WebSphere Business Integration Server
Express Plus

# コラボレーション開発ガイド

*V4.3*

WebSphere. software

WebSphere Business Integration Server
Express Plus

# コラボレーション開発ガイド

*V4.3*

**お願い**

本書および本書で紹介する製品をご使用になる前に、 487 ページの『特記事項』に記載されている情報をお読みください。

本書は、IBM WebSphere Business Integration Server Expres バージョン 4.3、IBM WebSphere Business Integration Server Express Plus バージョン 4.3、および新しい版で明記されていない限り、以降のすべてのリリースおよびモディフィケーションに適用されます。

本マニュアルに関するご意見やご感想は、次の URL からお送りください。今後の参考にさせていただきます。

`http://www.ibm.com/jp/manuals/main/mail.html`

なお、日本 IBM 発行のマニュアルはインターネット経由でもご購入いただけます。詳しくは

`http://www.ibm.com/jp/manuals/` の「ご注文について」をご覧ください。

(URL は、変更になる場合があります)

お客様の環境によっては、資料中の円記号がバックスラッシュと表示されたり、バックスラッシュが円記号と表示されたりする場合があります。


**原　典：** WebSphere Business Integration Server
Express Plus
Collaboration Development Guide

**発　行：** 日本アイ・ビー・エム株式会社

**担　当：** ナショナル・ランゲージ・サポート

第1刷 2004.8


この文書では、平成明朝体™W3、平成明朝体™W9、平成角ゴシック体™W3、平成角ゴシック体™W5、および平成角ゴシック体™W7を使用しています。この(書体*)は、(財）日本規格協会と使用契約を締結し使用しているものです。フォントとして無断複製することは禁止されています。

注* 平成明朝体™W3、平成明朝体™W9、平成角ゴシック体™W3、
平成角ゴシック体™W5、平成角ゴシック体™W7

# 目次

# まえがき

製品 IBM(R)WebSphere Business Integration Server Express および IBM(R) WebSphere Business Integration Server Express Plus は、InterChange Server Express、関連する Toolset Express、CollaborationFoundation、およびソフトウェア統合アダプターのセットで構成されています。Toolset Express に含まれるツールは、ビジネス・オブジェクトの作成、変更、および管理に役立ちます。プリパッケージされている各種アダプターは、お客様の複数アプリケーションにまたがるビジネス・プロセスに応じて、いずれかを選べるようになっています。標準的な処理のテンプレートである CollaborationFoundation は、カスタマイズされたプロセスを簡単に作成できるようにするためのものです。

本書では、Process Designer Express (WebSphere Business Integration Server Express Plus でのみ使用可能) を使用して、Business Integration Express インフラストラクチャーの一部であるコラボレーションを作成する方法について説明します。コラボレーションは、アプリケーション統合用のビジネス・ロジックを含むプログラムです。

特に明記されていない限り、本書の情報は、いずれも、IBM WebSphere Business Integration Server Express と IBM WebSphere Business Integration Server Express Plus の両方に当てはまります。WebSphere Business Integration Server Express およびそのバリアント (Business Integration Express など) は、同じ製品を指します。

## 対象読者

本書は、コラボレーションの作成や変更を担当するお客様、コンサルタント、および販売代理店を対象としています。読み進める前に、「システム・インプリメンテーション・ガイド」で説明されているすべての概念についてご理解ください。

コラボレーションを開発するには、プログラミングに関する標準的な概念および慣習を理解する必要があります。また、コラボレーションの開発には、Java® プログラム言語に関する多少の知識が必要です。コラボレーション API は、Java プログラム言語に基づいており、多くのコラボレーションが実行する操作 (ビジネス・オブジェクトの操作など) を処理します。プログラミングに関する経験があれば、Java に関する知識がなくとも、本書の例を参考にして単純なコラボレーションを作成できます。

## 本書の範囲

コラボレーションの開発過程全体には、多くのフェーズがあります。また、アプリケーション・エキスパート、ビジネス・アナリスト、およびプログラマーなど、多くの人々が関わることになります。コラボレーション開発チームは、アプリケーション統合問題を解析したら、WebSphere Business Integration システム内でこの問題を解決するためにビジネス・プロセスを構築します。通常、このチームは、フローチャートから作業を開始し、このフローチャートをコラボレーションに移行します。

本書では、仕様、フローチャート、または手書きの設計から開始することを前提としています。本書では、ビジネス・プロセスの分析、コネクターの開発、またはビジネス・オブジェクトの設計は扱いません。

**注:** 本書では、ディレクトリー・パスに円記号 (¥) を使用します。UNIX システムの場合には、円記号 (¥) をスラッシュ (/) に置き換えてください。ファイルのパス名はすべて、使用システムで製品がインストールされたディレクトリーを基準とした相対パス名です。

## 本書の使用法

本書の構成は、以下のとおりです。

**第 1 部: 始めに**

| | |
|---|---|
| 3 ページの『第 1 章 コラボレーション開発の概要』 | コラボレーションおよびコラボレーション開発環境の概要です。 |
| 17 ページの『第 2 章 Process Designer Express の概要』 | Process Designer Express インターフェースの詳細です。 |

**第 2 部: コラボレーション・テンプレートの作成**

| | |
|---|---|
| 33 ページの『第 3 章 コラボレーションの設計』 | コラボレーション開発の設計フェーズで役に立つ情報について説明します。 |
| 79 ページの『第 4 章 コラボレーション・テンプレートの構築』 | コラボレーション・テンプレートの定義の作成方法について説明します。 |
| 107 ページの『第 5 章 アクティビティー・ダイアグラムの使用』 | シンボルおよびその他のコンポーネントを使用してアクティビティー・ダイアグラムを構築する方法について説明します。 |
| 153 ページの『第 6 章 Activity Editor の使用』 | Activity Editor を使用して、コラボレーション・テンプレートにビジネス・ロジックを作成する方法を説明します。 |
| 169 ページの『第 7 章 例外処理』 | コラボレーション・テンプレートに例外処理を実装する方法を説明します。 |
| 187 ページの『第 8 章 ワークスペースとレイアウトのオプション』 | アクティビティー・ダイアグラムおよびダイアグラミング領域自体の中でシンボルを配置するためのオプションについて説明します。 |
| 203 ページの『第 10 章 コーディングのヒントと例』 | 共通操作を実行する方法を示すコードの断片およびヒントについて説明します。 |
| 197 ページの『第 9 章 メッセージ・ファイルの作成』 | すべてのコラボレーションがメッセージのロギングおよびトレースを保持する上で必要なファイルの設定方法について説明します。 |

**第 3 部: サポートされる機能ブロック**

| | |
|---|---|
| 471 ページの『標準的なコラボレーションに関する情報』 | CollaborationFoundation テンプレートを基本とするすべてのコラボレーションに共通のビジネス・プロセスとビジネス・プロパティーを説明します。 |
| 用語集 | 本書で使用される用語が定義されています。 |

## 関連文書

本製品に付属する資料全体を通して、すべての WebSphere Business Integration Server Express WebSphere Business Integration Server Express Plus のインストールに共通の機能とコンポーネントについて説明すると共に、特定のコンポーネントの参照資料も紹介しています。

関連文書は、www.ibm.com/websphere/wbiserverexpress/infocenter でダウンロード、インストール、および表示することができます。

このサイトには、資料のダウンロード、インストール、および表示に関する簡単な説明があります。

**注:** 本書の発行後に公開されたテクニカル・サポートの技術情報や速報に、本書の対象製品に関する重要な情報が記載されている場合があります。これらの技術情報や速報は、WebSphere Business Integration のサポート Web サイト (http://www.ibm.com/software/integration/websphere/support) で参照できます。適切なコンポーネント領域を選択し、「Technotes (技術情報)」セクションと「Flashes (速報)」セクションを参照してください。

## 表記上の規則

本書は下記の規則に従って編集されています。

| | |
|---|---|
| `Courier フォント` | コマンド名、ファイル名、入力情報、システムが画面に出力した情報などのリテラル値を示します。 |
| **太字** | 初出語を示します。 |
| *イタリック* | 変数名または相互参照を示します。資料を PDF ファイルとして開くと、相互参照はイタリックの青字で表示されます。相互参照をクリックすることにより、目的の情報にジャンプできます。 |
| *イタリック `Courier` フォント* | リテラル・テキスト内の変数名を示します。 |
| `枠付き Courier` | コード・フラグメントをテキストの他の部分と区別します。 |
| *青い文字* | オンラインで表示したときにのみ見られる青いアウトラインには、相互参照用のハイパーリンクです。アウトライン内をクリックすることにより、参照先オブジェクトに飛ぶことができます。 |
| { } | 構文の記述行の場合、中括弧 { } で囲まれた部分は、選択対象のオプションです。1 つのオプションのみを選択する必要があります。 |
| [ ] | 構文の記述行の場合、大括弧 [] で囲まれた部分は、オプションのパラメーターです。 |

| | |
|---|---|
| ... | 構文の記述行の場合、省略符号 ... は直前のパラメーターが繰り返されることを示します。例えば、`option[,...]` は、複数のオプションをコンマで区切って指定できることを意味します。 |

# 改訂の要約

## バージョン 4.3 の新機能

本書の最初のリリースです。

# 第 1 部 始めに

# 第 1 章 コラボレーション開発の概要

本書では、Process Designer Express とコラボレーション開発プロセスについて説明します。Process Designer Express は、IBM WebSphere Business Integration Server Express Plus でのみ使用可能で、コラボレーションを作成することができる強力なモデル化/コード生成ツールです。コラボレーションとは、複数のアプリケーションを必要とする企業ビジネス・プロセスを作成するプログラムです。

この章では、コラボレーション開発のプロセスとコラボレーション開発に使用するツールについて概説します。

## コラボレーションの概要

コラボレーション とは、ビジネス・プロセスを記述するソフトウェア・モジュールで、IBM InterChange Server Express (ICS) の中で実行されます。ビジネス・プロセスとは、アプリケーション統合のビジネス・ロジックを含むプログラムです。コラボレーションは、さまざまなタイプの Java 操作を実行することができます。しかし、ほとんどの場合、コラボレーションはビジネス・オブジェクトに対して以下の操作を実行します。

- トリガー・イベント内の 1 つ以上の値を取得および操作します。
- アプリケーションが指定のエンティティーを作成、削除、または更新できるように、ビジネス・オブジェクトを要求としてアプリケーションに送信します。
- エンティティーを検索する要求をアプリケーションに送信します。

表 1 に示すように、コラボレーションはリポジトリー定義とランタイム・オブジェクトという 2 つの構成要素から成るエンティティーです。

*表 1. コラボレーションの構成要素*

| リポジトリー・エンティティー | ランタイム・オブジェクト |
|---|---|
| コラボレーション・テンプレート | コラボレーション・オブジェクト |

コラボレーションをインストールするときには、コラボレーション・テンプレートをインストールします。コラボレーション・テンプレートにはコラボレーションの実行ロジックがすべて含まれますが、コラボレーションをすぐに実行できるわけではありません。コラボレーションを実行するには、最初にテンプレートからコラボレーション・オブジェクト を作成する必要があります。コラボレーション・オブジェクトを実行可能にするには、コラボレーション・オブジェクトをコネクターまたは別のコラボレーション・オブジェクトにバインドしてから他の構成プロパティーを指定することにより、コラボレーション・オブジェクトを構成します。

**注:** コラボレーションが IBM WebSphere Business Integration Server Express システムのコンポーネントとして機能する仕組みの概要については、「システム・インプリメンテーション・ガイド」を参照してください。このセクションでは、コラボレーションの開発方法の観点からコラボレーションの定義について説明します。

本書では、コラボレーション・テンプレートとコラボレーション・オブジェクトを区別する必要がない限り、どちらも単にコラボレーションと呼びます。

## コラボレーション・テンプレート

コラボレーションは、コラボレーション・テンプレートとして開始されます。コラボレーション・テンプレートとは、コラボレーションの中のロジックの仕様です。コラボレーション・テンプレートは Process Designer Express ツールを使用して定義します。このツールは、該当する情報を System Manager に格納します。コラボレーション・テンプレートの開発には以下のステップが含まれます。

- コラボレーション・テンプレートを作成します
- コラボレーション・テンプレートのパーツを構築します
- コラボレーション・テンプレートをコンパイルします

### コラボレーション・テンプレートの作成

コラボレーションを開発する際、Process Designer Express というツールを使用してコラボレーション・テンプレートを開発します。Process Designer Express には使いやすいグラフィカル・ユーザー・インターフェース (GUI) が用意されています。このためプログラムを開発するために通常必要となるコーディングの多くが不要になります。このインターフェースを使用すると、変数を宣言したりコード・フラグメントを記述するなどの作業が容易になります。Business Integration Express には、開発プロセスを促進する汎用コラボレーション・テンプレートも用意されています。Process Designer Express を使用すると、標準的なプログラム言語のプログラムを記述するよりも簡単にコラボレーション・テンプレートを開発できます。しかし、コラボレーション開発の最終的な結果は Java クラスの形式のプログラムです。

Process Designer Express は、配置までコラボレーション・テンプレート情報を System Manager に保管します。コラボレーションが配置されると、コラボレーション情報が InterChange Server Express で使用可能になり、コラボレーションがトリガー・イベントを受信したときにこの情報にアクセスできます。Process Designer Express の詳細については、17 ページの『第 2 章 Process Designer Express の概要』を参照してください。

### コラボレーション・テンプレートの構築

Process Designer Express を使用してコラボレーション・テンプレートを構築するには、以下の 2 段階の開発プロセスを実行します。

- アクティビティー・ダイアグラム
  - アクティビティー・ダイアグラムを作成します。アクティビティー・ダイアグラムとは、ビジネス・プロセスとそのフローをシンボルによってグラフィカルに記述するものです。
  - Activity Editor を使用して、ビジネス・プロセスの補足的な詳細部分を実装します。

  テンプレートをコンパイルすると、ダイアグラムとそれに関連するコードが実行可能 Java クラスに変換されます。
- メッセージ: メッセージを定義します。メッセージには、ロギング、トレース、および例外発生で使用されるテキストが保持されます。

テンプレートがコンパイルされると、メッセージの内容が System Manager 内のメッセージ・ファイルに格納されます。コラボレーションが配置されると、メッセージ・ファイルは InterChange Server Express にも格納され、実行時にアクセスされます。詳細については、197 ページの『第 9 章 メッセージ・ファイルの作成』を参照してください。

***アクティビティー・ダイアグラムの作成:*** コラボレーション・テンプレートは、以下の 2 つの要素から成ります。

- シナリオ。アクションのセットを指定します。

  コラボレーション・テンプレートを開発する際、最初にコラボレーションのビジネス・ロジックを 1 つ以上のシナリオに分割します。

  各コラボレーション・テンプレートは、シナリオと呼ばれる 1 つ以上の区分に分かれます。シナリオは、特定のフロー・トリガーに対してコラボレーションがどのように応答するかを正確に指定します。シナリオは、コラボレーションによって行われるアクションを記述するという点でメソッドに似ています。

  複数のシナリオを作成することができます。またコラボレーションのロジックすべてを 1 つのシナリオに入れることもできます。
- アクティビティー・ダイアグラム。個々のアクションを表すコード・フラグメントを使用して各アクションを記述します。

  各シナリオごとに、そのシナリオのプロセスをグラフィカルに記述するアクティビティー・ダイアグラムを作成します。アクティビティー・ダイアグラムはシナリオのグラフィカルな実装で、アクション、実行フロー、および外部呼び出しが含まれます。アクティビティー・ダイアグラムは、ビジネス・プロセスのモデル化の標準的な表記法である、UML (統一モデリング言語) に基づいています。ダイアグラムでは視覚的なプログラミングが使用されるため、シナリオの作成は簡単で、実際のコーディング量は少なくなります。

  アクティビティー・ダイアグラムのさまざまなステップは、個々のアクション、つまりコード・フラグメントです。

  各シナリオには少なくとも 1 つのトップレベル・ダイアグラムが含まれます。このダイアグラムは、実行時のシナリオのエントリー・ポイントを表し、そのシナリオの全体的なロジック・フローを含みます。サブダイアグラム は、シナリオのロジックの詳細を複数のネストされたレベルに分割します。

  アクティビティー・ダイアグラムの外見はフロー・チャートに似ています。しかしアクティビティー・ダイアグラムは、フロー・チャートと異なり、自身が表す実行可能 Java コードを作成できます。

図 1 に、アクティビティー・ダイアグラムの例を示します。

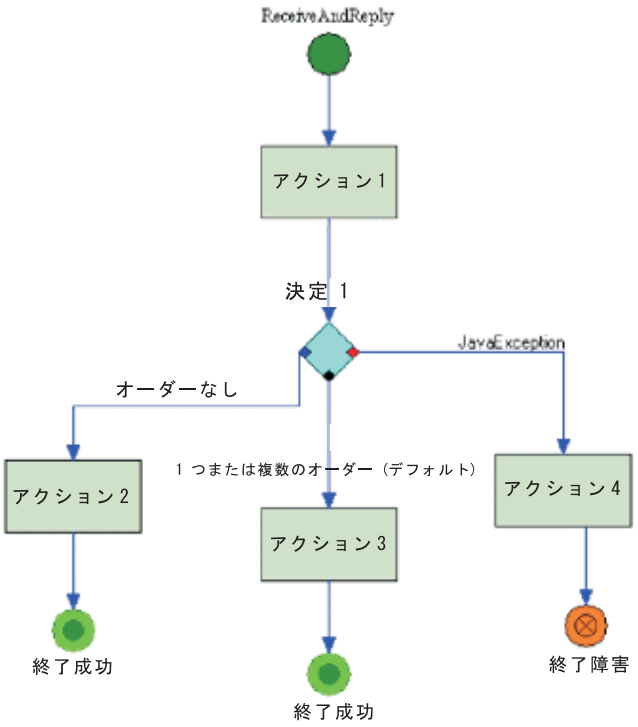



*図 1. アクティビティー・ダイアグラム*

アクティビティー・ダイアグラムの基本単位は アクションで、長方形で表されます。アクションでコラボレーションの作業単位を指定し、Java コード・フラグメントの作成および保管に使用されます。

アクティビティー・ダイアグラムは、実行時に可能な振る舞いをすべて表現します。図 1 のアクティビティー・ダイアグラムには複数の実行経路があります。*実行経路* は、一連のシンボルとリンクによって表され、一番上の開始シンボルから一番下の終了シンボルまで流れます。複数の出力リンクを持つシンボルは決定ノードで、コラボレーションが複数のロジック経路のうちいずれの経路に進むかを決定するポイントです。

***Java コード・フラグメントの実装:*** 各アクションには、Java プログラム言語のコード・フラグメント (アクティビティー定義 と呼ばれる) が含まれており、開発者は機能ブロックの形式でこのコード・フラグメントにさらにカスタム・コードを追加できます。Process Designer Express は、生成するコラボレーション・テンプレート・コードにアクティビティー定義を埋め込み、コラボレーション・オブジェクトが実行されるときに、コラボレーション・フローの一部としてその生成したコードを実行します。

必要であれば、カスタム・アクティビティー定義を追加できます。以下のことができます。

- 独自のアクティビティーを記述する。

  コラボレーション・テンプレートのビジネス・ロジックの多くは、Business Integration Express *collaboration API* への呼び出しで構成されています。ただし、アクティビティーはコラボレーション API への呼び出しに限定されておらず、記

述するすべての Java コードを含めることができます。独自のアクティビティー定義を追加するか、以下に示すいずれかの方法で既存のアクティビティーをカスタマイズします。

- アクティビティー定義の追加を支援する GUI である Activity Editor を使用する。従来の Java コードを記述、または*機能ブロック* を使用してプログラミング・ロジックをグラフィカルにモデル化できます。
- アクション・ノードの「Action Properties」ダイアログ・ボックスの「コード・フラグメント」ウィンドウに、Java コード・フラグメントを直接入力します (この機能を使用可能にするには、Process Designer のユーザー設定を設定する必要があります)。

- 別の Java クラスから (機能ブロックとして) コードをインポートする。

  Java クラスの外部パッケージをコラボレーションにインポートして、そのメソッドをアクションの中で使用できます。

  **注:** Process Designer Express によって生成されるクラスは、ICS の実行コンテキストの中で実行される必要があります。開発者自身の Java コードをインポートしたり記述したりできますが、コードはアクティビティー・ダイアグラムを強化するものでなければなりません。実行のフローを破壊したり過度のリソースを消費する操作を実行することは避けてください。

## コラボレーション・テンプレートのコンパイル

コラボレーション・テンプレートの定義を完了すると、つまりそのシナリオを定義し、アクティビティー・ダイアグラムを構築し、コード・フラグメントをカスタマイズして、メッセージ・ファイルを作成すると、テンプレート全体をコンパイルします。コラボレーション・コンパイル・プロセスでは、コラボレーション・ランタイムが使用する 3 種類のファイル (.class、.java、および .txt) が作成されます。

コラボレーションをコンパイルすると、これらのファイルは System Manager 内のユーザーの統合コンポーネント・ライブラリー (ICL) プロジェクトに自動的に作成されます。コラボレーション・オブジェクトをサーバーに配置すると、これらのファイルは *productDir*¥collaborations ディレクトリーに自動的に移動されます。表 2 に、ファイルと、コンパイルおよび配置後の場所を示します。

*表 2. コラボレーション・ファイル*

| ファイル・タイプ | 説明 | ロケーション |
|---|---|---|
| .class | コンパイル時に Process Designer Express によって生成される最終的な実行可能クラス・ファイルです。 | コンパイル後: *ICLProject*¥Templates¥Classes<br><br>配置後: classes¥UserCollaborations |
| .java | コード生成時に Process Designer Express によって作成されるソース・コード・ファイルです。 | コンパイル後: *ICLProject*¥Templates¥Src<br><br>配置後: classes¥UserCollaborations |

表 2. コラボレーション・ファイル (続き)

| ファイル・タイプ | 説明 | ロケーション |
|---|---|---|
| .txt | 開発時にテンプレートに追加したすべてのメッセージ・テキストを含むメッセージ・ファイルです。 | コンパイル後:<br>*ICLProject*¥Templates¥messages<br><br>配置後:<br>¥messages |

**要確認**

メッセージを変更するときは必ず Process Designer を使用し、メッセージ・テキスト・ファイルを直接変更することは絶対に行わないでください。コラボレーションが配置された後、このファイルはランタイム環境で使用されます。これを直接変更すると、エラーが発生する場合があります。

コラボレーション・テンプレートをコンパイルすると、System Manager を使用してコラボレーション・オブジェクトを作成し、これらのオブジェクトとテンプレートを InterChange Server Express に配置することができます。「*WebSphere InterChange Server システム・インプリメンテーション・ガイド*」を参照してください。

## コラボレーション・オブジェクト

コラボレーション・テンプレートにはコラボレーションの実行ロジックが含まれますが、コラボレーションを実行する前に以下のステップを行う必要があります。

1. コラボレーション・オブジェクトを作成します。

   コラボレーション・オブジェクトは、コラボレーション・テンプレートのインスタンスです。コラボレーション・オブジェクトを作成するには、System Manager を使用します。

2. コラボレーション・オブジェクトを構成します。

   コラボレーション・オブジェクトは、構成後に実行可能になります。コラボレーション・オブジェクトを構成するには、そのオブジェクトをコネクターまたは別のコラボレーション・オブジェクトにバインドします。その後、他の構成プロパティーを指定します。

   コラボレーション・オブジェクトが対話するオブジェクトを指定するプロセスのことをバインディング と呼びます。コラボレーション・オブジェクトは、以下の要素にバインドできます。

   - コラボレーション・オブジェクトが対話するコネクター、その他のコラボレーション・オブジェクト、またはアクセス・クライアント。コラボレーション・オブジェクトをバインドして構成プロパティーの値を指定すると、そのコラボレーション・オブジェクトは実行可能になります。

System Manager を使用してコラボレーション・オブジェクトを作成および構成する方法の詳細については、「*WebSphere InterChange Server システム・インプリメンテーション・ガイド*」を参照してください。

図 2 に、テンプレート OrderStatus から OrderStat というコラボレーション・オブジェクトを作成する方法を示します。

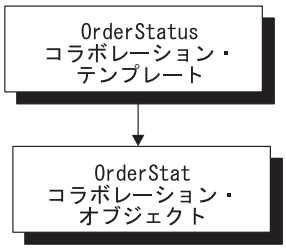



*図 2. コラボレーション・オブジェクトの作成*

OrderStatus コラボレーション・テンプレートは 2 つの定義済みポートと共に作成されています。これらのポートは、コラボレーション・オブジェクトがそのソース・オブジェクトおよび宛先オブジェクトと通信するときに使用されます。OrderStat コラボレーション・オブジェクトの構成の一環として、2 つの外部オブジェクトにオブジェクトをバインドします。図 3 では、OrderStat コラボレーション・オブジェクトが SAP コネクターと Vantive コネクターにバインドされています。

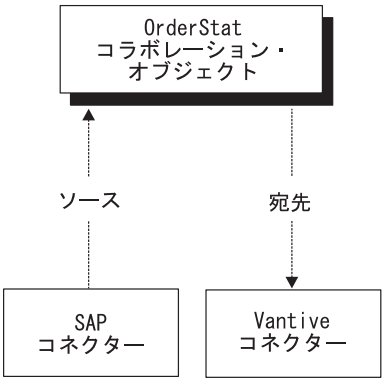



*図 3. コネクターにバインドされたコラボレーション・オブジェクト*

コラボレーション・オブジェクトをバインドし構成すると、System Manager を使用してランタイム環境でテストして配置できます。コラボレーション・オブジェクトは、単一スレッドまたはマルチスレッドで実行されるように構成できます。1 つのスレッドは 1 つのトリガー・イベントを扱います。複数のトリガー・イベントを並行処理させるためには、コラボレーション・オブジェクトをマルチスレッド・モードで実行してください。

## 長期存続ビジネス・プロセスとしてのコラボレーション

コラボレーション・オブジェクトは長期存続ビジネス・プロセスとして配置でき、ビジネス・プロセス間の非同期通信が可能になります。この結果、ビジネス・プロセスをより長い期間にすることができます。長期存続ビジネス・プロセスでは、イベント・フロー・コンテキスト (グローバル・テンプレートや Process Designer

Express で作成されたポート変数およびビジネス・オブジェクト変数、実行時ワークフロー情報を含む) はサービス呼び出しの間持続します。

非同期インバウンドおよび同期サービス呼び出しで長期存続ビジネス・プロセスのパラメーターをさらに定義するために、サービス呼び出しタイムアウト値を指定できます。

コラボレーション・オブジェクトを長期存続ビジネス・プロセスとして使用する計画がある場合は、それに応じてコラボレーション・テンプレートを構成する必要があります。コラボレーション・テンプレートを作成する前に、63 ページの『長期存続ビジネス・プロセスの設計』を参照してください。テンプレートを設計後、長期存続ビジネス・プロセスをサポートするのに必要な特別な構成タスクの詳細については、79 ページの『第 4 章 コラボレーション・テンプレートの構築』を参照してください。

## コラボレーションと WebSphere Business Integration Express システム

WebSphere Business Integration Express システムでは、ビジネス・オブジェクトを使用して、1 つのアプリケーションから別のアプリケーションに、データとアクションの要求を伝達します。コラボレーション・オブジェクトの中のシナリオが特定のビジネス・オブジェクトとアクション (動詞) を受け取ると、コラボレーションの実行が開始されます。ビジネス・オブジェクトと動詞のこの組み合わせをコラボレーションが受け取ることにより、シナリオの実行が開始されます。この組み合わせを、*フロー・トリガー* と呼びます。コラボレーションの開発者は、コラボレーション・テンプレートの設計部分として、各シナリオのフロー・トリガーとして機能するビジネス・オブジェクト (と動詞) を指定します。さらに、コラボレーション・オブジェクトの構成部分として、コラボレーションの受信ポートをフロー・トリガー用の特定のソースにバインドします。コラボレーションが受け取るフロー・トリガーのタイプは、フロー・トリガーを受信ポートに送信するソースのタイプによって決まります。

表 3 に示すように、フロー・トリガーのタイプは、複数あるタイプのうち、受信ビジネス・オブジェクトのソースに基づいて決まります。

*表 3. フロー・トリガーのタイプ*

| フロー・トリガー | 受信ビジネス・オブジェクトのソース |
|---|---|
| トリガー・イベント | コネクターまたは別のコラボレーション |
| トリガー・アクセス呼び出し | アクセス・クライアント (ICS 内のサーバー・アクセス・インターフェースを経由) |

**注:** アクセス・クライアントとは、サーバー・アクセス・インターフェース API を通じてコラボレーションの実行を要求できる外部プロセスです。詳細については、「*アクセス開発ガイド*」を参照してください。

フロー・トリガーの代表的なソースはコネクターです。そのため、用語「トリガー・イベント」はコラボレーションの受信ビジネス・オブジェクトを意味することがよくあります。例えば、「テンプレート定義」ウィンドウには、「ポートおよびトリガー・イベント」というタブがあります。このタブからは、コラボレーショ

ン・ポートを定義し、トリガー・イベントをシナリオに割り当てることができます。このタブとタブ内の関連タブのタイトルには「トリガー・イベント」という用語が含まれていますが、このタブは、トリガー・イベントとトリガー・アクセス呼び出しという 2 種類のフロー・トリガーの割り当てを扱います。シナリオがコネクターからビジネス・オブジェクトを受け取るときには、そのフロー・トリガーはトリガー・イベントです (タブ名が示すとおりです)。しかしシナリオがアクセス・クライアントからビジネス・オブジェクトを受け取るときには、フロー・トリガーはトリガー・アクセス呼び出しです。後者の場合にも、「ポートおよびトリガー・イベント」テーブルを使用して、シナリオにトリガー・アクセス呼び出しを割り当てます。

コラボレーションのフロー・トリガーのタイプは、次に示すように実際にコラボレーション・オブジェクトのポートが構成されるまでは決まりません。

- 内部ポート: ポートがコネクターにバインドされている場合、ポートはトリガー・イベントの形式でビジネス・オブジェクトを受け取ります。
- 外部ポート: ポートがアクセス・クライアントにバインドされている場合、ポートはトリガー・アクセス呼び出しの形式でビジネス・オブジェクトを受け取ります。

コラボレーション・オブジェクトの構成方法の詳細については、「*WebSphere InterChange Server システム・インプリメンテーション・ガイド*」を参照してください。「テンプレート定義」ウィンドウの「ポートおよびトリガー・イベント」タブの詳細については、94 ページの『ポートおよびトリガー・イベントの定義 (「ポートおよびトリガー・イベント」タブ)』を参照してください。

## コラボレーション開発用のツール

コラボレーション開発のプラットフォームは Windows 2000 です。コラボレーションは Java で記述されます。表 4 に、WebSphere Business Integration Express がコラボレーション開発用に提供するツールのリストを示します。

*表 4. コラボレーション開発用のツール*

| ツール | 説明 | 詳細 |
|---|---|---|
| Process Designer Express | コラボレーション・テンプレートを開発する際に役立つグラフィカルなツールです。 | 12 ページの『Process Designer Express』 |
| WebSphere Business Integration Express コラボレーション API | 生成されたコラボレーション・コードをカスタマイズすることのできる一連の Java クラスです。(API のメソッドには、Activity Editor 機能ブロックからもアクセスできます。) | 12 ページの『コラボレーション API』 |
| System Manager | コラボレーション・オブジェクトを作成し、構成するためのグラフィカルなウィンドウが用意されているツールです。 | 13 ページの『System Manager』 |
| 統合テスト環境 (Test Connector) | ビジネス・プロセスをテストするツール・セット。Test Connector ツール (統合テスト環境で、またスタンドアロン・ツールとして使用可能) を使用して汎用コネクターをシミュレートし、コラボレーションの設計を容易にテストできるようにします。 | 14 ページの『Test Connector』 |

## Process Designer Express

コラボレーション・テンプレートの作成、編集、コンパイル、および削除を実行するには、Process Designer Express を使用します。既存のテンプレートを変更するときには、Process Designer Express を使用します。テンプレートのプロパティーを編集したり、シナリオやアクティビティー・ダイアグラムの追加または編集を行うことができます。

Process Designer Express インターフェースの詳細については、 17 ページの『第 2 章 Process Designer Express の概要』を参照してください。

## コラボレーション API

Business Integration Express コラボレーション API には、コラボレーション・テンプレートで使用可能なメソッドを持つ数個のクラスが用意されています。以下のセクションでは、共通のコラボレーションの機能性を高める点で、これらのクラスが持つ役割について説明します。

**注:** コラボレーション API には、従来の Java 呼び出しと、Activity Editor でサポートされる機能ブロックの両方からアクセスできます。詳細については、 153 ページの『第 6 章 Activity Editor の使用』を参照してください。

### コラボレーション・オブジェクトとの対話

BaseCollaboration クラスは、構成プロパティーの値の取得、ログ・ファイルとトレースへのメッセージの書き込みなど、コラボレーションの全般的な振る舞いと機能を定義します。

コラボレーション・テンプレートを作成すると、BaseCollaboration のサブクラスである Java クラスを作成することになります。このため、コラボレーションは BaseCollaboration のメソッドをすべて継承します。これらのメソッドにより、コラボレーションは以下のような操作を実行することができます。

- 構成プロパティーの値を取得します。
- 例外を発生します。
- 情報メッセージ、警告メッセージ、およびエラー・メッセージをログ・ファイルに書き込みます。

BaseCollaboration クラスのメソッドの詳細については、 363 ページの『第 22 章 BaseCollaboration クラス』を参照してください。

### ビジネス・オブジェクトとの対話

一般にコラボレーションは、ビジネス・オブジェクトと対話してビジネス・オブジェクトを操作します。BusObj クラスのメソッドを使用すると、コラボレーションは以下のような操作を実行できます。

- ビジネス・オブジェクトの名前を取得します。
- ビジネス・オブジェクトのキー値を取得します。
- 階層ビジネス・オブジェクトに含まれている子ビジネス・オブジェクトの数を取得します。
- 2 つのビジネス・オブジェクトの属性値が等しいかどうかをテストします。

- 1 つのビジネス・オブジェクトの属性値を別のビジネス・オブジェクトにコピーします。

BusObj クラスのメソッドの詳細については、381 ページの『第 23 章 BusObj クラス』を参照してください。

### ビジネス・オブジェクト配列との対話

コラボレーションは、ビジネス・オブジェクトの属性の値を取得したり設定することがあります。ビジネス・オブジェクトが階層的である場合、1 つ以上の属性は子ビジネス・オブジェクトであるか、または多くの場合には子ビジネス・オブジェクトの配列です。子ビジネス・オブジェクトは、コラボレーションには配列として認識されます。

BusObjArray クラスのメソッドを使用することで、コラボレーションはビジネス・オブジェクトの配列と対話して操作することができます。これらのメソッドは、以下のような操作を実行します。

- 配列の要素を設定または取得します。
- 配列を別の配列にコピーします。
- ビジネス・オブジェクトを配列に追加します。
- 配列の要素数を取得します。

BusObjArray クラスのメソッドの詳細については、401 ページの『第 24 章 BusObjArray クラス』を参照してください。

### 例外との対話

コラボレーションでエラーが発生すると、そのコラボレーションまたはコラボレーション・ランタイム環境は例外を発生させます。例外は、CollaborationException クラスのオブジェクトの中に含まれます。このクラスを使用すると、コラボレーション・オブジェクトは例外オブジェクトと対話して、以下の操作を実行できます。

- 例外のタイプまたはサブタイプを取得します。
- 例外メッセージを取得します。

CollaborationException クラスのメソッドの詳細については、437 ページの『第 28 章 CollaborationException クラス』を参照してください。

## System Manager

System Manager は、ICS とそのリポジトリーとのインターフェースを提供するグラフィカルなツールです。これにより、以下のコラボレーション関連タスクを実行できます。

- コラボレーション・オブジェクトを作成します。
- コラボレーション・オブジェクトをバインドします。
- コラボレーション・オブジェクトのコラボレーション特定のプロパティーを設定します。
- コラボレーションの設計をテストします (Test Connector ツールを使用)。
- コラボレーション・オブジェクトをランタイム環境に配置します。

System Manager を使用してコラボレーション・オブジェクトを作成、構成、および配置する方法の詳細については、「*WebSphere InterChange Server システム・インプリメンテーション・ガイド*」を参照してください。

### Test Connector

Test Connector は、コラボレーションとコネクターをテストするためのグラフィカルなツールです。統合テスト環境内およびスタンドアロン・ツールとしての両方で使用可能です。

**注:** アクセス・クライアントをテストする場合、統合テスト環境で Test Connector を使用する必要があります。

Test Connector ツールは実際のコネクターをシミュレートします。これにより、トリガー・イベントやサービス呼び出し要求を送信することによって、コラボレーションの設計を簡単にテストすることができます。Test Connector の使用方法の詳細については、「*WebSphere InterChange Server システム・インプリメンテーション・ガイド*」を参照してください。

## 開発プロセスの概要

このセクションでは、次のハイレベル・ステップを含むコラボレーション開発プロセスの概要を説明します。

1. WebSphere Business Integration Express ソフトウェア (Java Development Kit およびその他の必要なサード・パーティー製品を含む) をインストールし、セットアップします。インストールおよび構成の具体的な説明については、「*WebSphere Business Integration Server Express インストール・ガイド*」を参照してください。
2. コラボレーションを設計、実装します。

### コラボレーションの開発ステージ

コラボレーションの開発ステージは次のとおりです。

1. コラボレーションによって実装されるビジネス・プロセスを設計します。
2. ビジネス・オブジェクト定義を作成します。
3. コラボレーション・テンプレートを作成し、メタ情報と定義を含めます。
4. 各シナリオとそのアクティビティー・ダイアグラムを作成します。
5. 必要なコード・フラグメントをカスタマイズします。
6. メッセージ・テキストを作成します。
7. テンプレートをコンパイルします。
8. コラボレーション・テンプレートからコラボレーション・オブジェクトを作成します。
9. コラボレーションをテストし、デバッグします。
10. コラボレーションをランタイム環境に配置します。

15 ページの図 4 に、コラボレーション開発プロセスの概要を視覚的に示します。また、特定のトピックについての情報が何章に記載されているかについても示します。

**注:** コラボレーションの開発タスク全体のうち一部は、コラボレーション・テンプレート開発の狭義の範囲外になるため、この資料では説明していません。これらの各タスクについての参照先として、WebSphere Business Integration Server Express ライブラリーの当該資料を 図 4 に示します。

コラボレーションの開発をチームで行う場合、コラボレーション開発の主要な作業をチームのメンバーが並行して行うことができます。

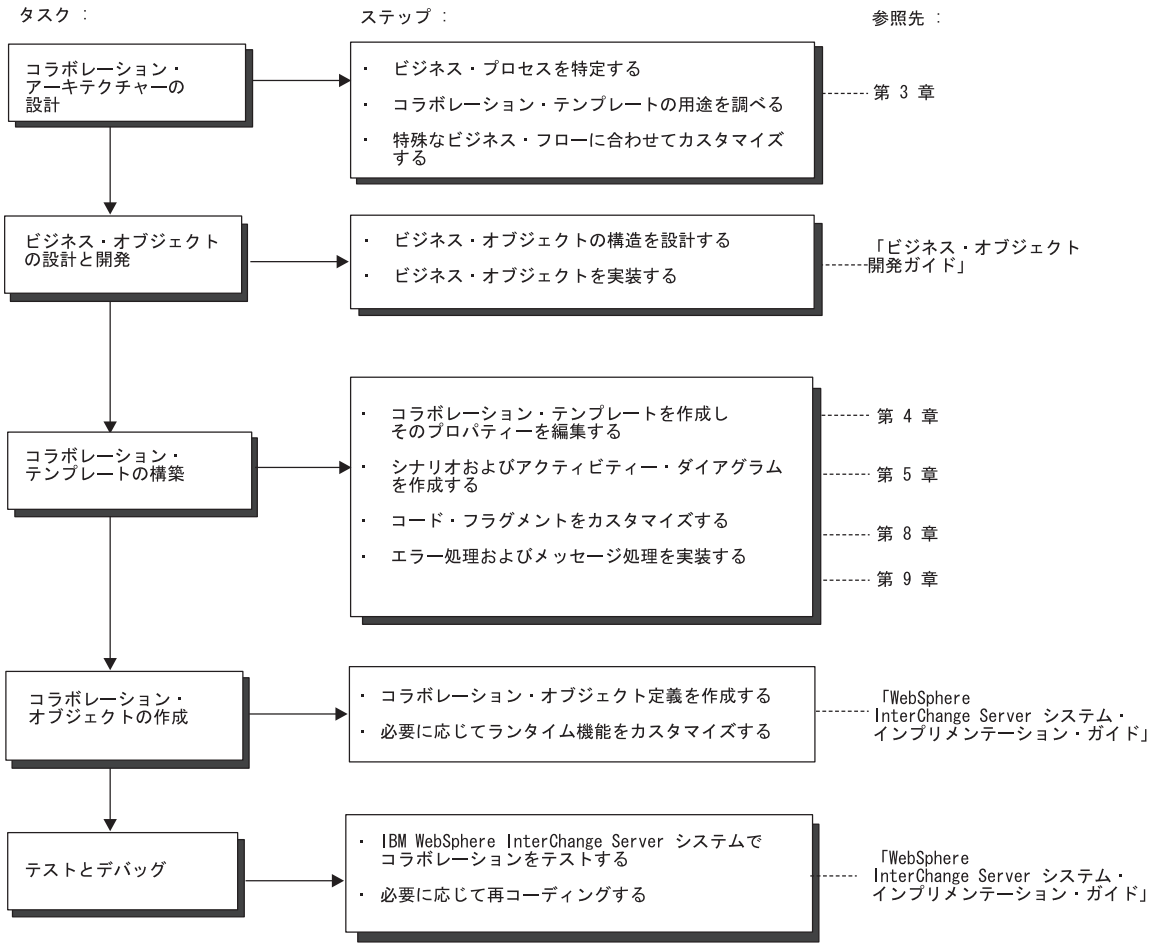



*図 4. コラボレーション開発タスクの概要*

# 第 2 章 Process Designer Express の概要

Process Designer Express を使用すると、以下のコラボレーション開発タスクを実行できます。

- 「テンプレート定義」ウィンドウで、テンプレート定義を作成、編集、コンパイル、または削除します。
- ダイアグラム・エディターを通じて、コラボレーション・テンプレートのシナリオのアクティビティー・ダイアグラムを定義または編集します。

この章では、Process Designer Express の概要を示します。インターフェースについて説明し、Process Designer Express のウィンドウ、メニュー、およびツールバーを使用してコラボレーション開発に必要なタスクを実行します。

## Process Designer Express の開始

Process Designer Express を始動するメソッドは、新規コラボレーション・テンプレートを作成している場合か、既存のコラボレーション・テンプレートを編集している場合かによって異なります。

**要確認**

Process Designer Express を始動するには、まず System Manager が実行中であることを確認する必要があります。

System Manager 内から Process Designer Express を始動するには、表 5 に示すように、いくつかの方法があります。

*表 5. System Manager 内からの Process Designer Express の始動*

| メソッド | 結果 |
|---|---|
| オブジェクト・ブラウザー・ビューからコラボレーション・テンプレート・フォルダーを右マウス・ボタン・クリックして、コンテキスト・メニューから「新規コラボレーション・テンプレートの作成」をクリックします。 | Process Designer Express が開き、「新規テンプレート」ダイアログ・ボックスが表示されます。 |
| コラボレーション・テンプレート・フォルダー内でコラボレーション・テンプレートをダブルクリックします。 | Process Designer Express が開き、ダブルクリックしたテンプレート定義が表示されます。 |
| 統合コンポーネント・ライブラリー内のコンポーネントへのユーザー・プロジェクト・ショートカットを作成し使用します。 | Process Designer Express が開き、ショートカットに関連付けされたテンプレート定義が表示されます。 |

「スタート」メニューから Process Designer Express を起動することもできます。「スタート」—>「プログラム」—>「IBM WebSphere Business Integration Express」—>「Toolset Express」—>「Development」—>「Process Designer Express」の順にクリックします。

Process Designer Express は、専用の連結可能なウィンドウの中に表示されます。一度に複数の Process Designer Express のインスタンスを立ち上げて、複数のコラボレーション・テンプレート定義を編集できます。

## Process Designer Express レイアウト

Process Designer Express を始動すると、図 5 に示されているようなメインウィンドウがデフォルトで表示されます。

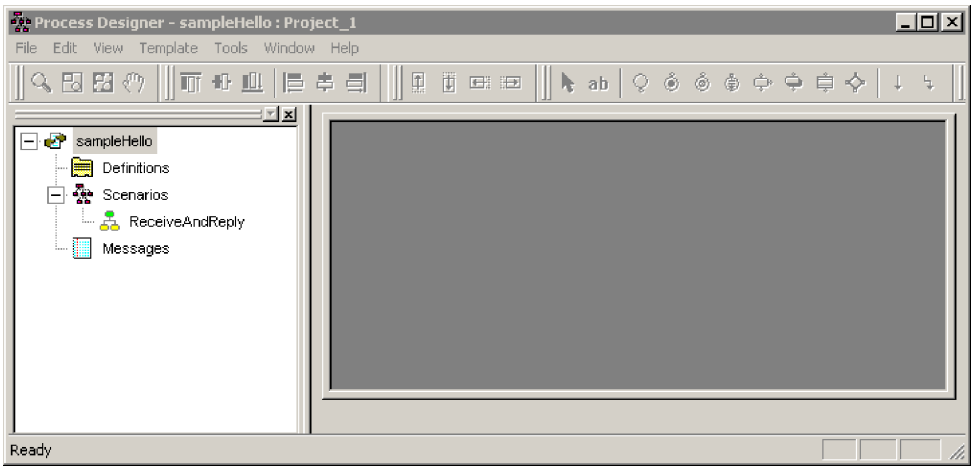



*図 5. Process Designer Express メインウィンドウ*

Process Designer Express ウィンドウのレイアウトは、以下の領域から構成されます。

- テンプレート・ツリー (連結可能)

  左側のテンプレート・ツリー・ビューには、コラボレーション・テンプレートの定義、シナリオ、およびメッセージが階層的にリスト表示されます。ツリー内の既存のシナリオ・ノードの横のプラス記号 (+) をクリックすると、そのサブツリーが展開されます。既存のシナリオとサブダイアグラムが存在すれば表示されます。

- メインウィンドウの作業域。ここは空白とするか、または以下のウィンドウを表示させることができます。
  - 「テンプレート定義」ウィンドウ

    このウィンドウには、コラボレーション・テンプレート、変数宣言、またはポート情報に関する一般情報が示されます。詳細については、 19 ページの『「テンプレート定義」ウィンドウ』を参照してください。
  - 「ダイアグラム」エディター・ウィンドウ

このウィンドウは、アクティビティー・ダイアグラムのノードを表示するために使用されます。詳細については、21 ページの『「ダイアグラム」エディター・ウィンドウ』を参照してください。

– 「テンプレート・メッセージ」ウィンドウ

このウィンドウは、テンプレートのメッセージ・ファイルを書き込みまたは編集するために使用されます。詳細については、22 ページの『「テンプレート・メッセージ」ウィンドウ』を参照してください。

「テンプレート定義」ウィンドウと「テンプレート・メッセージ」ウィンドウ、およびダイアグラム・エディターは、作業域内で、最小化または最大化することが可能で、ユーザー指定のサイズで開くこともできます。詳細については、29 ページの『作業域内でのウィンドウの表示』を参照してください。

- コンパイル出力ウィンドウ

コンパイル出力ウィンドウ (単に出力ウィンドウとも呼ばれます) には、コラボレーション・テンプレートのコンパイルの結果が表示されます。このウィンドウは、コラボレーション・テンプレートをコンパイルすると自動的に表示されます。詳細については、102 ページの『コラボレーション・テンプレートのコンパイル』を参照してください。

**注:** Process Designer Express のメインウィンドウの構成は、Process Designer Express を閉じると保管されます。このため、構成への変更は、Process Designer Express を次回開くと表示されます。(詳細については、27 ページの『メインウィンドウのカスタマイズ』を参照してください。) 図 5 に、メインウィンドウのデフォルト構成を示します。Process Designer Express の前回の起動時にこの構成が変更されていると、メインウィンドウはこの図と異なることがあります。

Process Designer Express の機能には、次のいずれかの方法でアクセスできます。

- ウィンドウ上部のプルダウン・メニュー
- ツールバーのアイコン
- コンテキストに依存したメニュー (マウスの右ボタンをクリックすると表示されるポップアップ・メニュー)
- キーボード・ショートカット

# Process Designer Express ウィンドウ

## 「テンプレート定義」ウィンドウ

「テンプレート定義」ウィンドウには、コラボレーション・プロパティーを定義するための以下の 3 つのタブがあります。

- 「一般」タブ: コラボレーション・テンプレートに関する一般情報 (名前、説明、最小トランザクション・レベル、パッケージなど) を指定するためのフィールドがあります。
- 「宣言」タブ: 変数宣言を指定するためのフィールドがあります。

- 「プロパティー」タブ: ユーザー定義のコラボレーション・テンプレート・プロパティーの名前、タイプ、値を指定するフィールドを示します。
- 「ポートおよびトリガー・イベント」タブ: ポート名とそれに関連するビジネス・オブジェクトおよび動詞を指定するためのフィールドがあります。

**要確認**

実稼働環境で実行中のコラボレーション・オブジェクトにビジネス・オブジェクトをバインドした後は、Business Object Designer または System Manager を使用してリポジトリーへのビジネス・オブジェクトの追加やリポジトリー内のビジネス・オブジェクトの変更または削除を行わないようにしてください。詳細については、94 ページの『ポートおよびトリガー・イベントの定義 (「ポートおよびトリガー・イベント」タブ)』を参照してください。

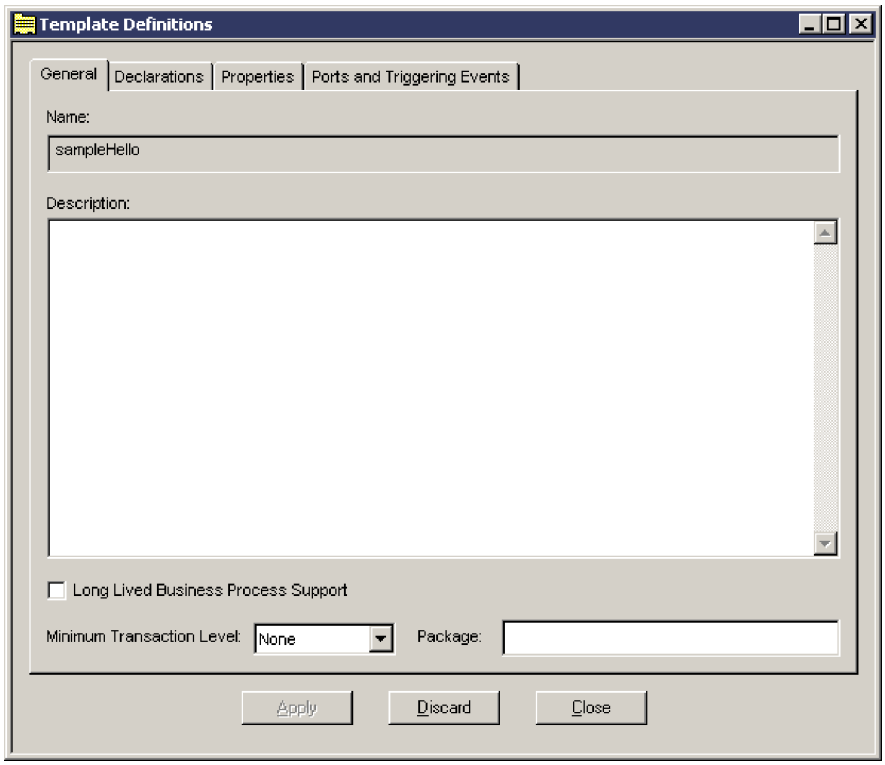



*図 6. 「テンプレート定義」ウィンドウ*

メインウィンドウの作業域に表示される「テンプレート定義」ウィンドウは、以下の方法で開くことができます。

- テンプレート・ツリー・ビューで、「定義」をダブルクリックします。
- テンプレート・ツリー・ビューで、「定義」を選択し、右マウス・ボタンでクリックして「テンプレート定義を開く」を選択します。
- 「テンプレート」プルダウン・メニューから、「テンプレート定義を開く」を選択します。
- ショートカット・キーの組み合わせ Ctrl+T を使用します。

「テンプレート定義」ウィンドウには、「適用」および「破棄」ボタンがあります。これらのボタンは、現在どのタブが表示されていてもウィンドウの下部に表示されます。「適用」ボタンによりテンプレートに対する変更が確定しますが、保管されません。すべての変更を保管するには、「ファイル」—> 「保管」コマンドを使用する必要があります。「破棄」ボタンを使用して、まだ保管していない変更内容を破棄し、前に保管した定義に復帰できます。

**注:** 「適用」および「破棄」ボタンは、現在表示されているタブのみではなく、すべてのタブに含まれるデータに影響します。

## 「ダイアグラム」エディター・ウィンドウ

ダイアグラム・エディターは Process Designer Express の中のツールです。このツールを使用してアクティビティー・ダイアグラムを作成したり編集したりできます。このウィンドウは、アクティビティー・ダイアグラムを開くとメインウィンドウの作業域に表示されます。ダイアグラム・エディターでは、ノード、サービス呼び出し、遷移リンクなどをアクティビティー・ダイアグラムに追加することができます。つまり、項目の配置変更、テキスト・ラベルとフォントの追加や編集、個々のコンポーネント・プロパティーの追加や変更を行うことができます。

ダイアグラム・エディターを開くには、以下の方法があります。

- テンプレート・ツリー・ビューで、シナリオ・ノードを展開してからダイアグラムの名前をダブルクリックするか、ダイアグラムの名前を右マウス・ボタンでクリックして「ダイアグラムを開く」を選択します。
- テンプレート・ツリー・ビューで、シナリオ・ノードを展開してからダイアグラムの名前を選択し、「テンプレート」プルダウン・メニューから「ダイアグラムを開く」を選択します。
- 「テンプレート」プルダウン・メニューから、「すべてのダイアグラムを開く」を選択します。

