB()`	`public void setSessionEJB()`

詳細については、Javadoc の `com.bea.wlpi.common.EJBDescriptor` を参照してください。

EJBInvocationDescriptor オブジェクト

`com.bea.wlpi.common.EJBInvocationDescriptor` オブジェクトでは、EJB メソッドの呼び出しの説明および実装が行われます。`EJBInvocationDescriptor` は、EJB リモート インタフェースを取得し、また関連メソッドを呼び出すために使用します。

新しい `EJBInvocationDescriptor` オブジェクトは以下のコンストラクタを使用して作成できます。

```
public EJBInvocationDescriptor()
```

```
public EJBInvocationDescriptor()
  java.lang.String description,
  com.bea.wlpi.common.EJBDescriptor beanDescriptor,
```

```
  com.bea.wlpi.common.MethodDescriptor homeMethodDescriptor,
  java.lang.String[ ] homeParmDescriptions,
  com.bea.wlpi.common.MethodDescriptor remoteMethodDescriptor,
  java.lang.String[ ] remoteParmDescriptions
)

public EJBInvocationDescriptor()
  java.lang.String description,
  com.bea.wlpi.common.EJBDescriptor beanDescriptor,
  com.bea.wlpi.common.MethodDescriptor homeMethodDescriptor,
  java.lang.String[ ] homeParmDescriptions,
  com.bea.wlpi.common.MethodDescriptor remoteMethodDescriptor,
  java.lang.String[ ] remoteParmDescriptions,
  boolean published
)
```

EJBInvocationDescriptor オブジェクト データ、データを定義する際に使用するコンストラクタ パラメータ、オブジェクト定義後のデータにアクセスする際に使用可能な取得および設定メソッドを次の表に示します。

表 C-4 EJBInvocationDescriptor オブジェクト データ

オブジェクト データ	パラメータ	取得メソッド	設定メソッド
EJB 起動記述子 ID	なし	public java.lang.String getId()	public void setId()
読み取り可能なフォーマットの EJB 記述子	description	public java.lang.String getDescription()	public void set(...) (表の下の「注意」を参照)
EJB デプロイメント名	なし	public java.lang.String getEJBDeploymentName()	public void setEJBDeploymentName(java.lang.String deploymentName)
EJB デプロイメント記述子	beanDescriptor	public java.lang.String	public void set(...) (表の下の「注意」を参照)
ホーム インタフェース名	なし	public java.lang.String getEJBHomeName()	public void setEJBHomeName(java.lang.String ejbHomeName)

表 C-4 EJBInvocationDescriptor オブジェクト データ (続き)

オブジェクト データ	パラメータ	取得メソッド	設定メソッド
ホーム インタフェース メソッド記述子	homeMethodDescriptor	public com.bea.wlpi.common.MethodDescriptor getEJBHomeMethod()	public void setEJBHomeMethod(com.bea.wlpi.common.MethodDescriptor ejbHomeMethod)
ホーム メソッド パラメータ記述子	homeParmDescriptions	public java.lang.String[ ] getHomeParmsDescriptions()	public void set(...) (表の下の「注意」を参照)
リモート インタフェース名	なし	public java.lang.String getEJBRemoteName()	public void setEJBRemoteName(java.lang.String ejbRemoteName)
リモート インタフェース メソッド記述子	remoteMethodDescriptor	public com.bea.wlpi.common.MethodDescriptor getEJBRemoteMethod( )	public void setEJBRemoteMethod(com.bea.wlpi.common.MethodDescriptor ejbRemoteMethod)
リモート メソッド パラメータ記述子	remoteParmDescriptions	public java.lang.String[ ] getRemoteParmsDescriptions()	public void set(...) (表の下の「注意」を参照)
情報をロックするかどうかを指定するブール フラグ	published	public boolean isPublished()	public void set(...) (表の下の「注意」を参照)
主キー クラス名	なし	public java.lang.String getEJBPrimaryKeyName()	public void setEJBPrimaryKeyName(java.lang.String ejbPrimaryKeyName)
戻り値型名	なし	public java.lang.String getReturnTypeName()	なし