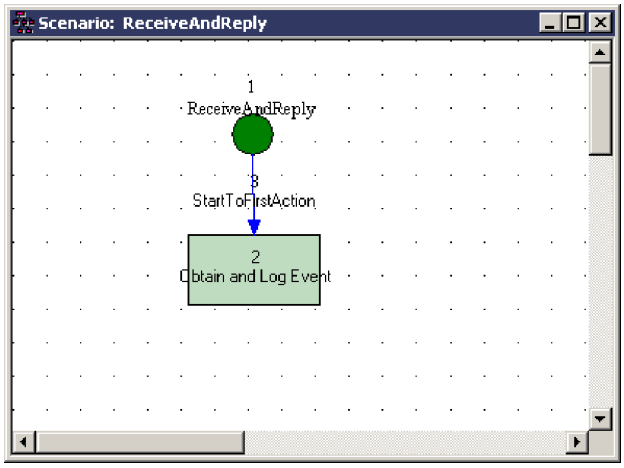



*図 7. 「ダイアグラム」エディター・ウィンドウ*

ダイアグラム・エディターの使用方法の詳細については、102 ページの『アクティビティー・ダイアグラムの作成』を参照してください。

## 「テンプレート・メッセージ」ウィンドウ

「テンプレート・メッセージ」ウィンドウには、テンプレートのメッセージ・ファイルを記述したり編集したりできる領域があります。テンプレートをコンパイルすると、メッセージ・テキストが System Manager 内の該当する統合コンポーネント・ライブラリー・プロジェクトの Template¥messages ディレクトリーに書き込まれます。

「テンプレート・メッセージ」ウィンドウを開くには、以下の方法があります。

- テンプレート・ツリー・ビューで、「メッセージ」をダブルクリックします。
- テンプレート・ツリー・ビューで、「メッセージ」を選択し、右マウス・ボタンでクリックして「メッセージを開く」を選択します。
- 「テンプレート」プルダウン・メニューから、「テンプレート・メッセージを開く」を選択します。
- ショートカット・キーの組み合わせ `Ctrl+M` を使用します。

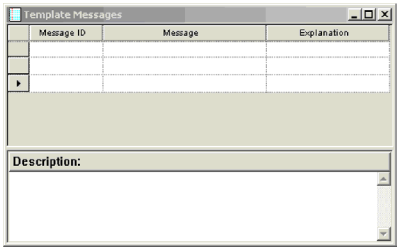



*図 8. 「テンプレート・メッセージ」ウィンドウ*

# Process Designer Express メニュー

Process Designer Express では、使用可能なメニューとメニュー・オプションは、作業域の表示内容によって異なります。以下のセクションでは、作業域が空であるかまたは「テンプレート定義」ウィンドウを表示しているときの、Process Designer Express のメイン・メニューについて説明します。

- 『「ファイル」メニューの機能』
- 24 ページの『「表示」メニューの機能』
- 25 ページの『「テンプレート」メニューの機能』
- 26 ページの『「ウィンドウ」メニューの機能』

**注:** 作業域にダイアグラム・エディターが表示されている場合は、「編集」メニューのオプションが使用可能になり、他の一部のメニューでも異なるオプションが使用可能になります。これらの機能のほとんどはアクティビティー・ダイアグラムで作業するときに使用します。詳細については、108 ページの『ダイアグラム・エディター機能へのアクセス: Process Designer Express メニュー』を参照してください。

## 「ファイル」メニューの機能

作業域が空であるか、「テンプレート定義」ウィンドウまたは「テンプレート・メッセージ」ウィンドウが表示されている場合には、「ファイル」メニューには以下のオプションが表示されます。

- **「新規」**—新規コラボレーション・テンプレートを作成します。
- **「開く」**—既存のコラボレーション・テンプレート定義を開きます。以下の 2 つのオプションがあります。
  - **「プロジェクトから」**—コラボレーション・テンプレートを統合コンポーネント・ライブラリーのユーザー・プロジェクトから開きます。
  - **「ファイルから」**—ファイル・システムに保管されている .cwt ファイルからコラボレーション・テンプレートを開きます。
- **「閉じる」**—コラボレーション・テンプレートを閉じます。
- **「保管」**—現在のコラボレーション・テンプレートを保管します。以下の 2 つのオプションがあります。
  - **「プロジェクトに」**—コラボレーション・テンプレートを統合コンポーネント・ライブラリーのユーザー・プロジェクトに保管します。
  - **「ファイルに」**—ファイル・システムに格納されている .cwt ファイルにコラボレーション・テンプレートを保管します。
- **「別名保管」**—現在のコラボレーション・テンプレートを別の名前で保管します。以下の 2 つのオプションがあります。
  - **「プロジェクトに」**—コラボレーション・テンプレートを統合コンポーネント・ライブラリーのユーザー・プロジェクトに保管します。
  - **「ファイルに」**—ファイル・システムに格納されている .cwt ファイルにコラボレーション・テンプレートを保管します。
- **「削除」**—「プロジェクトからテンプレートを削除」ダイアログ・ボックスが表示され、削除するコラボレーション・テンプレートを選択できます。
- **「コンパイル」**—コラボレーション・テンプレートをコンパイルします。詳細については、102 ページの『コラボレーション・テンプレートのコンパイル』を参照してください。
- **「すべてコンパイル」**—プロジェクトのすべてのコラボレーション・テンプレートをコンパイルするか、コンパイル対象のサブセットを指定できます。詳細については、103 ページの『複数のコラボレーション・テンプレートのコンパイル』を参照してください。
- **「インポート」**—テンプレート定義にファイルをインポートします。Process Designer Express Importer は、BPEL ファイルと UML ファイル (XMI 形式) をインポートできます。これらのファイルは、必要に応じて InterChange Server Express テンプレート・ファイルに変換されます。
- **「エクスポート」**—ファイルをエクスポートします。テンプレート・ファイルは、UML (XMI 形式) または BPEL 形式にエクスポートできます。Process Designer Express Exporter ツールは、必要なすべての形式変換を実行します。
- **「終了」**—Process Designer Express を閉じます。

作業域にダイアグラム・エディターが表示されている場合は、「ファイル」メニューに**「ページ設定」**、**「印刷プレビュー」**、および**「印刷」** オプションが追加表示されるので、アクティビティー・ダイアグラムを印刷できます。作業域に「テンプレート・メッセージ」ウィンドウが表示されている場合、「ファイル」メニューには、メッセージ・ファイルを印刷するための追加のオプションが表示されます。

## 「編集」メニューの機能

「編集」メニュー・オプションを使用できるのは、ダイアグラム・エディターがアクティブである場合のみです。オプションには、標準 Windows 編集コマンド (**「元に戻す」**、**「やり直し」**、**「切り取り」**、**「コピー」**、**「貼り付け」**、**「削除」**など) および以下の特別な Process Designer Express オプションなどがあります。

- **「すべて選択」**—現在のアクティビティー・ダイアグラム内のノードをすべて選択します。
- **「ID を検索」**—アクティビティー・ダイアグラム ID を検索します。
- **「テキストを検索」**—現在のアクティビティー・ダイアグラム内のテキストを検索します。
- **「テキストを置換」**—現在のアクティビティー・ダイアグラム内のテキストを検索および置換できます。
- **「プロパティー」**—選択されたシンボルのプロパティーを編集します。このオプションを使用できるのは、ワークスペースでシンボルが選択されている場合のみです。
- **「フォント」**—アクティビティー・ダイアグラムで選択したシンボルのテキスト・ラベルのフォントおよび色を変更します。現在選択しているシンボルおよびリンクのフォントを変更できます。また、すべてのコンポーネントのフォントを変更することもできます。この場合は、まず**「すべて選択」**オプションを使用してダイアグラム内のすべてのコンポーネントを選択し、その次にフォントの変更を適用します。このオプションを使用できるのは、ワークスペースでシンボルが選択されている場合のみです。

## 「表示」メニューの機能

「表示」メニューの機能は、Process Designer Express を最初に開き、作業域の表示がアクティビティー・ダイアグラムの外観に関連している場合に有効です。「表示」メニューの機能の多くは、オンとオフを切り替えることができます。

- **「設定」**—「ユーザー設定」ダイアログ・ボックスを開きます。これを使用すると、Process Designer Express での項目の表示を指定できます。
- **「テンプレート・ツリー」**—このオプションがオンの場合、「Process Designer Express」ウィンドウの左側にテンプレート・ツリー・ビューが表示されます。
- **「出力ウィンドウ」**—このオプションをオンにすると、Process Designer Express はテンプレートのコンパイル結果を表示します。
- **「ツールバー」**—Process Designer Express のさまざまなツールバーの表示を制御します。サブメニュー・オプションには以下のものが含まれています。
  - **「標準」**—このオプションをオンにすると、Process Designer Express により標準ツールバー用のボタンが表示されます。
  - **「シンボル」**—このオプションをオンにすると、Process Designer Express によりシンボル・ツールバー用のボタンが表示されます。
  - **「位置合わせ」**—このオプションをオンにすると、Process Designer Express により位置合わせツールバー用のボタンが表示されます。
  - **「微調整」**—このオプションをオンにすると、Process Designer Express により微調整ツールバー用のボタンが表示されます。

- **「ズーム」**—このオプションをオンにすると、Process Designer Express によりズーム/パン・ツールバー用のボタンが表示されます。
- **「プログラム」**—このオプションをオンにすると、Process Designer Express により他の Business Integration Express プログラムにアクセスするためのボタンが表示されます。

- **「ステータス・バー」**—このオプションをオンにすると、Process Designer Express により、メインウィンドウの下端に 1 行の状況メッセージが表示されます。

さらに、ダイアグラム・エディターがアクティブである場合、Process Designer Express では、以下のダイアグラム・オプションを使用できます。

- **「タイプを表示」**—オンの場合、シンボルのタイプが表示されます。このオプションを使用すると、ノードをその形状から認識することができます。
- **「UID を表示」**—オンの場合、各シンボルの固有 ID (UID) が表示されます。
- **「ラベルを表示」**—オンの場合、ユーザー提供のシンボル・ラベルが表示されます。
- **「ロック (読み取り専用)」**—オンの場合、アクティビティー・ダイアグラムが読み取り専用モードになります。
- **「最新表示」**—アクティビティー・ダイアグラムの表示を最新にします。
- **「グリッド」**—オンの場合、ワークスペースのグリッド・ラインが表示されます。オフの場合、グリッド・ラインは表示されません。
- **「グリッドに合わせる」**—オンの場合、新しいシンボルがアクティビティー・ダイアグラムに表示されているときに、それらがグリッド・ラインの位置に自動的に合わせられます。
- **「グリッド・プロパティー」**—グリッド・プロパティーを設定できます。 (「角度合わせ」をアクティビティー・ダイアグラムに適用することはできませんが、オン/オフを切り替えることはできます。)
- **「ページ境界」**—ページ境界が破線として表示されます。
- 「ズーム・コマンド」: アクティビティー・ダイアグラムの拡大またはあるセクションへのズームを実行できます。また、ズーム・ツールバーからズーム・コマンドを実行することもできます。ズーミングの詳細については、190 ページの『シンボルのズームまたはパン』を参照してください。

## 「テンプレート」メニューの機能

作業域が空であるか、「テンプレート定義」ウィンドウまたは「テンプレート・メッセージ」ウィンドウが表示されている場合、「テンプレート」メニューには以下のオプションが表示されます。

- テンプレート・ツリー・ビューでシナリオ以外のオブジェクトが選択されている場合には、以下のオプションを使用することができます。
  - **「すべてのダイアグラムを開く」**—コラボレーション・テンプレート用に定義されているすべてのアクティビティー・ダイアグラムを開きます。
  - **「すべてのダイアグラムを閉じる」**—開いているアクティビティー・ダイアグラムすべてを閉じます。
  - **「新規シナリオ」**—「新規シナリオ」ダイアログ・ボックスが表示されます。

- **「テンプレート定義を開く」**—「テンプレート定義」ウィンドウが表示されます。ここで、コラボレーション・テンプレートのプロパティーを変更できます。
- **「テンプレート・メッセージを開く」**—「テンプレート・メッセージ」ウィンドウが表示されます。ここで、コラボレーション・テンプレートに関連するメッセージ・ファイルを定義または変更できます。

• テンプレート・ツリー・ビューでシナリオが選択されている場合には、以下の追加メニュー項目を使用することができます。

- **「ダイアグラムを開く」**—現在のシナリオのアクティビティー・ダイアグラムを開きます。
- **「シナリオの名前を変更」**—現在のシナリオの名前を変更できます。
- **「シナリオを削除」**—現在選択されているシナリオとそのアクティビティー・ダイアグラムを削除します。
- **「シナリオ定義を開く」**—シナリオ・レベルの変数を編集します。

• ダイアグラム・エディターが開かれている場合は、前述の項目の他に、以下のメニュー項目を使用できます。

- **「ダイアグラムのサイズを変更」**—アクティビティー・ダイアグラムの大きさを縦と横のページ・カウント単位で変更します。これは、アクティビティー・ダイアグラムを印刷するときに使用します。「ダイアグラム・サイズ」ダイアログには、ページ情報を数値で入力するための回転コントロールが含まれます。ダイアグラムのサイズは、用紙の向き (横長または縦長) と用紙の選択に直接的に関係します。
- **「ダイアグラムをテキスト・ファイルとして保管」**—現在のアクティビティー・ダイアグラムをテキスト・フォーマット (.txt) でファイルに保管します。

### 「ツール」メニューの機能

「ツール」メニューにより、他の Business Integration Express ツールを起動できます。オプションは以下のとおりです。

- **「Map Designer」**—Map Designer を開きます。
- **「Business Object Designer」**—Business Object Designer Express を開きます。
- **「Relationship Designer」**—Relationship Designer Express を開きます。

### 「ウィンドウ」メニューの機能

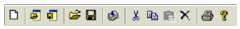
「ウィンドウ」メニューのプルダウン・オプションは、Multiple Document Interface (MDI) の標準的なウィンドウ表示機能をカバーします。タイル表示、カスケード表示、開いているウィンドウのアクティブ化などの表示機能の制御には、これらのオプションを使用します。

## Process Designer Express ツールバー

Process Designer Express には、実行する必要のある共通のタスクのためのツールバーが用意されています。これらのツールバーは連結可能で、メインウィンドウのパレットから切り離してメインウィンドウやデスクトップの上に浮動させることができます。

表 6 に、Process Designer Express に用意されているツールバーを示します。

*表 6. Process Designer Express ツールバー*

| ツールバー名 | ツールバーの外観 | 詳細 |
|---|---|---|
| 標準 |  | なし |
| シンボル |  | 108 ページの『シンボルの概要』 |
| 位置合わせ |  | 187 ページの『シンボルの位置合わせ』 |
| 微調整 |  | 189 ページの『シンボルの微調整』 |
| ズーム/パン |  | 190 ページの『シンボルのズームまたはパン』 |

# メインウィンドウのカスタマイズ

Process Designer Express では、メインウィンドウを以下の方法によってカスタマイズできます。

- 表示するウィンドウを選択する
- 連結可能なウィンドウを浮動させる
- 作業域内でのウィンドウの表示方法を選択する

## 表示するウィンドウを選択する

図 5 に示すように、Process Designer Express を最初に開いたときには、左側にテンプレート・ツリー・ビューが表示されます。作業域は右に表示され、中は空白です。出力ウィンドウは表示されません。メインウィンドウのオプションの外観は、「表示」メニューからカスタマイズできます。

表 7 には、「表示」プルダウン・メニューのオプションと、各オプションによって Process Designer Express のメインウィンドウの外観がどのように変わるかがまとめられています。

*表 7. メインウィンドウのカスタマイズ用の「表示」メニュー・オプション*

| 「表示」メニュー・オプション | 表示される要素 |
|---|---|
| テンプレート・ツリー | テンプレートの定義、シナリオ、およびメッセージが左側に表示されます。 |
| 出力ウィンドウ | 出力ウィンドウが、テンプレート・ツリー・ビュー (表示させる場合) と作業域の下に小さなウィンドウとして表示されます。 |

*表 7. メインウィンドウのカスタマイズ用の「表示」メニュー・オプション (続き)*

| 「表示」メニュー・オプション | 表示される要素 | |
|---|---|---|
| ツールバー | このメニューには、Process Designer Express の各ツールバーを表示するための以下のオプションがあります。 | |
| | 標準 | Process Designer Express パレット内のメイン・ツールバー。このツールバーには、ICS と接続/切断、サーバーまたはファイルからテンプレートを開く、テンプレートの保管およびコンパイル、テンプレートの切り取り、コピー、貼り付け、削除、および印刷の各ボタンが含まれます。 |
| | シンボル | ダイアグラム・シンボル・ツールバーには、アクティビティー・ダイアグラムに追加できるシンボルが含まれます。 |
| | 位置合わせ | 位置合わせツールバーには、アクティビティー・ダイアグラムのシンボルの位置合わせ機能が含まれます。 |
| | 微調整 | 微調整ツールバーには、選択されたアクティビティー・ダイアグラムのシンボルをわずかに動かす機能が含まれます。 |
| | ズーム | ズーム/パン・ツールバーには、選択されたアクティビティー・ダイアグラムのシンボルをズームまたは水平移動する機能が含まれます。 |
| 状況表示ウィンドウ | Process Designer Express が状況情報を表示する 1 行のペインです。 | |

各メニュー・オプションの左横にチェック・マークが付いているとき、対応する要素が画面に表示されます。要素を非表示にするには、対応するメニュー・オプションを選択します。選択すると、チェック・マークが消え、その要素は表示されないことを示します。逆に、表示されていない要素を表示させるには、対応するメニュー・オプションを選択します。選択すると、表示させる要素の横にチェック・マークが現れます。

## 連結可能なウィンドウの浮動

Process Designer Express のメインウィンドウの以下の部分は、連結可能なウィンドウです。

- テンプレート・ツリー・ビュー
- 出力ウィンドウ
- ツールバー

連結可能ウィンドウは、デフォルトでは通常メインウィンドウの端に沿って配置され、メインウィンドウの一部として移動します。連結可能ウィンドウを浮動させるときは、メインウィンドウから切り離します。これにより、独立したウィンドウと

して機能させることができます。連結可能ウィンドウを浮動させるには、マウスの左ボタンを押したままウィンドウの境界をつかみ、メインウィンドウまたはデスクトップ上にドラッグします。

## 作業域内でのウィンドウの表示

Process Designer Express のメインウィンドウの作業域内には、次のいずれかの方法でウィンドウを表示させることができます。

- 「最大化」: 1 つのウィンドウが作業域全体に表示されます。最大化状態のウィンドウを切り替えるには、「ウィンドウ」プルダウン・メニューから目的のウィンドウの名前を選択します。ワークブック・ダイアグラム・ウィンドウのデフォルトのユーザー設定が維持されている場合は、作業域の下の対応する「Workbook」タブを選択して、最大化状態のウィンドウを切り替えることもできます。詳細については、192 ページの『一般的な表示の変更』を参照してください。
- 「サイズ変更」: 各ウィンドウが作業域内のそれぞれ別の領域に表示されます。各ウィンドウのサイズを変更して、作業域内でウィンドウを重ねたり移動することができます。サイズ変更ウィンドウは、複数のアクティビティー・ダイアグラムを同時に表示するときや、アクティビティー・ダイアグラムと「テンプレート定義」ウィンドウまたは「テンプレート・メッセージ」ウィンドウを同時に表示するときに便利です。
- 「最小化」: 各ウィンドウは作業域の下部にアイコンとして表示されます。最小化状態のウィンドウを元に戻すには、アイコンをダブルクリックします。

# 第 2 部 コラボレーション・テンプレートの作成

# 第 3 章 コラボレーションの設計

この章では、コラボレーションの設計上の指針について説明します。この指針に従うことで、動作が良好で再使用可能なコラボレーションを設計することができます。

一般に、標準コラボレーション・テンプレートを開発することをお勧めします。これにより、ユーザー定義のコラボレーションの開発が容易になります。標準のテンプレートを使用することには、以下の利点があります。

- コラボレーションの設計上の一貫性

  標準のテンプレートに基づいてコラボレーションを設計すると、すべてのコラボレーションを以下のように動作させることができます。

  – 宛先アプリケーションにおいて、コラボレーションのフロー・トリガーに対応するビジネス・オブジェクトに対して同じ動詞操作を実行します。
  – 同じエラー処理機構を使用してエラーを処理します。これにより最終的なコラボレーションのテクニカル・サポートが大きく簡略化されます。
  – 同じ名前、タイプ、および予測される動作のポートを使用します。
- コラボレーションを容易に文書化

  コラボレーションの振る舞いを理解するうえで必要な情報のテンプレートも、コラボレーションに基づいて構成されています。したがって、標準テンプレートに基づくコラボレーションの振る舞いは、容易に文書化することができます。
- 「最良手法」の組み込み

  IBM によって推奨される最良手法 (41 ページの『コーディングの推奨事項』を参照) と、開発者自身のサイトで開発された最良手法とを標準のテンプレートに組み込み、その標準テンプレートに基づいてコラボレーションにこれらの最良手法を自動的に含めることができます。)

WebSphere Business Integration Express Plus には、標準コラボレーション・テンプレート (CollaborationFoundation) と標準ラッパー・コラボレーション・テンプレート (WrapperFoundation) があります。これらのテンプレートは、そのままの状態で使用することも、要件に合わせて変更することもできます。詳細については、 34 ページの『CollaborationFoundation テンプレート』および 54 ページの『WrapperFoundation テンプレート』を参照してください。

この章では、以下のタスクの指針についても説明します。

- コラボレーション・オブジェクト・グループの作成。詳細については、 60 ページの『コラボレーション・グループの作成』を参照してください。
- 並列処理 (イベントの順序付けおよびイベントの分離を含む)。詳細については、63 ページの『並列処理の設計』を参照してください。
- 国際化対応コラボレーション・テンプレートの作成。詳細については、 72 ページの『国際化対応コラボレーション』を参照してください。

## CollaborationFoundation テンプレート

CollaborationFoundation コラボレーション・テンプレートは、以下の機能を実行するユーザー定義のコラボレーションの作成を容易にするツールです。

- 同期

  コラボレーションは、アプリケーションから別のアプリケーションにデータを複製するときに同期をとります。同期では通常、結果を待機しないソース・アプリケーションのデータを処理します。

  CollaborationFoundation には、同期をとる多くのコラボレーションで使用される動詞ロジック、データ・フィルター操作、およびエラー処理が含まれます。つまりこれが基本同期ロジックです。CollaborationFoundation には、5 つの空のサブダイアグラムがあり、これらの空のサブダイアグラムはそれぞれ Additional Processing という名前が付いています。これらを使用すると、基本同期ロジックを変更せずにコラボレーションのビジネス・プロセスを拡張できます。

  このセクションでは、主に基本同期ロジックについて説明し、作成するコラボレーションの要件に合わせた CollaborationFoundation テンプレートの拡張についての推奨事項を提示します。

- データ・アクセス

  コラボレーションは、結果を待機するソース・アプリケーションからの Retrieve 要求によって起動されます。また、コラボレーションを起動するのは、同期イベントを送信してコラボレーションが宛先アプリケーションから完全なビジネス・オブジェクトを戻すのを待機する、ソース・アプリケーションの場合もあります。このようなイベントには、同期要求を送信するかサーバー・アクセス・インターフェースを使用することができるソース・アプリケーションが必要です。

  この章では、Retrieve プロセスおよび Additional Retrieve プロセスのアクセス機能について説明します。

- データ・パイピング

  コラボレーションは、データのフィルター操作や検証を実行せずにアプリケーションから別のアプリケーションにデータを移動するときに、データ・パイピングを実行します。このようなコラボレーションを CollaborationFoundation から作成するには、From ポートをソース・アプリケーションにバインドし、次に DestinationAppRetrieve と To ポートを宛先アプリケーションにバインドします。

  **注:** 製品に用意されているコラボレーションを変更して、ソースから宛先にデータを簡単に移動することもできます。このためには、From ポートをソース・アプリケーションにバインドし、DestinationAppRetrieve と To ポートを宛先アプリケーションにバインドし、その他のすべてのポートをポート・コネクターにバインドします。このとき、いずれの構成プロパティーのデフォルト値も変更しないでください。

以下のセクションでは、CollaborationFoundation テンプレートの詳細を説明します。


- 35 ページの『CollaborationFoundation の機能』
- 38 ページの『CollaborationFoundation テンプレートの使用』

- 39 ページの『CollaborationFoundation ポート』
- 41 ページの『CollaborationFoundation の拡張』


## CollaborationFoundation の機能

CollaborationFoundation テンプレートから生成されるコラボレーション・オブジェクトは、宛先アプリケーションのビジネス・オブジェクト上で Create、Retrieve、Delete、または Update アクションを実行できます。これらのアクションは、コラボレーションのトリガー・ビジネス・オブジェクトに対応します。

### トリガー・ビジネス・オブジェクトの処理

***Retrieve 動詞の振る舞い:*** CollaborationFoundation は、トリガー・ビジネス・オブジェクトで Retrieve 動詞を使用可能にするように設計されています。コラボレーションが同期要求を行うソース・コネクターにバインドされている場合、コラボレーションが処理を完了するとすぐに、ビジネス・オブジェクトの値がソース・コネクターに戻されます。値は、ビジネス・オブジェクトの値が参照によって受け渡された場合と同じように戻されます。

このテンプレートには、表 8 に示す標準プロパティーがあります。これらのプロパティーは、ビジネス・フローの振る舞いに影響します。

*表 8. ビジネス・フロー用 CollaborationFoundation 標準プロパティー*

| 標準プロパティー | 説明 |
|---|---|
| CONVERT_CREATE | トリガー・ビジネス・オブジェクトが宛先アプリケーションにすでに存在する場合に、コラボレーション・オブジェクトが宛先に送信された動詞を Create から Update に変換するよう構成します。 |
| CONVERT_UPDATE | ビジネス・オブジェクトが宛先アプリケーションに存在しない場合に、コラボレーション・オブジェクトが Update 動詞を Create 動詞に変換するよう構成します。 |
| USE_RETRIEVE | コラボレーション・オブジェクトが、データを同期する前に宛先アプリケーションからトリガー・ビジネス・オブジェクトを検索するように構成します。このプロパティーは、差し戻し処理を実行する場合と、ビジネス・オブジェクトがすでに宛先アプリケーションに存在するかどうかに基づいて動詞を設定する場合に便利です。 |

*表 8. ビジネス・フロー用 CollaborationFoundation 標準プロパティー (続き)*

| 標準プロパティー | 説明 |
| --- | --- |
| ADDITIONAL_RETRIEVE | コラボレーション・オブジェクトが、データを同期した後に宛先アプリケーションからビジネス・オブジェクトを検索するように構成します。このプロパティーは、ソース・アプリケーションが宛先アプリケーションから完全な値を持つビジネス・オブジェクトを戻すよう要求する場合で、宛先アプリケーションのコネクターがデータの作成または更新後に完全なビジネス・オブジェクトを戻さないときに便利です。 |

37 ページの図 9 は、USE_RETRIEVE プロパティーが `false` と評価された場合のコラボレーションのメイン・ビジネス・フローを示しています。

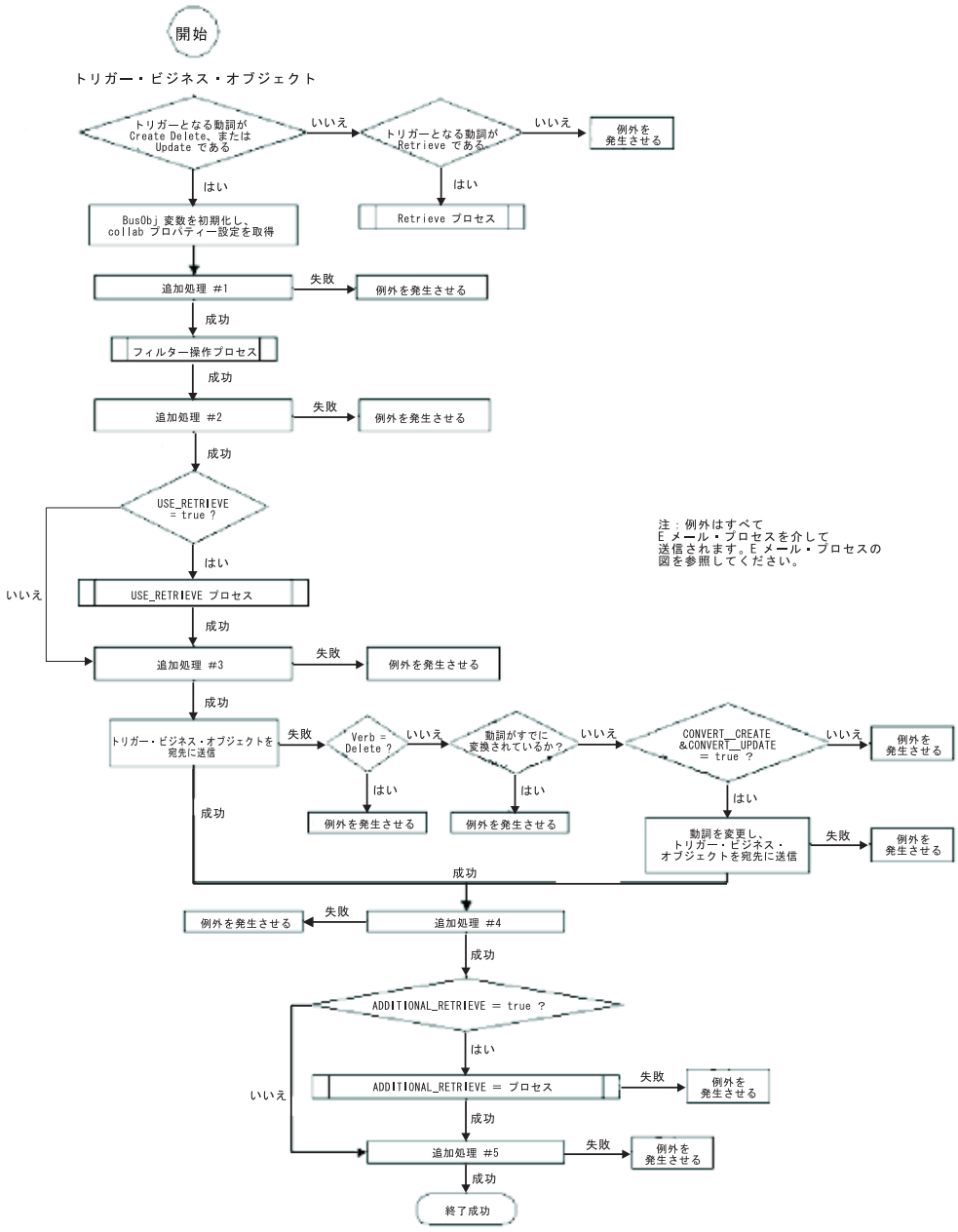



*図 9. CollaborationFoundation のメイン・ビジネス・プロセス・フロー*

***トリガー・ビジネス・オブジェクトでのデータのフィルター操作:*** コラボレーション・オブジェクトでは、トリガー・ビジネス・オブジェクトの特定の属性内のデータをフィルター操作するよう構成できます。フィルター操作の結果を使用して、コラボレーションが特定のデータを持つトリガー・ビジネス・オブジェクトと同期する必要があるか判断することができます。テンプレートには、データのフィルター操作用に以下の 4 つのプロパティーがあります。

- 1_FILTER_ATTRIBUTE
- 1_INCLUDE_VALUES
- 1_EXCLUDE_VALUES
- 1_FAIL_ON_INVALID_VALUE

複数のフィルター操作属性を指定できる追加プロパティーの追加方法については、471 ページの『標準的なコラボレーションに関する情報』の 1_FILTER_ATTRIBUTE プロパティーの説明を参照してください。また、476 ページの図 83 にコラボレーションのフィルター操作の振る舞いを示します。

2 番目の属性でフィルター操作をするために、CollaborationFoundation テンプレートには以下のプロパティー・セットがあります。

- 2_FILTER_ATTRIBUTE
- 2_INCLUDE_VALUES
- 2_EXCLUDE_VALUES
- 2_FAIL_ON_INVALID_VALUE

複数の属性でフィルター操作をするには、各プロパティーにフィルター・プロパティーのセットを指定します。プロパティーの各セットには、固有の数値を指定します。つまり、3 番目の属性のフィルター・プロパティーでは、プレフィックスとして 3_ を使用します。プレフィックスは属性のセットごとに異なる必要がありますが、プロパティー名自体は上記の名前と同一にする必要があります。

### エラー処理

CollaborationFoundation には、エラー処理用の標準プロパティーがあります。表 9 に、このプロパティーの説明を示します。

*表 9. エラー処理用の標準プロパティー*

| エラー処理プロパティー | 説明 |
|---|---|
| INFORMATIONAL_EXCEPTION | コラボレーション・オブジェクトが正常に終了して例外をトレースに送信するか、正常に終了しないで記録する例外を生成するかを指定します。 |
| SEND_EMAIL | INFORMATIONAL_EXCEPTION の値に関係なく、例外が発生した場合にコラボレーション・オブジェクトが特定のアドレスに E メールを送信するか指定します。 |

477 ページの『エラー処理の E メール・プロセス』は、コラボレーションのエラー処理時の E メール・プロセスを示しています。

いずれのエラー処理プロパティーの場合も、すべての例外、またはコンマで区切られたメッセージ番号の小さいセットに対して、振る舞いを指定できます。メッセージ番号は、コラボレーション・メッセージ・ファイル (collaborations¥messages¥CollaborationFoundation.txt) 内のメッセージ番号に対応します。

## CollaborationFoundation テンプレートの使用

CollaborationFoundation テンプレートを使用するには、以下の操作を実行します。

1. CollaborationFoundation テンプレート全体を開発域にコピーします。
2. コラボレーション・テンプレート名を、作成中のコラボレーションの名前に変更します。

3. Process Designer の「テンプレート定義」ウィンドウの「ポートおよびトリガー・イベント」タブで、既存のポートのタイプ (DestinationAppRetrieve、To、From) を変更して、該当するビジネス・オブジェクトをこれらのポートに割り当てます。詳細については、『CollaborationFoundation ポート』を参照してください。
4. コラボレーションで複数のビジネス・オブジェクト属性をフィルター操作する場合、フィルター・プロパティーの追加セットを作成します。

   WebSphere Business Integration Server Express には、一連の構成プロパティーがあります。このプロパティーにより、特定の属性の特定の値に基づいて同期対象にビジネス・オブジェクトを組み込んだり、同期対象から除外したりすることができます。複数の属性値に基づいて同期をフィルター操作する場合、評価する属性ごとに構成プロパティーの追加セットを作成する必要があります。詳細については、478 ページの『コラボレーション・テンプレートの標準プロパティー』を参照してください。
5. 必要な場合は、Additional Processing サブダイアグラムにコードを追加して、追加のコラボレーション機能を作成します。サブダイアグラムには、CollaborationFoundation テンプレートのような機能はありません。サブダイアグラムの目的は、コラボレーションのカスタマイズを容易にすることにあります。
6. コラボレーション・テンプレートを保管し、コンパイルします。

## CollaborationFoundation ポート

図 10 に、CollaborationFoundation のポートを示します。図に続く表には、各ポートの詳細を示します。

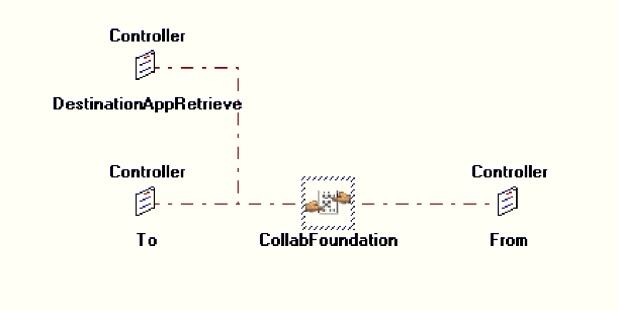



*図 10. CollaborationFoundation コラボレーションのポート*

### DestinationAppRetrieve ポート

表 10 に、CollaborationFoundation の DestinationAppRetrieve ポートの機能をリストします。

*表 10. ポート機能 (DestinationAppRetrieve ポート)*

| ポート機能 | 値 |
|---|---|
| ビジネス・オブジェクト | 参照値を持つ BusObj。デフォルト値は Controller です。 |
| 使用する動詞 | Retrieve |

*表 10. ポート機能 (DestinationAppRetrieve ポート) (続き)*

| ポート機能 | 値 |
|---|---|
| 差し戻し動詞 | なし |
| 目的 | 宛先から完全な値を持つビジネス・オブジェクトを検索し、それを使用して動詞を設定します。 |
| バインド先 | 宛先アプリケーションのコネクター |
| 構成プロパティー | USE_RETRIEVE |

## To ポート

表 11 は、CollaborationFoundation の To ポートの機能をリストしています。

*表 11. ポート機能 (To ポート)*

| ポート機能 | 値 |
|---|---|
| ビジネス・オブジェクト | 完全な値を持つ BusObj。デフォルト値は Controller です。 |
| 使用する動詞 | Create、Update、または Delete |
| 差し戻し動詞 | なし |
| 目的 | コラボレーションの外部にビジネス・オブジェクトを送信します。コラボレーションは、このポートを使用して宛先にトリガー・ビジネス・オブジェクトのコピーを送信することがあります。 |
| バインド先 | 宛先アプリケーションのコネクターまたは別のコラボレーション (追加処理用) |
| 構成プロパティー | なし |

## From ポート

表 12 に、CollaborationFoundation の From ポートの機能をリストします。

*表 12. ポート機能 (From ポート)*

| ポート機能 | 値 |
|---|---|
| ビジネス・オブジェクト | triggeringBusObj (デフォルト値は Controller) |
| 使用する動詞 | Create、Retrieve、Update、または Delete |
| 差し戻し動詞 | なし |
| 目的 | トリガー・ビジネス・オブジェクトを受け取ります。 |
| バインド先 | ソース・アプリケーションまたは呼び出し側コラボレーション |
| 構成プロパティー | なし |

## CollaborationFoundation の拡張

このセクションでは、CollaborationFoundation テンプレートに基づいてコラボレーションをカスタマイズする場合に役立つ情報を提供します。以下のセクションを参照してください。

- 『コーディングの推奨事項』
- 45 ページの『共通の変更事項』

## コーディングの推奨事項

このセクションでは、製品に用意されているコラボレーションのコードに合わせてコードを標準化するためのコーディング方法について説明します。

- 『命名規則』
- 『フロー・トリガーの処理』

### 命名規則

コラボレーション・テンプレートで使用する命名規則を確立することは優れた手法です。具体的な命名規則のいくつかを以下に示します。

- 各変数の型は、変数名の前に意味のある文字を付けることで識別します。例えば、ストリング変数名のプレフィックスには文字「s」を、Boolean 変数名のプレフィックスには文字「b」を付けます。次のコードは、この 2 つの変数を初期化します。

  ```
  String sExceptionType
  Boolean bBranch
  ```

- コラボレーションの構成プロパティーは、プログラム変数と区別しやすいようにすべて大文字にする必要があります。次のコードは、SEND_EMAIL プロパティーの値を取得します。

  ```
  bSendEmail = getConfigProperty("SEND_EMAIL");
  ```

### フロー・トリガーの処理

Process Designer では、triggeringBusObj という BusObj タイプの変数が自動的に宣言されます。この変数は、シナリオを実行するフロー・トリガー (通常はトリガー・イベント) を保持します。

サービス呼び出しを通じてフロー・トリガーが送信された後にフロー・トリガーにデータを追加したり、属性値を操作するなど、処理の完了後もフロー・トリガーで作業する必要のある状況がいくつかあります。このような状況は次のとおりです。

- トランザクション・コラボレーションの差し戻し段階中にロールバックを実行するために、サービス呼び出しを通じてフロー・トリガーを送信します。
- フロー・トリガー内の属性値と、サービス呼び出しまたはデータベース検索によって戻されるビジネス・オブジェクト内の属性値を比較します。

これらの状況を対処するには、フロー・トリガーを変更するのではなく、フロー・トリガーのコピーである中間 BusObj 変数を作成した後、必要に応じて中間変数を操作し、サービス呼び出しを通じてこれを送信することを推奨します。

**注:** ビジネス・オブジェクトのコピーを作成すると、システムのリソースが消費されます。ビジネス・プロセスで中間変数が必要とされない場合 (例えば、トラン

ザクションの要件がないうえに、特定の状況の前後に属性値を比較する必要がまったくないため)、リソースを保持するために、フロー・トリガーのコピーではなくフロー・トリガーを直接使用します。

コラボレーションが長期存続ビジネス・プロセスとして構成されている場合には、フロー・トリガー・ビジネス・オブジェクト (triggeringBusObj) の内容はサービス呼び出し間で保持されません。この場合は、必ずフロー・トリガーのコピーを作成してください。

あるビジネス・オブジェクトの内容を別のビジネス・オブジェクトにコピーする機能を提供する API がいくつかあります。それぞれの API には利点と欠点があり、『copy() メソッドの使用』および 『duplicate() メソッドの使用』で各アプローチについて説明します。

***copy() メソッドの使用:*** copy() メソッドを使用して、あるビジネス・オブジェクト変数の内容を同じタイプの別のビジネス・オブジェクト変数にコピーできます。WebSphere Business Integration Server Express に用意されているコラボレーション・テンプレートでも同じなので、この方法を使用することをお勧めします。用意されたコンポーネントとカスタム・ビルド・コンポーネントの間で一貫性を保てば、保守容易性も向上します。

このアプローチを使用するには、トリガー・ビジネス・オブジェクトと同じタイプの BusObj オブジェクトの新規インスタンスを作成する必要があります。このとき、シナリオのシナリオ定義でインスタンスを作成し、コピーの BusObj を格納する変数に命名することを推奨します。これらの要件および推奨事項を満たすために、シナリオのシナリオ定義に次のコードを追加します。

```
BusObj processingBusObj;
processingBusObj = triggeringBusObj.duplicate();
```

次に、processingBusObj 変数上で copy() メソッドを実行し、そこに triggeringBusObj 変数を引き数として渡す必要があります。シナリオの (変数を初期化するために排他的に使用する) トップレベル・ダイアグラムの最初のアクション・ノードでこれを実行することを推奨します。次のコード例は、triggeringBusObj 変数の内容を processingBusObj 変数にコピーします。

```
processingBusObj.copy(triggeringBusObj);
```

***duplicate() メソッドの使用:*** 例えば、次のコード・フラグメントは、フロー・トリガーと同じ型の変数を宣言し、そのフロー・トリガーのビジネス・オブジェクトの中の値をコピーして値を設定します。

```
BusObj processingBusObj;
processingBusObj = triggeringBusObj.duplicate();
```

コラボレーションは必要に応じて、processingBusObj を使用してデータを操作します。宛先アプリケーションにデータを送信する準備が整うと、コラボレーションは中間変数を ToBusObj 変数にコピーします。コラボレーションは、宛先アプリケーションへのサービス呼び出しで ToBusObj を使用します。次のコード・フラグメントは、データを ToBusObj にコピーするステートメントを示します。

```
ToBusObj.copy(processingBusObj);
```

サービス呼び出しの結果がコラボレーションに正常に戻ると、コラボレーションは次に示すように ToBusObj の値を triggeringBusObj にコピーします。

```
triggeringBusObj.copy(ToBusObj);
```

WebSphere Business Integration Server Express のコラボレーションでは通常、コラボレーションが To ポートから戻される ToBusObj を受け取るまでは、triggeringBusObj の元の値を変更しません。中間変数を使用すると、コラボレーションは宛先アプリケーションから正常に値を受け取った後にのみ triggeringBusObj の値を変更するようになります。

## 例外の生成

例外は、発生したレベルでキャッチし、コラボレーションの中のトップ・プロセスで生成させてください。例外をキャッチすることで、例外の処理方法を指定したり、ユーザーへの提示方法を制御できます。例えば、例外が生成したコンテキストを解明できます。さらに、例外処理のためのアクション・ノードを作成して、例外が生成する可能性のあるコード内のそれぞれの位置を示す文書を表示できます。

すべてのコラボレーションで、トラップした各例外をコラボレーション・ランタイム環境に達するまでに生成させる必要があります。別のコラボレーションを起動するサービス呼び出しを使用する場合、呼び出し側コラボレーションはそのサービス呼び出しの結果として例外が生成していないか検査する必要があります。

例外テキストを呼び出し側ダイアグラムに生成させるには、メッセージ・テキストと例外のタイプを格納するためにそれぞれ別個のストリング変数を宣言してください。例えば、次のコードはこのようなストリング変数を宣言します。

```
String sMessage
String sExceptionType
```

サービス呼び出しが成功したか、または失敗したかに応じて異なる動作を提供するには、分岐を使用します。失敗を処理する分岐では、2 つのストリング変数に値を割り当てます。以下に例を示します。

```
sMessage = currentException.getMessage();
sExceptionType = currentException.getType();
```

サービス呼び出しを行ったプロセスに制御を戻す前に、例外を生成させます。以下に例を示します。

```
raiseException(ServiceCallException, 4000, SendRefBusObj.getType(),
   SendRefBusObj.getVerb(), SendRefBusObj.keysToString(),
   sExceptionType, sMessage);
```

上記のコードは、エラー・メッセージ 4000 を指定します。これはコラボレーションの失敗に対する標準のエラー・メッセージです。メッセージ・ファイルには次のテキストが含まれます。

```
4000
コラボレーションが失敗しました: キー ({3}) を持つ {1}.{2} の同期が失敗しました。
例外は {4}.{5} です。[EXPL]
ビジネス・オブジェクトを宛先で同期させることができませんでした。
```

前述のテキストで、raiseException() メソッドは表 13 に示す値を置換します。

*表 13. raiseException() 呼び出しで置換される値*

| 変数 | 置換されるテキスト |
|---|---|
| {1} | SendRefBusObj.getType() が戻す値 |
| {2} | SendRefBusObj.getVerb() が戻す値 |
| {3} | SendRefBusObj.keysToString() が戻す値 |
| {4} | sExceptionType 変数の値 |
| {5} | sMessage 変数の値 |

サービス呼び出しを行うプロセスがコラボレーション内の最上位のプロセスではない 場合、そのサービス呼び出しを行うプロセスは呼び出し側プロセスに例外を生成させる必要があります。呼び出し側プロセスより上位の各プロセスも、最上位プロセスからエラー・メッセージを記録できるように例外を生成させる必要があります。

## 分岐

コラボレーション・ダイアグラムのフローが、コラボレーションの構成プロパティーの値に基づいて作成されていることがよくあります。コラボレーションはプロパティー値を使用して boolean 変数を設定することができます。この値は採用する経路を決定するために後で使用されます。例えば、次のコードは、bBranch という名前の boolean 変数を宣言して初期化します。

```
boolean bBranch = false;
```

InterChange Server Express では分岐変数の値は、コード内の条件に従って設定されます。これらの条件は、いくつかの boolean 変数の値に基づいて設定することができます。例えば、コラボレーションの CONDITION_ONE プロパティーが true と評価される場合のみコラボレーションが CONDITION_TWO プロパティーを評価するとします。

次のコードは、2 つの boolean 変数の値に基づいて分岐するようにします。

- bCondition1: コラボレーションの CONDITION_ONE プロパティー用に構成された値が含まれる。
- bCondition2: コラボレーションの CONDITION_TWO プロパティー用に構成された値が含まれる。

次のコードは、CONDITION_ONE が true、CONDITION_TWO が false と評価される場合に、bBranch の値を true に設定します。つまり、CONDITION_ONE が false と評価されるか、CONDITION_TWO が true と評価される場合は、bBranch の値を false に設定します。

```
if ( bCondition1 && !bCondition2 )
   {
   bBranch = true;
   }
else
{
   bBranch = false;
}
```

### ラッパー・コラボレーション

ラッパー・コラボレーション とは、別のコラボレーションのビジネス・オブジェクトの検査または同期を処理するコラボレーションです。呼び出し側コラボレーションは、自身のフロー・トリガーで参照されるトップレベルのビジネス・オブジェクトをラッパー・コラボレーションに送信します。

例えば、SalesOrderProcessing コラボレーションは汎用 Order ビジネス・オブジェクトを同期できます。汎用 Order には、注文を行う顧客を表す汎用 Customer ビジネス・オブジェクトへの参照が含まれます。また、汎用 Order には汎用 OrderLineItem ビジネス・オブジェクトの配列も含まれます。各 OrderLineItem は汎用 Item ビジネス・オブジェクトを参照し、注文された品目を表します。

コラボレーション・ロジックをモジュール化する目的で、汎用 Order とこれが参照する汎用ビジネス・オブジェクトを処理するための個別のコラボレーション・テンプレートを用意することができます。例えば、Customer ビジネス・オブジェクトと Item ビジネス・オブジェクトを参照する Order を処理するには、以下のテンプレートを使用できます。

- SalesOrderProcessing: 注文を処理する。
- CustomerWrapper および CustomerSync: 参照先顧客を処理する。
- ItemWrapper および ItemSync: 参照先品目を処理する。

ビジネス・オブジェクトの処理をそれぞれ異なる特定のコラボレーションに分けると、各コラボレーション・テンプレートの再利用が増えるだけでなく、2 つのコラボレーションが同じデータを同時に変更しないようにします。詳細については、64 ページの『並行処理の問題点』を参照してください。

## 共通の変更事項

このセクションでは、CollaborationFoundation コードに対する以下の共通の変更事項について説明します。

- 『従属データの処理』
- 46 ページの『従属データの検査の代行』
- 51 ページの『従属データの再帰的検査の実行』

このセクションで示す変更を行って、カスタマイズされたコラボレーションを製品に用意されているコラボレーションに合わせて標準化します。

### 従属データの処理

ビジネス・オブジェクトが自身の処理に影響する別のビジネス・オブジェクトを参照するとき、そのビジネス・オブジェクトは従属データを持つといいます。例えば、ユーザーのサイトで、Order、Customer、および Item ビジネス・オブジェクトを同期したいとします。SalesOrderProcessing コラボレーション・オブジェクトが各 Order を同期する前に、Order に参照される Customer が宛先アプリケーションにすでに存在することを確認する必要があります。さらに、各 OrderLineItem に参照される Item も宛先アプリケーションに存在することを確認したり、参照先ビジネス・オブジェクトが存在することを確認するまで Order の同期を停止したりする必要がある場合があります。このような場合、Order オブジェクトは参照先の Customer オブジェクトと Item オブジェクトの存在に依存します。

WebSphere Business Integration Server Express では、Customer または Item ビジネス・オブジェクトを参照するトリガー・ビジネス・オブジェクトを持つコラボレーションが複数用意されているので、単一のインストールでは、同じ Customer データまたは Item データの同期を試みる複数のコラボレーション・オブジェクトが同時に実行されることがあります。データの整合性を維持し、複数のコラボレーションが同じタイプのビジネス・オブジェクトの同じインスタンスを同時に変更しないようにするには、参照先ビジネス・オブジェクトの同期を代行させて、コラボレーション・オブジェクトを分離することをお勧めします。コラボレーションは中間ラッパー・コラボレーションを呼び出して、自身のトリガー・ビジネス・オブジェクトで参照しているビジネス・オブジェクトの検証または同期を処理します。ラッパー・コラボレーションは、複数のコラボレーションを実行している環境でのデータ分離を容易にします。

ラッパー・コラボレーションを使用して、複数のコラボレーションを実行している環境でデータの整合性を維持します。例えば、複数の CustomerSync コラボレーション・オブジェクトが同一インストール上で実行されているとします。1 つのオブジェクトは SalesOrderProcessing と共に実行され、その他のオブジェクトは個別に実行されます。この場合、独立した CustomerSync コラボレーション・オブジェクトは、SalesOrderProcessing コラボレーション・オブジェクトと同様に 2 つのアプリケーション間で Customer データを同期します。SalesOrderProcessing を使用して、参照値を持つ Customer ビジネス・オブジェクトの同期を中間 CustomerWrapper コラボレーション・オブジェクトに代行させる場合、ソフトウェアによってイベント分離が実行されます。つまり、独立型 CustomerSync コラボレーション・オブジェクトは、SalesOrderProcessing から呼び出された CustomerSync コラボレーション・オブジェクトと同時に同一のビジネス・オブジェクト上で作業することはできません。

トリガー・ビジネス・オブジェクトから参照されるビジネス・オブジェクトの同期についての詳細は、64 ページの『並行処理の問題点』を参照してください。

## 従属データの検査の代行

CollaborationFoundation を拡張して、依存関係の検査を別のコラボレーションに代行させることができます。この場合、以下の操作を実行します。

- 呼び出し先コラボレーションにバインドするポートを追加します。
- コラボレーションが参照先ビジネス・オブジェクトを検証または同期するかを指定できるようにする、コラボレーション構成プロパティーを追加します。参照先の Item または Contact ビジネス・オブジェクトを同期するよう選択した場合、新たに構成プロパティーを作成しなければならないことがあります。
- 検証または同期が失敗した時に情報を提示するメッセージを追加します。
- 依存関係検査を処理するシナリオを追加します。

***ポートの追加:*** 従属データを代行させるには、従属ビジネス・オブジェクトごとにポートを作成します。新規ポートに To*BOName*Wrapper と命名して、製品に用意されている他のコラボレーションとの整合性を維持します。ここで、*BOName* は参照値を持つ従属ビジネス・オブジェクトです。例えば、SalesOrderProcessing は ToCustomerWrapper というポートを使用して、参照値を持つ Customer ビジネス・オブジェクトを CustomerWrapper に送信します。

ラッパー・コラボレーションに送信される、参照値を持つビジネス・オブジェクトごとに、このようなポートを作成します。例えば SalesOrderProcessing は、SendContactRef や SendItemRef という命名されたポートも使用します。

***構成プロパティーの追加:*** 参照先ビジネス・オブジェクトの検査を別のコラボレーションに代行させるには、新規のコラボレーション構成プロパティーを作成します。このプロパティーを使用すると、コラボレーションを構成して、コラボレーションが参照先ビジネス・オブジェクトを検証または同期するか指定できるようになります。

製品に用意されているコラボレーションとの整合性を維持するには、新規プロパティーを VERIFY_SYNC_*BOName* と命名します。ここで、*BOName* は参照値を持つビジネス・オブジェクトです。例えば SalesOrderProcessing には、Customer ビジネス・オブジェクトの検査を代行させるために使用する VERIFY_SYNC_CUSTOMER というプロパティーが含まれます。このプロパティーに使用可能な値を以下のように設定します。

- `neither`: 代行を行わない。
- `sync`: コラボレーションは、Sync 動詞を使用して、参照値を持つビジネス・オブジェクトを同期のために適切なラッパー・コラボレーションに送信する。
- `verify`: コラボレーションは、Exists 動詞を使用して、参照値を持つビジネス・オブジェクトを検証のために適切なラッパー・コラボレーションに送信する。

デフォルトでは、VERIFY_SYNC_*BOName* プロパティーは `neither` に設定されます。

***エラー処理プロパティー:*** コラボレーションが参照値を持つビジネス・オブジェクトの配列を代行させる場合、配列の要素における同期または検証の障害を処理できるよう準備する必要があります。製品に用意されているコラボレーションには、検証または同期中に発生する個々の要素の障害を処理する方法を指定できる構成プロパティーがあります。

例えば汎用 Order ビジネス・オブジェクトには、OrderLineItem 子ビジネス・オブジェクトの配列が含まれています。これらの各 OrderLineItem オブジェクトは、Item ビジネス・オブジェクトを参照できます。したがって、Order ビジネス・オブジェクトを処理するコラボレーションは Item ビジネス・オブジェクトの配列を代行させることができます。同様に、汎用 Order ビジネス・オブジェクトには、OrderContactRef 子ビジネス・オブジェクトの配列が含まれています。これらの各 OrderContactRef オブジェクトは、Contact ビジネス・オブジェクトを参照できます。したがって、Order ビジネス・オブジェクトを処理するコラボレーションは Contact ビジネス・オブジェクトの配列を代行させることができます。

Order ビジネス・オブジェクトを処理する SalesOrderProcessing コラボレーションには、配列内の 1 つ以上の参照先ビジネス・オブジェクトが同期または検証に失敗した場合の対処方法を指定できる構成プロパティーがあります。

オプションの 1 つは、代行された参照先ビジネス・オブジェクトの配列内のビジネス・オブジェクトが同期または検証に失敗した場合に、トリガー・ビジネス・オブジェクトの同期を停止することです。もう 1 つのオプションは、失敗したビジネス・オブジェクトを配列から除去して、トリガー・ビジネス・オブジェクトの同期を続行することです。

失敗した参照値を持つビジネス・オブジェクトを処理するオプションをコラボレーションが評価するのは、そのビジネス・オブジェクトへの検証または同期を試行している場合のみです。したがって、コラボレーションがこのようなオプションを持つプロパティーを評価する必要があるのは、対応する VERIFY_SYNC_*BOName* プロパティーが `verify` または `sync` と評価される場合のみです。

Business Integration Express に用意されているコラボレーションには、配列内の参照先ビジネス・オブジェクトが検証または同期に失敗した場合にオプションを提供する、2 つの構成プロパティーがあります。プロパティーの 1 つは失敗した Contact ビジネス・オブジェクトを処理し、もう一方は失敗した Item ビジネス・オブジェクトを処理します。

Contact ビジネス・オブジェクトの配列の処理方法を以下の例に示します。この例で、SalesOrderProcessing および InstalledProductSync は、参照値を持つ Contact ビジネス・オブジェクトの配列を代行させます。どちらも FAIL_ON_CONTACT_ERROR プロパティーを使用して、呼び出し先コラボレーションが配列内の Contact オブジェクトの 1 つの検証または同期に失敗した場合のコラボレーションの振る舞いを決定します。

CollaborationFoundation を変更して Contact オブジェクトの配列を代行させる場合、他のコラボレーションとの整合性を維持してください。このためには、VERIFY_SYNC_CONTACT が `verify` または `sync` と評価され、ContactWrapper が参照先 Contact オブジェクトのいずれかの検証または同期に失敗した場合にのみ FAIL_ON_CONTACT_ERROR を評価するようにコラボレーションをコーディングします。製品に用意されているコラボレーションは、FAIL_ON_CONTACT_ERROR の値に基づいて以下の操作を実行します。

- True: コラボレーションは処理を停止する。
- False: コラボレーションは処理を続行し、子ビジネス・オブジェクトの配列から障害のある Contact オブジェクトを除去する。

デフォルトでは、FAIL_ON_CONTACT_ERROR プロパティーは True に設定されます。

Item ビジネス・オブジェクトの配列の処理方法を説明します。WebSphere Business Integration Server Express のビジネス・オブジェクトには、Item および Contact ビジネス・オブジェクトへの参照の配列を組み込むことができます。したがって、各汎用 Order は汎用 Item オブジェクトの配列を参照できます。このため、SalesOrderProcessing を構成して、参照先 Item ビジネス・オブジェクトの配列を ItemWrapper に代行させることができます。

検証または同期に失敗した個々の Item のエラーを処理するため、SalesOrderProcessing は FIND_ALL_ITEM_ERRORS プロパティーを使用します。コラボレーションが FIND_ALL_ITEM_ERRORS を評価するのは、VERIFY_SYNC_ITEM が `verify` または `sync` と評価される場合と、ItemWrapper が参照先 Item ビジネス・オブジェクトのいずれかの検証または同期に失敗した場合のみです。

図 11 は、VERIFY_SYNC_ITEM が sync と評価され、FIND_ALL_ITEM_ERRORS が true と評価される場合に、CollaborationFoundation が参照先 Item ビジネス・オブジェクトの配列を処理する方法を示しています。

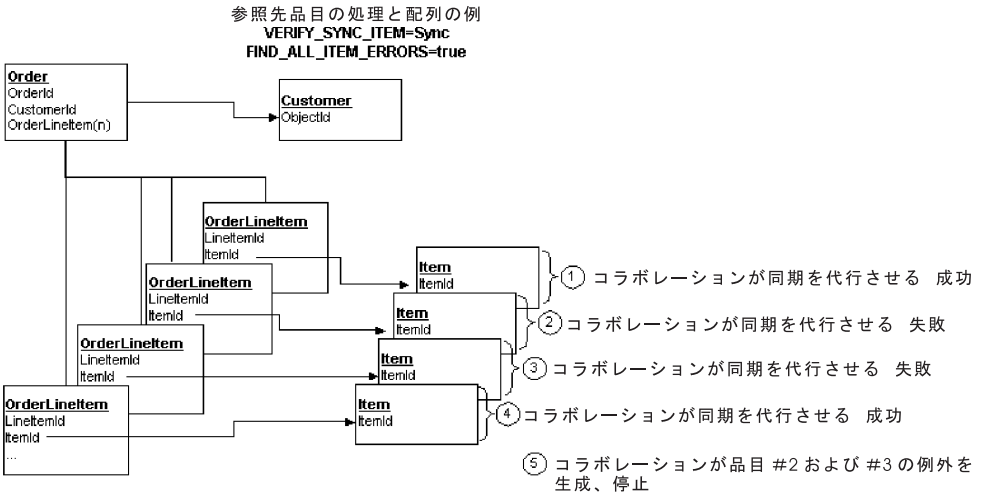



*図 11. 参照先 Item ビジネス・オブジェクトの配列の処理例*

コラボレーションは、FIND_ALL_ITEM_ERRORS の値に基づいて以下の操作を実行します。

- True: コラボレーションは、参照先 Item ビジネス・オブジェクトの処理および送信を続行します。配列の最後の Item の処理が終了すると、コラボレーションは失敗した Item ごとに例外を生成し、処理を停止します。この設定により、管理者はすべての品目エラーを一度に受け取ることができます。
- False: コラボレーションは処理を停止します。この設定では、各品目エラーが発生するごとに管理者がコラボレーションを再始動する必要があります。管理者は各エラーを検出および修正する必要があります。

デフォルトでは、FIND_ALL_ITEM_ERRORS の値は false です。

***複数の品目タイプの処理:*** WebSphere Business Integration Server Express には、表 14 のリストにある汎用 Item ビジネス・オブジェクトが用意されています。

*表 14. 汎用 Item ビジネス・オブジェクト*

| 汎用 Item ビジネス・オブジェクト | 説明 |
|---|---|
| ItemBasic | 品目データを使用するすべての論理組織に共通のデータ属性が含まれます。この属性は、他の Business Integration Express の汎用 Item ビジネス・オブジェクトの前提条件と見なされています。 |
| ItemOrder | 特定の注文管理組織エンティティー用に維持されるフィールドのセットを収集するデータ属性が含まれます。 |
| ItemPlanning | 品目の将来の所要量および在庫補充の計画に関するデータ属性が含まれます。 |

*表 14. 汎用 Item ビジネス・オブジェクト (続き)*

| 汎用 Item ビジネス・オブジェクト | 説明 |
|---|---|
| Item | 品目データを表すフラットなビジネス・オブジェクト。このビジネス・オブジェクトをプレースホルダーとして使用するコラボレーションもあります。汎用 Item を使用すると、コラボレーションが処理する Item ビジネス・オブジェクトの実際のタイプを実行時に決定できるようになります。したがって、汎用 ItemBasic、ItemOrder、および ItemPlanning を処理するコラボレーションにコネクターをバインドするためには、汎用 Item がインストール上に存在する必要があります。 |

トリガー・ビジネス・オブジェクトが Item ビジネス・オブジェクトを参照できる場合、コラボレーション・オブジェクトの構成担当者が、検証または同期する参照先 Item のタイプを指定できるようにすることが重要です。ITEM_TYPE プロパティーにはこの機能があります。

CollaborationFoundation から作成したコラボレーションに、Item ビジネス・オブジェクトを参照するトリガー・ビジネス・オブジェクトがある場合、ITEM_TYPE プロパティーを使用するようコラボレーションを拡張します。値のセットに、システム上で使用可能なすべてのタイプの Item ビジネス・オブジェクトが含まれるように定義します。

デフォルトでは、ITEM_TYPE プロパティーは `Item` に設定されます。

***メッセージの追加:*** CollaborationFoundation のメッセージを確認または変更するには、メッセージ・ファイル (collaborations¥messages¥CollaborationFoundation.txt) を編集します。依存関係検査の一部が失敗した場合は、情報を提供するメッセージを作成することによって、他の Business Integration Express のコラボレーションとの整合性を維持します。

WebSphere Business Integration Server Express の同期コラボレーションとラッパー・コラボレーションには固有のエラー・メッセージが用意されているので、CollaborationFoundation から作成されたコラボレーションには、関連するビジネス・オブジェクトの失敗した同期または検証それぞれに特定のエラー・メッセージを組み込む必要はありません。ただし、このようなカスタマイズされたコラボレーションでは、呼び出し先コラボレーションから受け取ったメッセージを明示的に処理する必要があります。これらのメッセージの処理についての詳細は、43 ページの『例外の生成』を参照してください。

***シナリオの追加:*** 参照先ビジネス・オブジェクトの検査を別のコラボレーションに代行させるには、そのコラボレーションが参照先ビジネス・オブジェクトを検証または同期するか評価するコラボレーション・シナリオを作成する必要があります。

シナリオでは、コラボレーションの VERIFY_SYNC_*BOName* プロパティーの値を確認して、コラボレーションが参照先ビジネス・オブジェクトを検証または同期す

るよう構成されたか判断する必要があります。また、ラッパー・コラボレーションを呼び出す前に参照先ビジネス・オブジェクトに他の予備検査を実行するシナリオを作成することもできます。

例えば以下のコードでは、必要でないかぎりコラボレーションが CustomerWrapper コラボレーションを呼び出さないようにすることにより、パフォーマンスを強化しています。

```
// Get the value of the configuration property
String vsc = getConfigProperty("VERIFY_SYNC_CUSTOMER");

// By default, do not branch to Wrapper call.
bBranch= false;

// If VERIFY_SYNC_CUSTOMER evaluates to "neither" or
// if the source application's business object does not contain the
// Customer's primary key, do not change the value of bBranch.
if (vsc.equals("neither") || (processingBusObj.isNull("CustomerID")))"
     {}
else
     {
     // Get the Customer's primary key from the source
     // and destination application's business objects.
     // Note: The DestinationAppRetrieveBusObj variable
     // contains a value only if the USE_RETRIEVE property evaluates
     // to "true" and has already been processed.
     String sc=processingBusObj.getString("CustomerID");
     String dc=DestinationAppRetrieveBusObj.getString("CustomerID");

     // If the Customer's primary key is the same in both the
     // source and destination application (therefore, no need to
     // synchronize), the verb is Update, and USE_RETRIEVE evaluates
     // to "true", do not change the value of bBranch.
     if (sVerb.equals("update") && bUseRetrieve && sc.equals.(DC) )
          {}
     else
          {
          // If the verb is Create, or if the Customer's primary
          // keys are not identical and the verb is Update, call
          // the Wrapper collaboration.
          bBranch = true;
          }
}
```

上記のコードを含むコラボレーション・テンプレートの例は、SalesOrderProcessing についての説明を参照してください。SalesOrderProcessing の Additional Processing 3 サブダイアグラムには、VERIFY_SYNC_CUSTOMER、VERIFY_SYNC_ITEM、および VERIFY_SYNC_CONTACT プロパティーの値を検査して、後続の複数の経路から 1 つを決定するコードが含まれます。

SalesOrderProcessing は USE_RETRIEVE ダイアグラムを処理してから Additional Processing 3 サブダイアグラムを処理するので、DestinationAppRetrieveBusObj 変数にすでに宛先アプリケーションのビジネス・オブジェクトが含まれている場合があります。

## 従属データの再帰的検査の実行

多くのビジネス・オブジェクトは、自身とは異なるタイプのビジネス・オブジェクトを参照します。例えば、Order は Customer、Contact、および Item を参照します。Order は別の Order ビジネス・オブジェクトを参照しません。

Customer、Contact、および Item オブジェクトを参照できる汎用ビジネス・オブジェクトは Order のみではないので、Order を変更するコラボレーションは、参照先ビジネス・オブジェクトの処理を適切なラッパー・コラボレーションに代行させる必要があります。複数のコラボレーションを実行する環境でデータの整合性を維持するため、製品に用意されている 2 つのコラボレーションが同一タイプのビジネス・オブジェクトを変更することはありません。

ただし、InstalledProduct および Item ビジネス・オブジェクトを変更するコラボレーションを分析する場合、ビジネス・オブジェクトの変更についての説明はより複雑になります。

***InstalledProduct ビジネス・オブジェクトとその親 InstalledProduct:*** Order と同様に、InstalledProduct ビジネス・オブジェクトは Customer、Contact、および Item を参照できます。ただし、他の汎用ビジネス・オブジェクトとは異なり、InstalledProduct はその親も参照できます。図 12 に、複数の InstalledProduct ビジネス・オブジェクト間の関係を示します。

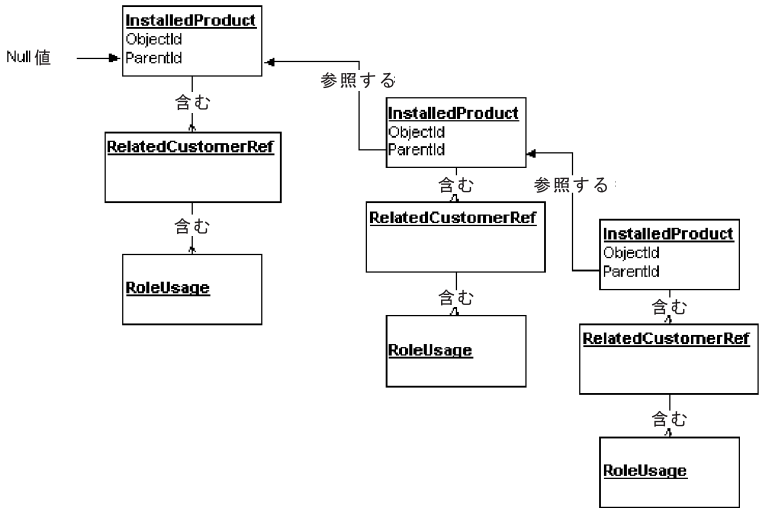



*図 12. 複数の InstalledProduct ビジネス・オブジェクト間の関係*

図 12 は、親 InstalledProduct に子 InstalledProduct が含まれていないことを示しています。その代わり、子の ParentId 属性には、String 型の親への参照が含まれています。ParentId 属性が null である InstalledProduct は、その製品階層のトップにあります。

***Item ビジネス・オブジェクトとその必須 Item オブジェクト:*** Item ビジネス・オブジェクトには、ビジネス・オブジェクトの変更における別のレベルの複雑さもあります。 49 ページの表 14 に示されるように、WebSphere Business Integration Server Express には Item ビジネス・オブジェクトのファミリーが用意されています。ビジネス・インテグレーション・システムでは、Item ビジネス・オブジェクトの柔軟性を使用します。コラボレーションは Item ビジネス・オブジェクトの ITEM_TYPE プロパティーを使用して、自身がどのタイプの Item を処理するよう構成されているか判断します。

Item は別の Item を親として参照しませんが、Item ビジネス・オブジェクトは Item ファミリー内の別のビジネス・オブジェクトを前提条件として参照できます。

例えば、ItemOrder と ItemPlanning は、どちらも ItemBasic を前提条件として参照できます。デフォルトでは、ItemSync コラボレーションの PREQ_ITEMORDER および PREQ_ITEMPLANNING プロパティーは、ItemBasic を Item ビジネス・オブジェクトの前提条件として指定します。

***同一タイプまたは同一タイプ・ファミリーのビジネス・オブジェクトの再帰的処理:*** ビジネス・オブジェクトが同一タイプまたは同一タイプ・ファミリーの別のビジネス・オブジェクトを参照する場合、トリガー・ビジネス・オブジェクトを変更する同一コラボレーションは参照先ビジネス・オブジェクトを変更できます。つまり、InstalledProductSync コラボレーションはトリガー・ビジネス・オブジェクトの参照先 Customer、Contact、および Item オブジェクトの処理を代行させる必要があります。

ただし InstalledProductSync は、起動する InstalledProduct のみでなく、トリガー・ビジネス・オブジェクトの親 InstalledProduct およびその親も同期できます。さらに、ItemSync は起動する ItemOrder のみでなく、トリガー・ビジネス・オブジェクトの ItemBasic 前提条件も同期できます。トリガー・ビジネス・オブジェクトの前提条件に固有の前提条件がある場合、ItemSync はそれらをすべて同期できます。

WebSphere Business Integration Server Express には、ビジネス・オブジェクトのスタックを再帰的に処理する 2 つのコラボレーションがあります。

- InstalledProductSync: 下位のトリガー・ビジネス・オブジェクトから開始して、最上位の親まで作成することにより、親 InstalledProduct ビジネス・オブジェクトのスタックを作成します。コラボレーションは、スタックの上位から下位に向かって、InstalledProduct ビジネス・オブジェクトを同期します。階層の各ノードで、コラボレーションが実際に InstalledProduct ビジネス・オブジェクトを同期する*前*に、コラボレーションは InstalledProduct タイプではないすべての参照先ビジネス・オブジェクトの処理を代行させます。つまり、InstalledProductSync は Last-In-First-Out (LIFO) メソッドを使用することにより、複数のビジネス・オブジェクトを再帰的に同期できます。

  例えば、車を製造する組織が、製品階層内の個別の取り付け製品としてコンポーネントを管理するとします。この場合、燃料噴射装置は取り付け製品として処理され、燃料噴射装置の親 (エンジン) も取り付け製品として処理され、エンジンの親 (車) も取り付け製品として処理されます。燃料噴射装置を同期するために、InstalledProductSync はスタックを構築します。燃料噴射装置はスタックの下位に位置付けされます。コラボレーションは、スタックの最上位に車を位置付けます。同期時に、InstalledProductSync は最初に車を同期し、最後に燃料噴射装置を同期します。

  InstalledProductSync は、InstalledProduct の親が InstalledProduct の作成より*前*に存在しているという前提事項を基づいています。これは、従業員が雇われるより前に社長が存在するのと同じです。
- ItemSync: 必須 Item ビジネス・オブジェクトの階層を順に処理します。このとき、必須ビジネス・オブジェクトが宛先アプリケーションにすでに存在する場合に、階層からビジネス・オブジェクトを除去します。つまり、階層内で必須ビジネス・オブジェクトを再帰的に同期する場合、ItemSync は深さ優先検索を実行します。

ItemSync の処理が InstalledProductSync の処理と異なる点は以下のとおりです。

– ItemSync は任意の数の依存関係を処理できます。このコラボレーションは、ユーザーのサイトで作成された汎用ビジネス・オブジェクトを容易に処理できるよう設計されています。適度に変更すれば、ItemSync はトリガー・ビジネス・オブジェクトまたは前提条件と同様にユーザー定義のビジネス・オブジェクトを受容できます。
– 前提条件のリストを下位に進むにつれて、ItemSync は失敗した前提条件を評価して、その前提条件に、階層がそれよりも上位で正常に検索された前提条件がないか判断します。そのような前提条件がある場合、ItemSync は失敗した前提条件のみをリストに保持します。

CollaborationFoundation を変更して再帰的処理を実行しようと計画している場合は、InstalledProductSync および ItemSync コラボレーションのコードを調べてください。これらのコラボレーションのシナリオを、CollaborationFoundation から作成したコラボレーションにコピーできます。次にこのシナリオを変更して、特定の要件に合わせることができます。これらのコラボレーションについての詳細は、InstalledProductSync および ItemSync の参照ページを参照してください。

## WrapperFoundation テンプレート

WrapperFoundation コラボレーション・テンプレートは、Business Integration Express 標準を順守するユーザー定義のラッパー・コラボレーションの開発を容易にするツールです。ラッパー・コラボレーションとは、別のコラボレーションのビジネス・オブジェクトの検査または同期を処理するコラボレーションです。呼び出し側コラボレーションは、自身のトリガー・ビジネス・オブジェクトで参照されるトップレベルのビジネス・オブジェクトをラッパー・コラボレーションに送信します。

例えば、SalesOrderProcessing コラボレーションは汎用 Order ビジネス・オブジェクトを同期します。汎用 Order には、注文を行う顧客を表す汎用 Customer ビジネス・オブジェクトへの参照が含まれます。また、汎用 Order には汎用 OrderLineItem ビジネス・オブジェクトの配列も含まれます。各 OrderLineItem は汎用 Item ビジネス・オブジェクトを参照し、注文された品目を表します。

このような状況でコラボレーション・ロジックをモジュール化するため、Business Integration Express では、汎用 Order とそれが参照する汎用ビジネス・オブジェクトを処理するための別々のコラボレーション・テンプレートが用意されています。例えば、Business Integration Express には、Customer および Item ビジネス・オブジェクトを参照する Order ビジネス・オブジェクトを処理するために、以下のテンプレートが用意されています。

- SalesOrderProcessing: 注文を処理する。
- CustomerWrapper および CustomerSync: 参照先顧客を処理する。
- ItemWrapper および ItemSync: 参照先品目を処理する。

ビジネス・オブジェクトの処理をそれぞれ異なる特定のコラボレーションに分けると、各コラボレーション・テンプレートの再利用が増えるだけでなく、2 つのコラボレーションが同じデータを同時に変更しないようにします。詳細については、64 ページの『並行処理の問題点』を参照してください。

すべての同期における整合性を維持し、コラボレーションを使用するため、WebSphere Business Integration Express では、CollaborationFoundation テンプレートからこれらのコラボレーション用のテンプレートが作成されます。すべてのラッパー・コラボレーションで整合性を維持するため、すべてのラッパー・コラボレーション・テンプレートは WrapperFoundation テンプレートから作成されます。WrapperFoundation を使用すると、ユーザー固有のラッパー・コラボレーションを作成できます。

## WrapperFoundation の機能

WrapperFoundation テンプレートから生成されたラッパー・コラボレーション・オブジェクトは、自身のトリガー・ビジネス・オブジェクトが宛先アプリケーションに存在するか検証したり、同期を容易にすることができます。このセクションの内容は次のとおりです。

- 『検証プロセス』
- 『同期プロセス』

### 検証プロセス

ラッパー・コラボレーションがその参照値と Exists 動詞を持つトリガー・ビジネス・オブジェクトを受け取ると、宛先アプリケーションでトリガー・ビジネス・オブジェクトを検索してそれが存在するか検証します。ラッパー・コラボレーションは、この検証に Retrieve 動詞を使用します。

検証が成功した場合、ラッパー・コラボレーションは呼び出し側コラボレーションに成功状況を戻します。

検証が失敗した場合、ラッパー・コラボレーションの振る舞いは CONTINUE_WITH_WARNING 構成プロパティーの設定によって異なります。

検証プロセスの詳細および図は、ご使用のラッパー・コラボレーション用のユーザーズ・ガイドを参照してください。

### 同期プロセス

ラッパー・コラボレーションがその参照値と Sync 動詞を持つトリガー・ビジネス・オブジェクトを受け取ると、ビジネス・オブジェクトの同期を容易にします。このためには、ラッパー・コラボレーションはソース・アプリケーションからすべての値を検索し、完全な値と Create 動詞を持つビジネス・オブジェクトを適切な同期コラボレーションに送信します。

同期コラボレーションが宛先アプリケーションでのビジネス・オブジェクトの作成に成功した場合、ラッパー・コラボレーションは呼び出し側コラボレーションに成功状況を戻します。

同期コラボレーションが宛先アプリケーションでのビジネス・オブジェクトの作成に失敗した場合、ラッパー・コラボレーションの振る舞いは CONTINUE_WITH_WARNING 構成プロパティーの設定によって異なります。

呼び出し先同期コラボレーションが既存オブジェクトの作成を試行してラッパー・コラボレーションに失敗状況を戻すことのないように、同期コラボレーションを以下のように設定してください。

- USE_RETRIEVE 構成プロパティーを true に設定する。
- INFORMATIONAL_EXCEPTIONS 構成プロパティーを 3010 に設定する。

同期プロセスの詳細および図は、ご使用のラッパー・コラボレーション・テンプレート用のユーザーズ・ガイドを参照してください。

## WrapperFoundation テンプレートの使用

WrapperFoundation テンプレートを使用する場合、以下の操作を実行します。

1. WrapperFoundation テンプレート全体を自身の開発域にコピーします。
2. テンプレート名を、作成中のコラボレーションの名前に変更します。
3. Process Designer の「テンプレート定義」ウィンドウの「ポートおよびトリガー・イベント」タブで、既存のポートのタイプ (DestinationAppRetrieve、From、SourceApp、To) を変更して、適切なビジネス・オブジェクトをこれらのポートに割り当てます。例えば InvoiceWrapper テンプレートを作成するには、各ポートのビジネス・オブジェクトをデフォルト値 (Controller) から Invoice ビジネス・オブジェクトに変更する必要があります。詳細については、『WrapperFoundation ポート』を参照してください。

## WrapperFoundation ポート

図 13 に、WrapperFoundation のポートを示します。図に続く表には、各ポートの詳細を示します。

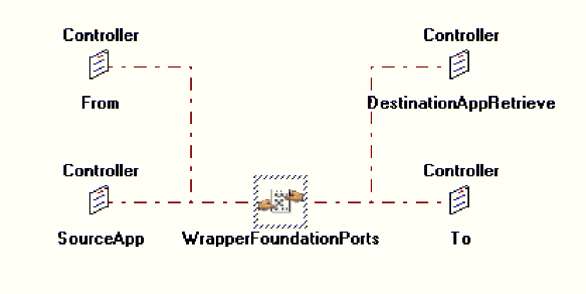



*図 13. WrapperFoundation コラボレーションのポート*

### WrapperFoundation の DestinationAppRetrieve ポート

表 15 に、WrapperFoundation テンプレートの DestinationAppRetrieve ポートの機能をリストします。

*表 15. WrapperFoundation DestinationAppRetrieve ポートのポート機能*

| ポート機能 | 値 |
|---|---|
| ビジネス・オブジェクト | ラッパー・コラボレーションの名前の由来となるビジネス・オブジェクト。例えば、CustomerWrapper は参照値を持つ Customer ビジネス・オブジェクトを送信します。このビジネス・オブジェクトのデフォルトは Controller です。 |
| 使用する動詞 | Retrieve |

*表 15. WrapperFoundation DestinationAppRetrieve ポートのポート機能 (続き)*

| ポート機能 | 値 |
|---|---|
| 目的 | 宛先アプリケーションからビジネス・オブジェクトを検索します。 |
| バインド先 | 宛先アプリケーションのコネクター |

## WrapperFoundation の From ポート

表 16 は、WrapperFoundation テンプレートの From ポートの機能をリストしています。

*表 16. WrapperFoundation From ポートのポート機能*

| ポート機能 | 値 |
|---|---|
| ビジネス・オブジェクト | ラッパー・コラボレーションの名前の由来となるビジネス・オブジェクト。例えば、CustomerWrapper は Customer ビジネス・オブジェクトによって起動されます。デフォルト値は Controller です。 |
| 使用する動詞 | Sync、Exists |
| 目的 | 呼び出し側コラボレーションからトリガー・ビジネス・オブジェクトを受け取ります。 |
| バインド先 | 呼び出し側コラボレーションの To*BusObj*Wrapper ポート |

## WrapperFoundation の SourceApp ポート

表 17 に、WrapperFoundation テンプレートの SourceApp ポートの機能をリストします。

*表 17. WrapperFoundation SourceApp ポートのポート機能*

| ポート機能 | 値 |
|---|---|
| ビジネス・オブジェクト | ラッパー・コラボレーションの名前の由来となるビジネス・オブジェクト。例えば、CustomerWrapper はソース・アプリケーションから Customer ビジネス・オブジェクトを検索します。このビジネス・オブジェクトのデフォルトは Controller です。 |
| 使用する動詞 | Retrieve |
| 目的 | ソース・アプリケーションからトリガー・ビジネス・オブジェクトを検索します。 |
| バインド先 | ソース・アプリケーションのコネクター |

## WrapperFoundation の To ポート

58 ページの表 18 に、WrapperFoundation テンプレートの To ポートの機能をリストします。

*表 18. WrapperFoundation To ポートのポート機能*

| ポート機能 | 値 |
|---|---|
| ビジネス・オブジェクト | ラッパー・コラボレーションの名前の由来となるビジネス・オブジェクト。例えば、CustomerWrapper は参照値を持つ Customer ビジネス・オブジェクトを送信します。このビジネス・オブジェクトのデフォルトは Controller です。 |
| 使用する動詞 | Create |
| 目的 | コラボレーションの外部にビジネス・オブジェクトを送信します。通常は、完全な値を持つビジネス・オブジェクトを関連する同期コラボレーションに送信するために使用します。 |
| バインド先 | そのビジネス・オブジェクトを同期するかサブスクライブしているコラボレーションの From ポート |

## WrapperFoundation の拡張

CustomerPartnerWrapper および ItemWrapper は、WrapperFoundation から作成され、特定の要件に合うように変更された、WebSphere Business Integration Express コラボレーションの例です。このセクションでは、WrapperFoundation テンプレートを変更する例としてこれらの変更について説明します。

### CustomerPartnerWrapper

異なるアプリケーションは明らかに異なる方法で CustomerPartner データを使用するので、CustomerPartnerWrapper はビジネス・オブジェクトのキーを固有の方法で処理します。一部のアプリケーションでは、CustomerPartner に関連する Customer オブジェクトとその ID を CustomerPartner に含める必要があります。他のアプリケーションは、Customer オブジェクトの ID を必要とせず、Business Integration Express システムにビジネス・オブジェクトを送信する場合に Customer オブジェクト ID を提供しません。

あるコラボレーションが Customer ID を提供しないシステムから CustomerPartner ビジネス・オブジェクトを受け取り、ID を要求するシステムとそのオブジェクトを同期する必要がある場合、コラボレーションは追加データを収集して、それを宛先アプリケーションに提供する必要があります。このため、CustomerPartnerWrapper はビジネス・オブジェクトのキーの設定に WrapperFoundation から提供される次のコマンドを使用しません。

```
SourceAppBusObj.setKeys(triggeringBusObj);
```

上記のコマンド (WrapperFoundation が Sync シナリオの Retrieve from Source ノードに用意している) を使用する代わりに、CustomerPartnerWrapper は以下の 2 つの個別のステートメントでキー値を取得します。

```
SourceAppBusObj.set("ObjectId",
     triggeringBusObj.getString("ObjectID"));
SourceAppBusObj.set("AdditionalKey",
     triggeringBusObj.getString("AdditionalKey"));
```

これらのステートメントも Sync シナリオの Retrieve from Source ノードにあり、どちらも潜在キーとして ObjectId 属性と AdditionalKey 属性を使用します。

WrapperFoundation は、Sync シナリオの Prepare Object for Sync Collaboration ノードに次のコードを持たせています。

```
ToBusObj =(SourceAppBusObj).duplicate();
ToBusObj.setKeys(triggeringBusObj);
```

等価のノードにおいて、CustomerPartnerWrapper は 2 つの個別ステートメントを使用して ToBusObj のキーを設定します。

```
ToBusObj =(SourceAppBusObj).duplicate();
ToBusObj.set("ObjectId", triggeringBusObj.getString("ObjectId"));
ToBusObj.set("AdditionalKey", triggeringBusObj.getString("AdditionalKey"));
```

## ItemWrapper

49 ページの表 14 は、WebSphere Business Integration Express に用意されている 4 タイプの汎用 Item ビジネス・オブジェクトを示しています。ユーザーは、自身の環境の特定の要件に合う固有のタイプの汎用 Item ビジネス・オブジェクトを定義することが前提となっています。したがって、ItemSync コラボレーションはユーザー定義のトリガー・ビジネス・オブジェクトと必須 Item ビジネス・オブジェクトを処理するために容易に拡張できるよう設計されています。詳細については、ItemSync コラボレーションのユーザーズ・ガイドを参照してください。

ItemWrapper はあらゆるタイプの汎用 Item ビジネス・オブジェクトによって起動されるので、コラボレーションはまず自身を起動したビジネス・オブジェクトのタイプを判別する必要があります。このため、ItemWrapper の Verify および Sync シナリオのシナリオ定義には、次のステートメントが含まれます。このステートメントは、他のラッパー・コラボレーション・テンプレートには含まれません。

```
BusObj getItemFlvrBusObj = new BusObj(triggeringBusObj.getType());
```

上記のステートメントにより、ItemWrapper 固有のビジネス・オブジェクト (getItemFlvrBusObj) が作成されます。また、このステートメントはトリガー・ビジネス・オブジェクトのタイプも取得し、それを使用して ItemWrapper の検証および同期シナリオで使用するビジネス・オブジェクトのタイプを設定します。

すべてのラッパー・コラボレーションは、コラボレーションのテンプレート定義の宣言節に DestinationAppBusObj および SourceAppBusObj ビジネス・オブジェクトを作成します。これらのビジネス・オブジェクトのタイプは、トリガー・ビジネス・オブジェクトのタイプと同じです。他のラッパー・コラボレーションは、DestinationAppBusObj および SourceAppBusObj を使用して宛先およびソース・アプリケーションからデータを検索しますが、ItemWrapper は getItemFlvrBusObj を使用します。Verify シナリオで宛先アプリケーションから項目を検索する前、または Sync シナリオでソース・アプリケーションから項目を検索する前に、ItemWrapper は triggeringBusObj のキー値を使用して getItemFlvrBusObj のキーを設定します。

```
getItemFlvrBusObj.setKeys(triggeringBusObj);
```

Sync シナリオでは、ItemWrapper は getItemFlvrBusObj の値を使用して、宛先への送信を行うビジネス・オブジェクトを以下のように作成します。

```
toItemFlvrBusObj = getItemFlvrBusObj.duplicate();
//toItemFlvrRef.copy(getItemFlvrRef);
toItemFlvrBusObj.setKeys(triggeringBusObj);
```

標準ラッパー・コラボレーション (ContactWrapper など) は、SourceAppBusObj からインスタンス化される ToBusObj を作成します。以下に、この作成およびインスタンス化を示します。

```
ToBusObj =(SourceAppBusObj).duplicate();
ToBusObj.setKeys(triggeringBusObj);
```

ItemWrapper は標準 FromBusObj を使用しないので、コラボレーションのコードでは Verify および Sync シナリオの Initialize Variables ノードのコードをコメント化します。

### テンプレートへのその他の拡張

WrapperFoundation を拡張して、追加のポート、プロパティー、処理ロジック、およびメッセージを組み込むことができます。詳細については、41 ページの『CollaborationFoundation の拡張』を参照してください。

## コラボレーション・グループの作成

コラボレーション・グループ とは、組み合わされたビジネス・プロセスを表す一連のコラボレーション・オブジェクトです。コラボレーション・グループを使用することで、異なるロジック単位を組み合わせることができます。コラボレーション・オブジェクトは同じタイプのポートを通じて互いにバインドされます。また、これらのポートを通じてコネクターにバインドすることもできます。

コラボレーション・グループには以下のメリットがあります。

- ロジックをモジュール化できます。いったん開発し、テストしたロジック単位は、何度でも配置できます。
- 既存のコラボレーションを拡張できます。既存のコラボレーションを呼び出す、または既存のコラボレーションによって呼び出されるコラボレーション・テンプレートを作成できます。

コラボレーション・グループは、2 つ以上のコラボレーションから成ります。グループ内のコラボレーションは他のコラボレーションとバインドされます。このとき、必ず呼び出し側コラボレーションと呼び出し先コラボレーションという概念が存在します。互いにバインドされている 2 つのコラボレーションでは、一方が呼び出し側コラボレーションで他方が呼び出し先コラボレーションです。呼び出し側コラボレーションは、そのサービス呼び出しの 1 つが、別のコラボレーションの実行を起動するビジネス・オブジェクトを送信するようにバインドされます。呼び出し先コラボレーションは、そのトリガー・イベントであるビジネス・オブジェクトを受け取ります。呼び出し先コラボレーションは、実行後に結果を呼び出し側に戻します。図 14 を参照してください。

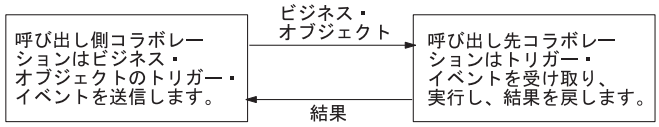



*図 14. 呼び出し側と呼び出し先コラボレーション*

コラボレーション・グループ内では、長期存続ビジネス・プロセスをサポートしていないコラボレーションは、長期存続ビジネス・プロセスとして配置されているコラボレーションにバインドできません。

## コラボレーション・グループの例: Collaboration for Customer Manager

コラボレーション・グループの一例は、以下のコラボレーションで構成される Business Integration Express 製品、Collaboration for Customer Manager です。

- CustomerSync
- CustomerWrapper
- CustomerPartnerSync
- CustomerPartnerWrapper

Collaboration for Customer Manager をインストールすると、すべてのコラボレーション・テンプレートを受け取ることになります。コラボレーションをさまざまな方法で構成し、バインドすることで (ポートを使用してコラボレーション間の通信を確立する) 統合プロセスを形成できます。

CustomerSync コラボレーションは SoldTo 顧客を同期します。つまり、CustomerSync コラボレーションは SoldTo 顧客を使用してイベントとデータを結び付けます。また、関連する顧客情報に関するデータとイベントを同期させることもできます。この場合、CustomerSync をいくつかのプリプロセスを実行する CustomerPartnerWrapper にバインドし、CustomerPartnerWrapper を CustomerPartnerSync にバインドできます。図 15 に、このバインディングのセットを示します。


*図 15. バインドされたコラボレーション・グループ*

## コラボレーション・グループの作成

このセクションでは、コラボレーション・グループを作成する一般的な手順を示します。

- 呼び出し側コラボレーションの場合

- 呼び出し先コラボレーションに渡すビジネス・オブジェクトのタイプ用のポートを作成します。
- ビジネス・オブジェクトを渡すサービス呼び出しと、呼び出し先コラボレーションに処理させる動詞をセットアップします。
- サービス呼び出しの結果を通常の方法で処理します。

**注:** コラボレーション・グループの 1 つのコラボレーションが、「Service Call In-Transit」永続性のために構成されている場合は、そのグループ内のすべてのコラボレーションが InterChange Server Express によって自動的に構成され、一貫性のあるリカバリーの振る舞いが維持されます。詳細については、182 ページの『サービス呼び出しおよび正確に 1 回のみの要求』を参照してください。

- 呼び出し先コラボレーションの場合
  - 呼び出し側コラボレーションから受け取るビジネス・オブジェクトのタイプ用のポートを作成します。
  - シナリオを作成し、そのシナリオにトリガー・イベントを割り当てます。

## Web サービスの組み込み

WebSphere Business Integration Express は、コラボレーションでの Web サービスの使用をサポートしています。*Web サービス* は、パブリック・インターフェースとバインディングが XML によって定義されるモジュラー・アプリケーションで、HTTP や SOAP などのオープン・プロトコルから利用できます。Web サービスには、コラボレーション・テンプレートのアクティビティー定義に含めることができます。対応するコラボレーション・オブジェクトが実行されると、Web サービスが呼び出されます。InterChange Server Express の変更は不要です。

System Manager を使用して場所を探索し、Web サービスを登録します。登録された Web サービスは、ICL (Integration Component Library) プロジェクトの一部になります。また、Web サービスに必要なすべてのビジネス・オブジェクトは自動的に生成され、ICL プロジェクトに配置されます。System Manager を使用して Web サービスを登録および管理する方法の詳細については、「システム・インプリメンテーション・ガイド」を参照してください。

コラボレーション・テンプレートに Web サービスを組み込むには、System Manager の ICL プロジェクトから Web サービスをエクスポートする必要があります。各メソッドは、機能ブロックとして Activity Editor にエクスポートされ、そこでアクティビティー定義に配置することができます。Web サービスのエクスポートとアクティビティー定義への追加方法の詳細については、160 ページの『Web サービス機能ブロック』を参照してください。

Web サービスのタイムアウト値を構成するには、ws_timeout と呼ばれるコラボレーションの構成プロパティーを追加します ( 91 ページの『コラボレーション構成プロパティーの定義 (「プロパティー」タブ)』参照)。タイムアウト値は、ミリ秒単位で指定されます。ws_timeout のデフォルト値は 10000 (10 秒) です。

## 長期存続ビジネス・プロセスの設計

コラボレーションを長期存続ビジネス・プロセスとして配置する場合には、コラボレーション・テンプレートの設計および作成時に以下の点に留意してください。

- ビジネス・プロセス中ずっと持続するデータにはグローバル・テンプレート変数またはポート変数を使用してください。
- すべての CwDBConnection オブジェクトへの参照は、長期存続ビジネス・プロセス環境でサービス呼び出しをコミットする前に解放され、すべてのアクティブなデータベース・トランザクションは暗黙にコミットされます。必要に応じて、サービス呼び出しの完了後に CwDBConnection オブジェクトを再獲得するようテンプレートを設計します。また、明示的なデータベース・トランザクション・スコープ設定を使用する場合には、サービス呼び出しの後にデータベース・トランザクション・コンテキストを再初期化してください。
- コラボレーションがアダプターにバインドされる場合には、トランスポート・メカニズムとして JMS を使用するようアダプターを構成してください。長期存続ビジネス・プロセスは、その他のタイプのトランスポートを使用するアダプターを使用できません。
- 長期存続ビジネス・プロセス・コラボレーションを 外部 Access Client へバインドすることはできません。
- コラボレーション・グループ内では、長期存続ビジネス・プロセスをサポートしていないコラボレーションを、長期存続ビジネス・プロセス・コラボレーションにバインドできません。

## 並列処理の設計

InterChange Server Express は並列処理環境を提供します。つまり、複数のコラボレーションを個別のスレッドで並行して実行できます。また、1 つのコラボレーション内の複数のスレッドを実行することもできます (マルチスレッド化と呼ばれます)。

**重要:** コラボレーションのスレッド・プールは、イベント・トリガー・フローに対してのみ 使用され、呼び出しによって起動されるフローには使用されません。ただし、呼び出しによって起動されるフローも、IBM Java Object Request Broker (ORB) のスレッド・プールを使用するという点において、その実行はマルチスレッド化されます。

### マルチスレッド化機能

各サーバーでは、ビジネス・オブジェクトのサブスクリプションを処理するために同時に発生させることのできるスレッドの最大数が決まっています。またユーザーは、それぞれの状況と、パフォーマンスの観点から最適と判断される条件に基づいて、発生させるスレッドの最大数を設定することができます。当然ながら、ユーザーが設定するスレッドの最大数がサーバーの許容最大数を超えることはできません。

作成可能なスレッドの最大数を設定するには、System Manager でスレッド数を指定します。

**注:** 宛先コネクターが並列処理用に構成されている場合は、要求が正常にアプリケーションに送信されたことを確認するようにコラボレーション・テンプレートをコーディングしてください。そのコードは、サービス呼び出しの例外遷移リンクの直後のノードに追加してください。詳細については、437 ページの『第 28 章 CollaborationException クラス』の getSubType() を参照してください。

**要確認:** 宛先コネクターが単一スレッドの場合は、マルチスレッドに対応するコラボレーションを生かすために、宛先コネクターを並列処理できるように構成する必要があります。詳細については、「*WebSphere InterChange Server システム・インプリメンテーション・ガイド*」を参照してください。

## 並行処理の問題点

並行処理環境では、データの矛盾が生じる危険性が常にあります。データの矛盾は、並行処理が複数のプロセスまたは複数のスレッドのいずれによって行われる場合にも発生します。2 つのプログラムまたは 2 つのスレッドが同じデータに同時にアクセスすると、一方がデータを変更し、それによって他方のプログラムまたはスレッドの動作に予期しない悪影響が及ぶという可能性が常にあります。並行処理環境では、共有データへのアクセスを同期することによってこの問題に対応します。つまりスレッドまたはプロセスは、データの一部をロックして、別のスレッドまたはプロセスが同時にそのデータにアクセスできないようにします。

ビジネス・インテグレーション環境で発生する可能性のあるこの問題の単純な例として、以下の状況を考えます。

- InterChange Server Express ソース・アプリケーションのアプリケーション・ユーザーは、一人の従業員の給料 $40,000 に $10,000 を追加する必要があるとします。
- アプリケーション・ユーザーが、誤って給料の増加分を $100,000 と入力します。このユーザーは入力が正しくないことに気付き、給料を再度更新し、今度は $100,000 を差し引きます。
- どちらの操作の場合も、同じ従業員 ID の Employee.Update イベントを InterChange Server Express システムに送信することになります。
- イベントは順不同で処理され、同期のため別のアプリケーションに送信されます。
- 宛先アプリケーションでは、最初の更新操作で、従業員の給料を -$60,000 に設定しようとします。なぜなら、従業員の給料 $40,000 から $100,000 を差し引こうとするからです。しかし従業員の給料は 0 以下になることはないため、これが原因で意味エラーが発生し、結果は予期しないものとなります。さらに 2 番目の更新でもエラーが発生することがあります。

InterChange Server Express には、データの整合性を確保してこの問題に対処するための以下の機能が用意されています。

- 『イベント順序付け』
- 65 ページの『イベント分離』

## イベント順序付け

*イベント順序付け* では、同じコラボレーションの 2 つのスレッドが同じデータを同時に操作することがなくなります。複数のイベントが同じビジネス・オブジェク

ト・タイプとキー値を持つ場合、サーバーはこれらをキューに入れ、到着の順にデリバリーします。最初のイベントを受け取ったコラボレーション・スレッドは、コラボレーションが次のイベントを受け取る前に完了する必要があります。このためイベント順序付けでは、マルチスレッドによる実行が存在してさまざまなスレッドがさまざまな速度で実行される場合でも、実行順序が維持されます。

イベント順序付けを使用するために特別な方法でコラボレーションを設計する必要はありません。イベント順序付けは自動的に行われます。

## イベント分離

*イベント分離* では、2 つのコラボレーションが同じデータを並行して操作することがなくなります。場合によっては、複数のコラボレーションが同じタイプのビジネス・オブジェクトを処理することがあります。 1 つのイベントが到着して、特定のコラボレーションを起動します。このコラボレーションはその実行を開始し、実行中は、InterChange Server Express 内のそのビジネス・オブジェクトのインスタンスに排他的にアクセスします。このとき同じデータに関連する別のイベントが到着すると、新たに到着したイベントは、実行中のコラボレーションが最初のイベントの処理を完了するまで InterChange Server Express によってキューに格納されます。この機能には以下に説明するような、いくつかの制約があります。

InterChange Server Express は、イベント分離を自動的に実行しません。コラボレーション開発者は、イベント分離を利用するための特定の方法によってテンプレートを設計する必要があります。このセクションでは、規則について説明し、この目標を達成するために設計時に決定するいくつかの事例を示します。

**注:** このセクションの指針は、複数のコラボレーションが使用される環境で動作し、データを変更する操作を実行するコラボレーションにのみあてはまります。開発しているコラボレーションが検索操作のみを実行し、サーバー上でそのビジネス・オブジェクト・タイプを使用する唯一のコラボレーションの場合には、このセクションの指針は無視してかまいません。

### イベント分離が適用される場合

イベント分離が適用されるかどうかは、到着するイベントとアクティブなコラボレーションのポートを InterChange Server Express が分析して実行時に決定されます。イベント分析の基準は、イベント順序付けの場合と同じで、ビジネス・オブジェクト・タイプとキー値が同じ場合にはイベントは同じであるとみなされます。

アクティブなコラボレーションの分析では、コネクターにバインドされている各コラボレーションの一連のポートが考慮されます。ポート・マッチング において、InterChange Server Express は以下のことを確認します。

- いずれかのコラボレーションの中でポートがコネクターの同じセットにバインドされているかどうか。
- コネクターの同じセットにバインドされているポートの中で、コネクターにバインドされているポートのビジネス・オブジェクト・タイプが同じであるかどうか。

例えば、2 つのコラボレーションの両方が以下のようにポートにバインドされている場合、これらのコラボレーションはマッチング・ポートを持ちます。

コネクター 1/ビジネス・オブジェクト・タイプ A

コネクター 2/ビジネス・オブジェクト・タイプ B

入力されるイベントと出力される要求/応答のいずれかにポートが使用されるかは問題ではありません。コネクター・バインディングとビジネス・オブジェクトのタイプのみが重要です。

イベント分離が適用されるコラボレーションを決定するときに、他のコラボレーションにバインドされているポートは考慮されません。

***ポート・マッチング例: マッチングするポート:*** 図 16 に、イベント分離が適用される 2 つのコラボレーション X と Y を示します。コラボレーションの端の小さな長方形はポートを示します。

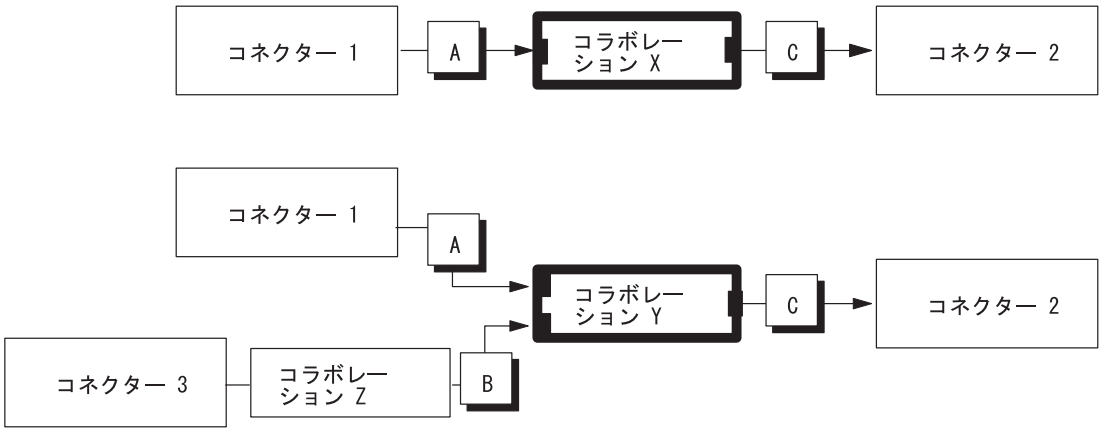



*図 16. マッチングするポート*

図 16 で、X には 2 つ、Y には 3 つのポートがあります。しかし、ポート・マッチングでは、コネクターにバインドされている Y の 2 つのポートのみが考慮されます。コラボレーション Z にバインドされているポートは無視されます。いずれのコラボレーションも、コネクターにバインドされている以下のポートがあります。

- ビジネス・オブジェクト・タイプ A 用に定義され、コネクター 1 にバインドされているポート
- ビジネス・オブジェクト・タイプ C 用に定義され、コネクター 2 にバインドされているポート

この例は、イベント分離の基準を満たすため、サーバーは入力イベントまたはトリガー・イベントを分離します。このため、イベント A のインスタンスはこれら 2 つのコラボレーションにおける分離の対象になります。

***ポート・マッチング例: マッチングしないポート:*** コラボレーションが比較されるとき、サーバーでは、すべてのポートが考慮されます。ポート・マッチングの分析対象となるポートは、トリガー・イベントを受け取るポートに限定されません。2 つのコラボレーションが同じコネクターから同じタイプのイベントを受け取るが、2 つの異なるコネクターに出力ビジネス・オブジェクトを送信する場合、これらのコラボレーションのイベントは分離されません。

図 17 に、出力ポートが異なるコネクターにバインドされている 2 つのコラボレーションを示します。これらのイベントのインスタンスは分離されません。

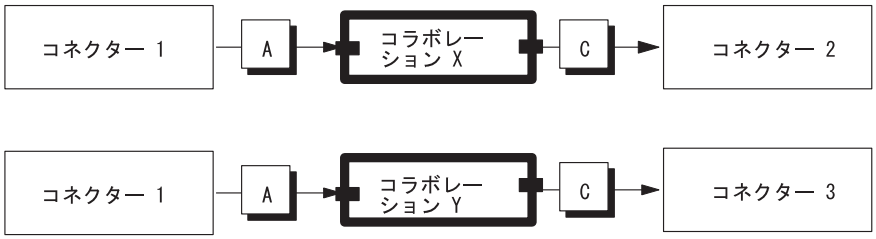



*図 17. マッチングしないポート*

## 設計規則

イベント・インスタンスの分離機能を利用する場合には、コラボレーションを特定の方法で設計する必要があります。このセクションでは、以下の方法について説明します。

- 代行を使用してコラボレーション・グループを形成する。
- 子ビジネス・オブジェクトを参照値オブジェクトとして処理する。

***代行の使用:*** ビジネス・オブジェクトを変更する各コラボレーション・テンプレートは、そのタイプのビジネス・オブジェクトのみを変更するように設計する必要があります。コラボレーションが、子ビジネス・オブジェクトなど別のタイプのビジネス・オブジェクトを変更する必要がある場合は、その別のタイプのビジネス・オブジェクトの変更のみを目的とする個別のコラボレーションを作成する必要があります。つまり、最初のコラボレーションは、他のビジネス・オブジェクトを変更するために、第 2 のコラボレーションに*代行させる* (渡す) ことになります。

1 つのコラボレーションが 1 種類のビジネス・オブジェクトのみを変更するようにする規則は、データの整合性を維持するうえで役立ちます。これにより、複数のコラボレーションが同じタイプのビジネス・オブジェクトの同じインスタンスを並行して変更することが防止されます。代行により、子ビジネス・オブジェクトのデータの整合性が、別のコラボレーションによって処理される同じビジネス・オブジェクトのインスタンスに関して維持されます。ビジネス・オブジェクトは 1 つのコンテキストでは子として使用し、別のコンテキストでは自身として使用することができます。

ビジネス・オブジェクト A と、これに子ビジネス・オブジェクトとして関連付けられている一連の情報 B とを扱うビジネス・プロセスを書く必要があるとします。この場合のビジネス・オブジェクトの構造は、図 18 に示すようになり、B ビジネス・オブジェクトは A ビジネス・オブジェクトの子オブジェクトです。