表 C-4 EJBInvocationDescriptor オブジェクト データ (続き)

オブジェクト データ	パラメータ	取得メソッド	設定メソッド
10-8 ページの「EJB 記述子の取得」に説明されているとおり、invoke() メソッド使用時にデバッグ メッセージを表示するかどうかを指定するブール フラグ	なし	public boolean isVerbose()	public void setVerbose (boolean verbose)
記述子に必要な値セットをすべて挿入するかどうかを指定するブール フラグ	なし	public boolean isFullyFormed()	なし
セッションまたはエンティティ EJB を指定するブール フラグ	なし	public boolean isSessionEJB()	なし

注意 : 上の表に指定されているように、以下のメソッドは特定の EJB 起動記述子情報の設定に使用できます。

```
public void set(
  java.lang.String description,
  com.bea.wlpi.common.EJBDescriptor beanDescriptor,
  com.bea.wlpi.common.MethodDescriptor homeMethodDescriptor,
  java.lang.String[ ] homeParmDescriptions,
  com.bea.wlpi.common.MethodDescriptor
    remoteMethodDescriptor,
  java.lang.String[ ] remoteParmDescriptions
)

public void set(
  java.lang.String description,
  com.bea.wlpi.common.EJBDescriptor beanDescriptor,
  com.bea.wlpi.common.MethodDescriptor homeMethodDescriptor,
  java.lang.String[ ] homeParmDescriptions,
  com.bea.wlpi.common.MethodDescriptor
    remoteMethodDescriptor,
  java.lang.String[ ] remoteParmDescriptions,
  boolean published
)
```

詳細については、Javadoc の
`com.bea.wlpi.common.EJBInvocationDescriptor` を参照してください。

MethodDescriptor オブジェクト

com.bea.wlpi.common.MethodDescriptor オブジェクトでは、サーバ側 Java クラス メソッドが説明されます。MethodDescriptor は、Java クラスの各コンストラクタおよびメソッドに関する情報をクライアントに渡すために使用されます。

新しい MethodDescriptor オブジェクトは以下のコンストラクタを使用して作成できます。

```
public MethodDescriptor(java.lang.reflect.Constructor constructor)
```

```
public MethodDescriptor(java.lang.reflect.Method method)
```

MethodDescriptor オブジェクト データおよびオブジェクト定義後のデータにアクセスする際に使用可能な取得メソッドを次の表に示します。

表 C-5 MethodDescriptor オブジェクト データ

説明	取得メソッド
メソッド名	`public final java.lang.String getMethodName()`
完全修飾の例外クラス名	`public final java.lang.String[ ] getExceptionTypes()`
完全修飾のパラメータ クラスまたはインタフェース名	`public final java.lang.String[ ] getParameterTypes()`
戻り値型	`public final java.lang.String getReturnType()`
メソッドの要約をするかどうかを指定するブール フラグ	`public final boolean isAbstract()`
メソッドを確定するかどうかを指定するブール フラグ	`public final boolean isFinal()`
メソッドをネイティブにするかどうかを指定するブール フラグ	`public final boolean isNative()`

表 C-5 MethodDescriptor オブジェクト データ (続き)

説明	取得メソッド
メソッドまたはコンストラクタを非公開にするかどうかを指定するブール フラグ	`public final boolean isPrivate()`
メソッドまたはコンストラクタを保護するかどうかを指定するブール フラグ	`public final boolean isProtected()`
メソッドを静的にするかどうかを指定するブール フラグ	`public final boolean isStatic()`
メソッドを厳密にするかどうかを指定するブール フラグ	`public final boolean isStrict()`
メソッドを同期させるかどうかかを指定するブール フラグ	`public final boolean isSynchronized()`