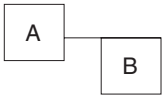



*図 18. 階層ビジネス・オブジェクトの例*

B ビジネス・オブジェクトを処理するコラボレーションがすでに存在する場合、B 子ビジネス・オブジェクトに対する操作をそのコラボレーションに代行させる必要があります。または、別のコラボレーションを作成する必要が生じることがあります。

A ビジネス・オブジェクトに関連付けられているデータを操作するときには、A ビジネス・オブジェクトとその一連の B データ (つまり子ビジネス・オブジェクト) の両方を操作する必要があります。このため、2 種類のコラボレーション・テンプレート (一方のコラボレーションはビジネス・オブジェクト A を変更し、他方は子ビジネス・オブジェクト B を変更する) を作成することになります。また場合によっては、これらの 2 つのテンプレートをコラボレーション・グループに結合することになります。各コラボレーション・テンプレートは、それぞれ 1 つのビジネス・オブジェクトに対する操作を処理します。

図 19 に、A プロセッサー・コラボレーションと B プロセッサー・コラボレーションが含まれるコラボレーション・グループ A/B を示します。A プロセッサー・コラボレーションは、A ビジネス・オブジェクトを処理します。A プロセッサー・コラボレーションが B 子ビジネス・オブジェクトを変更する必要があるとき、A プロセッサー・コラボレーションはサービス呼び出しを使用して B ビジネス・オブジェクトを B プロセッサー・コラボレーションに送信します。図 19 で、点線は代行を表します。

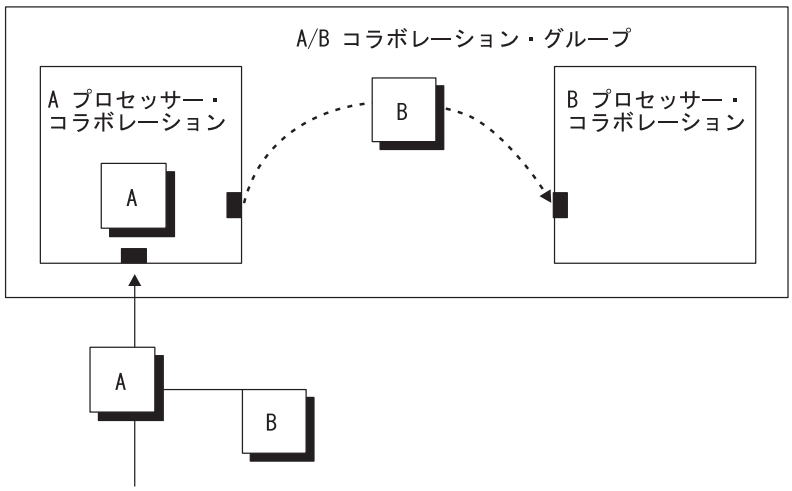



*図 19. 子ビジネス・オブジェクトの代行*

***参照値を持つビジネス・オブジェクトとして子ビジネス・オブジェクトを処理する:*** 代行された子ビジネス・オブジェクト (図 19 の B プロセッサー・コラボレーションの場合など) をコラボレーションが受け取ると、そのコラボレーションはそのビジネス・オブジェクトを、参照値を持つビジネス・オブジェクトとして処理する必要があります。*参照値を持つビジネス・オブジェクト* には、ビジネス・オブジェクトの基本キーとして定義されている属性の値のみが含まれます。これに対して、*完全な値を持つビジネス・オブジェクト* にはその他の属性値が含まれます。

この章の中の図では、参照値を持つビジネス・オブジェクトは (r) の印が付き、完全な値を持つビジネス・オブジェクトは (f) の印が付いています。図 20 は、参照値子ビジネス・オブジェクトを持つビジネス・オブジェクトの例です。

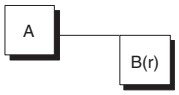



*図 20. 参照値子ビジネス・オブジェクトを持つ階層ビジネス・オブジェクト*

発信元コネクターに応じて、イベントは参照値子ビジネス・オブジェクトまたは完全な値を持つ子ビジネス・オブジェクトによって送信できます。このため、代行された子ビジネス・オブジェクトを受け取るコラボレーションは、その属性値すべてを受け取るか、または基本キー値のみを受け取ります。しかし、コラボレーションは、受信する代行された子ビジネス・オブジェクトを常に参照値を持つビジネス・オブジェクトとして処理する必要があります。基本キー値が正しい場合のみを想定し、基本キー以外の属性値は無視する必要があります。

コラボレーションが子ビジネス・オブジェクトのキー以外の属性に対する操作を実行する必要がある場合、ソース・アプリケーションからビジネス・オブジェクトの完全な値を持つバージョンを検索することによって参照を解決する必要があります。子ビジネス・オブジェクトが参照値を持つビジネス・オブジェクトの場合、検索操作により追加の属性値が取得されます。子ビジネス・オブジェクトが完全な値を持つビジネス・オブジェクトの場合、現在有効な関連データが検索されます。

図 19 に、参照値を持つビジネス・オブジェクトとしての B の代行と、ソース・コラボレーションからの検索による参照の解決を示します。

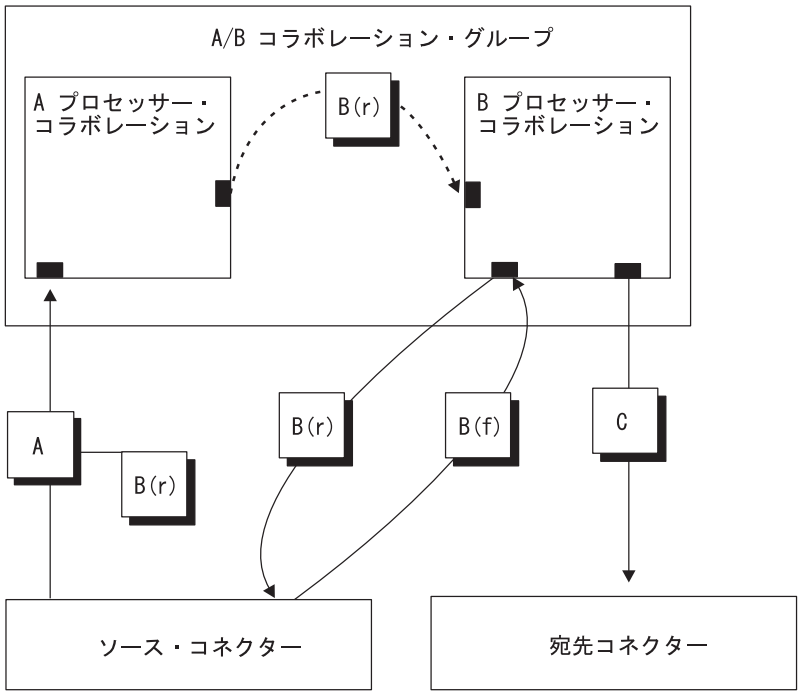



*図 21. 参照の解決*

## 例

図 22 に、B プロセッサー・コラボレーションと B-to-C コラボレーションという 2 つの異なるコラボレーション間でイベント分離が適用されている状況を示します。

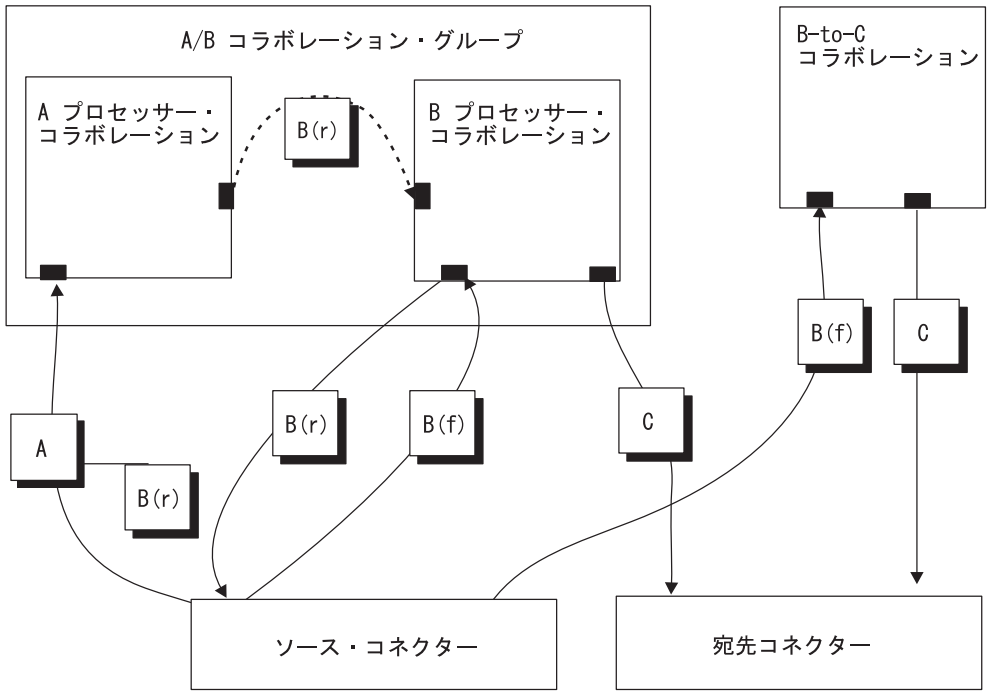



*図 22. イベント分離の対象となる 2 つのコラボレーション*

B プロセッサー・コラボレーションと B-to-C コラボレーションのいずれにも以下があてはまることに注意してください。

- タイプ B のビジネス・オブジェクトのイベントを受け取ります。
- タイプ C のビジネス・オブジェクトを生成します。
- 同じコネクター・セットにバインドされています。

このため、ポート・マッチングの分析結果として、これらのコラボレーションにはイベント分離が適用されます。

次の例 (図 23 を参照) は、同じ環境において 2 つの異なる方法で同じテンプレートから作成されたコラボレーション・オブジェクトを使用する方法を示します。この方法を使用すると、コラボレーションに機能を追加する場合など、既存のコラボレーション・テンプレートを再利用すると共に拡張することができます。

Y プロセッサー・コラボレーション・テンプレートが存在し、Y プロセッサー・テンプレートからインスタンス化されたコラボレーション・オブジェクトである Y プロセッサー・コラボレーションが使用されているとします。そして、Y プロセッサー・コラボレーション・テンプレートの機能を含めて拡張する新しいコラボレーション機能を作成したいとします。

これを行う 1 つの方法は、Y プロセッサー・コラボレーション・テンプレートを再利用して、コラボレーション・グループで使用する新しい Y プロセッサー・コラボレーション・オブジェクトを作成することです。つまり、Y プロセッサー・テンプレートから 2 番目の Y プロセッサー・コラボレーション・オブジェクトである Y プロセッサー・コラボレーション 2 をインスタンス化し、これをコラボレーション・グループに入れます。これで、イベント分離を必要とする 2 つの Y プロセッサー・コラボレーションが作成されます。中間コラボレーション (この例ではコラボレーション Z) は、追加の機能を提供し、Y プロセッサーへの変更を必要とせずにイベントを確実に分離します。

図 23 では、濃い線で囲まれたコラボレーション (コラボレーション Z と Y プロセッサー・コラボレーション 1) によって受信される Y ビジネス・オブジェクトにイベント分離が適用されます。数値は処理の順序を示します。

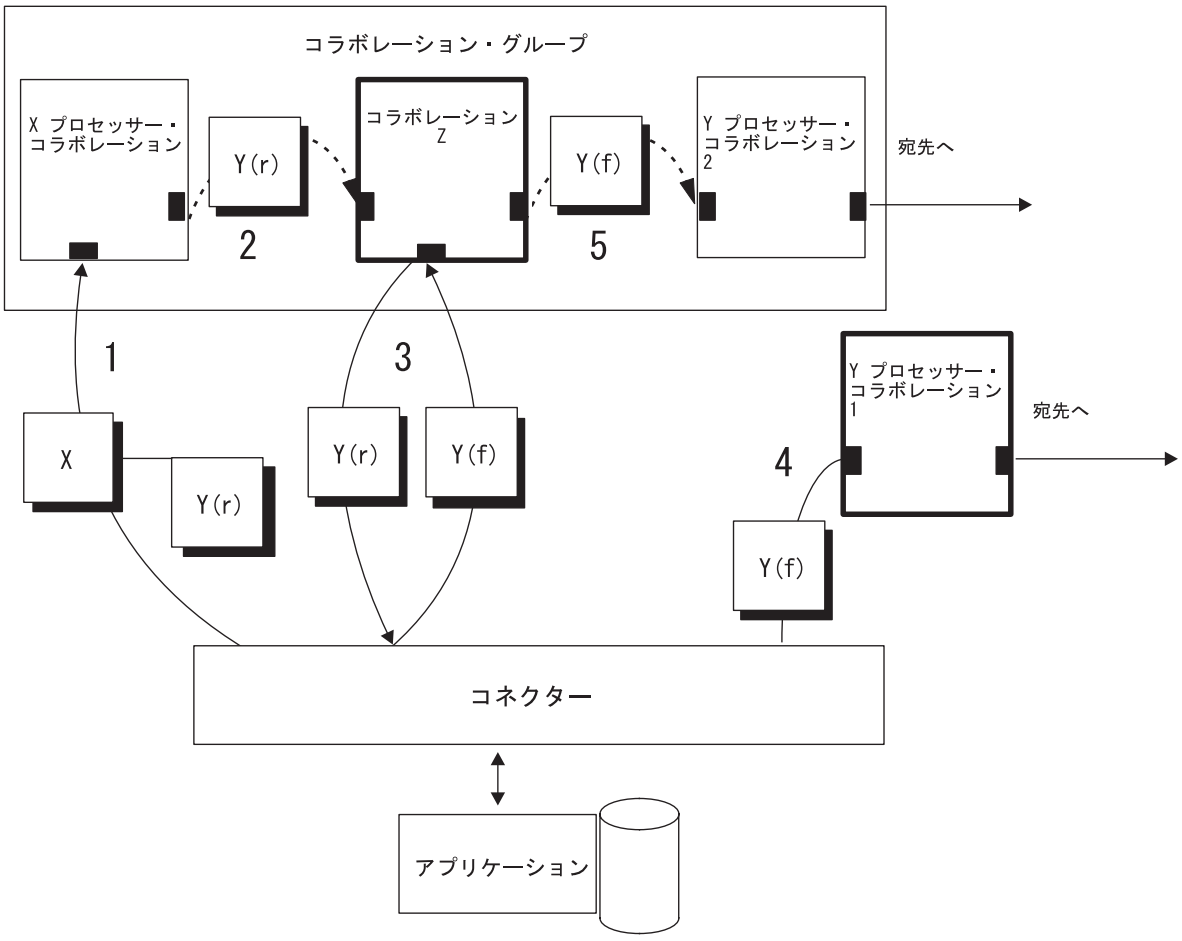



*図 23. 完全な値を持つビジネス・オブジェクトの検索*

コラボレーション Z と Y プロセッサー・コラボレーション 2 は、イベント分離に関しては協力して機能します。代行ビジネス・オブジェクトの指針は、コラボレーション Z が Y プロセッサー・コラボレーション 2 の代理となることで守られています。

# 国際化対応コラボレーション

*国際化対応コラボレーション* とは、特定のロケールに合わせてカスタマイズできるように作成されているコラボレーションです。*ロケール* とは、ユーザーの環境において、エンド・ユーザーの特定の国、言語、または地域に固有のデータの処理方法に関する情報を提供する部分です。ロケールは一般に、オペレーティング・システムの一部としてインストールされます。ロケール依存データを処理するコラボレーションを作成することを、コラボレーションの*国際化対応* (I18N) といいます。特定のロケールに従って国際化対応コラボレーションを処理することをコラボレーションの*ローカライズ* (L10N) と呼びます。

このセクションでは、国際化対応コラボレーションに関する以下の情報について説明します。

- 『ロケールとは』
- 73 ページの『国際化対応コラボレーションの設計上の考慮事項』

## ロケールとは

*ロケール*は、ユーザー環境の以下の情報を提供します。

- 言語および国 (または地域) に関連する国/地域別情報:
  - データ・フォーマット:
    - 日付: 曜日および月の名前とその省略名、および日付の構成 (日付の区切り文字を含む) を定義します。
    - 数値: 3 桁ごとの区切り記号や小数点記号、およびこれらの記号を数値中のどこに配置するかを定義します。
    - 時刻: 12 時間表示の標識 (AM および PM 標識など)、および時刻の構成を定義します。
    - 通貨値: 数値や通貨記号、およびこれらの記号を通貨値の中のどこに配置するかを定義します。
  - 照合順序: 特定の文字コード・セットおよび言語のデータをソートする方法を示します。
  - ストリング処理には、「大文字小文字」 (英大文字および英小文字) の比較、サブストリング、連結などの操作があります。
- 文字エンコード: 文字 (英字) を文字コード・セットの数値にマッピングします。例えば、ASCII 文字コード・セットでは文字 「A」を 65 にエンコードし、EBCIDIC 文字セットでは文字「A」を 43 にエンコードします。*文字コード・セット* には、1 つ以上の言語記号のすべての文字のエンコード方式が含まれます。

ロケール名のフォーマットは次のとおりです。

*ll_TT.codeset*

ここで *ll* は 2 文字の言語コード (通常は小文字)、*TT* は 2 文字の国および地域コード (通常は大文字)、*codeset* は関連する文字コード・セットの名前を表します。多くの場合、名前の *codeset* 部分はオプションです。ロケールは一般に、オペレーティング・システムのインストールの一部としてインストールされます。

# 国際化対応コラボレーションの設計上の考慮事項

このセクションでは、コラボレーションを国際化対応にする際の設計上の考慮事項を以下のように分類して説明します。

- 『ロケール依存設計の原則』
- 78 ページの『文字エンコード設計の原則』

## ロケール依存設計の原則

コラボレーションを国際化対応にするには、そのコラボレーションがロケール依存コラボレーションとしてコーディングされている必要があります。つまり、コラボレーションがロケール設定を反映して振る舞い、ロケールに基づいて適切なタスクを実行する必要があります。例えば英語を使用するロケールの場合、コラボレーションはそのエラー・メッセージを英語のメッセージ・ファイルから取得する必要があります。

Process Designer Express により作成されるコラボレーション・コードは、国際化に*対応していません*。Process Designer Express によりコラボレーション・コードが生成されたら、このセクションに記載されている手順に従い、コラボレーション・テンプレートを国際化対応にしてください。

表 19に、国際化対応コラボレーションが従う必要のあるロケール依存設計の原則を示します。

*表 19. コラボレーションのロケール依存設計の原則*

| 設計の原則 | 詳細 |
|---|---|
| エラー・メッセージと状況メッセージのテキストはすべて、コラボレーション・テンプレートから分離してメッセージ・ファイルへ保管し、ロケールの言語へ翻訳します。 | 『テキスト・ストリング』 |
| コラボレーションの実行時にビジネス・オブジェクトのロケールを保持します。 | 76 ページの『ビジネス・オブジェクト・ロケール』 |
| コラボレーション構成プロパティーは、マルチバイト文字を組み込むことができるように処理します。 | 77 ページの『コラボレーション構成プロパティー』 |
| その他のロケール固有タスクについて考慮します。 | 78 ページの『その他のロケール依存タスク』 |

***テキスト・ストリング:*** 国際化対応コラボレーションのプログラミング時には、コラボレーション・コードにテキスト・ストリングをハードコーディングするのではなく、国際化対応コラボレーションがテキスト・ストリングを必要とするときに外部メッセージ・ファイルを参照するように設計してください。テキスト・メッセージを生成する必要があるときは、コラボレーションがメッセージ番号を使用してメッセージ・ファイルから適切なメッセージを取得します。すべてのメッセージを 1 つのメッセージ・ファイルにまとめたら、このファイルのテキストを適切な言語へ翻訳してファイルをローカライズします。国際化対応メッセージ・ファイルの詳細については、 197 ページの『第 9 章 メッセージ・ファイルの作成』を参照してください。

ロギング操作、例外処理操作、E メール操作を国際化対応にするには、これらの各操作で、テキスト・メッセージの生成にメッセージ・ファイルが使用されていることを確認してください。メッセージ・ストリングをメッセージ・ファイルに記述し、各メッセージに固有の ID を割り当てます。表 20に、メッセージ・ファイルを使用する操作のタイプと、BaseCollaboration クラスの関連する Collaboration API メソッドを示します。これらのメソッドは、コラボレーション・テンプレートがメッセージ・ファイルからメッセージを取得するときに使用します。

*表 20. メッセージ・ファイルからメッセージを取得するメソッド*

| メッセージ・ファイル操作 | BaseCollaboration メソッド |
|---|---|
| ロギング | logInfo()、logError()、logWarning() |
| 例外処理 | raiseException() |
| E メール通知 | sendMail() |

**注:** InterChange Server Express 標準では、トレース・メッセージはコラボレーション・メッセージ・ファイルに含めない ことが推奨されています。トレース・メッセージは製品デバッグ処理で使用することを目的としているため、顧客ロケールの言語で表示する必要はありません。

***ロギングと例外メッセージの処理:*** ロギングと例外処理メッセージが常にコラボレーション・メッセージ・ファイルから取得されるようにするには、表 20 に示す形式のメソッドを*使用しない*でください。これらの形式を使用すると、呼び出し内で直接メッセージ・ストリングを指定できるようになります。例えばロギング宛先へエラーを記録するには、以下のように logError() を*呼び出さ*ないでください。

```
logError("Log this message to the log destination");
```

代わりにメッセージの固有 ID を作成し、コラボレーション・メッセージ・ファイルにテキストを挿入します。上記のメッセージに固有 ID として 712 が割り当てられている場合、メッセージ・ファイルのこのメッセージのエントリーは以下のようになります。

```
712
Log this message to the log destination.
```

必要に応じてこのメッセージへメッセージ・パラメーターを追加できます。

国際化対応コラボレーションでは、前述の logError() 呼び出しの代わりに、コラボレーション・メッセージ・ファイルからログ・メッセージを取得する以下の呼び出しを使用します。

```
logError(712);
```

同様に、例外メッセージはすべてコラボレーション・メッセージ・ファイルから取得します。そのために、raiseException() を以下の形式で使用することは避けてください。

```
void raiseException(String exceptionType, String message)
```

代わりに、メッセージ番号を指定する形式の raiseException() を 1 つ使用します。

***E メール・メッセージの処理:*** sendEmail() メソッドを使用して、指定の E メール宛先へメッセージを送信できます。国際化対応コラボレーションでは、E メール・

メッセージはコラボレーション・メッセージ・ファイルにまとめられます。ただし sendEmail() メソッドにはメッセージの固有 ID を指定できる形式はありません。したがって、E メール・メッセージを送信するには、まずメッセージ・ファイルからメッセージを抽出し、次に sendEmail() を使用して、取得したメッセージ・ストリングを送信します。表 21 に、コラボレーションでメッセージ・ファイルからメッセージを取得するときに使用できるメソッドを示します。

*表 21. メッセージ・ファイルからメッセージを取得するメソッド*

| コラボレーション・ライブラリー・クラス | BaseCollaboration メソッド |
|---|---|
| BaseCollaboration | getMessage() |

以下のコード・フラグメントでは、コラボレーション・メッセージ・ファイルからメッセージ 100 が取得され、E メール・メッセージに組み込まれます。

```
String retrievedMsg = getMessage(100);
sendEmail(retrievedMsg, subjectLine, recipientList);
```

***各種ストリングの処理:*** 表 20 で説明したメッセージ・ファイル操作の処理に加え、国際化対応コラボレーション・テンプレートには各種のハードコーディングされたストリングが組み込まれていてはなりません。各種ストリングはテンプレートから分離してメッセージ・ファイルにまとめてください。

ハードコーディングされたストリングを国際化対応にする手順は、以下のとおりです。

- ハードコーディングされたストリングに対し、コラボレーション・メッセージ・ファイルに固有の番号が付けられたメッセージを作成します。

  **注:** 必要に応じて、メッセージ・ファイルでは分離したストリングに関する説明を追加できます。この説明には、ストリングが使用されるシナリオの名前やアクション・ノードの番号を記述できます。この情報により、必要に応じてソースの位置を簡単に追跡し、変更することができます。
- コラボレーション・テンプレートで getMessage() メソッドを使用して、分離されたストリングをそのメッセージ番号で指定します。

例えばハードコーディングされたストリングに関する以下のコード行がコラボレーション・テンプレートにあるとします。

```
String imsg100 = "********Before entering order-to-ATP map********";
```

このハードコーディングされたストリングをコラボレーション・コードから分離するには、メッセージ・ファイルにメッセージを作成し、このメッセージに固有のメッセージ番号 (100) を割り当てます。

```
100
********Before entering order-to-ATP map********
[EXPL]
ATP Transaction: 162
```

コラボレーション・テンプレートで、ハードコーディングされたストリングを含むコードを、分離されたストリング (メッセージ 100) をメッセージ・ファイルから取得するコードに置き換えます。

```
String imsg100 = getMessage(100);
//retrieve the message numbered '    100'
String imsg100 = getMessage(100);
//display the retrieved message
```

メッセージ・ファイルの使用方法の詳細については、 197 ページの『第 9 章 メッセージ・ファイルの作成』を参照してください。

***ビジネス・オブジェクト・ロケール:*** コラボレーション・オブジェクトの実行時には 2 つのロケール設定があります。

- コラボレーションが実行されている InterChange Server Express インスタンスからこのコラボレーションに継承されるロケール (*コラボレーション・ロケール* と呼ばれます)。コラボレーション・ロケールにより、コラボレーションがロギング、トレース、例外処理、および E メールに使用するテキスト・メッセージのロケールが決まります。
- コラボレーションで、トリガー・ビジネス・オブジェクトに対して使用されるフロー・ロケール。フロー・ロケールにより、コラボレーション実行時に使用されるビジネス・オブジェクトのロケール設定が決まります。

作成されたビジネス・オブジェクトのデータには、常にロケールが関連付けられています。デフォルトでは、コラボレーションで作成されるすべてのビジネス・オブジェクトでコラボレーション・ロケールが使用されます。ただし、トリガー・ビジネス・オブジェクトのロケール (フロー・ロケール) がビジネス・オブジェクトに必要なことがよくあります。コラボレーション・ロケールはフロー・ロケールとは異なることがあるので、フロー・ロケールをビジネス・オブジェクトへ割り当てる必要があります。表 22 に、フローに関連付けられているロケールを取得するためにコラボレーションで使用できるメソッドを示します。

*表 22. コラボレーションのフロー・ロケールを取得するメソッド*

| コラボレーション・ライブラリー・クラス | メソッド |
|---|---|
| BaseCollaboration | getLocale() |

コラボレーション・シナリオのフロー中にビジネス・オブジェクトのロケールが正しく維持され、適切に使用されるようにコラボレーション・テンプレートを構成する必要があります。コラボレーションからこのロケールへアクセスするときに使用されるメソッドを表 23 に示します。

*表 23. ビジネス・オブジェクトのロケールへアクセスするメソッド*

| コラボレーション・ライブラリー・クラス | メソッド |
|---|---|
| BusObj | getLocale()、setLocale() |

Process Designer Express がコラボレーション・テンプレートの新しいポートを作成するときに、このポートの BusObj オブジェクトが新規に作成され、*portName*BusObj という名前が指定されます (*portName* はポート名です)。例えば To という名前のポートを作成すると、Process Designer Express により以下のようなコードを持つ ToBusObj という名前の BusObj オブジェクトが作成されます。

```
BusObj ToBusObj = new BusObj("Item");
```

BusObj クラスのコンストラクターにより BusObj オブジェクトが作成され、そのロケールにはコラボレーション・ロケールが設定されます。ビジネス・オブジェクトのデータをフロー・ロケールに関連付ける必要がある場合には、コラボレーション・テンプレートによりビジネス・オブジェクトのロケールを変更する必要があります。

例えば図 24 で、2 つのポート To と From に対応する BusObj オブジェクトが Process Designer Express により作成されるとします。

```
BusObj ToBusObj = new BusObj(triggeringBusObj.getType());
BusObj FromBusObj = new BusObj(triggeringBusObj.getType());
```

*図 24. ポート用のビジネス・オブジェクトを作成するためのコード*

図 24 で生成されたコードを国際化対応にするコード・フラグメントを以下に示します。このコードでは、この新しい BusObj オブジェクトにフロー・ロケールを設定しています。

```
BusObj ToBusObj = new BusObj(triggeringBusObj.getType());
BusObj FromBusObj = new BusObj(triggeringBusObj.getType());

// get flow locale from BaseCollaboration
triggerLocale = getLocale();

// set newly created BusObj objects' locale to flow locale
ToBusObj.setLocale(triggerLocale);
FromBusObj.setLocale(triggerLocale);
```

BusObj() コンストラクターは、ロケール名を引き数としてとります。したがって、別の方法として、フロー・ロケールをコンストラクター呼び出しに直接渡すように、図 24 で生成されたコードを以下のように書き直すこともできます。

```
// get flow locale from BaseCollaboration
triggerLocale = getLocale();

BusObj ToBusObj = new BusObj(triggeringBusObj.getType(), triggerLocale);
BusObj FromBusObj = new BusObj(triggeringBusObj.getType(), triggerLocale);
```

**注:** BusObj クラスの copy() メソッドと duplicate() メソッドにより、ビジネス・オブジェクト・ロケールの割り当てが自動的に処理されます。したがってソース・ビジネス・オブジェクトに正しいロケールが設定されている場合には、ターゲット・ビジネス・オブジェクトにもこのロケールが割り当てられます。

***コラボレーション構成プロパティー:*** 91 ページの『コラボレーション構成プロパティーの定義 (「プロパティー」タブ)』で説明したように、コラボレーション・テンプレートでは 2 種類の構成プロパティーを使用してテンプレート実行をカスタマイズできます。

- 標準構成プロパティー。すべてのコラボレーションで使用できます。
- コラボレーション固有の構成プロパティー。特定のコラボレーション・テンプレートで定義されており、このテンプレート固有のプロパティーです。

コラボレーション構成プロパティーの名前には、米国英語 (en_US) ロケールに関連付けられているコード・セットで定義されている文字のみ 使用してください。ただし、コラボレーション構成プロパティーの値には、コラボレーション・ロケールに関連付けられているコード・セットの文字を含めることができます。

コラボレーション・テンプレートは、BaseCollaboration クラスの getConfigProperty() メソッドまたは getConfigPropertyArray() メソッドを使用して構成プロパティーの値を取得します。これらのメソッドは、マルチバイト・コード・セットの文字を適切に処理します。ただし、コラボレーション・テンプレートを確実に国際化対応させるには、これらの構成プロパティー値を取得後、コラボレーション・テンプレートのコードによりこれらの値が正しく処理されなければなりません。コラボレーション・テンプレートでは、構成プロパティー値に 1 バイト文字だけが含まれていることを前提としないでください。

***その他のロケール依存タスク:*** 国際化対応コラボレーションは、以下のロケール依存タスクも処理する必要があります。

- データのソートまたは照合: コラボレーションは、ロケールの言語または国に基づく適切な照合順序を使用する必要があります。
- ストリング処理 (比較、サブストリング、大文字/小文字の比較など): コラボレーションは、ロケールの言語の文字に基づく適切な処理を実行する必要があります。
- 日付、数値、時刻のフォーマット: コラボレーションは、ロケールに基づき適切なフォーマット設定を実行する必要があります。

## 文字エンコード設計の原則

異なるコード・セットを使用する 2 つのロケーション間でデータが転送される場合、データを適切に保持するために何らかの文字変換を実行する必要があります。Java 仮想マシン (JVM) の Java ランタイム環境は、データを Unicode で表現します。Unicode 文字セットは、ほとんどの文字コード・セット (1 バイト文字およびマルチバイト文字) の文字のエンコードが含まれている汎用文字セットです。Unicode のエンコード形式は数種類あります。統合ビジネス・システムで最もよく使用されるエンコードを以下に示します。

- Universal multiple octet Coded Character Set: UCS-2

  UCS-2 エンコードは、2 バイト (オクテット) でエンコードされた Unicode 文字セットです。

- 8-bit UCS transformation format: UTF-8

  UTF-8 エンコードは、UNIX 環境での Unicode 文字データを使用可能にする目的で設計されています。すべての ASCII コード値 (0...127) をサポートしているので、これらの値が ASCII コード以外として解釈されることはありません。各コード値は 1 バイト値、2 バイト値、または 3 バイト値として表現されます。

InterChange Server Express やそのコラボレーション・ランタイム環境など、WebSphere Business Integration Server Express システムのほとんどのコンポーネントは、Java で記述されています。したがって、WebSphere Business Integration Server Express 内のコラボレーションと他のコンポーネント間で転送されるデータは、Unicode コード・セットにエンコードされ、文字変換の必要はありません。

# 第 4 章 コラボレーション・テンプレートの構築

この章では、コラボレーション・テンプレート定義の作成および変更方法について説明します。以下の操作を実行する必要があります。

1. テンプレート定義を作成します。詳細については、『コラボレーション・テンプレートの作成』を参照してください。
2. テンプレートのプロパティーに関する情報を提供します。詳細については、 81 ページの『テンプレート・プロパティー情報の提供』を参照してください。
3. シナリオを定義します。詳細については、 97 ページの『シナリオの定義』を参照してください。
4. 定義したシナリオのアクティビティー・ダイアグラムを作成します。一般情報については、 102 ページの『アクティビティー・ダイアグラムの作成』を参照し、詳細については、 107 ページの『第 5 章 アクティビティー・ダイアグラムの使用』を参照してください。
5. 必要に応じて追加のシナリオを定義して、さらに関連するアクティビティー・ダイアグラムを作成します。
6. テンプレートのメッセージ・ファイルを作成します。詳細については、 102 ページの『メッセージ・ファイルの作成』を参照してください。
7. コラボレーション・テンプレートをコンパイルします。詳細については、 102 ページの『コラボレーション・テンプレートのコンパイル』を参照してください。
8. 必要に応じてコラボレーションをテストします。統合テスト環境の Test Connector を使用して、コラボレーションが予定どおり機能していることを確認します。詳細については、 106 ページの『コラボレーションのテスト』を参照してください。

## コラボレーション・テンプレートの作成

Process Designer Express を使用して、コラボレーション・テンプレートを作成、編集、およびコンパイルします。以下の各セクションでは、新規テンプレートの定義方法についての説明、および必要となる基本情報を提供しています。

テンプレートを作成する前に、特定の情報を入力する必要があります。その他のタイプの情報は、開発中いつでも入力できます。テンプレートの作成には、以下の情報が必要になります。

| | |
|---|---|
| テンプレート名 | 80 ページの『テンプレート定義の作成』を参照 |
| ポート | 94 ページの『ポートおよびトリガー・イベントの定義 (「ポートおよびトリガー・イベント」タブ)』を参照 |
| シナリオ定義 | 97 ページの『シナリオの定義』を参照 |

以下の情報を使用するかどうかは任意であるため、開発過程のいずれの段階でも入力できます。

| | |
|---|---|
| 長期存続ビジネス・プロセスのサポート | 82 ページの『長期存続ビジネス・プロセスのサポートの追加』を参照 |
| パッケージ名 | 84 ページの『コラボレーション・パッケージの指定』を参照 |
| 最小トランザクション・レベル | 83 ページの『最小トランザクション・レベルの指定』を参照 |
| 構成プロパティー | 91 ページの『コラボレーション構成プロパティーの定義 (「プロパティー」タブ)』を参照 |
| テンプレート変数 | 85 ページの『テンプレート変数の宣言と編集 (「宣言」タブ)』を参照 |
| import ステートメント | 86 ページの『Java パッケージのインポート』を参照 |

## テンプレート定義の作成

新しいコラボレーション・テンプレートを作成するには、System Manager で次の操作を行います。

1. プロジェクトの「コラボレーション・テンプレート」フォルダーを右マウス・ボタンでクリックして、「新規コラボレーション・テンプレートの作成」オプションをクリックします。Process Designer Express が開き、図 25 に示すように、「新規テンプレート」ダイアログ・ボックスが表示されます。

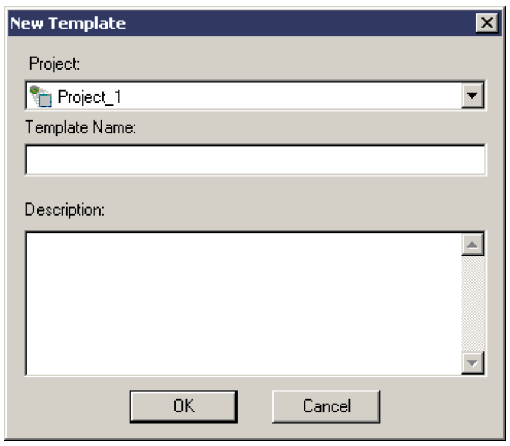



*図 25. 「新規テンプレート」ダイアログ・ボックス*

2. 「プロジェクト」フィールドで、そのテンプレートが含まれている統合コンポーネント・ライブラリー・ユーザー・プロジェクトの名前をドロップダウン・リストから選択します。
3. 「テンプレート名」フィールドにテンプレートの名前を入力します。テンプレート名には、英字、数値、および下線を使用できます。テンプレート名は、Process Designer Express がこれを基にソース・ファイル (.java) および Java クラス・ファイル (.class) を作成するため、以下の Java クラスの命名規則に従うことをお勧めします。
   - コラボレーション・テンプレート名の頭文字は大文字にしてください。
   - 名前に複数の単語が含まれる場合は、各単語の頭文字は大文字にしてください (例: CustomerSync)。
   - 名前では単語の間にスペースを使用しないでください。

詳細については、「*IBM WebSphere InterChange Server コンポーネント命名ガイド*」を参照してください。

4. 必要に応じてテンプレートの要旨を「説明」フィールドに入れてください。

   **注:** テンプレートの説明には、強制改行 (復帰) を組み込まないでください。

5. 「OK」をクリックします。Process Designer Express が開いて、「テンプレート・ツリー」ペインに新しいテンプレートとそのトップレベル・ツリーが表示されます。

## テンプレート・プロパティー情報の提供

コラボレーションの「テンプレート定義」ウィンドウには、表 24 にリストしているコラボレーション・テンプレートのプロパティーの定義に使用する 4 つのタブがあります。

*表 24. 「定義」ウィンドウのタブ*

| 「テンプレート定義」タブ | 説明 | 詳細 |
|---|---|---|
| その他 | コラボレーション・テンプレートに関する以下の情報を定義できます。<br>• テンプレートの説明<br>• 長期存続ビジネス・プロセス (LLBP) のサポート<br>• 最小トランザクション・レベル<br>• パッケージ情報 | 81 ページの『一般プロパティー情報の定義 (「一般」タブ)』 |
| 宣言 | テンプレート変数を定義したり、システム生成テンプレート変数を表示したりできます。 | 85 ページの『テンプレート変数の宣言と編集 (「宣言」タブ)』 |
| プロパティー | ユーザー定義のコラボレーション・テンプレート・プロパティーの名前、型、および値を指定できます。 | 91 ページの『コラボレーション構成プロパティーの定義 (「プロパティー」タブ)』 |
| ポートおよびトリガー・イベント | コラボレーション・テンプレートのポートおよびトリガー・イベントを定義できます。 | 94 ページの『ポートおよびトリガー・イベントの定義 (「ポートおよびトリガー・イベント」タブ)』 |

## 一般プロパティー情報の定義 (「一般」タブ)

「定義」ウィンドウの「一般」タブ (図 26 を参照) には、コラボレーション・テンプレートに関する一般プロパティー情報 (表 25 に記載されている情報を含む) が表示されます。

*表 25. 一般テンプレート定義情報*

| 一般テンプレート・プロパティー | 説明 | 詳細 |
|---|---|---|
| コラボレーション・テンプレートの説明 | コラボレーション・テンプレートのこのフィールドを使用するかどうかは任意です。このフィールドには、コラボレーション・テンプレートのすべてのユーザーが使用可能なテキストを入力できます。 | なし |

*表 25. 一般テンプレート定義情報 (続き)*

| 一般テンプレート・プロパティー | 説明 | 詳細 |
|---|---|---|
| 長期存続ビジネス・プロセスのサポート | テンプレートが長期存続ビジネス・プロセスをサポートするかどうかについて指定します。 | 82 ページの『長期存続ビジネス・プロセスのサポートの追加』 |
| トランザクション・レベル (トランザクション・コラボレーションのみ) | コラボレーションのすべての操作に対して最小トランザクション・レベルを設定します。 | 83 ページの『最小トランザクション・レベルの指定』 |
| コラボレーション・パッケージ | コラボレーションが格納されている Java パッケージ。 | 84 ページの『コラボレーション・パッケージの指定』 |

図 26 に、「定義」ウィンドウ内の「一般」タブを示します。

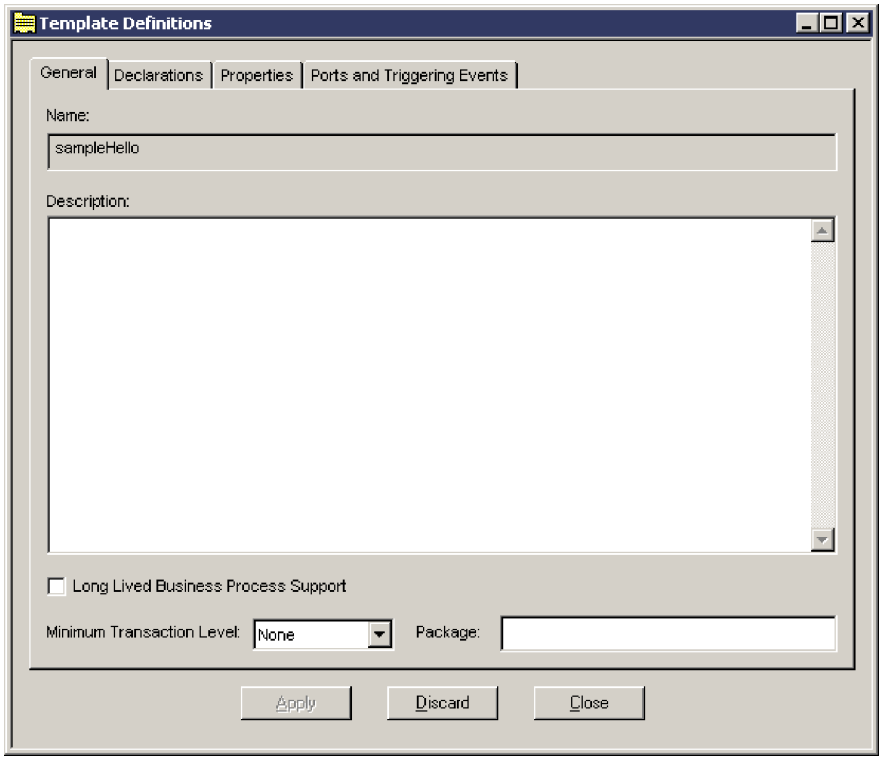



*図 26. 一般コラボレーション・プロパティー*

## 長期存続ビジネス・プロセスのサポートの追加

長期存続ビジネス・プロセスのサポートを使用すれば、コラボレーションを長期存続ビジネス・プロセスとして配置し、この環境でのサービス呼び出しのタイムアウト値を指定できます。この機能を使用するには、次の手順を実行する必要があります。

- 「一般」タブの「長期存続ビジネス・プロセスのサポート」オプションを選択します。
- 必要に応じて、長期存続ビジネス・プロセス処理で使用されるサービス呼び出しのタイムアウト値を示すユーザー定義のコラボレーション・テンプレート・プロ

パティーを作成します。詳細については、 91 ページの『コラボレーション構成プロパティーの定義 (「プロパティー」タブ)』を参照してください。

## 最小トランザクション・レベルの指定

コラボレーションがトランザクションである場合、InterChange Server Express は、トランザクションが失敗するとコラボレーションをロールバックします。ロールバックにより、テンプレート定義の差し戻しが実行され、コラボレーションのデータの変更が元に戻されます。トランザクション・コラボレーションについては、146 ページの『トランザクション機能の使用』を参照してください。

トランザクション・レベルにより、コラボレーションのシナリオを実行するためのメカニズムが決定します。コラボレーション・オブジェクトは、表 26 に記載されているトランザクション・レベルの 1 つで実行されます。

*表 26. トランザクション・レベル*

| トランザクション・レベル | 影響 | システムの振る舞い |
|---|---|---|
| なし | コラボレーションはトランザクションではありません。 | コラボレーションの実行時にエラーが発生すると、システムはこれをログに送信してから実行を終了します。 |
| 最小限の努力 | コラボレーションはトランザクションです。コラボレーションのシナリオのサブトランザクションに差し戻しが定義されています。 | このシナリオの実行時にエラーが発生すると、InterChange Server Express は、各サブトランザクション・ステップの差し戻しを実行し、シナリオをロールバックします。 |
| 最善的 | 最小限の努力と同じ処理を行います。差し戻し以外に、データ分離を使用して正確さを確保します。 | InterChange Server Express は、データの値が前回の使用以降に変更されているかどうかをチェックします。この方法で、トランザクション・コラボレーションでの使用期間にデータが実際に分離されているかどうかをチェックします。最善的分離チェックの実行時、データがその他のアプリケーション・トランザクションによる変更に対してぜい弱である場合、分離チェックの際小さい時間のウィンドウが表示されます。 |
| 厳重 | 最善的で行われる処理をすべて行います。ただし、ぜい弱点に関するデータ分離ウィンドウは表示されません。 | 分離のチェック時にアプリケーションによってデータがロックされます。API がアトミック「テストおよび設定」操作をサポートしているアプリケーションによってサポートされています。 |

コラボレーション・テンプレートの開発者は、テンプレートから作成したコラボレーション・オブジェクトに最小トランザクション・レベルを設定します。例えば、コラボレーションが重要なデータを処理する場合、処理に失敗するとコラボレーションが必ずロールバックされるようにするには、最小トランザクション・レベルを「最小限の努力」に設定します。トランザクションを実行するコラボレーションを設計するときに、トランザクション機能を使用しなくてもコラボレーションを正常に使用できるようにするには、最小トランザクション・レベルを「なし」に設定できます。

コラボレーション・オブジェクトのコネクターがより高いトランザクション・レベルをサポートしている場合、管理者は、コラボレーション・オブジェクトのトランザクション・レベルを上げることができます。ただし、コラボレーション・オブジェクトのトランザクション・レベルは、テンプレートに指定されている最小レベルより低いレベルにすることはできません。

**ヒント**

コラボレーション・テンプレートの最小トランザクション・レベルを「なし」に設定しているときに、差し戻しを作成してトランザクションを操作できるようにすることが可能です。より厳密なレベルが必要で、使用中のコネクターがより高いトランザクション・レベルをサポートできる場合、管理者は、バインド時にコラボレーション・オブジェクトのトランザクション・レベルを上げることができます。差し戻しの詳細については、131 ページの『差し戻しの定義』を参照してください。

コラボレーション・テンプレートに最小トランザクション・レベルを割り当てる手順は、以下のとおりです。

1. 「テンプレート定義」ウィンドウが開いており、「一般」タブが表示されているかどうかチェックします。
2. 「最小トランザクション・レベル」プルダウン・メニューを用いて、使用する最小トランザクション・レベルを選択します。トランザクションではないコラボレーション・テンプレートを編集している場合は、デフォルト値を「なし」のままにします。
3. 「適用」をクリックして変更内容を保管します。

## コラボレーション・パッケージの指定

パッケージは、関連する機能があるコラボレーションのグループです。Process Designer Express がアクセスするすべてのコラボレーションは、UserCollaborations パッケージまたは UserCollaborations のサブパッケージに含まれます。このため、UserCollaborations パッケージには以下が含まれます。

- WebSphere Business Integration Server Express ソフトウェアに付属のコラボレーション
- その他のコラボレーション開発者が作成したコラボレーション

UserCollaborations の下にサブパッケージを作成して、カスタム・コラボレーション・テンプレートをグループ化できます。例えば、オフィス用品を扱う複数のコラボレーション・テンプレートを作成する場合、OfficeSupplyMgmt と呼ばれるサブパッケージを作成することが可能です。その中に、PaperClipMgmt コラボレーションおよび PencilInventory コラボレーションを格納できます。

コラボレーション・テンプレートがパッケージに含まれていることを指定すると、Process Designer Express はパッケージ名を使用して、統合コンポーネント・ライブラリー・プロジェクトの Template¥Classes ディレクトリー内にサブディレクトリーを作成します。 (配置時にはパッケージ情報を格納するディレクトリーとして *ProductDir*¥collaborations¥classes¥UserCollaborations ディレクトリーが作成されます。)

製品のインストール時に、クラスパス内の UserCollaborations の下にすべてのコラボレーションを含めるように CLASSPATH 環境変数が設定されています。 Process Designer Express は、コラボレーション・テンプレートの .class および .java ファイルを、サブディレクトリー内に格納します。

コラボレーション・テンプレートを格納するパッケージを指定する手順は、以下のとおりです。

1. 「テンプレート定義」ウィンドウが開いており、「一般」タブが表示されているかどうかチェックします。
2. 「パッケージ」フィールドに、コラボレーション・テンプレートを格納するパッケージの名前を入力します。

   既存のパッケージに名前を指定すると、Process Designer Express によりコラボレーション・テンプレートがパッケージに追加されます。存在しないパッケージの名前を指定すると、Process Designer Express によってパッケージが作成されます。
3. 「適用」をクリックして変更内容を保管します。

既存のコラボレーション・テンプレート定義を変更して、いつでもパッケージ名を追加または変更することができます。

## テンプレート変数の宣言と編集 (「宣言」タブ)

「テンプレート定義」ウィンドウの「宣言」タブには、コラボレーション・テンプレートのテンプレート変数に関する情報が表示されます。テンプレート変数 は、コラボレーション内のすべてのシナリオを対象範囲としたコラボレーション変数です。つまり、テンプレート変数は、コラボレーション内のすべてのシナリオに対してグローバルな変数です。 (シナリオ変数は、Java プログラム言語のクラス変数に相当します。) 例えば、顧客トランザクションを含むコラボレーションには、すべてのシナリオについて顧客を識別する customerID テンプレート変数があります。テンプレート変数は、開発中、いつでも作成できます。

図 27 に、「テンプレート定義」ウィンドウ内の「宣言」タブを示します。

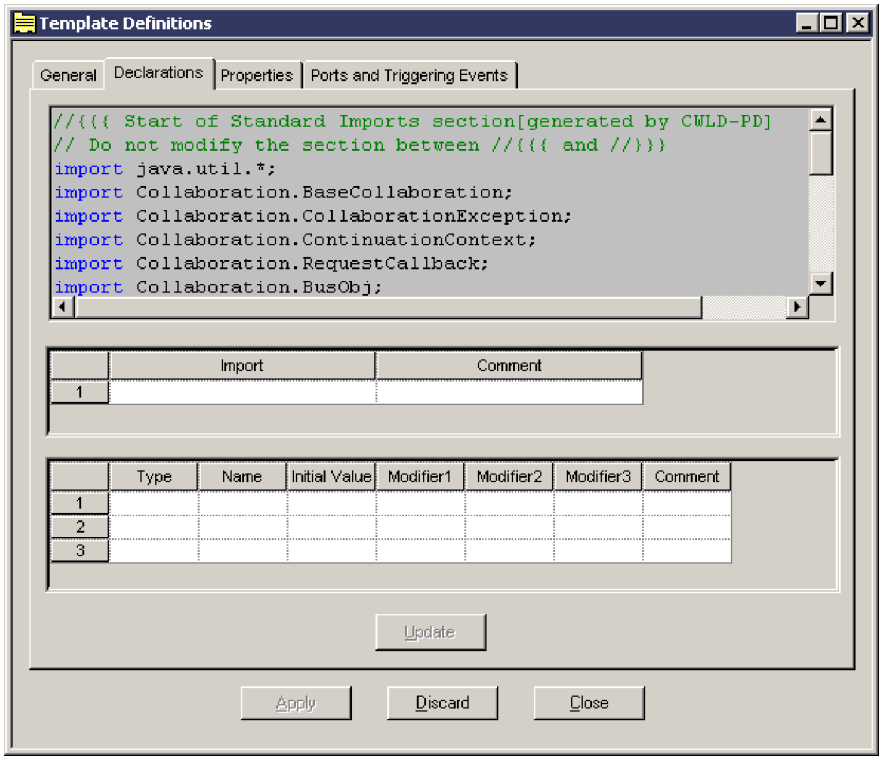



*図 27. 「テンプレート定義」ウィンドウの「宣言」タブ*

「宣言」タブでは、以下の操作を行うことができます。

- import ステートメントのコードを指定します。
- ユーザー定義のテンプレート変数の宣言テンプレートを入力します。
- システム生成テンプレート変数 (編集できません) を表示します。

## Java パッケージのインポート

「宣言」タブを使用して、特定の Java クラスをコラボレーションにインポートできます。 Java クラスは、その他のクラスのパッケージをインポートして、それらの機能にアクセスします。例えば、クラスが、java.math、java.security、および java.text パッケージをインポートして、それぞれの演算機能、セキュリティー機能、および国際化対応機能を使用することがあります。コラボレーション・テンプレートはクラスであるため、Java Development Kit またはサード・パーティー製品から提供されるクラスまたはクラスのグループ (パッケージと呼ぶ) を使用できます。

デフォルトでは、すべての Java クラスが、パッケージ java.lang 内のクラスを暗黙的にインポートします。また、Process Designer Express も、すべてのコラボレーション・テンプレートで使用するためにパッケージ java.util 内のクラスを暗黙的にインポートします。

以下の import ステートメントは、java.math クラスを JDK からインポートします。 (アスタリスクは、特定のパッケージ内のすべてのクラスをインポートすることを示します。)

```
java.math.*;
```

また、以下のステートメントにより、パッケージの BigDecimal クラスのみをインポートします。

```
java.math.BigDecimal;
```

import ステートメントは、コラボレーションの開発中はいつでもコードに追加できます。

Java クラスをインポートする手順は、以下のとおりです。

1. 「テンプレート定義」ウィンドウが開いており、「宣言」タブが表示されているかどうかチェックします。
2. インポート・テーブルの左側の見出しセルにカーソルを置きます。図 28 に示すように、右マウス・ボタンでクリックして、「追加」を選択します。テーブルに新しい行が追加されます。