詳細については、Javadoc の `com.bea.wlpi.common.MethodDescriptor` を参照してください。

D Studio および Worklist のロゴとテキストのカスタマイズ

BEA WebLogic Integration Studio および Worklist クライアントのロゴとテキストは、カスタマイズ可能です。

注意： Worklist クライアント アプリケーションは、このリリースの WebLogic Integration からは非推奨になります。Worklist クライアント アプリケーションに代わる機能については、『*WebLogic Integration リリース ノート*』を参照してください。

既にあるロゴ画像を Studio 用および Worklist 用にカスタマイズするには、`wlpi-studio.jar` および `wlpi-worklist.jar` の各ファイルにある次のファイルをカスタム ロゴと置換します。

`com/bea/wlpi/client/common/images/logo.gif`

既にある Studio 製品、Worklist 製品、およびサーバ メッセージ テキストをカスタマイズするには、必要に応じて以下のファイルをカスタム テキスト ファイルと置換します。

- `wlpi-studio.jar` および `wlpi-worklist.jar` (製品テキスト) 各ファイルの `com/bea/wlpi/client/common/text/text.properties`
- `wlpi-studio.jar` (Studio 製品テキスト) の `com/bea/wlpi/client/studio/text/text.properties`
- `wlpi-worklist.jar` (Worklist 製品テキスト) の `com/bea/wlpi/client/worklist/text/text.properties`
- `wlpi-studio.jar`、`wlpi-worklist.jar`、`wlpi-aux.jar` (サーバ メッセージ テキスト) の `com/bea/wlpi/common/message.properties`

E　データベース スキーマ

BPM データベース スキーマ エンティティを次の表に示します。

表 E-1 BPM データベース スキーマ エンティティ

エンティティ名	属性名	属性の定義	
		カラム データ型	カラム名
ACLENTRIES	PRINCIPAL	VARCHAR(254)	A_PRINCIPAL
	NAME	VARCHAR(254)	A_NAME
	PERMISSION	VARCHAR(254)	A_PERMISSION
ADDRESSEDMESSAGE	KEYVALUE	VARCHAR(254)	KEYVALUE
	EXPIRY	DATE	EXPIRY
	VALIDATED	BOOLEAN	Boolean
	MESSAGE	LARGE BINARY	MESSAGE
	MESSAGEID	VARCHAR(20)	MESSAGEID
BUSINESSCALENDAR	TIMEZONE	VARCHAR(50)	TIMEZONE
	NAME	VARCHAR(50)	NAME
	DATA	LARGE BINARY	DATA
	BUSINESSCALENDARID	VARCHAR(20)	ID
BUSINESSOPERATION	DESCRIPTION	VARCHAR(50)	DESCRIPTION
	LASTCHANGED	DATE	Datetime
	DEFINITION	LARGE BINARY	DEFINITION
	BUSINESSOPERATIONID	VARCHAR(20)	ID

表 E-1 BPM データベース スキーマ エンティティ (続き)

エンティティ名	属性名	属性の定義	
		カラム データ型	カラム名
EVENTKEY	PLUGIN	VARCHAR(50)	PLUGIN
	EXPRESSION	TEXT	EXPRESSION
	LASTCHANGED	DATE	Datetime
	EVENTDESCRIPTOR	VARCHAR(192)	EVENTDESCRIPTOR
	FIELDID	INTEGER	FIELDID
	CONTENTTYPE	VARCHAR(128)	CONTENTTYPE
EVENTWATCH	KEYVALUE	VARCHAR(254)	KEYVALUE
	CONDITION	TEXT	CONDITION
	PLUGIN	VARCHAR(50)	PLUGIN
	EVENTDESCRIPTOR	VARCHAR(192)	EVENTDESCRIPTOR
	CONTENTTYPE	VARCHAR(128)	CONTENTTYPE
	VALIDATED	BOOLEAN	Boolean
	TEMPLATEID	VARCHAR(20)	ID
	NODEID	VARCHAR(20)	NODEID
	FIELDID	INTEGER	FIELDID
	TEMPLATEDEFINITIONID	VARCHAR(20)	TEMPLATEDEFINITIONID
	STARTORG	TEXT	STARTORG
	INSTANCEID	VARCHAR(20)	INSTANCEID
GROUPMEMBER	GROUPID	VARCHAR(20)	GROUPID
	GROUPMEMBERID	VARCHAR(20)	GROUPMEMBERID
IDGENERATOR	MAXKEY	VARCHAR(20)	MAXKEY
	TABLENAME	VARCHAR(30)	TABLENAME