   **注:** テーブルの最後の行をクリックして、新しい行を追加することもできます。

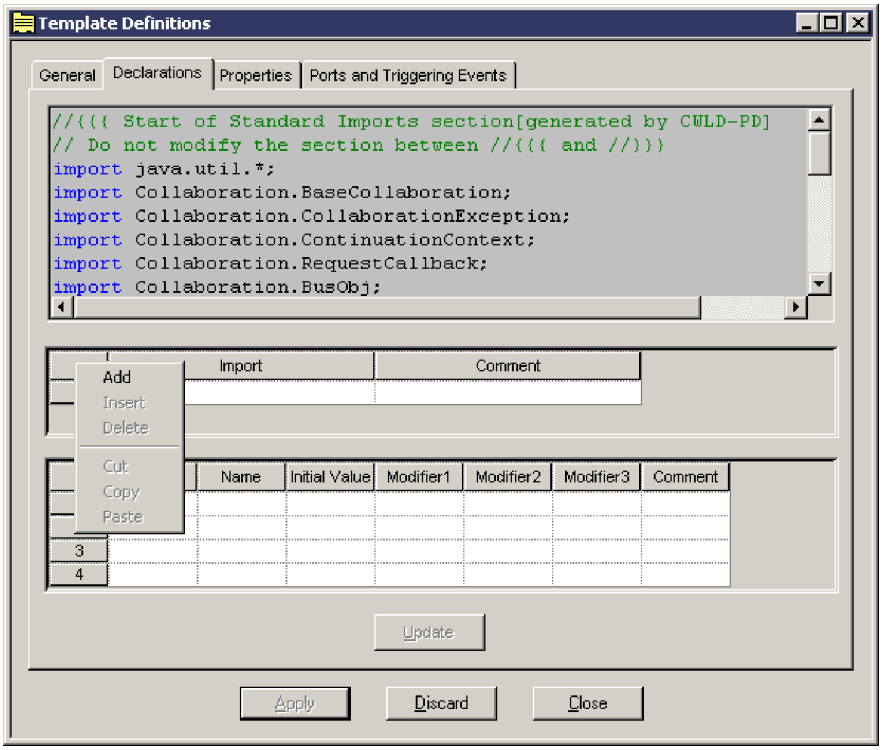



*図 28. import ステートメントの追加*

3. 「インポート」列内に任意の import ステートメントを入力します。以下に例を示します。

   ```
   java.math.*
   ```

4. 必要に応じて、「コメント」列に import ステートメントの要旨を入力します。
5. 「適用」をクリックして変更内容を保管します。
6. import ステートメントを追加するには、ステップ 2 からステップ 5 を繰り返します。コラボレーション・テンプレート内に格納できる import ステートメントの数は無限です。

インポートしたクラスが JDK ではなくサード・パーティー・パッケージのものである場合、*ProductDir*¥bin¥cwtools.cfg ファイルの [codeGeneration] セクションを編集し、テンプレートのコンパイル前のパッケージ・パスを反映する必要があります。

サード・パーティー・パッケージからインポートされたクラスを使用するコラボレーションを配置する前に、コラボレーションが配置されるシステムの JCLASSES 変数を更新する必要があります。インポートしたクラスが JDK ではなくサード・パーティーのものである場合、このクラスを、JCLASSES 変数のインポート済みクラスのパスに追加する必要があります。IBM では、なんらかの機構を使用して、JCLASSES に含まれるこれらの標準のクラスをカスタム・クラスと区別することを推奨します。例えば、以下のように、このようなカスタム・クラスを保持する新しい変数を作成し、この新しい変数を JCLASSES に追加します。

1. CwMacroUtils.jar ファイルを、専用ディレクトリーの中に格納します。例えば、製品ディレクトリーの下に ¥dependencies ディレクトリーを作成して、その中に .jar ファイルを格納します。
2. ICS を始動するときに使用されるファイル (デフォルトでは *ProductDir*¥bin¥start_server.bat または *ProductDir*/bin/CWSharedEnv.sh) を編集し、CWMacroUtils.jar ファイルの新しいパスを記述します。以下のエントリーをファイルに追加します。

   ```
   set DEPENDENCIES=ProductDir/dependencies/CwMacroUtils.jar
   ```

   ここで、*ProductDir* は、Business Integration Express がインストールされているロケーションです。
3. ご使用のオペレーティング・システムに応じた新しい DEPENDENCIES 変数を、次のように JCLASSES エントリーに追加します。

   UNIX システムでは次の構文を使用します。*ExistingJarFiles* は、JCLASSES に指定されている .jar ファイルです。

   ```
   set JCLASSES = $JCLASSES:ExistingJarFiles:$DEPENDENCIES
   ```

   Windows システムでは次の構文を使用します。*ExistingJarFiles* は、JCLASSES に指定されている .jar ファイルです。

   ```
   set JCLASSES = ExistingJarFiles;%DEPENDENCIES%
   ```
4. カスタム・クラスを使用しているコラボレーションごとに、CwMacroUtils.jar ファイルで指定されている *PackageName.ClassName* を含めます。
5. ICS を再始動し、コラボレーションに対してメソッドを有効にします。

カスタム・クラスのインポート時には、Business Integration Express ソフトウェアがそのカスタム・クラスを検出できなかったことを示すエラー・メッセージが表示されることがあります。その場合は、以下をチェックしてください。

- カスタム・クラスがパッケージの一部であるかどうかチェックします。プログラミングの慣例上、カスタム・クラスはパッケージに格納するのが理想的です。カスタム・クラス・コードに正しいパッケージ・ステートメントが含まれており、このステートメントがソース・ファイルの先頭 (任意のクラスまたはインターフェース宣言より前) にあることを確認します。
- コラボレーション・テンプレートの import ステートメントが正しいかどうか検証します。import ステートメントは正しいパッケージ名を参照している必要があり

ます。import ステートメントは、さらにカスタム・クラスの名前を指定したり、パッケージ内のすべてのクラスを参照できます。例えば、パッケージ名が COM.acme.graphics で、カスタム・クラスが Rectangle の場合、パッケージ全体をインポートできます。

```
COM.acme.graphics.*;
```

また、Rectangle カスタム・クラスのみをインポートすることもできます。

```
COM.acme.graphics.Rectangle;
```

- カスタム・クラスが格納されたパッケージ、またはカスタム・クラス自体 (パッケージに格納されていない場合) へのパスを含むよう CLASSPATH 環境変数が更新されていることを確認します。

例えば、カスタム・クラスのインポート時に、*ProductDir*¥lib¥com¥crossworlds¥*package* という名前のフォルダーを作成できます。ここで、*ProductDir* は Business Integration Express がインストールされているロケーションを示し、*package* はご使用のパッケージの名前を示します。次に、作成したフォルダーにカスタム・クラス・ファイルを格納します。最後に、start_server.bat ファイルの中の CLASSPATH 変数の中に、*ProductDir*¥lib というパスを含めます。

## テンプレート変数の宣言

「宣言」タブを使用して、コラボレーションが使用するユーザー独自のテンプレート変数を宣言することもできます。

変数を使用するには、タイプおよび名前を指定して、最初に変数を宣言する必要があります。コラボレーション・テンプレートの変数のデータ型は以下のいずれかです。

- 基本データ型 (byte、short、int、long、float、double、char、boolean など)
- Java クラス (String、Integer など)
- BusObj、BusObjArray、または CollaborationException などの Business Integration Expres 定義クラス
- ユーザーが定義するクラス (上級ユーザーである場合)

**注:** LongText および Date は、ビジネス・オブジェクト属性における特殊用途ストリング用の製品固有の指定です。 LongText または Date データ型のビジネス・オブジェクト属性に変数を指定するには、コードに String データ型を使用します。

テンプレート変数を宣言する手順は、以下のとおりです。

1. 「テンプレート定義」ウィンドウが開いており、「宣言」タブが表示されているかどうかチェックします。
2. 変数テーブルの左側の見出しセル内にカーソルを置きます。右マウス・ボタンでクリックして、「追加」を選択します。テーブルに新しい行が追加されます。

   **注:** テーブルの最後の行をクリックして、新しい行を追加することもできます。
3. 宣言する変数のタイプを指定するには、「タイプ」列のドロップダウン・メニューを使用します。
4. 「名前」列で変数の名前を指定します。

5. 「初期値」列で変数の初期値を指定します。

   **注:** ストリング値は、引用符で囲んで入力する必要があります (例えば、ストリング `Yes` は `"Yes"` のように入力します)。

6. 変数 (public、private、protected など) に適用させる修飾子を、「Modifier1」、「Modifier2」、および「Modifier3」列で指定します。すべての 3 つの列において修飾子を指定する必要はありません。

   **注:** テンプレート変数を定義するときには修飾子 `Static` を使用しないでください。

7. タブの上部にある宣言のリストに新しい変数を追加するには、「更新」をクリックしてから、「適用」をクリックして変更内容を保管します。

コラボレーションに対する複数回の呼び出しについて、その値が永続的となる変数を宣言できます。コラボレーション内でアクションのカウンター・データを記録し、このコラボレーションを実行するたびにこのカウンターを増分するとします。ctr という名前の public の integer 型変数を作成するには、「宣言」タブの中の変数テーブルを使用します。

次に、コラボレーション・コード自体の中で、カウンターを増分します。

```
ctr = ctr+1;
```

ctr 変数は、コラボレーションが実行されるごとに増えます。

***長期存続ビジネス・プロセスとともに使用するテンプレート変数に関する特殊考慮事項:*** コラボレーションを長期存続ビジネス・プロセスとして配置する場合には、永続させる変数をすべてグローバル・テンプレート変数またはグローバル・ポート変数として定義してください。

また、これらの変数のデータ型が次のいずれかでなければなりません。

- Java 直列化可能データ型。これには、byte、short、int、long、float、double、char、boolean、string、Integer、および Java Serializable インターフェースまたは Java Externalizable インターフェースを実装するユーザー定義のデータ型があります。
- Business Integration Express BusObj データ型
- Business Integration Express BusObjArray データ型

その他のデータ型の変数は、長期存続ビジネス・プロセスでは永続されません。

## システム生成変数

Process Designer Express は、以下のコラボレーション変数を自動的に宣言します。

- すべてのコラボレーションで使用可能な 2 つのコラボレーション変数 triggeringBusObj と currentException
- ポートごとに 1 つの変数

表 27 に、これらのシステム生成変数についての説明をリストします。

*表 27. システム生成変数*

| 変数 | 説明 |
|---|---|
| triggeringBusObj | triggeringBusObj 変数には、シナリオのフロー・トリガー (トリガー・イベントまたはトリガー・アクセス呼び出し) が含まれます。フロー・トリガーは、ビジネス・オブジェクトおよび動詞です。トリガー・イベントは、アプリケーション・イベントおよびそのデータを示します。フロー・トリガーが到着すると、シナリオの実行が開始されます。この変数はテンプレート変数です。つまり、その有効範囲はコラボレーション全体です。 |
| currentException | currentException 変数には、直前のアクション、サブアクティビティー、またはイテレーターによって発生した例外オブジェクトが含まれます。Process Designer Express は、currentException を暗黙的に宣言します。currentException のスコープは、例外の発生直後のアクションです。シナリオは、例外を生成したアクティビティーの直後の遷移リンクまたはコード・フラグメントにある currentException の値をチェックする必要があります。 |
| ポート変数 | Process Designer Express は、コラボレーションの各ポートに関連付けられているビジネス・オブジェクトのテンプレート変数を宣言します。このように生成された宣言は、「テンプレート定義」ウィンドウの「宣言」タブに表示されます。各ポート変数の名前は、`BusObj` が付加されたポートの名前です。例えば、ポート名が SourceInvoice の場合、変数名は SourceInvoiceBusObj になります。また、宣言により、ポートが定義されているものと同じ型の BusObj のインスタンスも生成されます。最初に、ビジネス・オブジェクトの属性が null に設定されます。これらのポート変数を使用して、トリガー・イベントを処理できます。詳細については、217 ページの『トリガー・イベントのコピー』を参照してください。 |

## コラボレーション構成プロパティーの定義 (「プロパティー」タブ)

コラボレーション・テンプレートには、2 つのタイプの構成プロパティーがあります。

- *標準プロパティー* には、すべてのコラボレーションに必要な情報 (トレース・レベルやメッセージ通知の E メール・アドレスなど) が含まれます。すべてのコラボレーションには、InterChange Server Express によって定義される同一の標準構成プロパティーがあります。
- *コラボレーション固有のプロパティー* はオプションです。これらは、コラボレーションの開発者によって定義されています。コラボレーションは、プロパティーの値を使用してその振る舞いの性質を決定します。プロパティーには、以下のようなタイプがあります。
  - Date
  - Double
  - Float
  - Integer
  - Boolean

- String
- Time
- URL

管理者は、コラボレーションを構成するときにこの 2 つのタイプのプロパティーを扱います。

コラボレーションの開発者は、コラボレーションにコラボレーション固有プロパティーが必要かどうかを決定します。必要な場合は、その名前およびデフォルト値を定義します。これらの構成プロパティーにより、コラボレーション・ユーザーは、コラボレーションの振る舞いを制御するデータを指定できます。

表 28 は、作成可能なプロパティーのタイプの例です。

*表 28. コラボレーション固有の構成プロパティーの例*

| プロパティーのタイプ | 例 |
|---|---|
| コラボレーションが属性の値を設定するために使用する値 | コラボレーションは、顧客に対する送り状の生成をアプリケーションに要求することがあります。コラボレーションは、Invoice ビジネス・オブジェクトの Rate 属性の値を設定することがあります。コラボレーションに BILLING_RATE と呼ばれるプロパティーがある場合、管理者は、現在の取引慣行に基づいて Rate 属性の値を上げたり下げたりすることができます。 |
| コラボレーションが特定の実行経路を取り込むかどうかについて判別するための true または false の値 | アプリケーション全体でエンティティーの変更を同期する Business Integration Express コラボレーションには、通常、CONVERT_CREATE と呼ばれるプロパティーがあります。コラボレーションは、Update イベントを受信すると、更新するエンティティーの宛先アプリケーションをチェックします。このエンティティーが存在しない場合、コラボレーションは、CONVERT_CREATE プロパティーの値をチェックします。プロパティーが true に設定されている場合は、コラボレーションは、Update 要求を Create 要求に変換します。 |

必要に応じてコラボレーション固有の構成プロパティーを使用できます。テンプレートで使用できるこの構成プロパティーの数に制限はありません。このプロパティーは開発中、いつでも追加できます。コラボレーションに必要なプロパティーが最初にわかっている場合は、シナリオをモデル化する前にプロパティーを作成できます。ただし、シナリオのモデルを作成中の場合は、追加のプロパティーを定義して、コラボレーションのロジックをサポートできます。

コラボレーション・テンプレートに対してコラボレーション固有の構成プロパティーを作成する手順は、以下のとおりです。

1. 「テンプレート定義」ウィンドウが開いており、「プロパティー」タブが表示されているかどうかチェックします。

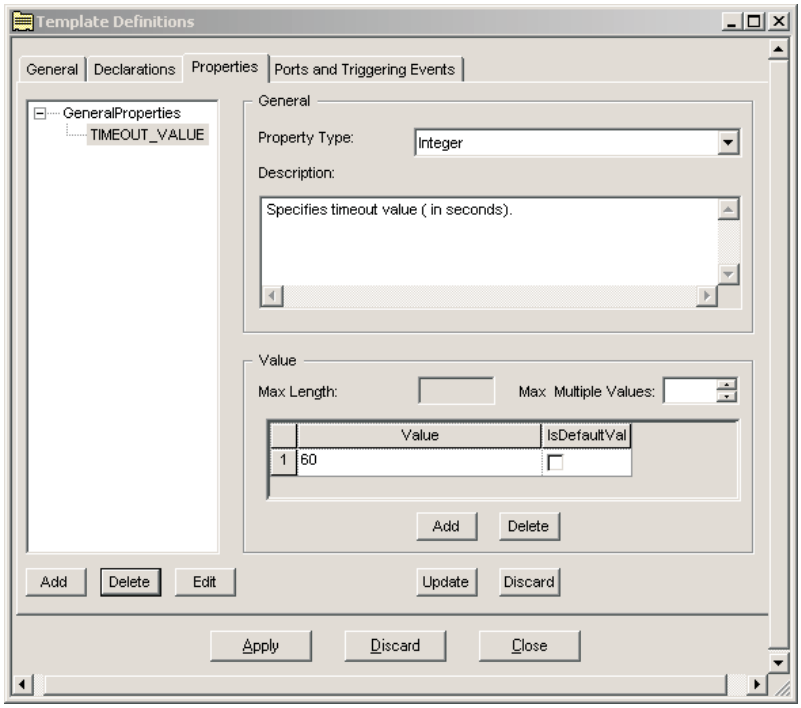



*図 29. コラボレーション固有のプロパティーの追加*

2. 「追加」ボタンをクリックし、構成プロパティーを作成します。「名前」ダイアログ・ボックスが開きます。
3. 「名前」フィールドにプロパティーの名前を入力して、「OK」をクリックします。

   **注:** 命名規則では、構成プロパティー名は大文字で書き、複数の単語は下線で区切ります。 IBM では、各プロパティー名を見れば管理者がその内容が理解できるようにする必要があるため、構成プロパティー名がプロパティーの機能を明確に表すようにしてください。
4. 「プロパティー・タイプ」ドロップダウン・メニューでプロパティーのタイプを選択します。
5. 必要に応じて、「説明」フィールドに新規プロパティーに関する説明を入力します。
6. プロパティー・タイプが string の場合は、「最大長」フィールドで値を指定します。
7. 必要に応じて「最大複数値」フィールドに、プロパティーに対して受け入れ可能な複数値の最大数を指定します。このフィールドで指定した数によって、プロパティーが持つことのできるデフォルト値の数も制限されます。例えば、最大複数値を 2 に設定すると、そのプロパティーに指定可能な値をいくつ関連付けても、プロパティーのデフォルト値で持つことができるのは 2 つのみです。このフィールドに値を指定しなければ、デフォルト値 1 が使用されます。

   **注:** コラボレーション固有のプロパティーの「最大複数値」属性は頻繁には使用されません。ほとんどのコラボレーション固有プロパティーでは、単一値だけが受け入れられます。

8. 「値」ペインの「追加」をクリックします。テーブルに新しい行が追加され、「プロパティー値」ダイアログ・ボックスが開きます。
9. 「値」フィールドに値を入力するか、または「範囲下限」フィールドで値の範囲を指定して、「OK」をクリックします。ダイアログ・ボックスが閉じて、「値列」列に情報が表示されます。
10. 値がデフォルト値の場合は、「IsDefaultValue」列内のチェック・ボックスをクリックします。
11. プロパティー定義に値を追加するごとに、ステップ 8 からステップ 10 を繰り返します。
12. 「更新」をクリックします。
13. 追加する構成プロパティーの数のみステップ 2 からステップ 12 を繰り返します。
14. 構成プロパティーの追加が終了したら、「適用」をクリックして変更内容を保管します。

コラボレーション固有の構成プロパティーを削除するには、タブの左側のペインにあるリストから削除するプロパティーの名前を選択して、「削除」をクリックします。

### 長期存続ビジネス・プロセスをサポートするプロパティーの追加

動的サービス呼び出しのタイムアウト値を使用して長期存続ビジネス・プロセス (LLBP) をサポートする場合は、Java 変数またはコラボレーション固有プロパティーを使用できます。コラボレーション固有プロパティーを使用する場合には、コラボレーション・テンプレート定義時にこのプロパティーを作成する必要があります。コラボレーション固有プロパティーを使用すると、タイムアウト値を実行時に設定できるため、サービス呼び出しの最初の作成時に提供される静的値を使用せずにすみます。動的タイムアウト値のプロパティーを作成するときには、integer データ型を使用してください。

例えば、create 要求によってビジネス・オブジェクトが送信される To ポートからサービス呼び出しを受け取る場合は、CreateTimeout と呼ばれるコラボレーション・プロパティーを定義します。このサービス呼び出しを定義するときに、CreateTimeout プロパティーを使用して、サービス呼び出しがタイムアウトになるポイントを指定します。サービス呼び出しの作成については、125 ページの『サービス呼び出し』を参照してください。

サービス呼び出しの作成および定義時に指定された固定タイムアウト値も使用できます。この場合、コラボレーション・プロパティーは不要です。129 ページの『サービス呼び出しタイプの定義』を参照してください。

## ポートおよびトリガー・イベントの定義 (「ポートおよびトリガー・イベント」タブ)

「テンプレート定義」ウィンドウの「ポートおよびトリガー・イベント」タブには、以下に関する情報が表示されます。

- コラボレーションのポート

コラボレーション・テンプレートでは、ポート は、コラボレーション・オブジェクトが実行時に受信または生成するビジネス・オブジェクトを表す変数です。

- コラボレーションのトリガー・イベント

  ビジネス・オブジェクトは、トリガー・イベントまたはアクションを示します。通常、コラボレーションは、コネクターからビジネス・オブジェクトを受信すると、アクションによって応答します。これらの受信ビジネス・オブジェクトは、トリガーまたはトリガー・イベントと呼ばれます。

  **注:** コラボレーションが受信するビジネス・オブジェクトを示す一般的な用語は、フロー・トリガーです。コラボレーションがコネクターからビジネス・オブジェクトを受信した場合、このフロー・トリガーはトリガー・イベントです。コラボレーションがアクセス・クライアントからビジネス・オブジェクトを受信した場合、このビジネス・オブジェクトはトリガー・アクセス呼び出しと呼ばれます。ビジネス・オブジェクトがトリガー・アクセス呼び出しに関連付けられているポートは、トリガー・イベントに関連付けられているポートと同じ方法で定義されます。

  「ポートおよびトリガー・イベント」タブの使用に関する詳細は、 98 ページの『シナリオへのトリガー・イベントの割り当て』を参照してください。

  通常、コラボレーションは、操作を完了すると、アクションを開始したコネクターにビジネス・オブジェクトを送信します。したがって InterChange Server Express は、イベントのトリガーまたは送信のために、ポートを頻繁に参照します。

**注:** Business Object Designer または System Manager を使用してリポジトリーとの間でオブジェクトを追加、変更、または削除すると、InterChange Server Express は、Process Designer Express に表示されるビジネス・オブジェクト定義のリストを動的に更新します。「テンプレート定義」ウィンドウの「ポートおよびトリガー・イベント」テーブルのビジネス・オブジェクト・フィールドに表示される動的変更結果を確認する場合は、InterChange Server Express または Process Designer Express を再始動する必要はありません。

**要確認**

IBM では、この動的更新機能は開発環境でのみ使用することを推奨します。ビジネス・オブジェクトの更新により、複雑な問題が発生する可能性があります。動的更新により、システム内のその他の機能が影響を受ける可能性があります。例えば、古いビジネス・オブジェクト定義を使用するイベントの処理方法や、初めに古いビジネス・オブジェクト定義に対してサブミットされた未解決のフローの再サブミット方法などが影響を受けます。このようなシナリオあるいはその他のシナリオにより、処理中のビジネス・オブジェクト定義とメモリー内のビジネス・オブジェクト定義の間でミスマッチが発生することがあります。このため、実動システムの場合、IBM では、システム上でイベントが処理されていない場合にのみビジネス・オブジェクト定義を更新することを推奨します。

コラボレーション・ポートの詳細については、「システム・インプリメンテーション・ガイド」を参照してください。

## ポートの作成

ポートを作成する手順は、以下のとおりです。

1. 「テンプレート定義」ウィンドウが開いており、「ポートおよびトリガー・イベント」タブが表示されているかどうかチェックします。

   ウィンドウ上部にあるテーブルには、ポート名、ビジネス・オブジェクト・タイプ、および動詞が表示されます。このコラボレーション・テンプレートのポートをまだ作成していない場合、このテーブルは空です。
2. 「ポートを追加」をクリックし、新しいポートを「ポート」テーブルに追加します。
3. テーブルの「ポート」列にポート名を入力します。

   ポート名の定義方法については、以下の指針に従ってください。
   - 名前の頭文字は英字にし、名前には英数字および下線シンボルのみを使用してください。
   - ポート名では大文字小文字は区別されませんが、ポートを示すときは常に、定義したとおりに大文字小文字を使用する必要があります。
   - 通常、ポート名には、ポートの目的がわかりやすい名前を割り当てると役立ちます。ポート名は開発プロセス全体にわたって使用されます。また、ポート名がわかりやすければ、作業が簡単になります。
   - 多くの場合、コラボレーションの開発者は、ビジネス・オブジェクト・タイプとその役割指定 (「In」と「Out」、または「ソース」と「宛先」など) を組み合わせることにより、ポート名を作成します。例えば、ポート SourceCase と命名することにより、これがソース・アプリケーションのポートであり、Case ビジネス・オブジェクトに対して構成されていることを示すことができます。
4. 「BO タイプ」列内のドロップダウン・リストからポートのタイプを選択します。これは、このポートがサポートするビジネス・オブジェクト定義のタイプです。
5. 「適用」をクリックして変更内容を保管します。

**注:** コラボレーション・オブジェクトの一部のポートが不要な場合があります。この場合は、未使用のポート (1 つまたは複数) へのサービス呼び出しを実行しないように、コラボレーション・ロジックを構成する必要があります。

InterChange Server Express では、すべてのコラボレーション・ポートをバインドする必要があるので、未使用ポート (1 つまたは複数) も Port コネクターにバインドする必要があります。*Port コネクター* は、未使用のポートを閉じるために使用される、汎用コネクター定義です。Port コネクターは、正しいコラボレーション・ロジックとともに使用する必要があります。Port コネクターにバインドされたポートに送信されたサービス呼び出しは、コラボレーション・スレッドをブロックします。

### ポート名の変更

ポート名を変更するには、このポートを削除し、新しい名前を使用してポートを再作成する必要があります。単にその名前を変更することはできません。ポート名を変更する手順は、以下のとおりです。

1. 「ポートおよびトリガー・イベント」タブのテーブルでポートを選択します。
2. 「ポートを削除」をクリックします。
3. 96 ページの『ポートの作成』の説明に従って、新しい名前のポートを作成します。

表 29 に、ポートを削除して再作成したときの影響についての要約を示します。

*表 29. ポート名の変更の結果*

| Process Designer Express による処理 | ユーザーが行う必要がある処理 |
| --- | --- |
| 変更後のポート名を使用するシステム生成テンプレート変数が変更されます。 | 古いポート名で宣言された変数を使用するコードがある場合は、コードの変数名を変更します。古いポート名で宣言された変数が表示されるすべてのアクション・ノードおよびサービス呼び出しを検索します。コンパイラーにより、残りの誤った名前が検出されます。 |
| フロー・トリガー (トリガー・イベントまたはトリガー・アクセス呼び出し) の割り当てが削除されます。 | フロー・トリガーを再度割り当てます。102 ページの『コラボレーション・テンプレートのコンパイル』を参照してください。 |

## シナリオの定義

シナリオ は、特定の受信ビジネス・オブジェクトまたはビジネス・オブジェクトの集合を処理するコラボレーション・テンプレート・コードです。このビジネス・オブジェクトには、イベント (コネクターからのもの) とアクセス呼び出し (アクセス・クライアントからのもの) があります。シナリオは、コラボレーション・テンプレート・クラスのイベント処理メソッドであるとみなすことができます。アクティビティー・ダイアグラムには、イベントの処理方法を示すコードが含まれます。

### シナリオの説明

シナリオを使用して、コラボレーションが解決するビジネス上の問題を区分します。コラボレーションのすべてのロジックを 1 つのシナリオにグループ化したり、複数のシナリオを作成してその 1 つのシナリオが問題の 1 つの局面を扱うようにすることができます。すべてのコラボレーション・ロジックを 1 つのシナリオにグループ化するのは、すべてのロジックが main() 関数に含まれるプログラムに似ています。一方、複数のシナリオを使用するのは、機能を分割して構築されたプログラムに似ています。

通常、シナリオの名前は、実行する機能に応じて付けます。コラボレーションに複数のシナリオがあり、各シナリオが 1 つのタイプのビジネス・オブジェクトを処理する場合、処理するビジネス・オブジェクトに応じて各シナリオに名前を付けることができます。例えば、コラボレーションが複数の動詞を持つ 1 つのタイプのビジネス・オブジェクトを処理する場合、Create、Update および Delete シナリオを開発

できます。さまざまなタイプのビジネス・オブジェクトをコラボレーションが処理する場合は、各ビジネス・オブジェクト定義に対してシナリオを開発できます。

シナリオは、1 回の実行につき 1 つのトリガー・フロー (トリガー・イベントまたはトリガー・アクセス呼び出し) のみを処理します。ただし、同じシナリオが複数のトリガー・フローを処理することもあります。例えば、同じシナリオが Create、Update、または Delete フローを処理することがあります。

通常、同じロジックがさまざまなタイプのビジネス・オブジェクトを処理する場合、これらのビジネス・オブジェクトに対して 1 つのシナリオを使用する方が効率的です。これにより、複数のコード・フラグメントをテストおよびデバッグする必要がなくなります。

**注:** シナリオは、同じコラボレーション内の別のシナリオに制御を渡すことはできません。コラボレーション・ロジックの区分化に関する事前の計画で、1 つのシナリオが別のシナリオを呼び出す必要がある場合は、すべてのコラボレーション・ロジックを同じシナリオに入れます。このシナリオでは、設計は非常に柔軟になります。また、コラボレーション・グループを作成し、このグループ内のコラボレーション間でロジックを分割することもできます。

## シナリオの作成

新しいシナリオを作成する手順は、以下のとおりです。

1. Process Designer Express 内から「テンプレート」—>「新規シナリオ」をクリックします。「シナリオを作成」ダイアログ・ボックスが表示されます。
2. 「シナリオ名」フィールドにシナリオの名前を入力します。

   シナリオ名のストリングに使用できるのは、英数字と下線です。特定の動詞を持つイベントをシナリオが処理する場合は、この動詞をシナリオ名に入れると役に立ちます。
3. 必要に応じて、「説明」フィールドに説明を入力します。
4. 「OK」をクリックします。テンプレート・ツリー・ビューでは、新しいシナリオの名前がシナリオ・ツリーに表示されます。さらに、ダイアグラム・エディターがメインウィンドウで開きます。
5. 少なくとも 1 つのフロー・トリガーをシナリオに割り当てる必要があります。フロー・トリガーの割り当てに失敗すると、実行時エラーが発生する可能性があります。新しいシナリオ定義に対するトリガー・イベントの割り当てについては、『シナリオへのトリガー・イベントの割り当て』を参照してください。

## シナリオへのトリガー・イベントの割り当て

シナリオへのトリガー・イベントの割り当ては、「ポートおよびトリガー・イベント」タブの「ポートおよびトリガー・イベント」テーブルで行います。作成する各シナリオには、そのトリガー・イベントを割り当てる必要があります。トリガー・イベントは、ビジネス・オブジェクトおよび動詞によって示されます。

**注:** シナリオの受信ビジネス・オブジェクトおよび動詞を示す一般的な用語は、「フロー・トリガー」です。フロー・トリガーがコネクターから送信されたものである場合、このフロー・トリガーはトリガー・イベントと呼ばれます。フロー・トリガーがアクセス・クライアントから送信されたものである場合、こ

のフロー・トリガーはトリガー・アクセス・メソッドと呼ばれます。「ポートおよびトリガー・イベント」タブでは、フロー・トリガーがトリガー・イベントとトリガー・アクセス・メソッドのいずれであるかにかかわらず、フロー・トリガーをシナリオに割り当てることができます。このセクションでは、「トリガー・イベント」と「イベント」という用語を使用します。これは、コネクターから受信したフロー・トリガーの方がはるかに一般的であるためです。

コラボレーションのポート定義により、コラボレーションが送受信できるビジネス・オブジェクトのタイプを指定します。コラボレーションのポートおよびシナリオを定義したら、以下を定義する必要があります。

- トリガー・イベントが送受信されるポート

  「ポートおよびトリガー・イベント」タブでは、トリガー・イベントが入力されるポート名と、イベントを示すビジネス・オブジェクト名に相当するテーブル内の行を選択します。
- コラボレーションの実行を起動するオブジェクト

  フロー・トリガーは、ポート・ビジネス・オブジェクトと動詞 (*business-object.verb* の組み合わせ) によって示されます。フロー・トリガーを定義するポートおよびビジネス・オブジェクトの行で、その動詞を選択してフロー・トリガーを指定します。
- 各フロー・トリガーを処理するシナリオ

図 30 は、ポート From がビジネス・オブジェクト・タイプ Widget をサポートしているコラボレーション・テンプレートにおける関連状況を示します。Create シナリオはトリガー・イベント Widget.Create を処理し、Delete シナリオはトリガー・イベント Widget.Delete を処理します。

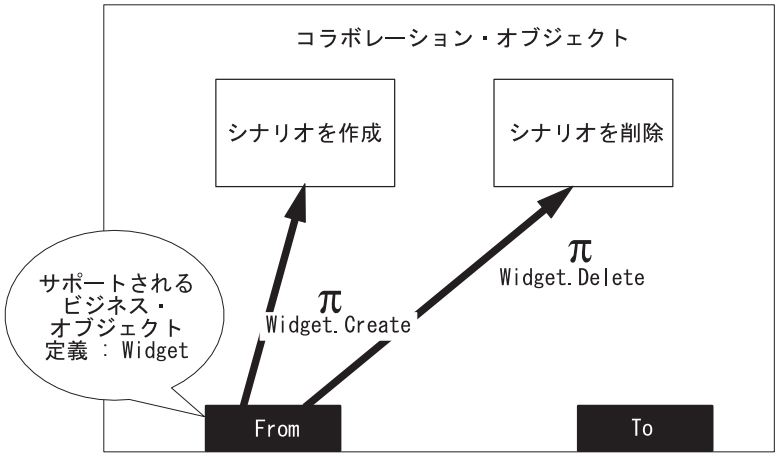



*図 30. ポート、トリガー・イベント、およびシナリオの関係*

各フロー・トリガーに対してシナリオを指定する手順は、以下のとおりです。

1. 「テンプレート定義」ウィンドウが開いており、「ポートおよびトリガー・イベント」タブが表示されているかどうかチェックします。

2. 「ポートおよびトリガー・イベント」テーブルで、フロー・トリガーを受信するポートと、フロー・トリガーを示すビジネス・オブジェクトを示す行を確認します。
3. この行で、「作成」列のドロップダウン・リストをクリックします。リストには、テンプレートに対して定義されたすべてのシナリオが含まれています。任意のシナリオを選択します。
4. フロー・トリガーを割り当てる各ポート、ビジネス・オブジェクト、および動詞に対して 2 および 3 を繰り返します。
5. トリガー・イベントの割り当てが完了したら、「適用」をクリックして割り当て内容を保管します。

## シナリオ変数の定義

シナリオの作成が完了すると、シナリオ固有の変数を「シナリオ定義」ダイアログ・ボックスに追加できるようになります (図 31 を参照)。

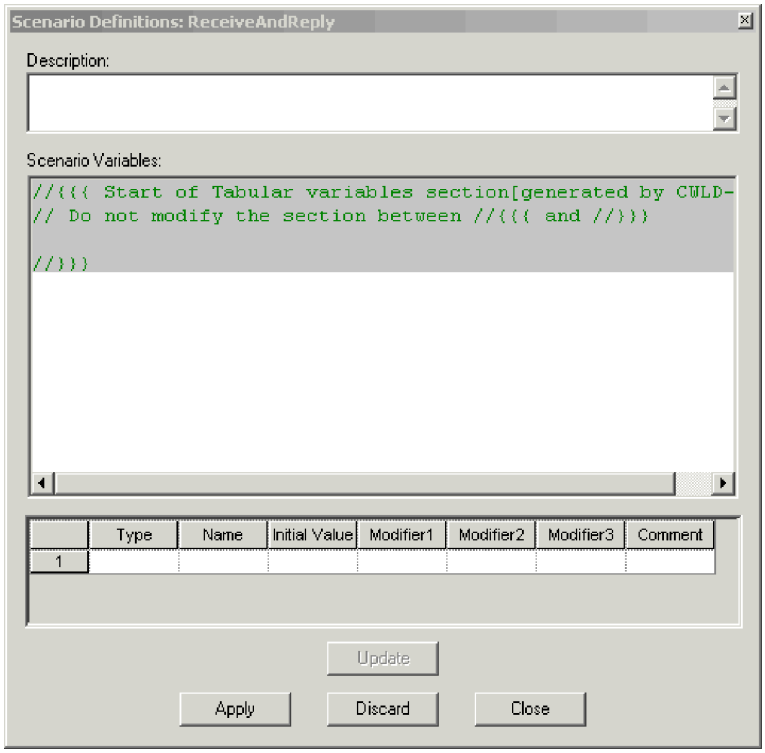



*図 31. 「シナリオ定義」ダイアログ・ボックス*

シナリオ変数 は、単一シナリオ内のすべてのアクションおよびリンクを範囲とするコラボレーション変数です。 (シナリオ変数は、Java プログラム言語のクラス変数に相当します。) シナリオ変数は、コラボレーション・テンプレート開発中にいつでも設定できます。

変数をシナリオ定義に追加する手順は、以下のとおりです。

1. 以下のいずれかの方法で「シナリオ定義」ダイアログ・ボックスを開きます。
   - テンプレート・ツリー・ビューでシナリオを選択し、「テンプレート」—> 「シナリオ定義を開く」を選択します。

- テンプレート・ツリー・ビューでシナリオを選択し、右マウス・ボタンでクリックしてコンテキスト・メニューを開きます。コンテキスト・メニューから「シナリオ定義を開く」を選択します。
- ダイアグラム・エディターのアクティビティー・ダイアグラムで、右マウス・ボタンでクリックしてコンテキスト・メニューを開きます。コンテキスト・メニューから「シナリオ定義を開く」を選択します。

2. 変数テーブルの左の見出しセルを右マウス・ボタンでクリックして、コンテキスト・メニューから「追加」を選択します。テーブルに新しい行が追加されます。

   **注:** テーブルの最後の行をクリックして、新しい行を追加することもできます。
3. 宣言する変数のタイプを指定するには、「タイプ」列のドロップダウン・リストを使用します。
4. 「名前」列で変数の名前を指定します。
5. 「初期値」列で変数の初期値を指定します。

   **注:** ストリング値は、引用符で囲んで入力する必要があります (例えば、ストリング `Yes` は `"Yes"` のように入力します)。
6. 変数 (transient、private、protected など) に適用させる修飾子を、「Modifier1」、「Modifier2」、および「Modifier3」列で指定します。すべての 3 つの列において修飾子を指定する必要はありません。

   **注:** シナリオ変数の宣言に、キーワード `public` および `static` を含めてはいけません。
7. タブの上部にある宣言のリストに新しい変数を追加するには、「更新」をクリックしてから、「適用」をクリックして変更内容を保管します。

### 長期存続ビジネス・プロセスのシナリオ変数に関する特殊考慮事項

シナリオ変数は、長期存続ビジネス・プロセスのイベント・フロー・コンテキストの一部として自動的に永続されるわけではありません。長期存続ビジネス・プロセス・コラボレーション内でシナリオ変数を使用する場合には、サービス呼び出しの前にシナリオ変数を手動で null に設定し、サービス呼び出しの完了後にシナリオ変数を再初期化する必要があります。これらの作業は、サービス呼び出しを実行するアクション・ノードで実行します。

以下に示す例では、アクション・ノードでシナリオ変数 poolName が null に設定されてから、サービス呼び出しが実行されます。

```
String poolName;
poolName = null;
```

サービス呼び出しの完了後、poolName はアクション・ノードで以下のように再初期化されます。

```
poolName = getConfigProperty("Pool_A");
```

## シナリオの削除

シナリオは、Process Designer Express を使用して削除できます。削除したシナリオを元に戻すことはできません。

シナリオ定義を削除する手順は、以下のとおりです。

1. テンプレート・ツリー・ビューで、削除するシナリオを選択します。
2. 「テンプレート」—>「シナリオを削除」をクリックします。削除について確認するダイアログ・ボックスが表示されます。
3. シナリオを削除する場合は「はい」をクリックします。

## アクティビティー・ダイアグラムの作成

各シナリオには、必ず 1 つのアクティビティー・ダイアグラムがあります。アクティビティー・ダイアグラム は、Unified Modified Language (UML) を使用してコラボレーションのビジネス・プロセスをモデル化します。 UML は、ビジネス・プロセスのステップおよび決定を示します。アクティビティー・ダイアグラムは、Process Designer Express のダイアグラム・エディターで作成します。

アクティビティー・ダイアグラムの作成については、 107 ページの『第 5 章 アクティビティー・ダイアグラムの使用』を参照してください。

## メッセージ・ファイルの作成

コラボレーション・テンプレートの作成プロセスの一部として、メッセージの定義があります。コラボレーション・ランタイム環境では、メッセージ・ファイルの内容をロギング、トレース、および例外メッセージのテキストとして使用します。

Process Designer Express には、メッセージの作成を支援する「テンプレート・メッセージ」ビューが用意されています。指定したメッセージ・テキストは、コラボレーション・テンプレートの一部として保管されます。テンプレートをコンパイルして配置すると、Process Designer Express によってメッセージの内容が抽出され、実行時に使用するメッセージ・ファイルが作成または更新されます。

メッセージ・ファイルの作成については、 197 ページの『第 9 章 メッセージ・ファイルの作成』を参照してください。

## コラボレーション・テンプレートのコンパイル

コラボレーション・テンプレートの作成には、テンプレートのコンパイルという最終作業が必要です。テンプレート・プロパティー、シナリオ、アクティビティー・ダイアグラム、およびメッセージ・ファイルの定義を終えたら、コラボレーション・テンプレートをコンパイルする必要があります。コンパイル時には、以下のファイルが作成されます。

- Java ソース・ファイル (*CollaborationName*.java)
- 実行可能ファイル (C*ollaborationName*.class)
- メッセージ・テキスト・ファイル (*CollaborationName*.txt)

Process Designer Express でのテンプレートのコンパイルが完了すると、System Manager の 統合コンポーネント・ライブラリー・ユーザー・プロジェクト内に上記のファイルが作成されます。(正確な位置については 7 ページの『コラボレーション・テンプレートのコンパイル』を参照してください。)

Process Designer Express には、コラボレーション・テンプレートのコンパイル方法として以下の 2 つの方法が用意されています。

- 『単一テンプレートのコンパイル』
- 『複数のコラボレーション・テンプレートのコンパイル』

## 単一テンプレートのコンパイル

単一コラボレーション・テンプレートのコンパイルを開始するには、以下のようないくつかの方法があります。

- Process Designer Express 内から「ファイル」—>「コンパイル」をクリックします。
- テンプレート・ツリー・ビューでテンプレート名を選択し、右マウス・ボタンでクリックしてコンテキスト・メニューを開き、「テンプレートをコンパイル」をクリックします。
- `Ctrl + F7` キーボード・ショートカットを使用します。

「Compile Output」ウィンドウがまだ開いていない場合は、メインウィンドウの下部で Process Designer Express が開き、コンパイル・メッセージが表示されます。

コンパイル時にエラーが発生した場合の手順は、次のとおりです。

1. 出力ウィンドウのエラー・メッセージをダブルクリックしてエラーをトレースします。

   障害ノードが選択された状態で、コンパイル・エラーを生成したコードを持つアクティビティー・ダイアグラムが表示されます。
2. 問題を修正し、再コンパイルしてください。以下のメッセージが表示されるまで、このプロセスを繰り返します。

   ```
   Code Generator: Code generation succeeded.
   ```

## 複数のコラボレーション・テンプレートのコンパイル

Process Designer Express の「ファイル」メニューの「すべてコンパイル」メニュー・オプションを使用すると、統合コンポーネント・ライブラリー・ユーザー・プロジェクトのすべてのコラボレーション・テンプレートまたはコラボレーション・テンプレートのサブセットをコンパイルできます。複数のテンプレートをコンパイルするには、以下の手順を実行します。

1. Process Designer Express でテンプレートを開いている場合には、テンプレートを閉じます。
2. 「ファイル」—>「すべてコンパイル」をクリックします。「すべてのテンプレートをコンパイル」ダイアログ・ボックスが表示されます。このダイアログ・ボックスには、ユーザー・プロジェクトのすべてのテンプレートのグリッドが表示されます。デフォルトでは、すべてのテンプレートがコンパイル対象として選択されています。
3. コンパイルしないテンプレートの隣にあるチェック・ボックスからチェックを外します。
4. 「継続」をクリックします。
5. コンパイル実行の確認プロンプトが出されたら、「はい」をクリックします。

# テンプレートの変換

Process Designer Express には、以下の変換機能が備わっています。

- インポート: Process Designer Express では、コラボレーション・テンプレートとして使用する Business Process Execution Language (BPEL) ファイルおよび統一モデリング言語 (XMI 1.1 の UML) ファイルをインポートできます。『ファイルのインポート』を参照してください。
- エクスポート: Process Designer Express では、コラボレーション・テンプレートを BPEL 形式または UML (XMI 1.1) 形式でエクスポートできます。『コラボレーション・テンプレートのエクスポート』を参照してください。

## ファイルのインポート

Process Designer Express では、BPEL および UML (XMI 1.1) ファイルをインポートし、コラボレーション・テンプレートで使用できます。これらのファイルの情報を使用して、新規テンプレート定義を作成します。

既存の BPEL ファイルまたは UML (XMI 1.1) ファイルに基づいて新規コラボレーション・テンプレートを作成する手順は以下のとおりです。

1. Process Designer Express が開かれていることを確認します。
2. 「ファイル」—> 「インポート」をクリックします。Process Designer Express Importer が開きます。
3. インポートするファイル・タイプを選択し、「次へ」をクリックします。
4. BPEL または UML ソース・ファイルのロケーションを選択します。
5. インポートするファイルを選択します。

   **注:** BPEL ファイルを使用する場合には、.bpel、.wsdl、および .bpelGUI.xml の 3 ファイルをすべてインポートする必要があります。Ctrl キーを使用して、これら 3 つのファイルをすべてインポート対象として選択します。
6. 「次へ」をクリックします。インポート・プロセスが開始されます。インポートが完了したら、「新規テンプレート」ダイアログ・ボックスが表示されます。
7. 「プロジェクト」フィールドに、テンプレートが含まれているユーザー・プロジェクトの名前を選択します。
8. 「テンプレート名」フィールドに作成するテンプレートの名前を入力します。テンプレート名には、英字、数値、および下線を使用できます。
9. 「OK」をクリックします。Process Designer Express により新規コラボレーション・テンプレートが作成され、ソース BPEL または UML ファイルの情報がこのテンプレートに挿入されます。

## コラボレーション・テンプレートのエクスポート

他のアプリケーションで使用できるようにコラボレーション・テンプレートを BPEL または UML (XMI 1.1) 形式でエクスポートできます。Business Integration Express コラボレーション・テンプレートを BPEL 形式でエクスポートすると、以下のファイルが作成されます。

- *.bpel: メイン・テンプレート情報が記述されているファイルです。
- *.wsdl: 外部インターフェースに関する情報が記述されているファイルです。

- *.bpelGUI.xml: アクティビティー・ダイアグラムのグラフィカル表現に関する情報が記述されているファイルです。このファイルは、BPEL ファイルが InterChange Server Express に再度インポートされる場合に使用されます。

コラボレーション・テンプレートを UML 形式 (XMI 1.1) でエクスポートすると、*.xmi ファイルが作成されます。

以下の手順を実行して、Business Integration Express コラボレーション・テンプレートをエクスポートします。

1. Process Designer Express が開いており、またコラボレーション・テンプレートが保管されていてエラーなしでコンパイルが完了していることを確認します。
2. 「ファイル」—>「エクスポート」をクリックします。Process Designer Express Exporter が開きます。
3. テンプレートのエクスポート形式を選択し、「次へ」をクリックします。
4. エクスポートするテンプレートの保管先を選択します。
5. 「ファイル名」フィールドに、エクスポートするテンプレート・ファイルの名前を指定します。BPEL 形式でエクスポートする場合には、「ファイル名」フィールドにはファイル名拡張子を*指定しないでください*。
6. 「次へ」をクリックします。エクスポート・プロセスが開始されます。「Process Designer Express Exporter」ダイアログに、変換処理の進行状況が表示されます。
7. エクスポート・プロセスが完了したら、「閉じる」をクリックします。

## コラボレーション・テンプレートの削除

**要確認**

コラボレーション・テンプレートがコラボレーション・オブジェクトに関連付けられており、これらのオブジェクトを削除する予定がない場合には、これらのテンプレートを削除しないでください。コラボレーション・テンプレートを削除すると、このテンプレートから作成されたすべてのオブジェクトが使用できなくなります。(System Manager でコラボレーション・オブジェクトを削除してからコラボレーション・テンプレートを削除する方法については、「*WebSphere InterChange Server システム・インプリメンテーション・ガイド*」を参照してください。)

コラボレーション・テンプレートのうち、コラボレーション・オブジェクトがそのテンプレートから構築されていないコラボレーション・テンプレートは、Process Designer Express を使用して削除できます。テンプレートを削除する手順は、以下のとおりです。

1. Process Designer Express を開きます。また、System Manager が実行中であることを確認します。
2. 「ファイル」—>「削除」をクリックします。Process Designer Express には、「プロジェクト '*ProjectName*' からテンプレートを削除してください。」ダイアログ・ボックスが表示されます。
3. 「プロジェクト」ドロップダウン・リストから、削除するテンプレートを含むプロジェクトの名前を選択します。

4. コラボレーション・テンプレートのリストから、削除するテンプレートの名前を選択して、「OK」をクリックします。
5. 削除について確認するプロンプトが出されます。「はい」をクリックします。

## コラボレーションのテスト

コラボレーション・テンプレートの作成とコンパイルが完了したら、テンプレートの設計をテストできます。コラボレーションが予定通りに機能することを確認するには、コラボレーション・オブジェクトを作成し、Test Connector ツールを使用してコラボレーション・オブジェクトの機能性をテストします。

Business Integration Express テスト環境の一部となっている Test Connector は、実際のコネクターをシミュレートします。Test Connector を使用して、コラボレーションへイベントと応答を送信します。これにより、コラボレーションの機能をテストするビジネス・オブジェクトとトリガー・イベントをセットアップできます。

テストするコラボレーションが 1 つのコネクターへのポートが 1 つの場合は、Test Connector のインスタンスを 1 つ開いてください。コラボレーションが 1 つのコネクターからのポートを受信ポートとし、別のコネクターへは別のポートを使用する場合、Test Connector の 2 つのインスタンス (各コネクター用に 1 つずつ) を開いてください。

「Test Connector」メニューを使用して、構成ファイルとエミュレートするコネクターの定義を指定します。選択されているビジネス・オブジェクトの値を設定してから、そのビジネス・オブジェクトを送信および受信します。

統合テスト環境および Test Connector の使用方法の詳細については、「*WebSphere InterChange Server システム・インプリメンテーション・ガイド*」を参照してください。

# 第 5 章 アクティビティー・ダイアグラムの使用

この章では、Process Designer Express を使用してアクティビティー・ダイアグラムを編集する方法について説明します。アクティビティー・ダイアグラム により、コラボレーションの特定の部分の制御フローが定義されます。アクティビティー・ダイアグラムはシナリオ作成時に自動的に作成されます。ダイアグラムは、特定の順序で実行されるステップの集合です。アクティビティー・ダイアグラムには、ステップを示すシンボル、ステップの順序、およびステップの実行方法を決定するロジックが含まれます。

ワークスペースのレイアウトおよび表示については、 187 ページの『第 8 章 ワークスペースとレイアウトのオプション』を参照してください。

シナリオに対して新しいアクティビティー・ダイアグラムを編集するには、以下のタスクを実行します。

1. 作業域にダイアグラム・エディターを表示します。

   ダイアグラム・エディターは、次のいずれかの方法で表示できます。

   - テンプレート・ツリー・ビューでシナリオ名またはダイアグラム名を選択し、「テンプレート」プルダウン・メニューから「ダイアグラムを開く」を選択します。
   - テンプレート・ツリー・ビューでシナリオ名またはダイアグラム名を選択し、右マウス・ボタンでクリックしてコンテキスト・メニューを開きます。コンテキスト・メニューから「ダイアグラムを開く」を選択します。
   - シナリオ名をダブルクリックします。

   選択されたシナリオのアクティビティー・ダイアグラムと共にダイアグラム・エディターを表示します。
2. ダイアグラム・エディターで、シンボル・ツールバーに示されているシンボルを使用してアクティビティー・ダイアグラムを編集します。
3. 「ファイル」メニューの「保管」オプションを使用するか、ショートカット・キーの組み合わせ `Ctrl+S` を使用し、アクティビティー・ダイアグラムを保管します。

## ダイアグラム・エディター機能の使用

ダイアグラム・エディターの機能には、以下のいずれかの方法でアクセスできます。

- ツールバー
- アクティビティー・ダイアグラムのシンボル上でのマウス移動

### ダイアグラム・エディター機能へのアクセス: Process Designer Express メニュー

Process Designer Express には、ダイアグラム関連の多くの操作を行うためのプルダウン・メニューが用意されています。これらの機能の一部には、キー・ストローク・ショートカットおよびコンテキスト・メニューがあります。これらのメニューについての詳細は、22 ページの『Process Designer Express メニュー』を参照してください。

### ダイアグラム・エディター機能へのアクセス: マウス移動

ダイアグラム・エディターでは、以下のマウス移動が認識されます。

- 左マウス・ボタンをクリックし、アクティビティー・ダイアグラム内のコンポーネントまたはシンボルを選択します。

  ダイアグラム・エディターでは、「*選択枠*」と呼ばれるグレーのアンカーの四角でシンボルが囲まれ、シンボルが選択されたことが示されます。シンボルの選択を解除するには、ワークスペース内の任意の位置をクリックします。
- 右マウス・ボタンをクリックし、シンボルのコンテキスト・メニューを選択して表示します。

  コンテキスト・メニューには、以下のオプションがあります。
  - 「プロパティー」: 選択されたシンボルに該当する「Symbol Properties」ダイアログを表示します。これは、「編集」メニューの「プロパティー」オプションと同じです。詳細については、24 ページの『「編集」メニューの機能』の「プロパティー」オプションの説明を参照してください。
  - 「フォント」: シンボル・テキストを表示するフォントを制御します。これは、「編集」メニューの「フォント」オプションと同じです。詳細については、24 ページの『「編集」メニューの機能』の「フォント」オプションの説明を参照してください。
- アクティビティー・ダイアグラム内の任意の位置 (ただし、特定のシンボル以外) で右マウス・ボタンをクリックし、アクティビティー・ダイアグラムの親ダイアグラムに移動します。

## アクティビティー・ダイアグラムのシンボル

アクティビティー・ダイアグラムでは、シンボルを使用して実行ステップを示します。このセクションでは、アクティビティー・ダイアグラムのシンボルに関する以下の情報について説明します。

- シンボルのタイプ
- フローチャートのシンボルとの比較
- すべてのシンボルのプロパティー

### シンボルの概要

109 ページの図 32 に、アクティビティー・ダイアグラムのシンボルと、シンボル・ツールバーの対応するボタンを示します。このツールバーは、作業域にダイアグラム・エディターが表示されている場合にアクティブになります。

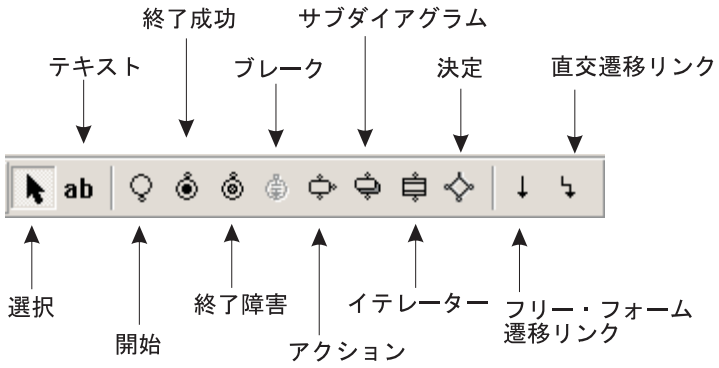



*図 32. シンボル・ツールバー*

アクティビティー・ダイアグラムには、ノード、遷移リンク、およびサービス呼び出しという 3 つのシンボルのメイン・タイプがあります。また、開始シンボルと終了シンボルも含まれています。

## 開始シンボルと終了シンボル

アクティビティー・ダイアグラムを作成すると、開始シンボルが自動的にダイアグラムに配置されます。開始シンボルはフローの開始を示します。各アクティビティー・ダイアグラムには開始シンボルが 1 つ必要です。

開始シンボルを使用して相関属性を初期化できます。詳細については、132 ページの『相関属性の使用』を参照してください。

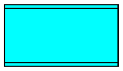
Process Designer Express では、アクティビティー・ダイアグラムの終了シンボルとして終了成功と終了障害の 2 つがあります。アクティビティー・ダイアグラムの各実行経路は、終了成功または終了障害のいずれかで終了している必要があります (ただし、ブレーク・シンボルで終了しているイテレーター・アクティビティー・ダイアグラムは例外です)。終了シンボルの使用方法の詳細については、146 ページの『実行経路の終了』を参照してください。

## ノード・シンボル

*ノード* は、コラボレーション内のステップを表すシンボルです。ノードには、*アクション*、*決定*、*サブダイアグラム*、および*イテレーター* という 4 つのタイプがあります。各ノードは、シンボル・ツールバーに固有のシンボルとして表されます (図 32 を参照)。

110 ページの表 30 に、ノードのタイプごとにアクティビティー・ダイアグラム内で表されるシンボルを示します。

*表 30. ノード・シンボル*

| ノード・タイプ | アクティビティー・ダイアグラム内のシンボル | 詳細 |
|---|---|---|
| アクション |  | 111 ページの『アクション・ノード』 |
| 決定 |  | 120 ページの『決定ノード』 |
| サブダイアグラム |  | 135 ページの『サブダイアグラム』 |
| イテレーター |  | 141 ページの『イテレーター』 |

## 遷移リンク・シンボル

*遷移リンク* は、ノード間の制御フローを表します。ダイアグラムのフローは上から下へと流れるため、遷移リンクは常に縦方向の向きとなります。ノードから複数の経路がある場合、決定ノードに遷移リンクを使用する必要があります。決定ノードのロジックにより、採用されるパスが決まります。

ダイアグラム・エディターでは、遷移リンクをフリー・フォーム・リンクまたは直交リンクのいずれかの方法で表します。表 31 に、それぞれのタイプの遷移リンクを表すアクティビティー・ダイアグラム・シンボルを示します。

*表 31. 遷移リンク・シンボル*

| 遷移リンク・タイプ | アクティビティー・ダイアグラム内のシンボル | 詳細 |
|---|---|---|
| フリー・フォーム遷移リンク |  | 116 ページの『遷移リンク』 |
| 直交遷移リンク |  | 116 ページの『遷移リンク』 |

## サービス呼び出しシンボル

*サービス呼び出し* は、外部エンティティーとの間でポート経由で送受信される応答または要求を表します。サービス呼び出しは、常に横方向の向きとなります。サービス呼び出しはアクション・ノードに接続されます。デフォルトでは、サービス呼び出しのラベルにサービス呼び出しのタイプが記述されます。サービス呼び出しには以下の種類があります。

- 同期サービス呼び出し
- 非同期アウトバウンド・サービス呼び出し
- 非同期インバウンド・サービス呼び出し

シンボル・ツールバーにはサービス呼び出しのシンボルは含まれません。サービス呼び出し機能は、アクション・ノードを右マウス・ボタンでクリックすると表示されるコンテキスト・メニューから使用できます。

サービス呼び出しのタイプとそれらをアクティビティー・ダイアグラムに組み込む方法については、 125 ページの『サービス呼び出し』を参照してください。

## ダイアグラム・シンボルのプロパティー

アクティビティー・ダイアグラム内のシンボルには、表 32 のようなプロパティーがあります。

*表 32. シンボルのプロパティー*

| シンボル・プロパティー | 説明 |
|---|---|
| 固有 ID (UID) | アクティビティー・ダイアグラム内のすべてのシンボルには、固有 ID (UID) があります。ダイアグラムに UID を表示するかどうかは選択できます。シンボルには独自のラベルを割り当てることができますが、独自のラベルによって UID が置き換えられることはありません。UID は、コンパイル・メッセージおよびトレース・メッセージのシンボルを示します。「表示」プルダウン・メニューの「UID を表示」オプションを使用して、UID を表示するかどうかを選択できます。 |
| オプションのラベル | このラベルは、アクティビティー・ダイアグラムをよりわかりやすくするための記述名を示します (ラベルが表示される場合)。「表示」プルダウン・メニューの「ラベルを表示」オプションを使用して、ラベルを表示するかどうかを選択できます。 |
| オプションの説明 | この説明はコメントです。 |
| タイプ固有のプロパティー | 一部のシンボル (アクション・ノードなど) には、コード・フラグメントが関連付けられています。 |

ほとんどのシンボルは、プロパティーを編集できます。以下のいずれかの方法により、「Symbol Properties」ダイアログを表示します。

- アクティビティー・ダイアグラムのシンボルを右マウス・ボタンでクリックしてコンテキスト・メニューを表示し、「プロパティー」を選択します。
- アクティビティー・ダイアグラムのシンボルを選択し、「編集」プルダウン・メニューから「プロパティー」を選択します。
- アクティビティー・ダイアグラムのシンボルをダブルクリックし、「Symbol Properties」ダイアログを表示します。
- `Ctrl + Enter` のショートカットの組み合わせを使用します。

# アクション・ノード

アクション・ノード (場合によっては単にアクション と呼ばれます) は、コラボレーションにおけるステップを示します。アクション・ノードは、コラボレーション・ロジックの構築ブロックです。コラボレーション・ロジックをアクション・ノードに分割するかどうかは、完全にユーザーの自由です。多数の行から成る複雑なコードを単一アクションで記述することも、ロジックを多数の個別アクションに分割することもできます。コラボレーション・ロジックをアクション・ノードに分割することは、プログラム・コードの開発に似ています。サブルーチンあるいはメソッド呼び出しの集合を呼び出す短いメインルーチンを使用してプログラムを作成し、プログラムの機能を実行することができます。また、すべてのプログラム・ロジックをインラインに含む長いメインルーチンを作成することもできます。

## ダイアグラムへのアクションの追加

アクション・ノードをアクティビティー・ダイアグラムに追加する手順は、以下のとおりです。

1. シンボル・ツールバーの「アクション」ボタンをクリックします。
2. ワークスペース内の任意の位置をクリックしてアクション・シンボルを配置します。

**注:** サービス呼び出しシンボルをアクションに接続すると、アクション・ノードによってサービス呼び出しが作成されます。サービス呼び出しについては、125 ページの『サービス呼び出し』を参照してください。

## アクション・ノード・プロパティーの定義

アクティビティー・ダイアグラムにアクション・ノードが表示されたら、「Action Properties」ダイアログ・ボックスを使用して以下のいずれかのノード用プロパティーを定義します。

- 「ラベル」: アクション・ノードにラベルを提供します。デフォルトの UID の代わりに記述テキストを使用すると、ダイアグラムの読み取りおよび使用が容易になります。このプロパティーはオプションです。
- 「説明」: アクション・ノードの目的を説明します。このプロパティーはオプションです。
- 「コード・フラグメント」: アクション・ノードの動作を定義します。詳細については、113 ページの『アクション・ノードへのアクティビティー定義の追加』を参照してください。

以下のいずれかの方法で「Action Properties」ダイアログ・ボックスを開きます。

- 選択されているアクション・ノードをダブルクリックします。
- アクション・ノードを右マウス・ボタンでクリックしてコンテキスト・メニューを表示し、「プロパティー」を選択します。
- アクション・ノードを選択し、「編集」プルダウン・メニューから「プロパティー」を選択します。
- アクション・ノードを選択し、キーボード・ショートカット `Ctrl + Enter` を使用します。

「Action Properties」ダイアログの上部には、アクション・ノードの名前が表示されます。この名前のフォーマットは、以下のとおりです。

`Action_`*`UID`*

ここで、*UID* はアクション・ノードの固有 ID です。 113 ページの図 33 に「Action Properties」ダイアログ・ボックスを示します。

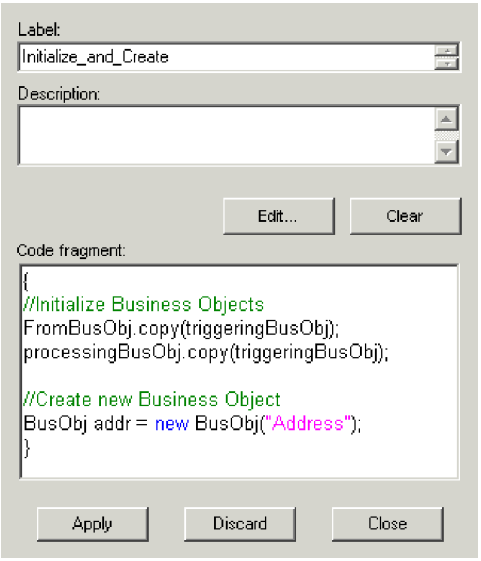



*図 33. 「Action Properties」ダイアログ*

## アクション・ノードへのアクティビティー定義の追加

アクション・ノードにはアクティビティー定義が含まれています。アクティビティー定義 (コード・フラグメント とも呼ばれる) は、コラボレーション API への呼び出しまたはその他の Java コードで構成されており、次のような操作を含めることができます。

- ビジネス・オブジェクトの属性値の取得および設定
- 入力イベントの動詞のチェック
- 属性値と定数またはその他の属性値との比較
- サービス呼び出しで使用するビジネス・オブジェクト変数の設定
- メッセージのロギング
- Web サービスの呼び出し

アクティビティー定義には、任意の Java プログラム言語構造を含めることができます。

アクティビティー定義をアクション・ノードに追加するには、次の 2 つの方法があります。

- 「Action Properties」ダイアログ・ボックスの「コード・フラグメント」ウィンドウに Java コードを直接入力する。 114 ページの『「Action Properties」ダイアログ・ボックスでのアクティビティー定義の追加』を参照してください。
- Activity Editor を使用して、グラフィカルな機能ブロックまたは従来の Java コードを追加する。 114 ページの『Activity Editor でのアクティビティー定義の追加』を参照してください。

## 「Action Properties」ダイアログ・ボックスでのアクティビティー定義の追加

「Action Properties」ダイアログ・ボックスの「コード・フラグメント」ウィンドウに Java コードを直接追加する手順は、以下のとおりです。

1. 「コード・フラグメント」ウィンドウの直接編集を使用可能にするため、次の操作を行います。
   a. Process Designer で、「表示」—>「設定」をクリックします。「ユーザー設定」ダイアログ・ボックスが表示されます。
   b. 「ダイアグラム」タブで「新規アクション・ノード (空コード) のインプレース編集を使用可能にする」オプションと「既存のアクション・ノードのインプレース編集を使用可能にする」オプションを選択します。
   c. 「適用」をクリックして「OK」をクリックします。
2. アクション・ノードを右マウス・ボタンでクリックしてコンテキスト・メニューを表示させます。
3. コンテキスト・メニューから「プロパティー」をクリックします。「Action Properties」ダイアログ・ボックスが表示されます。
4. 「コード・フラグメント」ウィンドウに Java コードを入力します。
5. 「適用」をクリックして変更内容を保管します。

## Activity Editor でのアクティビティー定義の追加

Activity Editor を使ってアクティビティー定義を追加する手順は、以下のとおりです。

1. アクション・ノードを右マウス・ボタンでクリックしてコンテキスト・メニューを表示させます。
2. コンテキスト・メニューから「プロパティー」をクリックします。「Action Properties」ダイアログ・ボックスが表示されます。
3. 「編集」をクリックし、Activity Editor を表示します。デフォルトでは、Activity Editor はグラフィック表示で表示されます。

   **注:** このノードに対しアクティビティー定義をすでに作成しており、グラフィック表示で認識できないカスタム Java コードがこの定義に含まれている場合には、Java 表示で表示されます。
4. アクティビティー定義を追加します。詳細については、160 ページの『アクティビティー定義』を参照してください。
5. Activity Editor を閉じます。「Action Properties」ダイアログ・ボックスがまだ開いている場合には、アクティビティー定義に関連するコード・フラグメントが表示されます。
6. 「適用」をクリックして変更内容を保管します。

## アクティビティー定義での Java 関係演算子の使用

アクティビティー定義では Java 演算子を使用できます。具体的には、算術演算子のほかに、関係演算子や条件演算子を使用できます。

Java クラスはすべて基本 Object クラスから継承されます。このため、各クラスには使用可能な equals() メソッドが含まれます。このシグニチャーは次のとおりです。

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

equals() メソッドは、値の等価性に関するメソッドです。このメソッドは、このメソッドを呼び出すオブジェクトと、obj によって参照されるオブジェクトを比較して等価性を調べます。値が同じであれば true を戻し、等しくなければ false を戻します。これは、等価演算子 == および != とは異なります。これら 2 つの等価演算子は、2 つの参照がオブジェクトの値に関係なく同じオブジェクトを参照しているかどうかを判断します。この相違は重要です。

表 33 に、使用可能な等価演算子をまとめてあります。各等価演算子または関係演算子は、boolean 値の結果を生成します。

*表 33. Java 関係演算子*

| 関係演算子 | 意味 |
|---|---|
| > | より大きい |
| >= | より大きいか等しい |
| < | より小さい |
| <= | より小さいか等しい |
| == | 等しい |
| != | 等しくない |
| ! | 単項演算子。boolean 値を反転します。 |

boolean 式の結果は、条件付き AND (&&) および条件付き OR (||) シンボルと結合できます。

表 34 に、算術演算子 (増分演算子と減分演算子を含む) についてまとめてあります。

*表 34. Java 算術演算子と代入演算子*

| 算術演算子 | 意味 |
|---|---|
| + | 加算 |
| - | 減算 |
| * | 乗算 |
| / | 割り算 |
| % | 余り |
| - | 否定、または数値の符号を反転する単項演算子 |
| ++ | 増分演算子 |
| -- | 減分演算子 |
| = | 代入演算子 |

## アクティビティー定義での Web サービスの使用

コラボレーション・テンプレートのアクティビティー定義で Web サービスを使用する場合は、まず System Manager の ICL (Integration Component Library) プロジェクトから Web サービスをエクスポートする必要があります。Web サービスの各メソッドは、機能ブロックとして Activity Editor にエクスポートされ、そこでアクティビティー定義に配置することができます。詳細については、160 ページの『Web サービス機能ブロック』を参照してください。

### アクティビティー定義に関する情報の取得

アクション・ノードのロジックのコーディングについては、次の表を参照してください。

*表 35. コード・フラグメントに関する情報の取得*

| 追加リソース | ロケーション |
|---|---|
| Activity Editor の詳細記述 (機能ブロックの使用例を含む) | 153 ページの『第 6 章 Activity Editor の使用』 |
| コード・フラグメントの例 | 203 ページの『第 10 章 コーディングのヒントと例』 |
| 個々の機能ブロックの参照ページ | 第 11 章から第 21 章 |
| コラボレーション API の個々のメソッドの参照ページ | 第 22 章から第 32 章 |

## 遷移リンク

遷移リンクは、アクティビティー・ダイアグラムの制御フローを示します。遷移リンクは、アクティビティーが発生するノード (アクション、決定、サブダイアグラム、イテレーターなど) を接続し、これらのノードを開始シンボルと完了シンボルに接続します。遷移リンクには、ビジネス・オブジェクト・インスタンス値をモニターするビジネス・オブジェクト・プローブを組み込むことができます。

**注:** アクティビティー・ダイアグラムでは、遷移リンクはデータ・フローを示しません。 1 つのノードが変数を設定し、別のノードがこの変数にアクセスすると、データがノード間で渡されます。アクティビティー・ダイアグラムのデータ・フロー・メカニズムは、他のコードが使用する変数がコードによって設定されるという点で、クラスやプログラムのメカニズムに似ています。このモデルは、リンクに沿ったデータの移動を示す、イベント渡しモデリング・ツールによって使用されるモデルとは異なります。

Process Designer Express には、直交リンクとフリー・フォーム・リンクの両方があります。直交リンクは、可能ならいつでも使用できます。フリー・フォーム・リンクを使用するのは、直交リンクから目的の形状が得られない場合にしてください。リンクを右マウス・ボタンでクリックし、リンクの直交に関するコンテキスト・メニューを表示します。このコンテキスト・メニューを使用して、リンクの直交とフリー・フォームを切り替えます。

### 同時に使用できるリンクの数

表 36 に、異なるノードが使用できる入力および出力リンクの数を示します。

*表 36. ノード・タイプ別に使用できる入力および出力リンク*

| ノード・タイプ | 入力リンク | 出力リンク |
|---|---|---|
| アクション | 無制限 | 1 |
| 決定 | 1 | 7 |
| サブダイアグラムまたはイテレーター | 無制限 | 1 |

## 遷移リンクの作成

遷移リンクを作成するには、接続する 2 つのシンボルがワークスペース上で使用可能である必要があります。

遷移リンクをアクティビティー・ダイアグラムに追加する手順は、以下のとおりです。

1. シンボル・ツールバーの「遷移リンク」ボタンをクリックします。
2. ワークスペースで、遷移リンクを開始するシンボルの下端をクリックします。
3. 遷移リンクを終了するシンボルの上端をクリックします。

Process Designer Express では、シンボル間で有効なリンクを設定できます。リンクが無効な 2 つのシンボルを接続することはできません。Process Designer Express では、無効な遷移リンクを作成することはできません。この場合、エラー・メッセージは表示されません。

Process Designer Express では、シンボルに遷移リンクを設定しようとすると、この試みが有効かどうか示されます。シンボルの端にマウス・ポインターを (リンクを使用して) 置くと、この接続が有効であれば、マウス・ポインターが丸で囲まれた正符号 (+) に変わります。これをクリックすると、シンボルにリンクを設定できます。この接続が無効な場合は、マウス・ポインターは正符号には変わらず、リンクを設定することはできません。

### リンクの取り消し

遷移リンクは、Escape (ESC) キーを押すことにより、打ち切る (取り消す) ことができます。ESC キーを押すたびに、接続線の最後の線分が元に戻されます。アクティビティー・ダイアグラムに有効なシンボルがない接続を取り消す場合、ESC キーを使用するのが唯一の方法です。

例えば、ダイアグラムに開始シンボルと終了シンボルの 2 つのシンボルを設定するとします。次に、遷移リンクを選択し、開始シンボルをクリックします。これにより、開始シンボルに接続された遷移リンク線分が表示されます。ただし、この遷移リンク線を接続できるシンボルまたはポートはありません。この場合、接続アクティビティーを取り消して編集セッションを続ける唯一の方法は、ESC キーを押す方法です。

## 遷移リンク・プロパティーの定義

遷移リンクがアクティビティー・ダイアグラムに表示されたら、「Link Properties」ダイアログでそのプロパティーを定義できます (図 34 を参照)。

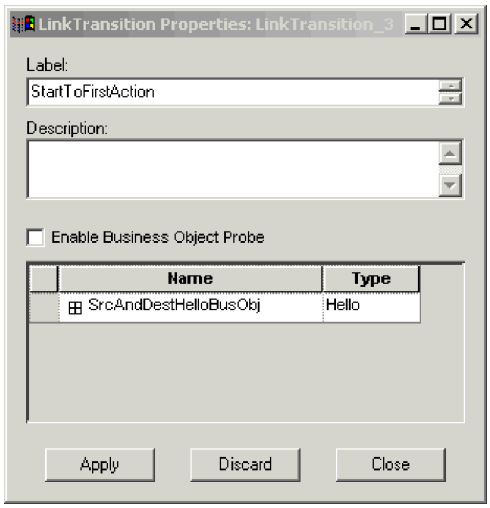



*図 34. 遷移リンクの「Link Properties」ダイアログ・ボックス*

「Link Properties」ダイアログでは、ダイアログの上部に次のフォーマットで名前が表示された状態で遷移リンクが表示されます。

`LinkTransition_`*`UID`*

ここで、*UID* はリンクの固有 ID です。

遷移リンクのプロパティーには、ラベル、定義、およびビジネス・オブジェクト・プローブの有無が含まれます。

遷移リンク・プロパティーを定義する手順は、以下のとおりです。

1. 「Link Properties」ダイアログを表示します。

   このダイアログは、以下のいずれかの方法で表示できます。
   - 選択されている遷移リンクをダブルクリックします。
   - 遷移リンクを右マウス・ボタンでクリックしてコンテキスト・メニューを表示し、「プロパティー」を選択します。
   - 遷移リンクを選択し、「編集」プルダウン・メニューから「プロパティー」を選択します。
   - 遷移リンクを選択し、キーボード・ショートカット `Ctrl + Enter` を使用します。

   「Link Properties」ダイアログが表示されます。図 34 に、「Link Properties」ダイアログのサンプルが示されています。
2. オプションで、この遷移リンクにラベルおよび説明を指定します。

   リンク・ラベルの詳細については、118 ページの『リンクのラベル付け』を参照してください。
3. 「適用」をクリックしてリンク・プロパティーを保管します。

## リンクのラベル付け

リンク・ラベルを使用することで、アクティビティー・ダイアグラムを読みやすくすることができます。フローチャートで決定ノードにラベルを付けるときと同じ方

法で決定ロジックを使用するようにしてください。リンク・ラベルに名前を論理的に付けることにより、シナリオ・フローを説明できます。以下に例を示します。

- 2 つの遷移リンク分岐が CONVERT_VERB という構成プロパティーの値に基づいている場合、ラベルを DoConvert と DoNotConvert にします。
- 1 つの遷移リンクが成功サービスを処理し、別の遷移リンクがサービス呼び出し例外を処理する場合、ラベルは Success や ServiceCallException などとすることができます。

遷移リンクにラベルを付けるには、「Link Properties」ダイアログの「ラベル」ボックスにリンク・ラベルのテキストを入力します。このラベルは、テキスト・ボックスとまったく同じようにアクティビティー・ダイアグラムに表示されます。ラベルが他のリンクに重ならないようにラベルを複数の行に分割するには、テキスト・ボックスで改行を使用します。

## ビジネス・オブジェクト・プローブの使用

ビジネス・オブジェクト・プローブは、実行時にビジネス・オブジェクト・インスタンスの値をモニターします。プローブはアクティビティー・ダイアグラム作成時に遷移リンクに配置され、実行時に System Manager の「コラボレーション・プロパティー」ダイアログ・ボックスで活動化または非活動化されます。

デフォルトでは、ビジネス・オブジェクト・プローブは赤い四角としてアクティビティー・ダイアグラムの遷移リンク上に表示されます。図 35 では、デフォルトの分岐リンクにビジネス・オブジェクト・プローブが含まれています。

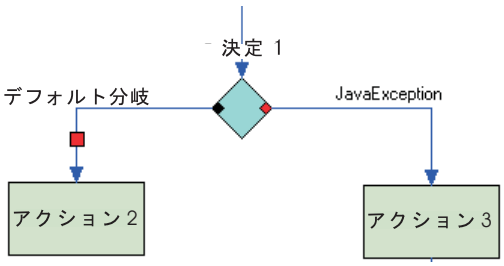



*図 35. ビジネス・オブジェクト・プローブのある遷移リンク*

ビジネス・オブジェクト・プローブを使用すれば、「テンプレート定義」ウィンドウの「ポートおよびトリガー・イベント」タブで指定されたビジネス・オブジェクトをモニターできます。ただし、以下の例外があります。

- ビジネス・オブジェクト・プローブは、決定ノードの受信遷移リンクでは使用できません。
- ビジネス・オブジェクト・プローブは、サービス呼び出しリンクでは使用できません。
- ビジネス・オブジェクト・プローブは、N カーディナリティーの子ビジネス・オブジェクトをプローブできません。プローブは、単一カーディナリティーの子ビジネス・プロジェクト内のデータにのみアクセスできます。

ビジネス・オブジェクトごとにモニターしたい特定の属性を選択できます。これらの属性のインスタンス値はすべて、System Monitor により作成されるレポートに記述されます。

ビジネス・オブジェクト・プローブを追加するタスクは、以下のとおりです。

1. Process Designer Express が開いており、アクティビティー・ダイアグラムが表示されていることを確認します。
2. ビジネス・オブジェクト・プローブを追加する遷移リンクを右マウス・ボタンでクリックします。
3. コンテキスト・メニューから「プロパティー」をクリックします。「Link Transition Properties」ダイアログ・ボックスが表示されます。
4. 「ビジネス・オブジェクト・プローブを使用可能にする」チェック・ボックスをクリックします。
5. モニターするビジネス・オブジェクトの横にある正符号 (+) をクリックし、このオブジェクトの属性リストを表示します。
6. モニターする特定の属性を選択します。
7. 「適用」をクリックして変更内容を保管します。
8. 実行時に必要に応じて System Manager でビジネス・オブジェクト・プローブの使用可能/使用不可を切り替えます。

### 遷移リンクの変更

遷移リンクを切断または再接続したり、遷移線分の外観を変更できます。

- 遷移リンクを切断または再接続するには、マウスを使用してリンク点 (線の端) を選択し、目的の方向にドラッグします。切断または再接続により、遷移リンクを移動できるようになります。
- 既存の遷移リンクに新しい遷移線分を作成するには、Ctrl キーを押したまま線分の真中をクリックします。 2 つの線分の間の結合部分が四角でマークされます。
- 遷移線分を除去するには、Ctrl キーを押したまま線分の四角をクリックします。

**注:** 線分の変更は、直交リンクには適用されません。

遷移線分が直角を保つようにするには、Shift キーを押したまま新しい線分を作成します。

## 決定ノード

条件に関係なく、あるアクションから次のアクションに移す場合、遷移リンクが必要です。ただし、一連の条件に基づいて複数のアクションに分岐させる場合は、決定ノードを組み込む必要があります。 最も一般的な使用法では、決定ノードはアクションを可能性のあるすべての結果 (他のアクション、サブダイアグラム、終了シンボルを含む) に接続します。決定ノードは、アクション・ノード、サブダイアグラム・ノード、およびイテレーター・ノードと共に使用できます。開始シンボルの後に直接決定ノードを配置しないでください。

一般に決定ノードには最低 2 つの分岐があります。分岐は最大 7 つまで指定可能です。各分岐にはそれに関連する条件があり、その条件によって分岐を採用するかどうかが決定します。

**要確認**

決定ノードを実装するときには、常に 1 つの条件だけが true として評価されるように条件を定義してください。決定ノードの条件すべてが true にならない場合には、実行時エラーが発生します。

決定ノードの分岐には、以下の 3 つのタイプがあります。

- 標準: 標準分岐には、それに関連する条件があります。条件が合うと、分岐が採用されます。標準分岐は複数持つことができます。デフォルトでは、標準分岐は青色の四角で表されます。
- 例外: 例外分岐には、それに関連する特定の例外タイプがあります。例外分岐の条件により、システム変数 currentException が、分岐に設定した例外タイプと等しいかどうかテストされます。例外分岐は複数持つことができます。デフォルトでは、例外分岐は赤色の四角で表されます。
- デフォルト: デフォルト分岐は、他の分岐の条件がいずれも true でない場合に採用されます。各決定ノードには 1 つ (1 つのみ) のデフォルト分岐を設定できます。この分岐はオプションです。デフォルトでは、この分岐は黒色の四角で表されます。

これらの分岐を定義し、その条件を設定するには、図 36 に示す「Decision Properties」ダイアログ・ボックスを使用します。

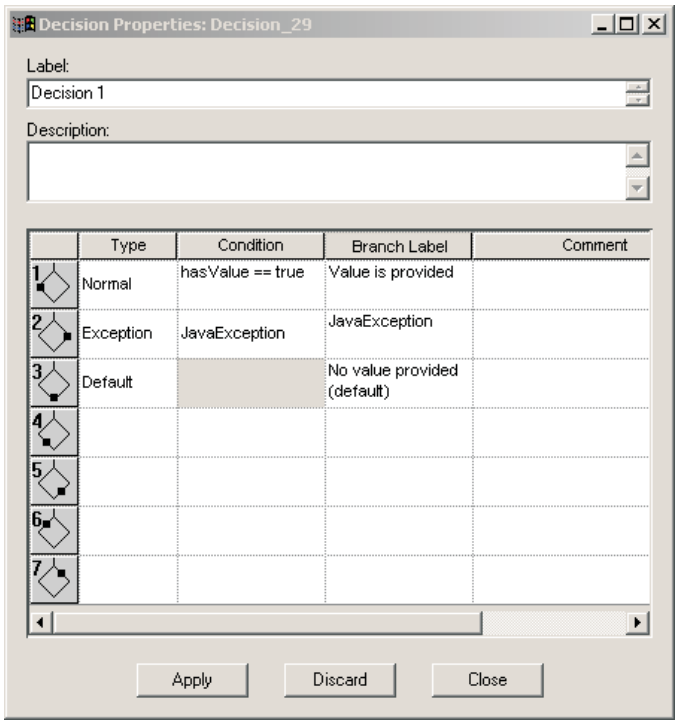



*図 36. 「Decision Properties」ダイアログ・ボックス*

決定ノードの各定義済み分岐には、ノードを関連する結果 (アクション・ノードや終了シンボルなど) に接続する遷移リンクを 1 つ設定する必要があります。

図 37 に、決定ノードを使用するアクティビティー・ダイアグラムの例を示します。この例で、決定ノードには 3 つの分岐があります。標準分岐は、条件が true と評価されるとアクション 2 にフローを移します。例外分岐は、JavaException 例外がスローされると終了障害にフローを移します。デフォルト分岐は、標準分岐の条件が false と評価され、JavaException がスローされない場合に、アクション 3 にフローを移します。

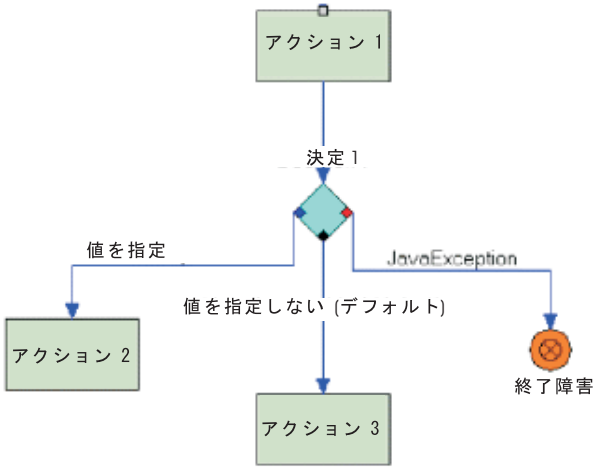



*図 37. 決定ノードを使用するアクティビティー・ダイアグラム*

決定ノードをアクティビティー・ダイアグラムに追加する手順は、以下のとおりです。

1. ダイアグラム・エディターが開いていることと、決定ノードへフローするシンボルをすでに配置していることを確認します。決定ノードは、アクション、サブダイアグラム、またはイテレーター・ノードにより使用できます。
2. シンボル・ツールバーの「決定ノード」ボタンをクリックします。
3. ダイアグラムで、決定ノードを使用するシンボルの下にカーソルを置いてクリックします。ノードがダイアグラムに配置されます。
4. 決定ノードとそれを呼び出すシンボルの間に遷移リンクを作成します。遷移リンクの作成についての詳細は、117 ページの『遷移リンクの作成』を参照してください。

## 標準分岐の定義

各標準分岐には条件が必要です。これらの条件は、ユーザーがコラボレーション・テンプレートまたはシナリオで定義した変数を使用して作成します。標準分岐を作成するには、まず条件に必要な変数を定義する必要があります。詳細については、85 ページの『テンプレート変数の宣言と編集 (「宣言」タブ)』および 100 ページの『シナリオ変数の定義』を参照してください。

決定ノードで標準分岐を定義する手順は、以下のとおりです。

1. Activity Editor で、決定ノード・シンボルをダブルクリックします。「Decision Properties」ダイアログ・ボックスが表示されます。
2. 作成中の分岐の行で、「タイプ」列のテーブル・セルをクリックし、分岐タイプのドロップダウン・リストから「標準」を選択します。

3. 「条件」列のテーブル・セルを右マウス・ボタンでクリックし、コンテキスト・メニューから「条件ビルダー」を選択します。

   **注:** 「条件エディター」を使用せずに、条件を直接「条件」テーブル・セルに入力することもできます。

   「条件エディター」ダイアログ・ボックスが表示されます。

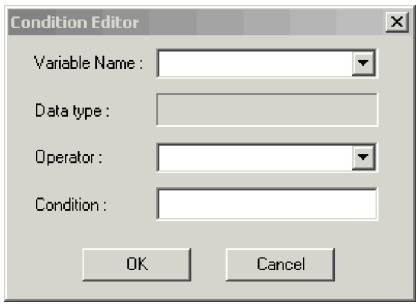



*図 38. 条件エディター*

4. 「変数名」フィールドで、ドロップダウン・リストを使用して、条件を評価する変数を選択します。このリストには、シナリオで定義済みのコラボレーション変数がすべて含まれています。

   変数を選択すると「データ型」フィールドが自動的に更新されて、変数の型が表示されます (例えば Boolean や String)。
5. 「演算子」フィールドで、ドロップダウン・リストを使用して、変数の評価に使用する演算子を選択します。リストには、ユーザーが使用している変数の型でサポートされる演算子のみが含まれます。
6. 「条件」フィールドで、条件に使用する値を入力します。(例えば、hasValue という名前の Boolean 変数を使用している場合、条件を true または false に設定できます。)
7. 「OK」をクリックして「条件エディター」を閉じ、「Decision Properties」ダイアログ・ボックスに戻ります。
8. オプションで、「分岐ラベル」テーブル・セルに分岐のラベルを入力します。分岐にラベルを付けると、アクティビティー・ダイアグラムが読みやすくなります。
9. オプションで、「コメント」テーブル・セルに分岐の説明を入力します。
10. 「適用」をクリックして、決定ノードに分岐を追加します。アクティビティー・ダイアグラムの決定ノードには、作成したばかりの標準分岐を示す青色の四角が含まれているはずです。

標準分岐を追加したら、遷移リンクを使用してそれを関連する結果に接続する必要があります。

## 例外分岐の定義

決定ノードで例外分岐を定義する手順は、以下のとおりです。

1. Activity Editor で、決定ノード・シンボルをダブルクリックします。「Decision Properties」ダイアログ・ボックスが表示されます。

2. 作成中の分岐の行で、「タイプ」列のテーブル・セルをクリックし、分岐タイプのドロップダウン・リストから「例外」を選択します。
3. 「条件」列のテーブル・セルをクリックし、例外タイプのドロップダウン・リストから例外のタイプを選択します。
4. オプションで、「分岐ラベル」テーブル・セルに分岐のラベルを入力します。分岐にラベルを付けると、アクティビティー・ダイアグラムが読みやすくなります。
5. オプションで、「コメント」テーブル・セルに分岐の説明を入力します。
6. 「適用」をクリックして、決定ノードに分岐を追加します。アクティビティー・ダイアグラムの決定ノードには、作成したばかりの例外分岐を示す赤色の四角が含まれているはずです。

例外分岐を追加したら、遷移リンクを使用してそれを関連する結果に接続する必要があります。

## デフォルト分岐の定義

各決定ノードにはデフォルト分岐を 1 つのみ追加できます。デフォルト分岐の追加はオプションです。

デフォルト分岐を決定ノードに追加する手順は、以下のとおりです。

1. Activity Editor で、決定ノード・シンボルをダブルクリックします。「Decision Properties」ダイアログ・ボックスが表示されます。
2. 作成中の分岐の行で、「タイプ」列のテーブル・セルをクリックし、分岐タイプのドロップダウン・リストから「デフォルト」を選択します。
3. オプションで、「分岐ラベル」テーブル・セルに分岐のラベルを入力します。分岐にラベルを付けると、アクティビティー・ダイアグラムが読みやすくなります。
4. オプションで、「コメント」テーブル・セルに分岐の説明を入力します。
5. 「適用」をクリックして、決定ノードに分岐を追加します。アクティビティー・ダイアグラムの決定ノードには、作成したばかりのデフォルト分岐を示す黒色の四角が含まれているはずです。

デフォルト分岐には、条件を指定できません。デフォルト分岐の条件は暗黙的であり、他の分岐に関連するすべての条件が false と評価される場合に true と評価されます。

デフォルト分岐を追加したら、遷移リンクを使用してそれを関連する結果に接続する必要があります。

## 分岐ロジックにおける例外と条件の結合

分岐は標準分岐または例外分岐のいずれかです。1 つの分岐に両方を設定することはできません。ただし場合によっては、例外が発生し、別の条件が true であるという 2 つの条件が同時に成立する状況にとる実行経路を指定する必要があります。この組み合わせは、条件式で AND 演算子を使用するのと同じです。

例えば、以下の 2 つの条件をモデル化したいとします。

```
Exception == JavaException && hasValue == false
Exception == JavaException && hasValue == true
```

このような構成をモデル化するには、以下に示すように 2 つのレベルの決定ノードを作成し、これらノードの間にアクション・ノードを作成します。

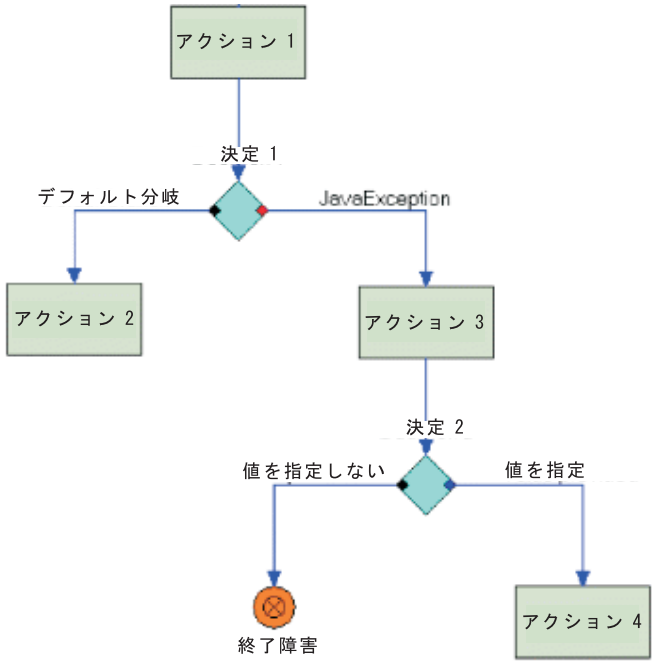



*図 39. 分岐ロジックにおける例外と条件の結合*

## サービス呼び出し

アクション・ノードは、それ自体ではコネクターまたは別のコラボレーションに要求を送信することはできません。代わりに、ユーザーがアクション・ノードをサービス呼び出しに接続する必要があります。InterChange Server Express では、同期および非同期のサービス呼び出しがサポートされています。以下は、Retrieve 要求を使用する同期サービス呼び出しの例です。

```
status = retrieve(business-object, port);
```

完了呼び出しの状況　要求タイプ　送信元の変数から送信され、同じ変数に戻されるデータ

サービス呼び出しは常にアクションに接続されます。アクションは、サービス呼び出しを生成し、サービス呼び出しの結果をその出力遷移リンク上で処理します。アクションとサービス呼び出しはペアとして機能し、サービス呼び出しはアクションのリモート入出力関数を実行します。

図 40 に、アクション・ノードとサービス呼び出し間の関係を示します。この図は、コラボレーション・ランタイム環境がアクションおよびそのサービス呼び出しを処理する順序を示します。

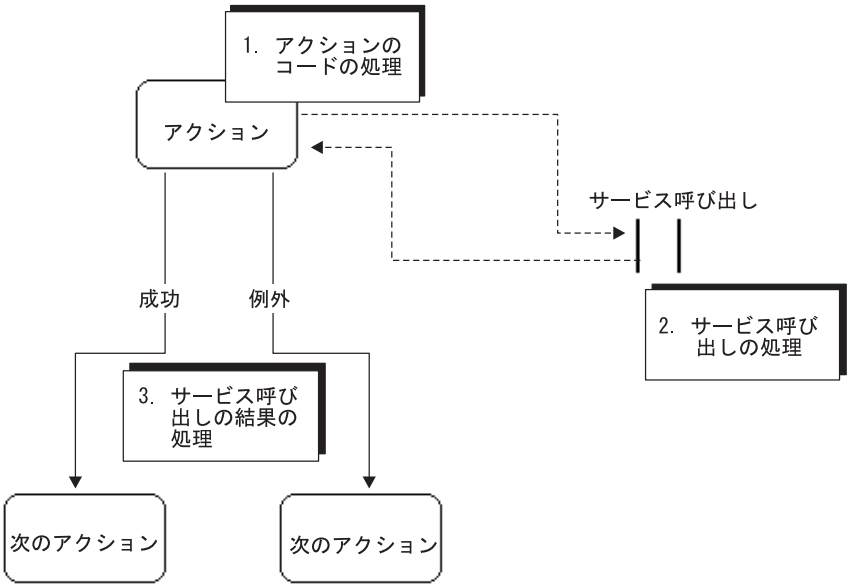



*図 40. アクション・ノードとサービス呼び出しの関係*

図 40 に、コラボレーション・ランタイム環境が最初にアクションのコードを処理し、次にサービス呼び出しを実行する様子を示します。サービス呼び出しが完了すると、アクション・ノードの出力遷移リンクがその結果を処理します。サービス呼び出しを生成するアクション・ノードの場合、ServiceCallException 例外をチェックする出力例外リンクを組み込むことをお勧めします。

## サービス呼び出しのタイプ

InterChange Server Express では、同期、非同期アウトバウンド、非同期インバウンドの 3 つのタイプのサービス呼び出しがサポートされています。以下のセクションでは、呼び出しの各タイプについて説明します。

### 同期サービス呼び出し

このタイプのサービス呼び出しは、同期要求/応答機構を使用します。サービス呼び出しは、要求を送信しますが、応答が到着して処理されるまではその処理が完了しません。

同期サービス呼び出しでは、差し戻しがサポートされています。また、長期存続ビジネス・プロセスのタイムアウト値もサポートされています。

デフォルトでは、アクティビティー・ダイアグラムに追加されるすべてのサービス呼び出しは同期サービス呼び出しです。必要に応じて、シナリオでタイプを変更できます。

### 非同期アウトバウンド・サービス呼び出し

非同期アウトバウンド・サービス呼び出しは要求を送信しますが、応答を待たずに処理を続行します。非同期アウトバウンド・サービス呼び出しを使用するために、コラボレーション・テンプレートは長期存続ビジネス・プロセスをサポートしている必要があります。

非同期アウトバウンド・サービス呼び出しに設定されているポートが、コネクターではなくコラボレーションにバインドされている場合、このサービス呼び出しは自動的に同期サービス呼び出しになります。

「非同期アウトバウンド」サービス呼び出しでは、差し戻しはサポートされますが、長期存続ビジネス・プロセスのタイムアウト値はサポートされません。

### 非同期インバウンド・サービス呼び出し

非同期インバウンド・サービス呼び出しは、受信イベントを識別する相関属性または相関属性のセットに基づいて受信イベントの受信を待機します。この呼び出しは長期存続ビジネス・プロセスとともに使用されます。(詳細については、 132 ページの『相関属性の使用』を参照。)

非同期インバウンド・サービス呼び出しが作成されると、タイムアウト値が付与されます。サービス呼び出しがこのタイムアウト値以内に受信イベントを受信しない場合、例外 TimeoutException が発生します。

非同期インバウンド・サービス呼び出しを使用できるのは、コラボレーション・テンプレートで長期存続ビジネス・プロセスがサポートされる場合のみです。このサポートは、テンプレート定義の「一般」タブで「長期存続ビジネス・プロセスのサポート」オプションをクリックして、テンプレート作成時にいつでも使用可能にできます。

非同期インバウンド・サービス呼び出しでは、差し戻しはサポートされません。

## サービス呼び出しの作成

サービス呼び出しをアクティビティー・ダイアグラムに追加する手順は、以下のとおりです。

1. ダイアグラム・エディターが開いていることを確認します。
2. ワークスペースで、サービス呼び出しを関連付けるアクション・ノードのシンボルを右マウス・ボタンでクリックします。
3. コンテキスト・メニューから「サービス・ノードを追加」をクリックします。これでサービス呼び出しがアクティビティー・ダイアグラムに追加され、サービス呼び出しとアクション・ノードが点線で結ばれます。デフォルトでは、サービス呼び出しのタイプは同期です。

## サービス呼び出しの定義

サービス呼び出しを作成したら、それを定義する必要があります。「Service Call Properties」ダイアログ・ボックスを使用して、以下の必須プロパティーを指定します。

- サービス呼び出しの送信先のポート

- 送信するビジネス・オブジェクトを含む変数
- ビジネス・オブジェクトの動詞
- 相関セット (非同期インバウンド・サービス呼び出しに必要です)

オプションで、以下も指定できます。

- 差し戻しのサポート (トランザクション・コラボレーションを使用している場合)。詳細については、131 ページの『差し戻しの定義』を参照してください。
- サービス呼び出しのタイプ (デフォルトではすべてのサービス呼び出しは同期ですが、非同期アウトバウンドまたは非同期インバウンド呼び出しに変更できます)。詳細については、 129 ページの『サービス呼び出しタイプの定義』を参照してください。
- 同期および非同期インバウンド・サービス呼び出しで使用されるタイムアウト値。詳細については、 129 ページの『サービス呼び出しタイプの定義』を参照してください。
- 属性のマッチングに使用される相関セット (同期または非同期アウトバウンド・サービス呼び出しの場合)。

**ヒント**

サービス呼び出しが戻されたとき、ビジネス・オブジェクト変数には呼び出しの結果が含まれます。サービス呼び出しがビジネス・オブジェクトに関する新しいデータを取得すると、元のビジネス・オブジェクトのデータは失われます。このため、元のビジネス・オブジェクト値が必要になると予期される場合は、サービス呼び出しを呼び出すアクションの一時変数に元のビジネス・オブジェクトをコピーしておくと便利です。

通常のサービス呼び出し (差し戻しや相関セットを使用しないサービス呼び出し) を定義する手順は、以下のとおりです。

1. アクティビティー・ダイアグラムで、作成済みのサービス呼び出しシンボルをダブルクリックします。 129 ページの図 41 に示すように、「Service Call Properties」ダイアログ・ボックスが表示されます。サービス呼び出しの「ラベル」フィールドに値を直接入力することはできません。サービス呼び出し定義が完了すると、ラベルが割り当てられます。

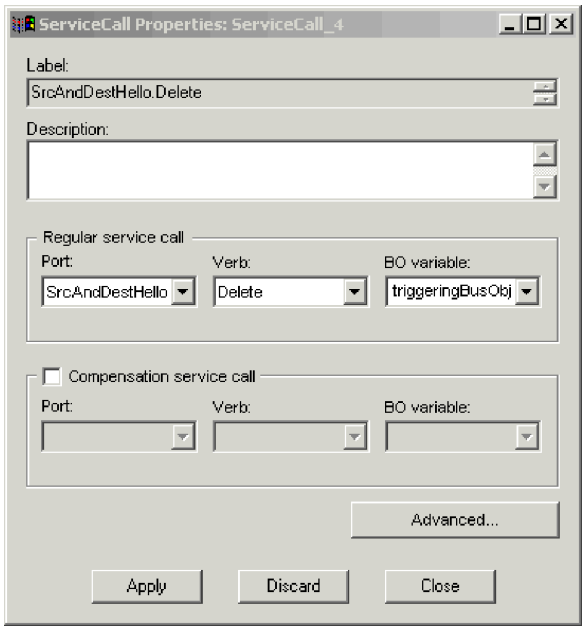



*図 41. 「Service Call Properties」ダイアログ・ボックス*

2. オプションで、このサービス呼び出しの説明を入力します。
3. 「ポート」ドロップダウン・リストを使用して、サービス呼び出しが要求の送受信に使用するポートを選択します。
4. 「動詞」ドロップダウン・リストを使用して、要求に使用する動詞を指定します。例えば、サービス呼び出しが送信するビジネス・オブジェクトに含まれるデータを使用してアプリケーションを更新する場合は、Update 動詞を使用します。
5. 「BO 変数」ドロップダウン・リストを使用して、サービス呼び出しが送信するビジネス・オブジェクトが含まれる変数を選択します。長期存続ビジネス・プロセスをサポートする場合には、この変数はグローバル・テンプレート変数またはポート変数でなければなりません。長期存続ビジネス・プロセスをサポートする場合にはシナリオ変数は使用できません。
6. 「適用」をクリックして定義を保管します。

## サービス呼び出しタイプの定義

サービス呼び出しのタイプを変更する手順は、以下のとおりです。

1. 「Service Call Properties」ダイアログ・ボックスが開いていない場合、アクティビティー・ダイアグラムのサービス呼び出しシンボルをダブルクリックして表示します。
2. サービス呼び出しに、必須のポート名、動詞、ビジネス・オブジェクト変数名が指定されているか確認します。
3. 「Service Call Properties」ダイアログ・ボックスの「拡張」ボタンをクリックします。「Service Call Advanced Properties」ダイアログ・ボックスが表示されます。

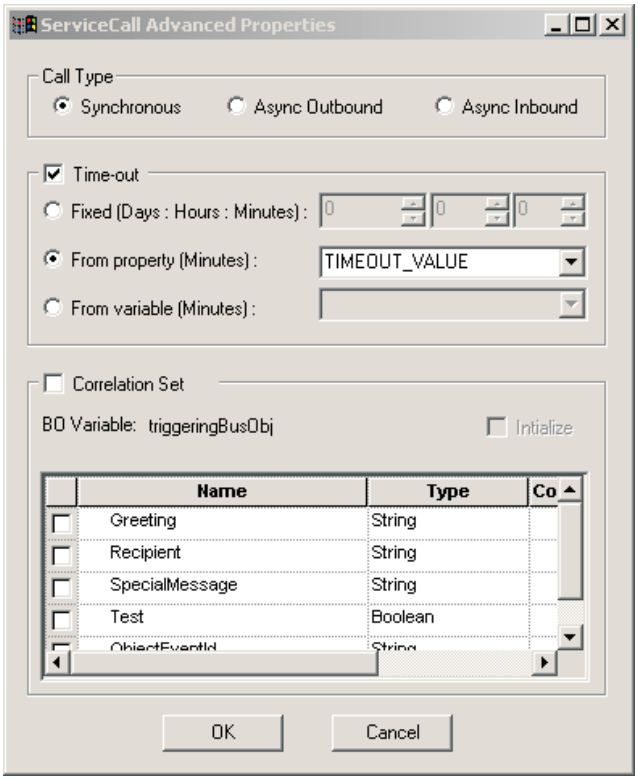



*図 42. 「Service Call Advanced Properties」ダイアログ・ボックス*

4. 「呼び出しタイプ」ボックスで、使用するサービス呼び出しのタイプの横にあるラジオ・ボタンをクリックします。

   **注:** 「非同期インバウンド」オプションが選択可能なのは、テンプレート定義で長期存続ビジネス・プロセスのサポートを使用可能にしてある場合のみです。

5. 同期サービス呼び出しまたは非同期インバウンド・サービス呼び出しを使用していて、長期存続ビジネス・プロセスに使用するタイムアウト値を指定する場合、「タイムアウト」チェック・ボックスをクリックして、以下のいずれかのタイムアウト値を設定します。
   - 「固定」: このオプションには、タイムアウト値を日、時間、分単位で指定する必要があります。サービス呼び出しに常に同じタイムアウト値を使用する場合、このオプションを選択します。この値はコラボレーションの構成時に変更できません。
   - 「プロパティーから」: このオプションを使用すると、コラボレーション固有のプロパティーを使用して、コラボレーションの構成時にタイムアウト値 (秒単位) を動的に指定できます。「プロパティーから」ドロップダウン・リストで、作成したプロパティーを選択してタイムアウト値を示します。
   - 「変数から」: このオプションを使用すると、グローバル Java オブジェクト・タイプの変数を使用して、実行時のタイムアウト値 (秒単位) を設定できます。「変数から」ドロップダウン・リストから、該当する変数を選択します。

**注:** 同期サービス呼び出しのポートがコネクターではなくコラボレーションにバインドされている場合には、指定されるタイムアウト値は無視されます。

6. 「OK」をクリックして「Service Call Advanced Properties」ダイアログ・ボックスを閉じます。
7. 「Service Call Properties」ダイアログ・ボックスの「適用」ボタンをクリックして変更内容を保管します。

## 差し戻しの定義

コラボレーションのサブトランザクション・ステップにより、トランザクション・コラボレーションのトランザクションの振る舞いを定義します。サブトランザクション は、アプリケーション・データ・ストアのトランザクション・データを変更する要求をコラボレーションが送信する操作です。サービス呼び出しにより、サブトランザクション・ステップが実装されます。Create、Delete、または Update 要求を持つサービス呼び出しは、サブトランザクション・ステップです。ただし、Retrieve 要求はサブトランザクション・ステップではありません (データが変更されないため)。その他の動詞は、アプリケーション・データ・ストアのデータを変更するかどうかにより、トランザクションであるかどうかが決まります。

**注:** Retrieve 要求を使用したサービス呼び出しはサブトランザクション・ステップとしてみなされませんが、この呼び出しの差し戻しを指定できます。

トランザクションの振る舞いをサポートするコラボレーション・テンプレートを作成するには、サブトランザクション・ステップごとに差し戻し を定義します。差し戻しは論理的な取り消しアクションです。つまり、コラボレーション・オブジェクトの実行が失敗すると、前に実行された操作がロールバックされます。ロールバックが行われると、コラボレーション・ランタイム環境ステップが実行経路に応じて逆戻りします。この場合、すでに実行され、差し戻しが定義されているすべての通常ステップに対して差し戻しステップが実行されます。この結果、ロールバックにより、データは、トランザクション・コラボレーションが実行される前の状態に戻されます。