表 E-1 BPM データベース スキーマ エンティティ (続き)

エンティティ名	属性名	属性の定義	
		カラム データ型	カラム名
INSTANCE	STARTED	DATE	STARTED
	COMPLETED	DATE	COMPLETED
	INITIATOR	VARCHAR(50)	INITIATOR
	TEMPLATEDEFINITIONID	VARCHAR(20)	ID
	NAME	VARCHAR(50)	NAME
	ORGID	VARCHAR(20)	ORG
	TEMPLATEID	VARCHAR(20)	ID
	PARENTID	VARCHAR(20)	PARENTID
	DATA	LARGE BINARY	DATA
	INSTANCEID	VARCHAR(20)	ID
	IDSTRING	VARCHAR(50)	IDSTRING
	STATE	INTEGER	STATE
	ATTACHMENTS	INTEGER	ATTACHMENTS
	WORKFLOWCOMMENT	VARCHAR(50)	WORKFLOWCOMMENT
MESSAGERECIPIENT	RECIPIENTTYPE	INTEGER	RECIPIENTTYPE
	CREATED	DATE	Datetime
	RECIPIENTID	VARCHAR(20)	RECIPIENTID
	MESSAGEID	VARCHAR(20)	MESSAGEID
ORG	DESCRIPTION	VARCHAR(254)	DESCRIPTION
	BUSINESSCALENDARID	VARCHAR(20)	ID
	ORGID	VARCHAR(20)	ORGID

表 E-1 BPM データベース スキーマ エンティティ (続き)

エンティティ名	属性名	属性の定義	
		カラム データ型	カラム名
ORGMEMBER	USERID	VARCHAR(20)	U_NAME
	ORGID	VARCHAR(20)	ORGID
PLUGIN	CONFIG	TEXT	CONFIG
	STARTMODE	INTEGER	STARTMODE
	VERSION	VARCHAR(20)	VERSION
	NAME	VARCHAR(50)	NAME
REROUTE	ROUTETYPE	INTEGER	ROUTETYPE
	EFFECTIVE	DATE	EFFECTIVE
	ROUTETO	VARCHAR(50)	ROUTETO
	ROUTEFROM	VARCHAR(50)	ROUTEFROM
	REROUTEID	VARCHAR(20)	ID
	EXPIRY	DATE	EXPIRY
	ORGID	VARCHAR(20)	ORGID
STATS	STARTED	DATE	STARTED
	COMPLETED	DATE	COMPLETED
	TASKID	VARCHAR(20)	TASKID
	TEMPLATEDEFINITIONID	VARCHAR(20)	TEMPLATEDEFINITIONID
	ISROLE	BOOLEAN	ISROLE
	ORGID	VARCHAR(20)	ORG
	ASSIGNEE	VARCHAR(20)	ASSIGNEE

表 E-1 BPM データベース スキーマ エンティティ (続き)