Process Designer Express に表示されるとおり、サブトランザクション・ステップは、Create、Update、または Delete などの操作を要求するサービス呼び出しです。これらの操作により、常にアプリケーション内のトランザクションのデータが変更されます。また、Retrieve 操作を要求するサービス呼び出しの差し戻しも指定できます。ただし Retrieve 操作実行時にはデータは変更されないため、差し戻しを実行する必要はありません。サービス呼び出しは、コラボレーションが別のコラボレーションまたはコネクターに送信する特定の動詞を持つビジネス・オブジェクトによって定義されます。この操作の差し戻しは、別のビジネス・オブジェクトおよび動詞です。サービス呼び出しに対しては、任意のビジネス・オブジェクトおよび動詞を差し戻しとして使用できます。

表 37 に、差し戻しの共通タイプの一部を示します。

*表 37. 差し戻しの例*

| アクション | 差し戻し |
|---|---|
| ビジネス・オブジェクトの作成 | ビジネス・オブジェクトの削除 |
| ビジネス・オブジェクトの削除 | ビジネス・オブジェクトの作成 |

*表 37. 差し戻しの例 (続き)*

| アクション | 差し戻し |
| --- | --- |
| ビジネス・オブジェクトの更新 | 以前の値を復元して、ビジネス・オブジェクトを更新 |

差し戻しは、同期サービス呼び出しと非同期アウトバウンド・サービス呼び出しでサポートされます。コラボレーションがトランザクションであり、このサービス呼び出しの動詞がデータの変更を要求する場合、通常のサービス呼び出しをロールバックする差し戻し操作を指定できます。差し戻しを定義する手順は、以下のとおりです。

1. 「Service Call Properties」ダイアログ・ボックスが開いていない場合、アクティビティー・ダイアグラムのサービス呼び出しシンボルをダブルクリックして表示します。
2. 「差し戻し」チェック・ボックスをクリックします。「差し戻しサービス呼び出し」ボックスの「ポート」、「動詞」、「BO 変数」の各フィールドがアクティブになります。
3. 「差し戻しサービス呼び出し」ボックスで、差し戻しサービス呼び出しに使用するポート、動詞、ビジネス・オブジェクト変数を選択します。差し戻しサービス呼び出しでは、通常サービス呼び出しとして同一のポート、動詞、およびビジネス・オブジェクトを使用できます。または、異なるポート、動詞、およびビジネス・オブジェクトを指定することもできます。
4. 「適用」をクリックして変更内容を保管します。

トランザクション・コラボレーションについては、146 ページの『トランザクション機能の使用』を参照してください。

## 相関属性の使用

相関属性により会話が識別されます。*会話* とは、2 つのビジネス・プロセス間での首尾一貫した情報伝達の単位です。2 つ以上のビジネス・プロセスが相互に通信する場合には複数の会話が発生するため、相関属性によって会話を識別する必要があります。相関属性は会話の UID として考えることができます。この ID は会話開始時に初期化されます。後続の参加プログラムが会話に加わるときには、この ID を使用する必要があります。

InterChange Server Express ではシナリオ当たり 1 つの会話のみがサポートされています。複数の会話が必要な場合には、コラボレーション・テンプレート内で複数のシナリオを使用するか、または複数のコラボレーション・テンプレートを使用する必要があります。また、相関属性の初期化はシナリオ内で 1 回のみ実行できます。

相関属性を使用するには、以下の条件を満たしていることを確認してください。

- コラボレーション・テンプレートを定義したときに、長期存続ビジネス・プロセスのサポートを追加している。
- 相関属性値を収集するときに使用されるテンプレート変数を 1 つ以上作成している。相関として選択する各ビジネス・オブジェクト属性に対し、その値を収集できる固有のテンプレート変数を設定する必要があります。

サービス呼び出しで相関属性を使用すると、ビジネス・オブジェクトが自動的に判別されます。これは常に、サービス呼び出しで使用されるポートに割り当てられているビジネス・オブジェクトです。

シナリオで初期化された相関属性をアウトバウンド・サービス呼び出しに設定し(『相関属性の設定』を参照)、非同期インバウンド・サービス呼び出しで属性をマッチングできます ( 134 ページの『相関属性のマッチング』を参照)。

***相関属性の初期化:*** 相関属性を使用するには、最初に初期化を行います。相関属性を初期化するときに、ビジネス・オブジェクト属性と、属性の値を収集するために使用するテンプレート変数を指定します。アウトバウンド・サービス呼び出しに相関属性を設定する前、またはランタイム環境で相関属性のマッチングを実行する前に初期化を実行してください。

開始ノード、非同期アウトバウンド・サービス呼び出し、または同期サービス呼び出しで初期化を実行できます。

サービス呼び出しの相関属性を定義および初期化する手順は、以下のとおりです。

1. Process Designer Express が開いており、「Service Call Properties」ダイアログ・ボックスが表示されていることを確認します。
2. 「拡張」をクリックします。「Service Call Advanced Properties」ダイアログ・ボックスが表示されます。
3. 「相関セット」をクリックします。ビジネス・オブジェクト変数が、サービス呼び出しのアウトバウンド・ポートへ割り当てられる変数として自動的に定義される点に注意してください。
4. 「初期化」をクリックします。
5. 相関に使用する各ビジネス・オブジェクト属性の隣のチェック・ボックスをクリックします。
6. 選択した属性ごとに、属性値の収集および保管に使用されるテンプレート変数を「相関」列のドロップダウン・リストから選択します。
7. 「適用」をクリックします。

開始ノードの相関セットを初期化する手順は、「初期化」チェック・ボックスをクリックする点を除き上記の手順と同じです。開始ノードは相関属性のマッチングに関連していないので、初期化は暗黙に実行されます。

***相関属性の設定:*** 相関セットの定義と初期化が完了したら、相関セットをアウトバウンド・サービス呼び出しに割り当てることができます。次にサービス呼び出しは、相関セットが組み込まれている要求を送信します。相関属性を設定できるアウトバウンド・サービス呼び出しの数に制限はありません。

アウトバウンド・サービス呼び出しで相関属性を設定する手順は、以下のとおりです。

1. Process Designer Express が開いており、アウトバウンド・サービス呼び出しの「Service Call Properties」ダイアログ・ボックスが表示されていることを確認します。
2. 「拡張」をクリックします。「Service Call Advanced Properties」ダイアログ・ボックスが表示されます。

3. 「相関セット」をクリックします。
4. 初期化済み相関セットで定義されている各ビジネス・オブジェクト属性の隣のチェック・ボックスをクリックします。
5. 選択した属性ごとに、属性値の収集および保管に使用されるテンプレート変数を「相関」列のドロップダウン・リストから選択します。この変数は、相関設定の初期化時に使用された変数でなければなりません。
6. 「適用」をクリックします。

***相関属性のマッチング:*** 相関セットの初期化が完了したら、特定の相関セットと一致する応答を受信できるように非同期インバウンド・サービス呼び出しを構成できます。

非同期インバウンド・サービス呼び出しに相関セットを指定する手順は、以下のとおりです。

1. Process Designer Express が開いており、非同期インバウンド・サービス呼び出しの「Service Call Properties」ダイアログ・ボックスが表示されていることを確認します。
2. 「拡張」をクリックします。「Service Call Advanced Properties」ダイアログ・ボックスが表示されます。
3. 「相関セット」をクリックします。
4. 初期化済み相関セットで定義されている各ビジネス・オブジェクト属性の隣のチェック・ボックスをクリックします。
5. 選択した属性ごとに、属性値の収集および保管に使用されるテンプレート変数を「相関」列のドロップダウン・リストから選択します。この変数は、相関設定の初期化時に使用された変数でなければなりません。
6. 「適用」をクリックします。

実行時に、非同期インバウンド・サービス呼び出しの属性が相関セットで定義されている属性と一致する場合には、サービス呼び出しはシナリオにより呼び出されます。属性が一致しない場合には、一致する相関セットが定義されている他のシナリオまたはコラボレーションへサービス呼び出しの経路が設定されます。

## 結果の処理

サービス呼び出しが実行されると、シナリオは、状況とビジネス・オブジェクトという 2 つの戻り値を受け取ります。表 38 で、それぞれの使用について説明します。

*表 38. サービス呼び出しの戻り値*

| 戻り値 | 説明 |
|---|---|
| 状況 | サービス呼び出しを生成したアクションの通常の出力遷移リンクは、サービス呼び出しが成功かどうかテストします。例外遷移リンクは、`ServiceCallException` 例外をチェックし、サービス呼び出しが失敗かどうかテストします。サービス呼び出しは、トランスポート問題やアプリケーション問題が原因で失敗することがあります。転送障害によってデータが重複する可能性があるため、サービス呼び出しの障害が転送問題が原因であるかどうかを確認することが重要です。詳細については、181 ページの『特定のサービス呼び出し例外の処理』を参照してください。 |
| ビジネス・オブジェクト | サービス呼び出しによってアプリケーションが変更された場合は、サービス呼び出しが完了すると、サービス呼び出しが使用したビジネス・オブジェクト変数に新しいデータ値が含まれます。 |

- Create サービス呼び出しの完了後は、ビジネス・オブジェクトの値をチェックする必要はありません。サービス呼び出しが正常に戻された場合、Create 操作は正常に実行されています。
- Delete 要求の場合、必ずしもアプリケーション・データが実際に削除されるわけではありません。多くのアプリケーションは削除処理をサポートしていないため、Delete 動詞はアプリケーションの規則に応じてコネクターによって処理されます。例えば、コネクターは、アプリケーション・エンティティーを非アクティブ状況に更新することにより、Delete 要求を Update 要求に変換することがあります。

### パフォーマンスの考慮

コラボレーションのパフォーマンスは、サービス呼び出しの数や、サービス呼び出しによって渡されるビジネス・オブジェクトのサイズの影響を受けます。ビジネス・オブジェクトのサイズは変更できませんが、コラボレーションが作成するサービス呼び出しの数は減らすことができます。

例えば、シナリオが階層ビジネス・オブジェクト内の子ビジネス・オブジェクトに対して操作を行う必要があるとします。場合によっては、サービス呼び出しを実行する必要がある子ビジネス・オブジェクトごとにサービス呼び出しを作成する代わりに、階層オブジェクト全体を検索し、階層オブジェクトを通じてローカルに操作を繰り返す方が効率的なことがあります。

## サブダイアグラム

アクティビティー・ダイアグラムのロジックが複雑になった場合、ロジックを区分し、ロジックの個別単位を *サブダイアグラム* として使用する方が役に立つことがあります。各サブダイアグラムは、特定のメインダイアグラムに関連付けられます。

図 43 に、アクティビティー・メインダイアグラムに Retrieve サブダイアグラムと Delete サブダイアグラムという 2 つのサブダイアグラムへの参照が含まれるシナリオを示します。

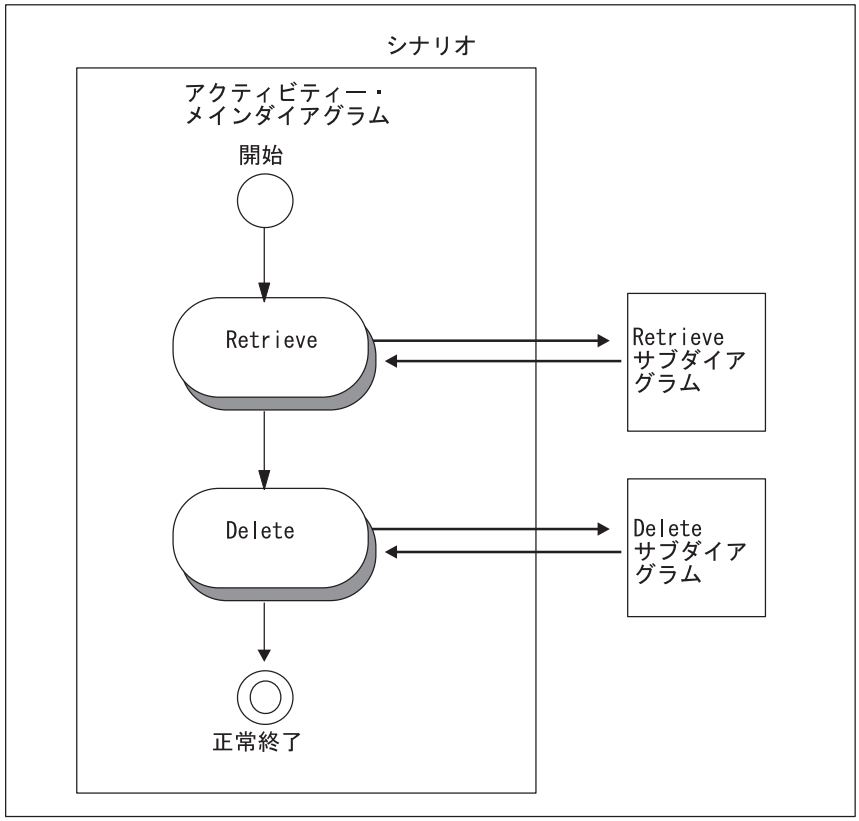



*図 43. 2 つのサブダイアグラムがあるシナリオ*

**注:** イテレーターは、サブダイアグラムの特殊な形式です。サブダイアグラムに関するすべての基本情報は、イテレーターに適用されます。特にイテレーターに関する詳細については、141 ページの『イテレーター』を参照してください。

シナリオ内のアクティビティー・ダイアグラムは、階層的に配置されています。シナリオ内のすべてのサブダイアグラムとイテレーターは、シナリオのアクティビティー・メインダイアグラムより下位にあります。図 44 にこの関係を示します。サブダイアグラム・シンボルが表示されるアクティビティー・ダイアグラムは、サブダイアグラムの*親ダイアグラム* と呼ばれます。

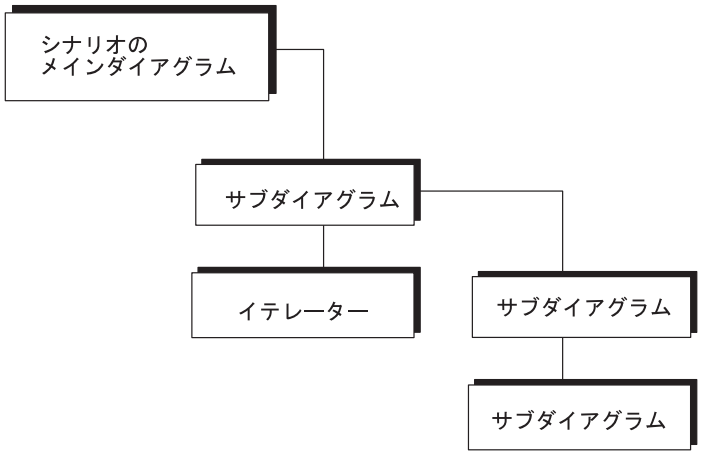



*図 44. メインダイアグラムとサブダイアグラムの関係*

サブダイアグラムは、すべてのコラボレーション・テンプレート・プロパティーおよびすべてのシナリオ変数にアクセスできます。表 39 に、サブダイアグラムとメインダイアグラムの相違点を要約します。

*表 39. メインダイアグラムとサブダイアグラムの比較*

| 相違点 | メインダイアグラム | サブダイアグラム |
|---|---|---|
| 設計時の作成方法 | シナリオの作成時に自動的に作成されます。 | 親ダイアグラムでサブダイアグラム・シンボルによって制御されます。 |
| 実行時の実行要因 | コラボレーション・ランタイム環境によってトリガー・イベントが渡されるときに実行が開始されます。 | 親ダイアグラムの実行経路がこのサブダイアグラムに到達すると実行が開始されます。つまり、トリガー・イベントはありません。 |
| 未処理例外または発生した例外の処理方法 | コラボレーション・ランタイム環境に渡します。 | 親ダイアグラムに渡します。 |
| 実行時の完了方法 | コラボレーション・ランタイム環境に戻されます。 | 親ダイアグラムに戻されます。 |

## サブダイアグラムの作成

サブダイアグラムをアクティビティー・ダイアグラムに追加する手順は、以下のとおりです。

1. シンボル・ツールバーの「サブダイアグラム」ボタンをクリックします。
2. アクティブなアクティビティー・ダイアグラム内でサブダイアグラム・シンボルを配置する位置をクリックします。

   サブダイアグラムの固有 ID が表示され、シナリオ・ツリーの階層内で親ダイアグラムの下に配置されます。シナリオ・ツリーでは、名前が以下のフォーマットで表示されます。

   (*UID*)

   サブダイアグラムにラベルを使用する場合、シナリオ・ツリーでは、名前が以下のフォーマットで表示されます。

   *label* (*UID*)

UID は、シナリオ・ツリー内のサブダイアグラム・オブジェクトの名前でもある固有 ID です。その他のシンボルの UID のように、サブダイアグラムの UID を表示するかどうかを選択できます。UID の表示のオン/オフを切り替えるには、テンプレート・ツリーのシナリオ・ノードのコンテキスト・メニューを使用します。

3. シナリオ・ツリー内のサブダイアグラム名をダブルクリックするか、ダイアグラム・エディター・ウィンドウ内のノードを右マウス・ボタンでクリックし、「サブダイアグラムを開く」を選択します。

   新しいアクティビティー・ダイアグラムを定義するための新しいウィンドウが作業域に表示されます。

アクティビティー・メインダイアグラムの場合と同じように、サブダイアグラムも開始シンボルから始まって終了成功シンボル (および必要に応じて 1 つ以上の終了障害シンボル) で終わります。サブダイアグラムには、すべてのダイアグラミング・コンポーネント (サブダイアグラムおよびイテレーターを含む) を入れることができます。

## サブダイアグラムの定義

サブダイアグラムがアクティビティー・ダイアグラムに表示されたら、「Subdiagram Properties」ダイアログでそのプロパティーを定義できます。サブダイアグラムのプロパティーは、そのラベルおよび説明です。これらのプロパティーを使用するかどうかは任意です。

サブダイアグラム・プロパティーを定義する手順は、以下のとおりです。

1. 「Subdiagram Properties」ダイアログを表示します。

   このダイアログは、以下のいずれかの方法で表示できます。
   - 選択されているサブダイアグラムをダブルクリックします。
   - 右マウス・ボタンでクリックしてコンテキスト・メニューを表示し、「プロパティー」を選択します。
   - 「編集」プルダウン・メニューから「プロパティー」を選択します。
   - キーボード・ショートカット `Ctrl + Enter` を使用します。
2. オプションで、このサブダイアグラムにラベルおよび説明を指定します。

   UID より詳しいテキストを使用してサブダイアグラムにラベルを付けることにより、アクティビティー・ダイアグラムがよりわかりやすくなります。「説明」フィールドは、サブダイアグラムの目的を示すコメント用フィールドです。
3. 「適用」をクリックしてサブダイアグラム・プロパティーを保管します。プロパティーを消去する場合は、「破棄」をクリックします。サブダイアグラム定義を取り消す場合は、「閉じる」をクリックします。

## サブダイアグラムの削除

サブダイアグラムを削除するには、親ダイアグラムまたはサブダイアグラムを表示し、以下の手順を実行します。

1. 削除するサブダイアグラムのシンボルを選択します。

2. 「編集」メニューから「削除」オプションを選択します。または、DEL (Delete) キーをクリックします。

シナリオ・ツリーで親ダイアグラムを展開すると、サブダイアグラムが削除されている場合はサブダイアグラム名が表示されません。

## サブダイアグラムの完了状況の処理

親ダイアグラムの実行は、サブダイアグラムの実行状況に対応しています。サブダイアグラムの例外時の振る舞いと完了状況を決定するのは、サブダイアグラムの開発者です。コラボレーションは以下のいずれかの手順でサブダイアグラムを意図的に終了できます。

- サブダイアグラムの実行経路の終端に、サブダイアグラム実行の正常終了を示す終了成功ノードを配置する。
- サブダイアグラムの実行経路の終端に、サブダイアグラム実行の失敗を示す終了障害ノードを配置する。

### サブダイアグラムの正常な実行の処理

終了成功の終了ノードは、実行が正常に終了していることを示します。サブダイアグラムの終端に終了成功ノードが配置されている場合には、コラボレーション・ランタイム環境によりサブダイアグラムが終了され、制御が親ダイアグラムに渡されます。親ダイアグラムのフローは、サブダイアグラム・ノードの次のノードに進みます。一般にこの次のノードは、サブダイアグラムの状況を調べる決定ノードです。この決定ノードには、サブダイアグラムの実行状況を評価する以下の分岐を設定できます。

- ブール値条件をテストする標準分岐。このブール値条件はユーザーが設定できます。

  コラボレーションが標準状態で実行されると、コラボレーション・ランタイム環境により標準分岐の条件が評価されます。true と評価される標準分岐がない場合にはデフォルトの分岐が実行されます。
- 特定の例外が発生しているかどうかを調べる例外分岐。

  コラボレーションが例外状態で実行される場合、コラボレーション・ランタイム環境により例外分岐の条件が評価されます。

サブダイアグラムが正常終了する場合、以下のいずれかの方法で終了します。

- サブダイアグラムがタスクを完了し、例外が検出されない。

  コラボレーションは標準状態で実行されます。制御が親ダイアグラムに渡され、親ダイアグラムの次のノードが決定ノードの場合、コラボレーション・ランタイム環境により標準分岐がすべて評価されます。
- サブダイアグラムは例外を検出して (例外を生成しないで) 処理し、意図的に成功で終了する。

  コラボレーションは標準状態で実行されます。制御が親ダイアグラムに渡され、親ダイアグラムの次のノードが決定ノードの場合、コラボレーション・ランタイム環境により標準分岐がすべて評価されます。

- サブダイアグラムが例外を検出し、親ダイアグラムで例外を生成してこの例外を処理し、意図的に成功で終了します。

  この場合、コラボレーションは例外状態で実行されます。制御が親ダイアグラムに渡され、親ダイアグラムの次のノードが決定ノードの場合、コラボレーション・ランタイム環境により例外分岐がすべて評価されます。この例外分岐は、例外を処理するアクション・ノードにも到達する必要があります。例外を処理する最善の方法は、この例外を再度発生させ、例外の理由として例外テキストを含めるという方法です。

  アクティビティー・ダイアグラムに複数のレベルがあるコラボレーションの場合、`raiseException()` メソッドを使用して各レベルを通して例外を生成し、未解決のフローのリストに元の例外テキストを表示する必要があります。以降の各 `raiseException()` 呼び出しは、例外を生成して元の例外テキストを渡します。例外がメインダイアグラムに到達すると、コラボレーション・ランタイム環境は、例外処理操作 (ログへの書き込みなど) を実行するか、コラボレーションがトランザクションである場合はロールバックを開始します。

**注:** 開発者は、例外が検出された場合でも、例外が処理される限り、成功で終了するサブダイアグラムを使用できます。正常終了とは、サブダイアグラムの実行が終了成功シンボルに到達し、発生したあらゆる例外をそのコードが処理できた状態だけを意味します。

決定ノードの作成方法の詳細については、 120 ページの『決定ノード』を参照してください。例外処理の実装方法の詳細については、 169 ページの『第 7 章 例外処理』を参照してください。

## サブダイアグラムの実行不成功の処理

終了障害の終了ノードは、実行が正常に終了していない ことを示します。サブダイアグラムの終端に終了障害ノードが配置されている場合には、コラボレーション・ランタイム環境によりこのサブダイアグラムとコラボレーション全体が終了されます。制御がコラボレーション・ランタイム環境に渡されます。これにより、コラボレーションのログ記録先にエントリーが記述され、未解決のフローが作成されます。

サブダイアグラムを失敗で終了する場合、以下のいずれかの方法で終了します。

- サブダイアグラムは例外を検出して処理し、意図的に失敗で終了します。この場合、例外の原因として例外テキストをサブダイアグラムに挿入する必要があります。
- サブダイアグラムは 予期されない例外 (サブダイアグラムで*処理されない* 例外) を検出して、意図的に失敗で終了します。

サブダイアグラムが終了障害ノードで終わっている場合には、コラボレーション・ランタイム環境によりコラボレーション全体 が終了されます。サブダイアグラムで検出された例外を処理する方法については、 173 ページの『サブダイアグラムまたはイテレーターの正常終了』を参照してください。コラボレーション・ランタイム環境による未解決のフローの作成方法については、 172 ページの『メインダイアグラムの正常終了』を参照してください。

## イテレーター

イテレーターは、ループまたは反復を実装するサブダイアグラムの特殊な形式です。イテレーターを使用して、以下に対して操作を実行します。

- ビジネス・オブジェクトのすべての属性
- ビジネス・オブジェクト配列のすべての要素

イテレーターをループとして使用することもできます。ループの初期化、テスト、増分に必要な値を指定するには、「Iterator Properties」ダイアログ・ボックスを使用します。

イテレーター・ダイアグラムは、サブダイアグラムまたはその他のイテレーターを呼び出すことができます。階層ビジネス・オブジェクトまたは階層ビジネス・オブジェクト配列を処理するには、イテレーターの階層が必要です。

親ダイアログの実行がイテレーター・シンボルに到達すると、制御がイテレーター・アクティビティー・ダイアグラムに渡されます。コラボレーションは、ビジネス・オブジェクトの各属性またはビジネス・オブジェクト配列内の各ビジネス・オブジェクトに対してイテレーター・ダイアグラムの実行を繰り返します。「Iterator Properties」ダイアログに指定されているイテレーター変数を介して、現在イテレーター内にある項目にアクセスできます。

イテレーターの実行が完了すると、制御は親ダイアグラムに渡されます。

### イテレーターの作成

イテレーターをアクティビティー・ダイアグラムに追加する手順は、以下のとおりです。

1. シンボル・ツールバーの「イテレーター」ボタンをクリックします。
2. ワークスペース内の任意の位置をクリックしてイテレーター・シンボルを配置します。

   シナリオ・ツリーで、イテレーターの固有 ID が親ダイアグラムの下に階層的に配置された状態で表示されます。シナリオ・ツリーでは、名前が以下のフォーマットで表示されます。

   (*UID*)

   イテレーターにラベルを使用する場合、シナリオ・ツリーでは、名前が以下のフォーマットで表示されます。

   *label* (*UID*)

   UID は、シナリオ・ツリー内のイテレーター・オブジェクトの名前でもある固有 ID です。その他のシンボルの UID のように、イテレーターの UID を表示するかどうかを選択できます。UID の表示のオン/オフを切り替えるには、テンプレート・ツリーのシナリオ・ノードのコンテキスト・メニューを使用します。

### イテレーター変数の作成

ビジネス・オブジェクトの配列またはビジネス・オブジェクトの属性を処理するイテレーターには、反復で処理される項目を保持する変数が設定されている必要があ

ります。このイテレーター変数は、実際に「シナリオ定義」ダイアログ・ボックスで作成され、初期化されるシナリオ変数です。イテレーターのプロパティーを定義する前にイテレーター変数を作成してください。

イテレーターがビジネス・オブジェクトの属性を処理する場合は、Object を使用して現在の属性を保持できます。例えば、以下のように宣言できます。

```
Object iterAttr = null;
```

イテレーターが配列内のビジネス・オブジェクトを処理する場合は、タイプ BusObj の変数を使用して現在のビジネス・オブジェクトを保持できます。以下に例を示します。

```
BusObj iterBusObj = new BusObj("LineItem");
```

イテレーターがループとして使用される場合には、イテレーター変数を作成する必要はありません。処理中にシステムによりイテレーター変数が自動的に作成されます。ループ索引変数は getCurrentLoopIndex() API を使用して検索できます。

## イテレーターの定義

イテレーターがアクティビティー・ダイアグラムに表示されたら、「Iterator Properties」ダイアログ・ボックスでそのプロパティーを定義できます (図 45 を参照)。

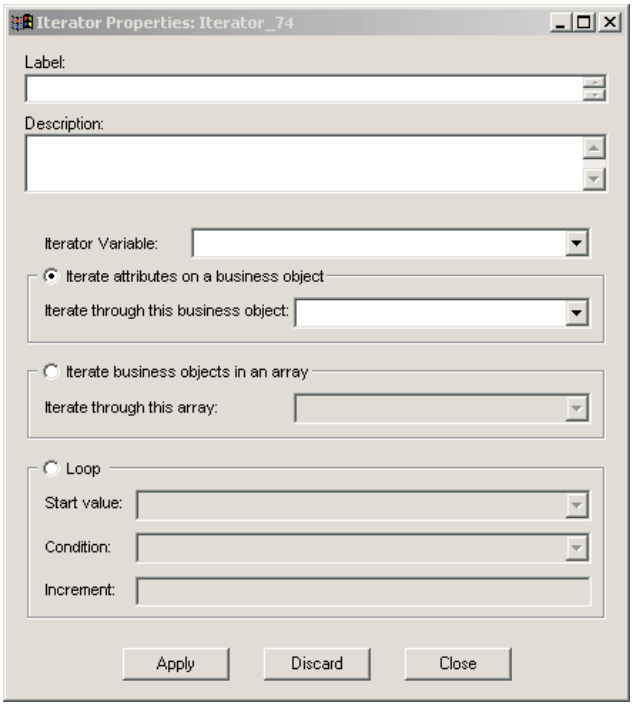



*図 45. 「Iterator Properties」ダイアログ・ボックス*

「Iterator Properties」ダイアログ・ボックスを開くには、イテレーターを右マウス・ボタンでクリックし、コンテキスト・メニューから「プロパティー」を選択します。イテレーターのラベルを定義し、説明を入力できます。ラベル定義と説明入力のプロパティーはオプションです。使用するイテレーターのタイプによっては、他

のプロパティーも定義する必要があります。以降のセクションでは、各タイプのイテレーターを定義する際の要件について説明します。

## ビジネス・オブジェクトの属性に対するイテレーターの使用

ビジネス・オブジェクトの属性を繰り返し操作するには、以下の手順を実行します。

1. 「イテレーター変数」フィールドに、反復で処理する項目を保持するための変数を指定します。このフィールドのドロップダウン・リストには、テンプレート変数とシナリオ変数がすべて表示されます。リストから変数を選択するか、またはフィールドに変数名を直接入力できます。
2. 「ビジネス・オブジェクトの属性について繰り返す」ラジオ・ボタンをクリックします。
3. 「このビジネス・オブジェクト全体について繰り返す」フィールドに、属性を繰り返し操作するビジネス・オブジェクトを指定します。このフィールドに名前を直接入力するか、またはドロップダウン・リストからビジネス・オブジェクトを選択します。
4. 「適用」をクリックします。

イテレーターのプロパティーを定義したら、次にそのアクティビティー・ダイアグラムを編集し、処理対象の各属性に対するイテレーターの処理内容を定義する必要があります。イテレーターのアクティビティー・ダイアグラムを開くには、Process Designer Express のテンプレート・ツリー・ビューでイテレーター名をダブルクリックします。

## 配列のビジネス・オブジェクトに対するイテレーターの使用

配列のビジネス・オブジェクトを繰り返し操作するには、以下の手順を実行します。

1. 「イテレーター変数」フィールドに、反復で処理する項目を保持するための変数を指定します。このフィールドのドロップダウン・リストには、テンプレート変数とシナリオ変数がすべて表示されます。リストから変数を選択するか、またはフィールドに変数名を直接入力できます。
2. 「配列のビジネス・オブジェクトについて繰り返す」ラジオ・ボタンをクリックします。
3. 「この配列全体について繰り返す」フィールドに、繰り返し操作する配列を指定します。このフィールドに名前を直接入力するか、またはドロップダウン・リストから配列を選択します。

   フィールドに値を直接入力する場合には以下の構文を使用してください。

   *BusinessObjectVariable.AttributeName*

   *BusinessObjectVariable* は親ビジネス・オブジェクトの名前、*AttributeName* は子ビジネス・オブジェクトの配列を示す属性の名前です。

   例えば、変数 order に含まれるビジネス・オブジェクトの Item 属性によって示される配列のビジネス・オブジェクトを繰り返すには、「この配列全体について繰り返す」フィールドに `order.Items` と入力します。
4. 「適用」をクリックします。

イテレーターのプロパティーを定義したら、次にそのアクティビティー・ダイアグラムを編集し、処理対象の各ビジネス・オブジェクトに対するイテレーターの処理内容を定義する必要があります。イテレーターのアクティビティー・ダイアグラムを開くには、Process Designer Express のテンプレート・ツリー・ビューでイテレーター名をダブルクリックします。

### ループとしてのイテレーターの使用

ループを定義する手順は、以下のとおりです。

1. 「ループ」ラジオ・ボタンをクリックします。
2. 「開始値」フィールドに、カウンター変数の初期値を指定します。値を保持する変数をドロップダウン・リストから選択するか、または値をこのフィールドに直接入力できます。
3. 「条件」フィールドに、ループ実行条件を指定します。この条件が true の場合にループが実行されます。ドロップダウン・リストには、テンプレートで定義されているブール値変数がすべて表示されます。使用する変数を選択するか、またはフィールドに条件を直接入力します。
4. 「増分」フィールドに、カウンター変数の値の増分方法を指定します。
5. 「適用」をクリックします。

イテレーターのプロパティーを定義したら、次にそのアクティビティー・ダイアグラムを編集し、処理対象の各属性または各ビジネス・オブジェクトに対するイテレーターの処理内容を定義する必要があります。イテレーターのアクティビティー・ダイアグラムを開くには、Process Designer Express のテンプレート・ツリー・ビューでイテレーター名をダブルクリックします。

## ブレークの追加

イテレーターのアクティビティー・ダイアグラムにブレークを追加すると、反復が強制的に早期終了されます。イテレーターの実行経路がブレーク・シンボルに到達すると、イテレーターが終了し、制御が親ダイアグラムに渡されます。ブレークはすべてのタイプのイテレーターに使用できます。

イテレーターのアクティビティー・ダイアグラムに、以下のようにブレーク・シンボルを配置します。

1. シンボル・ツールバーの「Break」ボタンをクリックします。
2. アクティビティー・ダイアグラムでブレークを配置する位置にカーソルを合わせ、クリックします。ブレーク・シンボルがダイアグラムに追加されます。

*図 46. ブレーク・シンボル*

必要に応じて、ブレーク・シンボルにラベルと説明を追加できます。「Break Properties」ダイアログ・ボックスを表示してプロパティーを編集するには、アクティビティー・ダイアグラムのブレーク・シンボルをダブルクリックします。

## シンボル・ツールバーのその他の機能の使用

ダイアグラム・シンボル・ツールバーには「選択」ボタンと「テキスト」ボタンがあります。「テキスト」ボタンを使用してアクティビティー・ダイアグラムにテキスト・ボックスを追加します。この手順については『テキスト・ボックス機能の使用』を参照してください。操作を取り消すには「選択」ボタンを使用します。この手順については『操作の取り消し』を参照してください。

### テキスト・ボックス機能の使用

アクティビティー・ダイアグラムにテキスト・ボックスを挿入し、このボックスにテキストを入力したり、既存のテキストを編集できます。テキストをアクティビティー・ダイアグラムに追加する手順は、以下のとおりです。

1. シンボル・ツールバーの「テキスト」ボタンをクリックします。
2. ワークスペース内の任意の位置をクリックしてテキスト・シンボルを配置します。

   テキスト・ボックス内に「テキスト」という単語が表示されます。
3. 単語「テキスト」をダブルクリックし、必要なテキストを入力します。

テキスト・ボックスは、選択してダイアグラム内の別の位置にドラッグできます。

### 操作の取り消し

アクティビティー・ダイアグラムに特定のシンボルを挿入する必要がなくなった場合は、この操作を取り消すことができます。ダイアグラム・シンボル・ツールバーのボタンをクリックした後にこの操作を取り消すには、シンボル・ツールバーの「選択」ボタンをクリックします。

遷移リンクを取り消すには、Esc キーを使用します。Esc キーを押すたびに、遷移リンクの最後の線分が元に戻されます。

## コラボレーション構成プロパティーの値の取得

通常、アクション・ノードおよび決定ノードは、実装者または開発者がコラボレーション・オブジェクトの構成プロパティーに対して設定する値を取得して評価する必要があります。コラボレーション・オブジェクトは、その構成プロパティーをコラボレーション・テンプレートから継承します。これらのコラボレーション・プロパティーの設定方法の詳細については、91 ページの『コラボレーション構成プロパティーの定義 (「プロパティー」タブ)』を参照してください。

プロパティーの値を取得するため、シナリオは getConfigProperty() メソッドを呼び出します。プロパティーが値のリストである場合、シナリオは getConfigPropertyArray() を呼び出します。詳細については、208 ページの『コラボレーション構成プロパティーの検索』を参照してください。

## トランザクション機能の使用

**トランザクション・コラボレーション**は、その実行を停止するエラーを検出すると、ロールバックします。ロールバックが正常に行われるには、各シナリオのサブトランザクション・ステップごとに差し戻しが指定されている必要があります。コラボレーションのトランザクション・プロパティーを設定するには、表 40 に説明されている手順を実行する必要があります。

*表 40. トランザクション・コラボレーションの定義*

| ステップの定義 | 説明 | 詳細 |
|---|---|---|
| コラボレーション・テンプレートに対する最小トランザクション・レベルの割り当て | トランザクション・コラボレーションは、最小トランザクション・レベルを使用して、トランザクション・ロールバックを実行するタイミングを決定します。 | 83 ページの『最小トランザクション・レベルの指定』 |
| サブトランザクション・ステップに対する差し戻しの指定 | トランザクション・コラボレーションは、サブトランザクションに定義されている差し戻しを使用して、実際のロールバックを実行します。 | 131 ページの『差し戻しの定義』 |

トランザクション・コラボレーションの詳細については、「システム・インプリメンテーション・ガイド」を参照してください。

## 実行経路の終了

アクティビティー・ダイアグラムの各実行経路は、成功と失敗のいずれかで終了する必要があります。

### 成功での終了

コラボレーションの成功での終了とは、アクティビティー・ダイアグラムの実行経路によってトリガー・イベントが正常に処理される状況を意味します。つまり、すべての実行が成功で終了したか、または実行時にアクティビティー・ダイアグラムで例外が発生し、このコラボレーションが成功で終了する方法でこの例外が処理されています。詳細については、169 ページの『第 7 章 例外処理』を参照してください。

成功での終了を示すには、実行経路の終端に終了成功シンボルを配置します。コラボレーション・ランタイム環境で終了成功が実行されると、現在の実行経路が終了され、上位の実行レベル (親ダイアグラム) がある場合にはこの実行レベルへ制御が渡されます。サブダイアグラムまたはイテレーターが終了成功ノードに到達すると、制御が親ダイアグラムに渡されます。アクティビティー・メインダイアグラムが終了成功ノードに到達すると、コラボレーション・ランタイム環境に制御が渡され、環境固有のエラー処理アクションが実行されます。

#### 終了成功シンボルの追加

実行経路を成功で終了するには、アクティビティー・ダイアグラムに終了成功シンボルを配置し、これを接続します。終了成功シンボルをアクティビティー・ダイアグラムに追加する手順は、以下のとおりです。

1. ダイアグラム・シンボル・ツールバーの「終了成功」ボタンをクリックします。
2. ワークスペース内の任意の位置をクリックして終了成功シンボルを配置します。

### 終了成功シンボルの定義

終了成功シンボルがアクティビティー・ダイアグラムに表示されたら、「End Success Properties」ダイアログでシンボルのプロパティーを定義できます。終了成功シンボルのプロパティーは、そのラベルおよび説明です。これらのプロパティーを使用するかどうかは任意です。

終了成功プロパティーを定義する手順は、以下のとおりです。

1. 「End Success Properties」ダイアログを表示します。

   このダイアログは、以下のいずれかの方法で表示できます。
   - 選択されている終了成功ノードをダブルクリックします。
   - 右マウス・ボタンでクリックしてコンテキスト・メニューを表示し、「プロパティー」を選択します。
   - 「編集」プルダウン・メニューから「プロパティー」を選択します。
   - キーボード・ショートカット `Ctrl + Enter` を使用します。

   「End Success Properties」ダイアログが表示されます。
2. オプションで、このアクション・ノードにラベルおよび説明を指定します。

   UID より詳しいテキストを使用して終了成功シンボルにラベルを付けることにより、アクティビティー・ダイアグラムがよりわかりやすくなります。「説明」フィールドは、終了成功シンボルの目的を示すコメント用フィールドです。
3. 終了成功プロパティーを保管する場合は、「適用」をクリックします。プロパティーを消去する場合は、「破棄」をクリックします。この終了成功定義を取り消す場合は、「閉じる」をクリックします。

## 失敗での終了

コラボレーションが失敗で終了する場合、アクティビティー・ダイアグラムが適切に実行できなくなるので、実行を停止する必要があります。失敗での終了を示すには、終了障害シンボルを実行経路の終端に配置します。コラボレーション・ランタイム環境は終了障害の実行時に、コラボレーション全体を終了します。サブダイアグラムまたはイテレーターが終了障害ノードに到達すると、サブダイアグラムまたはイテレーターと、すべての親ダイアグラムが終了します。コラボレーション・ランタイム環境は、固有のエラー処理アクションを実行します。

**注:** 終了障害シンボルへ到達すると、*常に* コラボレーション実行が停止します。ただし、終了障害での終了により、トリガー・イベントが自動的に失敗するわけではありません。コラボレーションが例外状態で実行されている場合にのみ、コラボレーション・ランタイム環境によりトリガー・イベントの未解決のフローが作成されます。詳細については、169 ページの『第 7 章 例外処理』を参照してください。

### 終了障害シンボルの追加

実行経路を失敗で終了するには、アクティビティー・ダイアグラムに終了障害シンボルを配置してこれを接続します。終了障害シンボルをアクティビティー・ダイアグラムに追加する手順は、以下のとおりです。

1. ダイアグラム・シンボル・ツールバーの「終了障害」ボタンをクリックします。

2. ワークスペース内の任意の位置をクリックして終了障害シンボルを配置します。

### 終了障害シンボルの定義

終了障害シンボルがアクティビティー・ダイアグラムに表示されたら、「End Failure Properties」ダイアログでシンボルのプロパティーを定義できます。終了障害シンボルのプロパティーは、そのラベルと説明です。これらのプロパティーを使用するかどうかは任意です。

終了障害プロパティーを定義する手順は、以下のとおりです。

1. 「End Failure Properties」ダイアログを表示します。

   このダイアログは、以下のいずれかの方法で表示できます。
   - 選択されている終了障害ノードをダブルクリックします。
   - 右マウス・ボタンでクリックしてコンテキスト・メニューを表示し、「プロパティー」を選択します。
   - 「編集」プルダウン・メニューから「プロパティー」を選択します。
   - キーボード・ショートカット `Ctrl + Enter` を使用します。
2. オプションで、このアクション・ノードにラベルおよび説明を指定します。

   UID より詳しいテキストを使用して終了障害シンボルにラベルを付けることにより、アクティビティー・ダイアグラムがよりわかりやすくなります。「説明」フィールドは、終了障害シンボルの目的を示すコメント用フィールドです。
3. 終了障害プロパティーを保管する場合は、「適用」をクリックします。プロパティーを消去する場合は、「破棄」をクリックします。この終了障害定義を取り消す場合は、「閉じる」をクリックします。

## その他のアクティビティー・ダイアグラム操作

このセクションでは、Process Designer Express のアクティビティー・ダイアグラムで使用可能な以下の追加操作について説明します。

- 『アクティビティー・ダイアグラムの開閉』
- 149 ページの『アクティビティー・ダイアグラムの文書化』
- 150 ページの『アクティビティー・ダイアグラムのコピー』
- 150 ページの『アクティビティー・ダイアグラム内での削除』

## アクティビティー・ダイアグラムの開閉

Process Designer Express には、アクティビティー・ダイアグラムを開くまたは閉じるための機能が用意されています。

### アクティビティー・ダイアグラムを開く

アクティビティー・ダイアグラムを開くには、以下のいずれかを行うことができます。

- 「テンプレート」メニューから「すべてのダイアグラムを開く」を選択します。

  「すべてのダイアグラムを開く」を選択すると、テンプレートに定義されているアクティビティー・ダイアグラムごとにウィンドウが表示されます。

- シナリオ・ツリーでシナリオ、サブダイアグラム、またはイテレーターを選択します。
  - 「テンプレート」メニューから「ダイアグラムを開く」を選択し、関連付けられているアクティビティー・ダイアグラムのウィンドウを開きます。
  - シナリオ、サブダイアグラム、またはイテレーターの名前をダブルクリックします。
- アクティビティー・ダイアグラムがサブダイアグラムまたはイテレーターである場合は、ワークスペースを右マウス・ボタンをクリックしてコンテキスト・メニューを表示してから「親ダイアグラムを開く」を選択することにより、親ダイアグラムを開くことができます。

### アクティビティー・ダイアグラムを閉じる

アクティビティー・ダイアグラムを閉じるには、以下のいずれかを行うことができます。

- 「テンプレート」メニューから「すべてのダイアグラムを閉じる」を選択します。

  少なくとも 1 つのアクティビティー・ダイアグラムが開いている場合、「すべてのダイアグラムを閉じる」を選択すると、すべてのアクティビティー・ダイアグラムのウィンドウが閉じられます。
- シナリオ、サブダイアグラム、またはイテレーターが表示されているウィンドウの上部にある「閉じる」ボタンを選択します。

## アクティビティー・ダイアグラムの文書化

アクティビティー・ダイアグラムを文書化するには、以下の 2 つの方法があります。

- ダイアグラム、サブダイアグラム、またはイテレーターのグラフィック表示を印刷します。
- アクティビティー・ダイアグラム全体をテキスト・ファイルに保管します。

### ダイアグラムの印刷

各ダイアグラム、サブダイアグラム、およびイテレーターのグラフィック表示を印刷できます。表記が長い場合は、印刷は複数のページに渡ります。印刷の手順は、以下のとおりです。

1. ダイアグラムを開きます。これを行うには、テンプレート・ツリー・ビューでダイアグラムを選択し、その名前を右マウス・ボタンでクリックしてコンテキスト・メニューから「ダイアグラムを開く」をクリックするか、「テンプレート」メニューから「ダイアグラムを開く」を選択します。
2. 「ファイル」メニューから「印刷」を選択するか、ショートカット・キーの組み合わせ `Ctrl+P` を使用します。

### テキスト・ファイルとしてのダイアグラムの保管

アクティビティー・ダイアグラム全体をテキスト・ファイルとして保管できます。この手順は、以下のとおりです。

1. 「テンプレート」メニューから「ダイアグラムをテキストとして保管...」を選択します。

2. 「ダイアグラムをテキストとして保管」ダイアログ・ボックスでファイル名を指定します。

## アクティビティー・ダイアグラムのコピー

アクティビティー・ダイアグラム全体をシナリオと分けてコピーすることはできませんが、アクティビティー・ダイアグラムの内容をコピーすることはできます。アクティビティー・ダイアグラムの内容をコピーする手順は、以下のとおりです。

1. ソース・アクティビティー・ダイアグラムを表示します。
2. コピーするオブジェクトを選択します。
   - 一部のオブジェクトを選択するには、左マウス・ボタンを押したままマウスを移動し、コピー対象の長方形領域を定義します。領域が決まったらマウス・ボタンを放します。
   - 特定のオブジェクトを 1 つずつ選択するには、1 つのオブジェクトをクリックしてから Shift キーを押したまま他のオブジェクトをクリックします。Shift キーを押したまま、選択対象から外すオブジェクトをクリックします。
   - すべてのオブジェクトを選択するには、「編集」メニューから「すべて選択」(Ctrl+A ショートカット) を選択します。
3. 「編集」メニューから「コピー」(Ctrl+C ショートカット) を選択します。
4. 宛先アクティビティー・ダイアグラムを表示します。
5. 「編集」メニューから「貼り付け」(Ctrl+V ショートカット) を選択します。
6. 必要に応じて、プロパティー、ポート、および変数に対する参照が正しくなるようにシンボルの定義を変更します。

コピー作業は反復プロセスではありません。サブダイアグラムまたはイテレーターをコピーして貼り付けると、このサブダイアグラムまたはイテレーター・ダイアグラムを示す空のアクティビティー・ダイアグラム (現在のダイアグラムの従属ダイアグラム) がシナリオ・ツリーに作成されます。サブダイアグラムおよびイテレーター・ダイアグラムの内容は、手動でコピーおよび貼り付ける必要があります。コピー操作時にイテレーター・ノードでブレーク・シンボルが選択された場合には、別のイテレーター・ノードにのみブレーク・シンボルを貼り付けることができます。

**要確認:** 重複した開始シンボルをアクティビティー・ダイアグラムに貼り付けることはできません。コピーおよび貼り付けのシンボルを選択する場合、すでに開始シンボルがあるダイアグラムに開始シンボルをコピーしないでください。ダイアグラムの「ユーザー設定」のデフォルト値をそのまま使用すると、開始シンボルは新しいダイアグラムごとに自動的に配置されます。詳細については、192 ページの『表示の変更: ユーザー設定』を参照してください。

## アクティビティー・ダイアグラム内での削除

アクティビティー・ダイアグラム内のシンボルを削除するには、削除対象のシンボルを選択し、以下のいずれかの作業を行います。

- 「編集」メニューから「削除」を選択します。
- DEL (Delete) キーを押します。

アクティビティー・ダイアグラム全体を削除するには、アクティビティー・ダイアグラムが定義されているシナリオを削除します。詳細については、101 ページの『シナリオの削除』を参照してください。サブダイアグラムまたはイテレーターがあるシナリオの場合、親ダイアグラムからそのシンボルを削除することにより、このサブダイアグラムまたはイテレーターを削除できます。

# 第 6 章 Activity Editor の使用

Activity Editor は、Java コードを記述せずにコラボレーションのビジネス・ロジックを指定できるグラフィカル・インターフェースです。この章では、Activity Editor インターフェースおよびアクティビティー定義のコンポーネントについて説明し、ビジネス・ロジック作成でサポートされる機能ブロックのリストを示し、Activity Editor の使用例を紹介します。この章の内容は、以下のとおりです。



## Activity Editor の開始

コラボレーション・テンプレート・シナリオのアクション・ノード内から、以下の手順で Activity Editor を起動します。

1. ビジネス・ロジックを追加するアクション・ノードを選択します。
2. 以下のいずれかの方法でアクション・ノードの「Action Properties」ダイアログ・ボックスを開きます。
   - アクション・ノードをダブルクリックします。
   - アクション・ノードを右マウス・ボタンでクリックし、ポップアップ表示されたコンテキスト・メニューから「プロパティー」をクリックします。
3. 「編集」をクリックします。新規ウィンドウで Activity Editor が開きます。

## Activity Editor インターフェース

154 ページの図 47 に、Activity Editor の標準画面を示します。

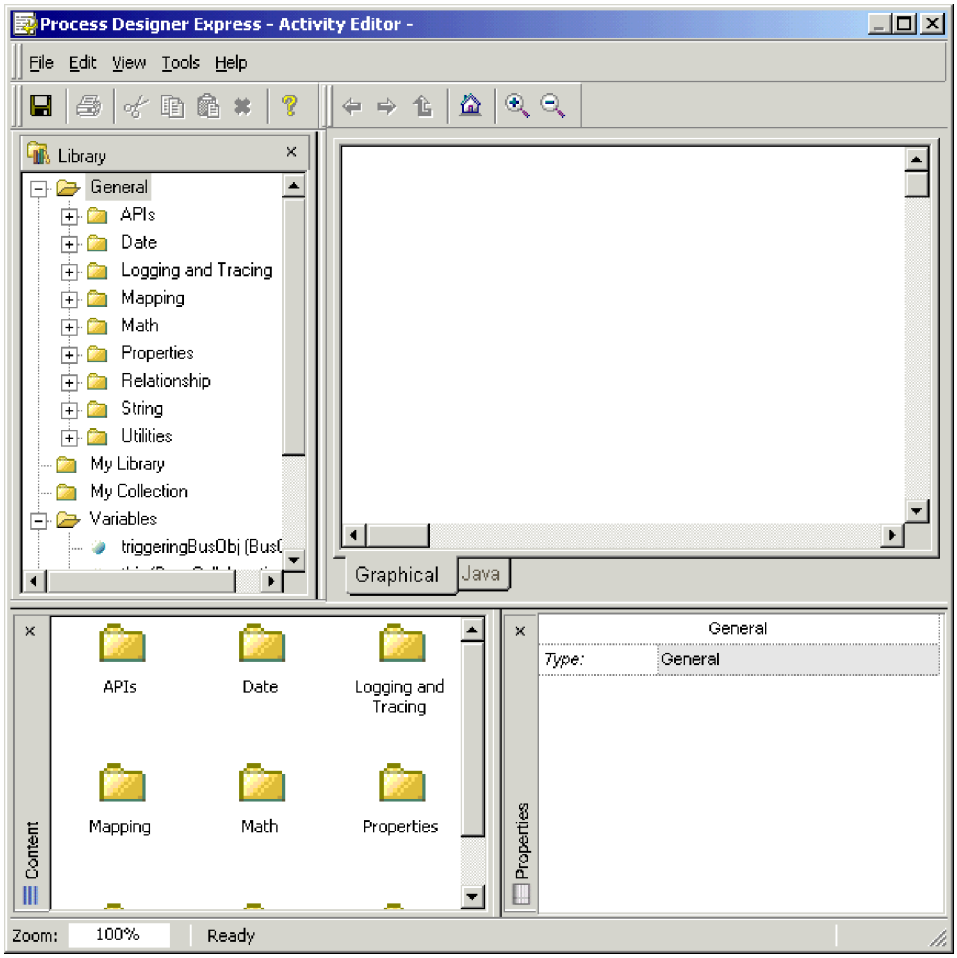



*図 47. Activity Editor*

Activity Editor のグラフィック表示には、Activity Workbook ウィンドウ、ライブラリー・ウィンドウ、コンテンツ・ウィンドウ、およびプロパティー・ウィンドウという 4 つのメイン・ウィンドウがあります。

- Activity Workbook ウィンドウ: このウィンドウ (通常は、編集キャンバスと呼ばれる) 上で、アクション・ノードのビジネス・ロジックを構成する機能ブロックをドラッグ・アンド・ドロップします。
- ライブラリー・ウィンドウ: このウィンドウには、使用可能な機能ブロックのツリー表示が含まれます。オプションで、名前付きのグループも表示されます。機能ブロックは、用途別にフォルダーに分類されています (詳細は、 163 ページの『サポートされる機能ブロック』を参照してください)。さらに、このウィンドウには以下のフォルダーがあります。
  - 「System」: このフォルダーには、編集キャンバスに追加可能なシステム要素が含まれます。システム要素には、コメント、記述、ラベル、処理待ちタグ、および定数があります。(これらのコンポーネントの使用については、 161 ページの『New Constant 機能ブロック』および 162 ページの『アクティビティー定義のタグ』を参照してください。)
  - 「Library」: このフォルダーを使用すれば、ライブラリー・ウィンドウをカスタマイズできます。このフォルダーには、System Manager の「Activity