エンティティ名	属性名	属性の定義	
		カラム データ型	カラム名
TASK	MODIFIABLE	BOOLEAN	MODIFIABLE
	UNMARKDONE	BOOLEAN	UNMARKDONE
	REASSIGNMENT	BOOLEAN	REASSIGNMENT
	STARTED	DATE	STARTED
	COMPLETED	DATE	COMPLETED
	NAME	VARCHAR(50)	NAME
	DOITIFDONE	BOOLEAN	DOITIFDONE
	PRIORITY	INTEGER	PRIORITY
	DONEWITHOUTDOIT	BOOLEAN	DONEWITHOUTDOIT
	TEMPLATEID	VARCHAR(20)	ID
	TEMPLATEDEFINITIONID	VARCHAR(20)	ID
	WORKFLOWID	VARCHAR(50)	WORKFLOWID
	INSTANCEID	VARCHAR(20)	Id
	TASKID	VARCHAR(20)	TASKID
	TASKCOMMENT	VARCHAR(50)	TASKCOMMENT
	ASSIGNEE	VARCHAR(20)	ASSIGNEE
	ORGID	VARCHAR(20)	ORGID
	WORKFLOW	VARCHAR(50)	WORKFLOW
	ISROLE	BOOLEAN	ISROLE
	DUE	DATE	DUE

表 E-1 BPM データベース スキーマ エンティティ (続き)

エンティティ名	属性名	属性の定義	
		カラム データ型	カラム名
TEMPLATE	NAME	VARCHAR(50)	NAME
	LASTCHANGED	DATE	Datetime
	DATA	LARGE BINARY	DATA
	TEMPLATEID	VARCHAR(20)	ID
TEMPLATEDEFINITION	ACTIVE	BOOLEAN	ACTIVE
	MANUAL	BOOLEAN	MANUAL
	EXPIRY	DATE	EXPIRY
	EFFECTIVE	DATE	EFFECTIVE
	LASTCHANGED	DATE	Datetime
	TEMPLATEDEFINITIONID	VARCHAR(20)	ID
	CALLED	BOOLEAN	CALLED
	TEMPLATEID	VARCHAR(20)	ID
	DATA	LARGE BINARY	DATA
TEMPLATEORG	TEMPLATEID	VARCHAR(20)	ID
	ORGID	VARCHAR(20)	ORGID
TIMEDEVENT	INSTANCEID	VARCHAR(20)	ID
	TEMPLATEID	VARCHAR(20)	ID
	TRIGGERTIME	DATE	TRIGGERTIME
	TEMPLATEDEFINITIONID	VARCHAR(20)	ID
	NODEID	VARCHAR(20)	NODEID
	STARTORG	TEXT	STARTORG

表 E-1 BPM データベース スキーマ エンティティ (続き)

エンティティ名	属性名	属性の定義	
		カラム データ型	カラム名
TRANSACTIONINFO	RESULTOBJECT	LARGE BINARY	RESULTOBJECT
	COMPLETEDTIME	DATE	COMPLETEDTIME
	TRANSACTIONID	VARCHAR(40)	TRANSACTIONID
USERMEMBER	USERID	VARCHAR(20)	U_NAME
	GROUPID	VARCHAR(20)	GROUPID
WLSGROUP	GROUPID	VARCHAR(20)	GROUPID
WLSUSER	PASSWORD	VARCHAR(254)	U_PASSWORD
	USERID	VARCHAR(20)	U_NAME
WORKFLOWROLE	BUSINESSCALENDARID	VARCHAR(20)	ID
	GROUPID	VARCHAR(20)	GROUPID
	DESCRIPTION	VARCHAR(20)	DESCRIPTION
	ORGID	VARCHAR(20)	ORGID
	ROLEID	VARCHAR(20)	ROLEID
WORKFLOWUSER	FULL_NAME	VARCHAR(254)	FULL_NAME
	DEFAULTORGID	VARCHAR(20)	DEFAULTORGID
	EMAILADDRESS	VARCHAR(100)	EMAILADDRESS
	BUSINESSCALENDARID	VARCHAR(20)	ID
	USERID	VARCHAR(20)	U_NAME

索引

E

G

H

I

J

T

U

V

W

X

あ

い

え

お

か

き

く

け

こ

さ

し

す

せ

た

つ

て

と

な

の

は

ひ

ふ

へ

ほ

め

も

ゆ

よ

り

れ

ろ

わ