Settings」ビューで指定されたユーザー定義の機能ブロックが含まれます。このフォルダーには、System Manager からエクスポートされた Web サービス機能ブロックも保管されています。

  - 「My Collection」: このフォルダーを使用すれば、頻繁に使用するコンポーネントのコレクションを作成できます。いつも使用する機能ブロックをこのフォルダーに置くことも、あるいは再使用可能なコンポーネント・グループを独自に作成することもできます (詳細は、 162 ページの『コンポーネント・グループ』を参照してください)。
  - 「Variables」: このフォルダーには、現在のアクティビティーで使用可能なグローバル変数が含まれます。一般的には、ポートのビジネス・オブジェクト変数や、シナリオに定義されたその他のすべてのビジネス・オブジェクトおよび変数が入っています。

- コンテンツ・ウィンドウ: このウィンドウには、ライブラリー・ウィンドウで現在選択されているフォルダー内で使用可能な機能ブロックの大アイコンのリストが表示されます。機能ブロックを選択すると、その説明とプロパティーがプロパティー・ウィンドウに表示されます。また、機能ブロック・アイコンを編集キャンバスにドラッグ・アンド・ドロップすれば、アクティビティー・フローの一部を作成できます。
- プロパティー・ウィンドウ: このウィンドウには、現在選択されている機能ブロックのプロパティーが表示されます。プロパティーは、このウィンドウにグリッドで表示されます。一部のプロパティーはプロパティー・ウィンドウで直接編集できますが、その他のプロパティーは読み取り専用です。

以降のセクションでは、Activity Editor インターフェースの一部であるビュー、メニュー、ツールバー、およびキーボード・ショートカットについて説明します。

## Activity Editor の表示方法

Activity Editor には、グラフィック表示と Java 表示という 2 つの表示方法があります。 154 ページの図 47 に示されるようなグラフィック表示により、機能ブロックをドラッグ・アンド・ドロップして、アクティビティー定義を作成できます。

Java 表示を使用すると、アクティビティー定義に従来の Java コードを追加できます。グラフィックで表現する代わりにコードを記述する場合、またはグラフィック表示の機能ブロックにはアクティビティー定義のための十分な機能が備わっていない場合は、この表示を使用します。グラフィック表示でアクティビティー定義を作成した場合は、Java 表示にグラフィック機能ブロックに相当する Java コードが表示されます。

**要確認**

Java 表示でコードを入力または変更すると、グラフィック表示は使用できなくなります。アクティビティー定義に関するその他の作業すべては、Activity Editor の Java 表示で実行するか、「Action Properties」ダイアログ・ボックスの「コード・フラグメント」ウィンドウで直接実行する必要があります。

# Activity Editor のメニュー

このセクションでは、Activity Editor のメニューから使用できる機能について説明します。

## 「ファイル」メニュー

Activity Editor の「ファイル」メニューには、以下のオプションがあります。

- **「保管」**—アクティビティーを Process Designer Express に保管します。
- **「印刷設定」**—印刷オプションを指定する「印刷設定」ダイアログ・ボックスを開きます。
- **「印刷プレビュー」**—エディターを印刷プレビュー・モードに切り替えます。このメニュー・オプションは、Java 表示でのみ使用可能です。
- **「印刷」**—現在のアクティビティーを印刷する「印刷」ダイアログ・ボックスを開きます。このメニュー・オプションは、Java 表示でのみ使用可能です。
- **「閉じる」**—Activity Editor を閉じます。

## 「編集」メニュー

Activity Editor の「編集」メニューには、以下のオプションがあります。

- **「元に戻す」**—Java コードに対する最後の変更を取り消し、その前の状態に復元します。このメニュー・オプションは、Java 表示でのみ使用可能です。
- **「やり直し」**—**「元に戻す」**コマンドによって以前に削除された変更を復元します。このメニュー・オプションは、Java 表示でのみ使用可能です。
- **「切り取り」**—選択した項目を削除します。この項目は、クリップボードにコピーされます。
- **「コピー」**—選択した項目をクリップボードにコピーします。
- **「貼り付け」**—現在クリップボード内にあるオブジェクトを、カーソルが置かれている場所にあるアクティビティーに貼り付けます。
- **「削除」**—選択したオブジェクトを削除します。
- **「すべて選択」**—アクティビティー内のオブジェクトをすべて選択します。
- **「検索」**—指定されたテキストを編集域で検索します。このメニュー・オプションは、Java 表示でのみ使用可能です。
- **「置換」**—指定されたテキストを検索し、指定された新しいテキストで置き換えます。このメニュー・オプションは、Java 表示でのみ、また「ツール」—>「コードを編集」を選択した後でのみ使用可能です。
- **「行に移動」**—指定された行にカーソルを移動します。このメニュー・オプションは、Java 表示でのみ使用可能です。

## 「表示」メニュー

Activity Editor の「表示」メニューには、以下のオプションがあります。

- **「デザイン・モード」**—「デザイン・モード」と「クイック表示モード」を切り替えます。
- **「クイック表示モード」**—「クイック表示モード」と「デザイン・モード」を切り替えます。
- **「移動」**—アクティビティー内をナビゲートするために以下のサブメニューを開きます。

  - **「戻る」**—ナビゲーション・ヒストリーを後方に移動します。
  - **「進む」**—ナビゲーション・ヒストリーを前方に移動します。
  - **「1 レベル上へ」**—ダイアグラムを 1 レベル上から表示します。
  - **「ホーム」**—グラフィック表示のトップレベルのダイアグラムに移動します。
- **「ズームイン」**—エディターの内容を拡大します。
- **「ズームアウト」**—エディターの内容を最小化します。
- **「倍率指定ズーム」**—特定のズーム・レベルを指定する「ズーム」ダイアログ・ボックスを開きます。
- **「ライブラリー・ウィンドウ」**—ライブラリー・ウィンドウのオン/オフを切り替えます。
- **「コンテンツ・ウィンドウ」**—コンテンツ・ウィンドウのオン/オフを切り替えます。
- **「プロパティー・ウィンドウ」**—プロパティー・ウィンドウのオン/オフを切り替えます。
- **「ツールバー」**—ツールバーを表示および終了するために以下のサブメニューを開きます。
  - **「標準」**—標準ツールバーのオン/オフを切り替えます。
  - **「グラフィックス」**—グラフィックス・ツールバーのオン/オフを切り替えます。
  - **「Java」**—Java ツールバーのオン/オフを切り替えます。
- **「ステータス・バー」**—ステータス・バーのオン/オフを切り替えます。
- **「設定」**—Activity Editor のデフォルトの振る舞いを指定する「設定」ダイアログ・ボックスを開きます。

## 「ツール」メニュー

Activity Editor の「ツール」メニューには、以下のオプションがあります。

- **「変換」**—現在のアクティビティーを Java コードに変換して、Java 表示を開きます。このウィンドウに表示される Java コードは、「ツール」—>「コードを編集」をクリックするまで読み取り専用です。
- **「コードを編集」**—Java 表示でのコードの編集を使用可能にします。このオプションが選択されるまで、表示されるコードは読み取り専用です。このメニュー・オプションは、Java 表示でのみ使用可能です。
- **「対応しない区切り文字を検査」**—Java コード内の対応しない区切り文字を検出します。このメニュー・オプションは、Java 表示でのみ使用可能です。
- **「式ビルダー」**—式ビルダー・ツールを起動します。

## 「ヘルプ」メニュー

Activity Editor の「ヘルプ」メニューには、以下のオプションがあります。

- **「ヘルプ・トピック」**—ヘルプ・トピックを開きます。
- **「文書」**—WebSphere Business Integration Express 文書を開きます。

### 「コンテキスト」メニュー

Activity Editor の「コンテキスト」メニューにアクセスするには、編集キャンバス上で右マウス・ボタンをクリックします。このメニューには、以下のオプションがあります。

- **「新規定数」**—新しい定数コンポーネントを編集キャンバス上に作成します。
- **「ラベルを追加」**—新しいラベル・コンポーネントを編集キャンバス上に作成します。
- **「記述を追加」**—新しい記述コンポーネントを編集キャンバス上に作成します。
- **「コメントを追加」**—新しいコメント・コンポーネントをキャンバス上に作成します。
- **「予定を追加」**—新しいメモ・コンポーネントをアクティビティーに作成します。
- **「ユーザー・コレクションに追加」**—再利用のための新しいコンポーネント・グループをライブラリー・ウィンドウに作成します。

## Activity Editor のツールバー

Activity Editor には、標準ツールバー、グラフィックス・ツールバー、および Java ツールバーの 3 つのツールバーがあります。

標準ツールバーには、アクティビティーの保管と印刷、アクティビティー内の要素の切り取り、コピー、貼り付け、削除、およびヘルプ表示の機能があります。

グラフィックス・ツールバーには、アクティビティー内をナビゲートする機能があります。このボタンは、「表示」 —> 「ズーム」、および「表示」 —> 「移動」メニュー項目に相当します。

Java ツールバーには、Java コードを編集するための機能が表示され、コードをすばやく検出および置換し、対応しない区切り文字を検査し、式ビルダーを開くことができます。

## Activity Editor のキーボード・ショートカット

表 41 に、Activity Editor メニュー項目と、それに関連付けられたキーボード・ショートカットを示します。

*表 41. Activity Editor のキーボード・ショートカット*

| メニュー | メニュー項目 | キーボード・ショートカット |
|---|---|---|
| 「ファイル」メニュー | 保管 | Ctrl+S |
| | 印刷設定 | Ctrl+Shift+P |
| | 印刷プレビュー | 使用可能なショートカットなし |
| | 印刷 | Ctrl+P |
| | 閉じる | 使用可能なショートカットなし |

*表 41. Activity Editor のキーボード・ショートカット (続き)*

| メニュー | メニュー項目 | キーボード・ショートカット |
|---|---|---|
| 編集 | 元に戻す | Ctrl+Z |
| | やり直し | Ctrl+Y |
| | 切り取り | Ctrl+X |
| | コピー | Ctrl+C |
| | 貼り付け | Ctrl+P |
| | 削除 | Del |
| | すべて選択 | Ctrl+A |
| | 検索 | Ctrl+F |
| | 置換 | Ctrl+H |
| | 行に移動 | Ctrl+G |
| 表示 | デザイン・モード | 使用可能なショートカットなし |
| | クイック表示モード | 使用可能なショートカットなし |
| | 移動/戻る | Alt+左矢印 |
| | 移動/進む | Alt+右矢印 |
| | 移動/1 レベル上へ | 使用可能なショートカットなし |
| | 移動/ホーム | Alt+Home |
| | ズームイン | Ctrl++ |
| | ズームアウト | Ctrl+ - |
| | 倍率指定ズーム | Ctrl+M |
| | ライブラリー・ウィンドウ | 使用可能なショートカットなし |
| | コンテンツ・ウィンドウ | 使用可能なショートカットなし |
| | プロパティー・ウィンドウ | 使用可能なショートカットなし |
| | ツールバー | 使用可能なショートカットなし |
| | ステータス・バー | 使用可能なショートカットなし |
| | 設定 | Ctrl+U |
| ツール | 変換 | Ctrl+T |
| | コードを編集 | Ctrl+E |
| | 対応しない区切り文字を検査 | 使用可能なショートカットなし |
| | 式ビルダー | 使用可能なショートカットなし |
| ヘルプ | ヘルプ・トピック | F1 |
| | 文書 | 使用可能なショートカットなし |

# アクティビティー定義

Activity Editor はアクティビティー定義 の作成に使用されますが、このアクティビティー定義は、コラボレーション・テンプレート内の各アクション・ノードのビジネス・ロジックを指定します。各アクション・ノードには、関連するアクティビティー定義が 1 つあります。

## 機能ブロック

アクティビティー定義は*機能ブロック* を基にしています。機能ブロックは、アクティビティー定義の個別のパーツを表しており、例えば定数、変数、あるいは特定の機能部分 (プログラミング・メソッドなど) があります。Activity Editor の機能ブロックの多くは、コラボレーション API の個々のメソッドに対応します。

機能ブロックは、ライブラリー・ウィンドウまたはコンテンツ・ウィンドウからドラッグ・アンド・ドロップして、編集キャンバスに置きます。機能ブロックを編集キャンバスに一度ドロップすれば、その機能ブロックを必要に応じて移動させることができます。キャンバス上で機能ブロック・アイコンをクリックして選択し、希望する場所にドラッグしてください。

機能ブロックには入力、出力、またはその両方があります。機能ブロックごとに入力と出力が事前定義され、その指定された値タイプのみが受け入れられます。機能ブロックが編集キャンバスにドロップされると、その入力ポートと出力ポートは、図 48 に示すような矢印で表されます。

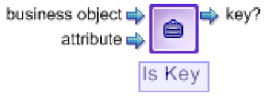



*図 48. 入力ポートと出力ポートがある機能ブロック*

これらのポートは、その機能ブロックと他のコンポーネントをリンクする接続点になります。デフォルトでは、各入出力の名前は、接続ポートの隣に表示されます (「表示」 —> 「設定」オプションを使って、名前を非表示にできます)。

### Web サービス機能ブロック

登録済みの Web サービスは、System Manager からエクスポートして Activity Editor で使用できます。エクスポート処理では、Web サービスの各メソッドが機能ブロックに変換され、変換された機能ブロックは、My Library¥Web Services フォルダーに配置されます。Web サービス・メソッドがエクスポートされ、機能ブロックに変換されれば、機能ブロックをアクティビティー定義で使用できます。他の機能ブロック同様、編集キャンバスでドラッグ・アンド・ドロップして、必要な入出力を指定できます。

Web サービスをエクスポートして、そのメソッドを機能ブロックに変換するには、以下の手順を実行します。

1. System Manager が開いていることを確認します。

2. 適切な統合コンポーネント・ライブラリー (ICL) からエクスポートする Web サービスを見付けます。
3. Web サービス名を右マウス・ボタンでクリックし、コンテキスト・メニューを開きます。
4. コンテキスト・メニューの「Activity Editor へエクスポート」をクリックします。「Activity Editor へエクスポート」ダイアログ・ボックスが表示されます。
5. エクスポートする 1 つまたは複数の Web サービスの横にあるチェック・ボックスを選択し、「Finish」をクリックします。Web サービスのメソッドが機能ブロックに変換され、Activity Editor の My Library¥Web Services フォルダーにエクスポートされます。
6. 開いている Activity Editor セッションがある場合は、Activity Editor を終了して再始動し、新しい機能ブロックを編集できるようにする必要があります。

## 接続リンク

機能ブロックは*接続リンク* によって接続されます。接続リンクは、アクティビティー定義におけるさまざまなコンポーネント間のアクティビティーの流れを定義します。接続リンクは、ある機能ブロックの出力ポートを別の機能ブロックの入力ポートに接続します。

**注:** 出力ポートは複数の接続リンクに接続できますが、入力ポートは単一の接続リンクしか受け入れることができません。

2 つの機能ブロック間に接続リンクを追加する手順は、以下のとおりです。

- 第 1 の機能ブロック (機能ブロック A) の出力ポートを左マウス・ボタンでクリックし、そのまま押し続けます。
- 左マウス・ボタンを押したまま、カーソルを第 2 の機能ブロック (機能ブロック B) の入力ポート上に移動します。
- マウス・ボタンを放します。これで、2 つの機能ブロック間に接続リンクが配置されます。接続リンクは、2 つのコンポーネントを結ぶ直角線でグラフィカルに表現されます。

入力ポートにすでに既存の接続リンクがある場合は、より新しい接続リンクに置き換わります。

## New Constant 機能ブロック

Activity Editor には New Constant 機能があります。この機能を編集キャンバス上にドラッグ・アンド・ドロップし、他の機能ブロックへの入力として設定し使用する定数値を定義することができます。

New Constant 機能ブロックは、ライブラリー・ウィンドウおよびコンテンツ・ウィンドウの「System」フォルダーにあります。 162 ページの図 49 は、編集キャンバスにドロップされたときの New Constant 機能ブロックの表示を示します。

*図 49. New Constant 機能ブロック*

定数の値を入力するためのテキスト編集ボックスが機能ブロックの上部に表示されます。(値を編集する必要がある場合は、Constant 機能ブロックの内側をクリックして、新しい値を入力します。) 定数には単一の出力ポートが含まれる点に注意してください。

**注:** Constant 機能ブロックは、値について単一行のみを受け入れる唯一のアクティビティー定義コンポーネントです。定数は Java String オブジェクトに変換されますが、複数行の定数値は変換できません。複数行の入力にする必要がある場合は、プログラミング規則「¥n」を使って、定数の行を区切ってください。(例えば、値「line1¥nline2」は、この値が 2 行で出力されることを示します。)

## アクティビティー定義のタグ

「System」フォルダー (ライブラリー・ウィンドウおよびコンテンツ・ウィンドウ内にある) には、コメント、記述、ラベル、および処理待ちタグをアクティビティー定義に追加する機能ブロックが含まれています。これらのタグは、各アクティビティーまたはサブアクティビティーの識別に役立ちます。また、実施すべき事項の覚え書にもなります。これらの機能ブロックを、他の機能ブロックと同様に編集キャンバス上にドラッグ・アンド・ドロップします。ただし、入出力ポートはありません。

新規タグを編集するには、タグの中心をシングルクリックします。カーソルが I ビームに変わり、テキストを入力できます。タグでは、長すぎるテキスト行は自動的に折り返されます。Enter を押して、新しい行にテキストを入力することもできます。

タグのサイズを変更する場合は、タグの右下の隅を左マウス・ボタンでクリックし、マウス・ボタンを押したまま、タグを希望のサイズにドラッグします。タグには最小サイズ要件があるため、最小サイズよりも小さいサイズには変更できません。

## コンポーネント・グループ

編集キャンバス上の機能ブロックのセットは、一つにまとめて保管し、後で別のアクティビティー定義に再使用することができます。この保管されたコンポーネント・グループは、実際には機能ブロックとして動作します。

希望するアクティビティー・フローを編集キャンバス上に作成した後に、そのフローの全部または一部を再使用可能なコンポーネント・グループとして保管する手順は、以下のとおりです。

1. 一つにまとめたい機能ブロックを選択します。複数の機能ブロックを選択するには、Ctrl キーを押したまま選択します。
2. 編集キャンバス上で右マウス・ボタンをクリックして、コンテキスト・メニューを開きます。

3. 「ユーザー・コレクションに追加」をクリックします。「ユーザー・コレクションに追加」ダイアログ・ボックスが表示されます。
4. 作成するコンポーネント・グループの名前と (オプションで) 説明を入力します。
5. コンポーネント・グループを表すのに使用するアイコンを選択して、「OK」をクリックします。

ライブラリー・ウィンドウおよびコンテンツ・ウィンドウの「My Collection」フォルダーに新しいコンポーネント・グループ・アイコンが追加されます。このアイコンは、編集キャンバス上にある、コラボレーション・シナリオ内の任意のアクティビティー定義にドラッグ・アンド・ドロップできます。

## サポートされる機能ブロック

Activity Editor の機能ブロックは、ライブラリー・ウィンドウの「General」フォルダー下、およびコンテンツ・ウィンドウの対応するフォルダー内に編成されています。表 42 に、機能ブロックの編成を示します。

*表 42. 機能ブロックの編成*

| 機能ブロック・フォルダー | 説明 | 詳細 |
|---|---|---|
| General¥APIs¥Business Object | ビジネス・オブジェクトの処理に関する機能ブロック | 247 ページの『第 11 章 ビジネス・オブジェクトの機能ブロック』 |
| General¥APIs¥Business Object¥Array | BusObj クラスの Java 配列の処理に関する機能ブロック | 247 ページの『第 11 章 ビジネス・オブジェクトの機能ブロック』 |
| General¥APIs¥Business Object¥Constants | BusObj クラスの Java 定数の処理に関する機能ブロック | 247 ページの『第 11 章 ビジネス・オブジェクトの機能ブロック』 |
| General¥APIs¥Business Object Array | ビジネス・オブジェクト配列の処理に関する機能ブロック | 275 ページの『第 12 章 ビジネス・オブジェクト配列の機能ブロック』 |
| General¥APIs¥Collaboration Exception | コラボレーション例外を処理する機能ブロック | 323 ページの『第 16 章 例外機能ブロック』 |
| General¥APIs¥Collaboration Template | コラボレーション・オブジェクトを操作する機能ブロック | 289 ページの『第 13 章 コラボレーション・テンプレートの機能ブロック』 |
| General¥APIs¥Collaboration Template¥Exception | コラボレーション・テンプレート内に新しい例外オブジェクトを作成する機能ブロック | 289 ページの『第 13 章 コラボレーション・テンプレートの機能ブロック』 |

表 42. 機能ブロックの編成 (続き)

| 機能ブロック・フォルダー | 説明 | 詳細 |
|---|---|---|
| General¥APIs¥Collaboration Template¥Constants | コラボレーション例外オブジェクト内で特定の例外タイプを表すために使用される機能ブロック | 289 ページの『第 13 章 コラボレーション・テンプレートの機能ブロック』 |
| General¥APIs¥Database Connection | データベース接続の作成および保守に関する機能ブロック | 303 ページの『第 14 章 データベース接続の機能ブロック』 |
| General¥APIs¥DB Stored Procedure Param | データベースのストアード・プロシージャー・パラメーターを使用して動作する機能ブロック | 319 ページの『第 15 章 データベース・ストアード・プロシージャーの機能ブロック』 |
| General¥APIs¥Execution Context | コラボレーション実行コンテキストを設定および保守する機能ブロック | 329 ページの『第 17 章 実行機能ブロック』 |
| General¥APIs¥Identity Relationship | ID 関係の処理に関する機能ブロック | 「マップ開発ガイド」 |
| General¥APIs¥Maps | マップ実行に必要なランタイム値の照会および設定に関する機能ブロック | 「マップ開発ガイド」 |
| General¥APIs¥Maps¥Constants | 機能ブロック定数 | 「マップ開発ガイド」 |
| General¥APIs¥Maps¥Exception | マップの新規例外オブジェクト作成に関する機能ブロック | 「マップ開発ガイド」 |
| General¥APIs¥Participant | ID 関係の参加者の値の設定および検索に関する機能ブロック | 「マップ開発ガイド」 |
| General¥APIs¥Participant¥Array | 参加者配列の作成および処理に関する機能ブロック | 「マップ開発ガイド」 |
| General¥APIs¥Participant¥ Constants | 参加者に使用する機能ブロック定数 | 「マップ開発ガイド」 |
| General¥APIs¥Relationship | 関係のランタイム・インスタンスの操作に関する機能ブロック | 「マップ開発ガイド」 |
| General¥Date | 日付の処理に関する機能ブロック | 331 ページの『第 18 章 日付の機能ブロック』 |
| General¥Date¥Formats | さまざまな日付形式の指定に関する機能ブロック | 331 ページの『第 18 章 日付の機能ブロック』 |
| General¥Logging and Tracing | ログ・メッセージおよびトレース・メッセージの処理に関する機能ブロック | 337 ページの『第 19 章 ログおよびトレースの機能ブロック』 |
| General¥Logging and Tracing¥Log Error | エラー・メッセージのフォーマット設定に関する機能ブロック | 337 ページの『第 19 章 ログおよびトレースの機能ブロック』 |
| General¥Logging and Tracing¥Log Information | 情報メッセージのフォーマット設定に関する機能ブロック | 337 ページの『第 19 章 ログおよびトレースの機能ブロック』 |

*表 42. 機能ブロックの編成 (続き)*

| 機能ブロック・フォルダー | 説明 | 詳細 |
|---|---|---|
| General¥Logging and Tracing¥Log Warning | 警告メッセージのフォーマット設定に関する機能ブロック | 337 ページの『第 19 章 ログおよびトレースの機能ブロック』 |
| General¥Logging and Tracing¥Trace | トレース・メッセージのフォーマット設定に関する機能ブロック | 337 ページの『第 19 章 ログおよびトレースの機能ブロック』 |
| General¥Mapping | 指定されたコンテキスト内でのマップ実行に関する機能ブロック | 「*マップ開発ガイド*」 |
| General¥Math | 基本的な計算タスクの機能ブロック | 「*マップ開発ガイド*」 |
| General¥Properties | 構成プロパティー値の検索に関する機能ブロック | 「*マップ開発ガイド*」 |
| General¥Relationship | ID 関係の保守および照会に関する機能ブロック | 「*マップ開発ガイド*」 |
| General¥String | String オブジェクトの操作に関する機能ブロック | 345 ページの『第 20 章 ストリングの機能ブロック』 |
| General¥Utilities | 例外のスローとキャッチ、およびループ、属性の移動、条件の設定に関する機能ブロック | 353 ページの『第 21 章 ユーティリティーの機能ブロック』 |
| General¥Utilities¥Vector | Vector オブジェクトの処理に関する機能ブロック | 353 ページの『第 21 章 ユーティリティーの機能ブロック』 |
| General¥Utilities¥Locale and General¥Utilities¥Locale¥Constants | ロケールを設定および照会する機能ブロック | 353 ページの『第 21 章 ユーティリティーの機能ブロック』 |

## 例: 日付形式の変更

この例は、Activity Editor を使ってソース属性の日付形式を変更し、形式を設定し直した値を宛先属性に割り当てる方法を示します。この例では、ソース属性は QuoteSchedule.ExpireDate で、宛先属性は Invoice.PostingDate です。オリジナルの日付形式は yyyyMMdd で、更新された日付形式は yyyy-MM-dd です。この例では、ビジネス・オブジェクトおよび属性はすでに作成され、コラボレーション・テンプレート・シナリオに宣言されていることを前提としています。

ソース属性の日付形式を変更して、その形式を宛先属性に割り当てるには、以下の手順を行う必要があります。

1. Activity Editor が開いていることを確認します。
2. QuoteSchedule.ExpireDate 変数機能ブロックを編集キャンバスにドラッグ・アンド・ドロップします。(シナリオで使用可能なビジネス・オブジェクト、属性、および変数を表す機能ブロックは、ライブラリー・ウィンドウおよびコンテンツ・ウィンドウの「Variables」フォルダーにあります。)

3. Format Change 機能ブロックを編集キャンバス上の QuoteSchedule.ExpireDate 機能ブロックの右側にドラッグ・アンド・ドロップします (図 50 を参照)。

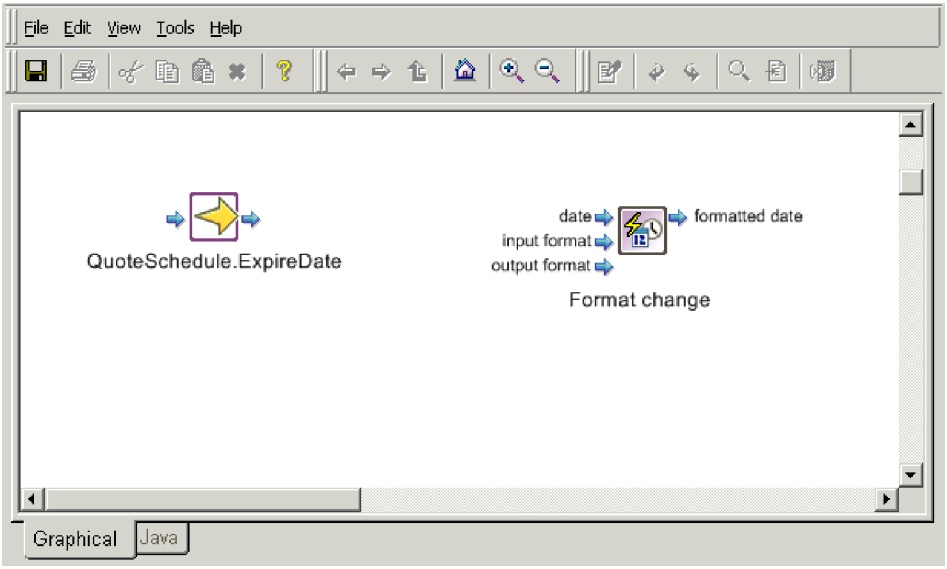



*図 50. Format Change 機能ブロックの配置*

4. QuoteSchedule.ExpireDate 機能ブロックの出力ポートと Format Change 機能ブロックの「Date」入力の間に接続リンクを置きます。
5. yyyyMMdd 機能ブロック定数を編集キャンバスにドラッグ・アンド・ドロップして、QuoteSchedule.ExpireDate 機能ブロックおよび Format Change 機能ブロックの下側に置きます。この機能ブロックは、QuoteSchedule.ExpireDate 属性の現在の形式を表します。
6. yyyyMMdd 機能ブロックの出力ポートと Format Change 機能ブロックの「Input Format」入力の間に接続リンクを置きます (167 ページの図 53 を参照)。

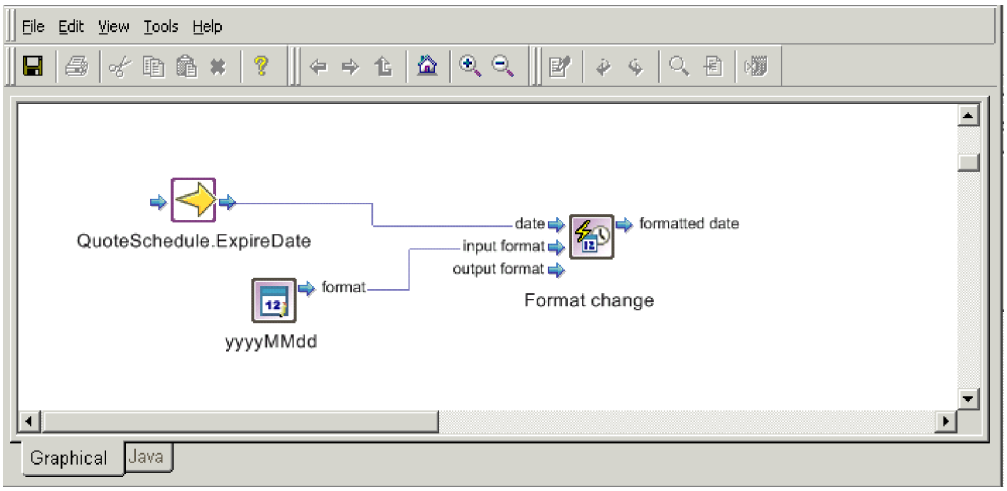



*図 51. 入力日付形式の指定*

7. yyyy-MM-dd 機能ブロック定数を編集キャンバスにドラッグ・アンド・ドロップして、yyyyMMdd 機能ブロックの近くに置きます。この機能ブロックは、QuoteSchedule.ExpireDate 属性の新しい形式を表します。
8. yyyy-MM-dd 機能ブロックの出力ポートと Format Change 機能ブロックの「Output Format」入力の間に接続リンクを置きます (図 53 を参照)。

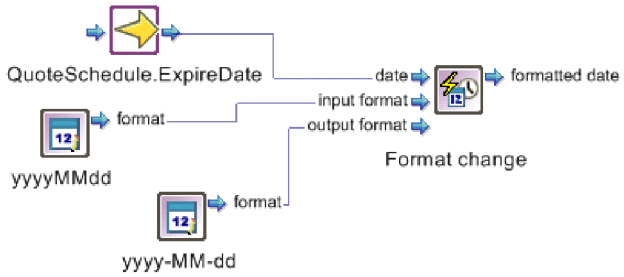



*図 52. 出力日付形式の指定*

9. Invoice.PostingDate 機能ブロックを編集キャンバスにドラッグ・アンド・ドロップします。これは宛先属性です。この機能ブロックを Format Change 機能ブロックの右側に置きます。
10. Format Change 機能ブロックの出力を Invoice.PostingDate 属性に割り当てるには、Format Change 機能ブロックの出力ポートと Invoice.PostingDate の入力ポートの間に接続リンクを置きます (図 53 を参照)。

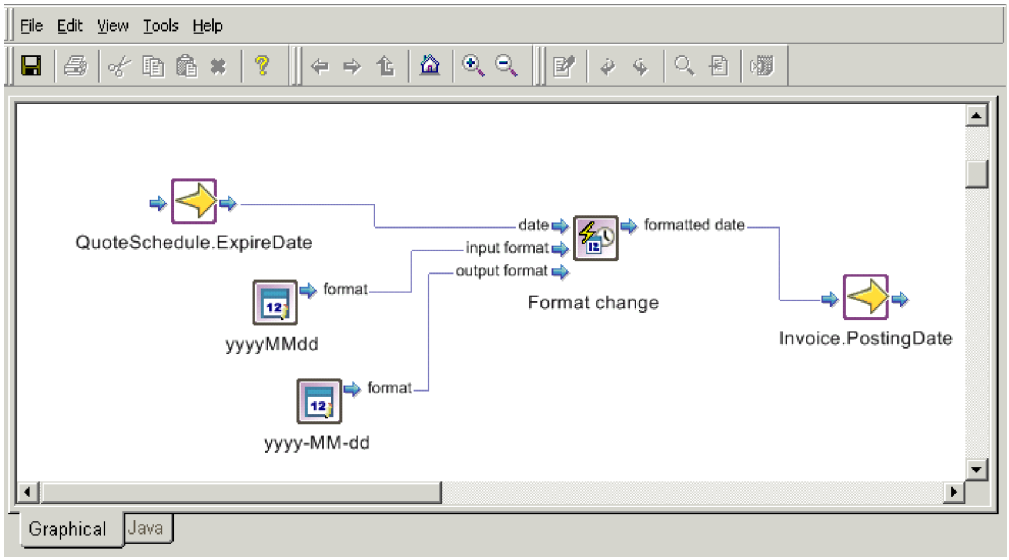



*図 53. 出力を宛先属性に割り当てる*

11. アクティビティー定義を保管するため、「ファイル」 —> 「保管」をクリックします。

## 例: 複製ビジネス・オブジェクトの作成

以下の例は、ビジネス・オブジェクトを複製する方法を示します。この例では、オリジナル・オブジェクトはトリガー・ビジネス・オブジェクト (triggeringBusObj) で、複製は inPort というポート変数のビジネス・オブジェクト (inPortBusObj) になります。どちらのビジネス・オブジェクトも、ライブラリー・ウィンドウおよびコンテンツ・ウィンドウの「Variables」フォルダーにあります。

この例の複製ビジネス・オブジェクトを作成するには、以下の手順を実行します。

1. Activity Editor が開いていることを確認します。
2. triggeringBusObj 変数を編集キャンバスにドラッグ・アンド・ドロップします。
3. Duplicate 機能ブロックをキャンバスにドラッグ・アンド・ドロップして、triggeringBusObj 機能ブロックの右側に置きます。

   **注:** Duplicate 機能ブロックは、ライブラリー・ウィンドウおよびコンテンツ・ウィンドウの General¥APIs¥Business Object フォルダーにあります。
4. triggeringBusObj 変数の出力ポートと Duplicate 機能ブロックの「Original」入力の間に接続リンクを置きます。
5. inPortBusObj 変数をキャンバスにドラッグ・アンド・ドロップして、Duplicate 機能ブロックの右側に置きます。
6. オリジナル・ビジネス・オブジェクトの値を新規ビジネス・オブジェクトに割り当てるには、Duplicate 機能ブロックの「duplicate」出力と inPortBusObj 変数の入力の間に接続リンクを置きます。

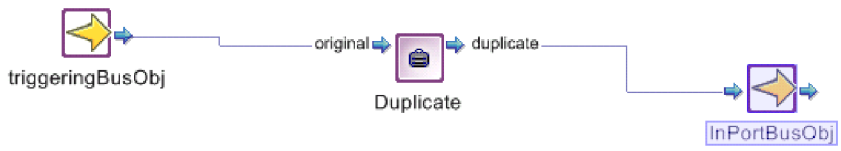



*図 54. ビジネス・オブジェクトの複製*

7. アクティビティー定義を保管するため、「ファイル」 —> 「保管」をクリックします。

# 第 7 章 例外処理

*例外* とは、アクティビティー・ダイアグラム内で明示的に処理されない場合、コラボレーションの実行が停止することがあるエラー状態を表します。例外処理の目的は、以下を確実にすることです。

- 可能であれば、例外を発生させたエラー条件が訂正されるかまたは対象範囲を減少して、コラボレーションの実行を継続できるようにします。
- エラー条件が訂正できず、シナリオを正常に終了できない場合は、コラボレーションの実行を終了する必要があります。この場合には、エラー条件の原因について、コラボレーションでできるだけ多くの情報を提供する必要があります。この情報により、管理者はこのエラーについての修正方法と今後このエラーが発生するのを防止する方法を判別できます。

したがって、ユーザーのコラボレーション・テンプレートおよびコラボレーション・ランタイム環境の両方による、例外の処理方法を理解することが重要です。このセクションでは、例外処理について、以下の内容を説明します。

- 『コラボレーション例外とは』
- 171 ページの『例外の処理方法』
- 174 ページの『例外の処理方法』

**要確認**

この章では、SEND_EMAIL プロパティーを使用して例外を処理する方法を紹介します。SEND_EMAIL プロパティーは、CollaborationFoundation テンプレートでのみ使用でき、現在は Process Designer Express に同梱されていません。

## コラボレーション例外とは

コラボレーション API には、コラボレーションで発生する例外を表現するための、*例外オブジェクト* が用意されています。 図 55 に示すように、この例外オブジェクトは、例外を発生させた条件についての情報を含んでいます。

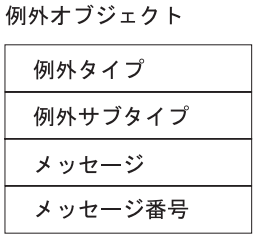



*図 55. CollaborationException 例外オブジェクト*

この例外オブジェクトは、`CollaborationException` クラスのインスタンスで、このクラスは Java `Exception` クラスの拡張です。表 43 に、例外オブジェクト内の情報

を取得するために CollaborationException クラスに用意されている accessor メソッドを示します。

*表 43. 例外オブジェクト内の情報*

| メンバー | accessor メソッド |
|---|---|
| 例外タイプ | getType() |
| 例外サブタイプ | getSubType() |
| メッセージ・テキスト | getMessage()、toString() |
| メッセージ番号 | getMsgNumber() |

**注:** 例外が発生すると、コラボレーション・ランタイム環境により、currentException と呼ばれるシステム変数に例外についての情報が設定されます。 currentException 変数は CollaborationException クラスのインスタンスです。したがって、表 43 のメソッドを使用して、currentException 変数から例外情報を取得できます。

コラボレーション例外の原因を識別するために、例外オブジェクトには、表 44 にリストされている*例外タイプ* の 1 つが含まれています。例外タイプは、Java 静的定数で宣言されたストリング値です。

*表 44. 例外タイプ*

| 例外タイプ定数 | 説明 |
|---|---|
| AnyException | 任意のタイプの例外。2 つの例外分岐があり、1 つは特定タイプの例外をテストし、もう 1 つは AnyException をテストする場合、特定タイプの例外をテストする分岐が先に検査されます。現在の例外が特定の例外と一致しない場合、AnyException についてテストする分岐が次に検査されます。 |
| AttributeException | 属性のアクセス問題。例えば、コラボレーションが、String 属性に対して getDouble() を呼び出す場合や、存在しない属性に対して getString() を呼び出す場合など。 |
| JavaException | コラボレーション API の一部ではない Java コードに関する問題。 |
| ObjectException | メソッドに渡された無効なビジネス・オブジェクト。 |
| OperationException | サービス呼び出しが適切に設定されませんでした。あるいは送信されませんでした。 |
| ServiceCallException | コラボレーション以外の理由によって失敗したサービス呼び出し。例えば、コネクターまたはアプリケーションが使用できない場合や、ネットワークに障害がある場合など。 |
| SystemException | InterChange Server Express システムの内部エラー。 |
| TimeoutException | 同期または非同期インバウンド・サービス呼び出しのタイムアウト。 |
| TransactionException | トランザクション・コラボレーションの、トランザクションの振る舞いに関連するエラーです。例えば、ロールバックに失敗しました。あるいは、コラボレーションが差し戻しを適用できませんでした。 |

**注:** 決定ノードで例外分岐を定義する場合、例外分岐の条件で検査する例外タイプを指定します。詳細については、 175 ページの『例外のキャッチ』を参照してください。

これらの例外タイプの一部には、例外を発生させる多くの状態があります。このような例外タイプの場合、例外オブジェクトに通常、*例外サブタイプ* が含まれ、例外の原因についての追加情報が提供されます。例外サブタイプを使用する 2 つの主な例外タイプは、`JavaException` と `ServiceCallException` です。詳細については、438 ページの『getSubType()』を参照してください。

## 例外の処理方法

コラボレーションの実行は、次の 2 つの*実行状態* の内の 1 つになります。

- 通常状態は、次のいずれかの状態を示します。
  - 例外は発生しなかった。
  - 例外が発生したが、コラボレーション・テンプレートが例外をキャッチした。
- 例外状態は、例外が発生し、コラボレーション・テンプレート内で処理*されなかった* ことを示します。コラボレーションは、次のいずれかの状態が発生すると、例外状態になります。
  - 次のいずれかの状態などの、コラボレーション例外が発生。
    - サービス呼び出しが失敗。つまり、例外タイプ `ServiceCallException` のコラボレーション例外が発生。
    - Java 例外が発生。つまり、例外タイプ `JavaException` のコラボレーション例外が発生。
  - コラボレーション・テンプレートでの `raiseException()` メソッドの呼び出しで、任意の有効な例外タイプのコラボレーション例外が発生。

コラボレーションは、実行時にこれらの 2 つの状態間を切り替えることができます。例外が発生するか `raiseException()` を実行すると、例外状態になります。例外分岐を使用してコラボレーション・テンプレートが例外をキャッチすると、通常状態に戻ります。アクティビティー・ダイアグラムが実行する (または実行しない) 例外処理に関係なく、コラボレーション・ランタイム環境は、例外の発生後もダイアグラムのロジックの実行を継続します。このロジックは、最終的に、終了成功ノードまたは終了障害ノードで終了します。コラボレーション・ランタイム環境では、コラボレーションの実行状態を使用して、コラボレーションが終了した際に未解決のフローを作成するかどうかを判別します。終了ノードの詳細については、146 ページの『実行経路の終了』を参照してください。未解決のフローの詳細については、172 ページの『例外状態の処理』を参照してください。

### 通常状態の処理

コラボレーション・ランタイム環境でコラボレーションが正常に実行されている間は (アクティビティー・ダイアグラムのロジックで定義されたように)、コラボレーションの実行は通常状態です。実行経路を終了するために可能な方法には、以下のものがあります。

- 終了成功ノード

コラボレーション・ランタイム環境は現行のダイアグラムの実行を停止して、次に高い実行レベルに制御を渡します。

– 終了成功ノードでメインのアクティビティー・ダイアグラムが終了すると、コラボレーション・ランタイム環境でコラボレーションが終了します。コラボレーションの実行が通常状態の場合、コラボレーション・ランタイム環境では未解決のフローは作成されません。
– 終了成功ノードでサブダイアグラムまたはイテレーターが終了すると、コラボレーション・ランタイム環境で次のステップが実行されます。
  - 現行の実行レベルを終了して、親ダイアグラムに制御を渡します。
  - 親ダイアグラムのロジックに定義されているように、親ダイアグラムの実行を継続します。しかし、コラボレーションの実行が通常状態の場合、コラボレーション・ランタイム環境では例外分岐を検査しません。

• 終了障害ノード

コラボレーション・ランタイム環境はコラボレーションの実行を停止して、通常状態のコラボレーションのステップを実行します ( 147 ページの『失敗での終了』を参照)。

## 例外状態の処理

コラボレーションの実行が例外状態になっても、コラボレーション・ランタイム環境は実行を停止しません。それに対し、通常状態での実行の場合のように、アクティビティー・ダイアグラムのロジックで定義されている実行を継続します。この実行経路を終了するために可能な方法には、以下のものがあります。

• 終了成功ノード

コラボレーション・ランタイム環境は現行のダイアグラムの実行を停止して、次に高い実行レベルに制御を渡します。それには以下の場合があります。

– 終了成功ノードでメインのアクティビティー・ダイアグラムが終了すると、次に高い実行レベルはコラボレーション・ランタイム環境です。詳細については、『メインダイアグラムの正常終了』を参照してください。
– 終了成功ノードでサブダイアグラムまたはイテレーターが終了すると、次に高い実行レベルは親ダイアグラムです。詳細については、 173 ページの『サブダイアグラムまたはイテレーターの正常終了』を参照してください。

• 終了障害ノード

コラボレーション・ランタイム環境でコラボレーションの実行を停止します。コラボレーションの実行が例外状態の場合、コラボレーション・ランタイム環境は次に例外を処理します。例外処理ステップについては、『メインダイアグラムの正常終了』を参照してください。

### メインダイアグラムの正常終了

終了成功ノードでメインダイアグラムが終了すると、コラボレーション・ランタイム環境でコラボレーションが終了します。コラボレーションの実行が例外状態の場合、コラボレーション・ランタイム環境は次のステップを実行して例外を処理します。

1. エラーをコラボレーションのログ宛先に記録します。ログ宛先は、標準出力 (STDOUT) またはログ・ファイルにすることができ、InterChange Server Express のログ宛先の構成方法によって異なります。
   - コラボレーションがトランザクションでない 場合、コラボレーション・ランタイム環境はエラーをログに記録します。
   - コラボレーションがトランザクションである場合、コラボレーション・ランタイム環境は、これをロールバックしてコラボレーションの差し戻しステップを実行します。

     ロールバック時に例外が発生すると、コラボレーション・ランタイム環境は、コラボレーションを終了してエラーをログに記録します。この時点では、コラボレーション・オブジェクトは「未確定」状態です。管理者は、残りの差し戻しステップを実行または破棄することにより、コラボレーション・オブジェクトのトランザクション状況を手動で解決できます。
2. 不成功のコラボレーションのために*未解決のフロー* を作成します。

   コラボレーションが例外状態で実行を終了すると、未解決のフローはそのままになります。未解決のフローには以下のものが含まれています。
   - **失敗したイベント**。これは不成功のコラボレーションを起動したオリジナルのイベント (ビジネス・オブジェクトと動詞) です。
   - 失敗の原因を説明する例外メッセージ。

   コラボレーション・ランタイム環境によりこの未解決のフローが InterChange Server のイベント再サブミット・キューに送信され、そこでサーバー管理者は再サブミットのためにこれを解析および評価できます。Flow Manager ツールを使用すると、管理者はイベント再サブミット・キューにアクセスできます。管理者は、終了したコラボレーションの名前、およびエラー条件を説明するメッセージなどの、未解決のフローについての情報を調査することができます。

   **注:** Flow Manager の使用法の詳細については、「*システム管理ガイド*」を参照してください。

デフォルトでは、コラボレーション・ランタイム環境は、次の非常に単純な例外メッセージを未解決のフローに関連付けしています。

```
Scenario failed.
```

このデフォルトの例外メッセージでは、管理者が未解決のフローの原因をトラブルシューティングするのに使用できる情報はあまり提供されません。しかし、コラボレーション・テンプレートをコーディングして例外を生成すると、発生した実際のエラー状態についての詳細情報を例外メッセージに設定できます。コラボレーション・ランタイム環境で例外を処理する場合、この詳細な例外メッセージを未解決のフローと関連付けできます。詳細については、177 ページの『例外の発生』を参照してください。

## サブダイアグラムまたはイテレーターの正常終了

終了成功ノードでサブダイアグラム (またはイテレーター) が終了し、コラボレーションの実行が例外状態の場合、コラボレーション・ランタイム環境で次のステップが実行されます。

1. 制御を親ダイアグラムに渡します。親ダイアグラムとは、サブダイアグラム (またはイテレーター) ノードを含むダイアグラムです。
2. サブダイアグラム (またはイテレーター) を例外処理ノードと接続する親ダイアグラムの決定ノードですべての例外分岐を検査します。以下のいずれかのアクションを実行します。
   - 現行の例外をキャッチする例外分岐が存在する場合、コラボレーション・ランタイム環境は、例外分岐が指す例外処理ノードに制御を渡します。この例外処理ノードが完了すると、例外処理ノードを含むアクティビティー・ダイアグラムの分岐で定義されたように、実行が継続します。

     **注:** 例外分岐が実行されると、コラボレーションの実行は通常状態に変わります。したがって、コラボレーション・テンプレートにより実行経路のどこかで例外が生成されなければ、コラボレーション・ランタイム環境でコラボレーションの未解決のフローは作成されません。例外処理コードの実装方法の詳細については、『例外の処理方法』を参照してください。
   - 例外分岐で現行の例外をキャッチしない場合、コラボレーション・ランタイム環境では、親ダイアグラムをそのロジックで定義されているように継続して実行します。しかし、コラボレーションの実行は依然として例外状態です。他のいずれかの実行レベルで例外をキャッチしない限り、コラボレーションが終了するときに、コラボレーション・ランタイム環境でコラボレーションの未解決のフローが作成されます。

ダイアグラムが終了成功または終了障害のいずれかで終了するまで、コラボレーション・ランタイム環境で親ダイアグラムのロジックは継続して実行されます。コラボレーション実行が例外状態である限り、コラボレーションが終了するときにランタイム環境で例外が処理されます。各実行レベルが終了成功で終了する場合、メインダイアグラムに達するまで、次に高いレベルに制御が渡されます。メインダイアグラムで例外をキャッチしない場合、このコラボレーションは終了され、制御はコラボレーション・ランタイム環境に渡されます。

## 例外の処理方法

コラボレーションが処理できる例外には 2 つのカテゴリーがあります。

- ビジネス・プロセス例外

  ビジネス・プロセス例外は、コラボレーション API メソッドを使用するコードから発生します。例えば、シナリオでビジネス・オブジェクト属性の値を設定し、コネクターにサービス呼び出し要求を送信する場合などで、ビジネス・プロセス例外が発生することがあります。特定のサービス呼び出し例外の処理方法の詳細については、181 ページの『特定のサービス呼び出し例外の処理』を参照してください。
- ネイティブ Java 例外

  Java 例外は、ネイティブな Java メソッドを使用する開発者自身のコードが原因で発生します。このような例外が発生すると、例外タイプが `JavaException` で、その例外サブタイプに特定の Java 例外を含むコラボレーション例外を生成できます。コラボレーション・ランタイム環境は、開発者自身のコードから発生する Java 例外をキャッチし処理します。

例外が発生すると、アクティビティー・ダイアグラム階層の所定レベルでの例外処理で、次のいずれかの方法で例外を処理できます。

- 現行の実行レベルで例外をキャッチしない。
- 決定ノードの例外分岐で例外をキャッチする。

## 例外をキャッチしない

アクティビティー・ダイアグラムで、例外分岐を使用して例外を明示的にキャッチしない 場合、コラボレーションの実行は (例外が発生したときになった) 例外状態のままです。例外が発生しても、コラボレーション・ランタイム環境はダイアグラム内の実行を停止しません。その代わりにアクティビティー・ダイアグラムのロジックに従って実行を継続し、終了成功ノードまたは終了障害ノードで終了します。

- 実行経路が終了障害で終わる場合、コラボレーション・ランタイム環境によりコラボレーションが終了され、未解決のフローが作成されます。

  したがって、サブダイアグラムでの例外をキャッチし、さらに終了障害ノードにトラバースする場合は、コードでサブダイアグラム内の例外を確実にキャッチする必要があります (終了障害ノードの前)。
- 実行経路が終了成功で終わる場合、コラボレーションにより次に高いレベルに制御が渡されます。

  階層アクティビティー・ダイアグラムでは、例外分岐を使用してある実行レベルでの例外をキャッチしない場合、終了成功を使用して次に高いレベルのダイアグラムに制御を渡すことができます。この高位レベルのダイアグラムで、イベントをキャッチして、例外を処理するかまたは次に高いレベルに例外を発生することができます。

  コラボレーション・テンプレートが、この実行経路内のどこにおいても例外をキャッチしなかった場合、その実行は例外状態のままになっています。この場合、コラボレーション・ランタイム環境は、 172 ページの『例外状態の処理』で説明するように、そのまま例外を処理します。コラボレーション・テンプレートは例外をキャッチしていないため、コラボレーション・ランタイム環境に未解決のフローを使用して独自のデフォルト例外メッセージ (`Scenario failed.`) を組み込む必要があります。

## 例外のキャッチ

コラボレーション・テンプレートはアクティビティー・ダイアグラム内に、その分岐タイプが `Exception` に設定されている決定ノード内の分岐である、*例外分岐* を使用した例外処理を組み込むことができます。決定ノードにより、アクション・シンボルがその可能性ある決定結果に接続されます。例外分岐により、例外が発生するアクション・シンボルが例外を処理するアクション・シンボルに経路付けされます。例外分岐には、例外分岐がキャッチする例外タイプを指定する例外条件が含まれます。 170 ページの表 44 に、例外条件を定義する際に選択できる、コラボレーション例外タイプをリストします。

**注:** アクティビティー・ダイアグラムへの例外分岐の追加方法の詳細については、123 ページの『例外分岐の定義』を参照してください。

例外が発生すると、コラボレーション・ランタイム環境により、currentException システム変数が設定されます。例外分岐に従うかどうかを判別するために、コラボレーション・ランタイム環境で例外分岐の条件での例外タイプと currentException システム変数内の例外タイプを比較して、例外分岐の例外条件を評価します。

- これらの例外タイプが一致すると、例外条件は *true* となり、アクティビティー・ダイアグラムで例外をキャッチします。

  コラボレーション・ランタイム環境でコラボレーションの実行は通常状態に変更され、例外分岐が指すアクション・シンボルに制御が渡されます。このアクション・シンボルには、例外条件で指定された例外タイプを処理する、例外処理コードを組み込むことができます。このコードで currentException システム変数にアクセスして例外情報を取得できます。
- これらの例外タイプが一致しない場合、例外条件は false となり、アクティビティー・ダイアグラムで例外はキャッチされません。

  実行は、決定ノードのデフォルトの分岐 (存在する場合) に渡され、その後アクティビティー・ダイアグラムのロジック内の次のアクション・シンボルに渡されます。デフォルトの分岐で例外が処理されない場合、この状態はアクティビティー・ダイアグラムのこのレベルで例外が処理されない ことを意味します。また、コラボレーションの実行は例外状態のままであることも意味します。

**注:** コラボレーション・ランタイム環境では、コラボレーションの実行が例外状態の場合のみ、例外分岐を検査します。

指定された決定ノードは、最大 7 つの分岐を持つことができます。したがって、多くの例外タイプの例外処理を設定できます。各例外分岐にその例外条件での異なる例外タイプを指定して、その例外タイプについての例外処理ノードを指すことができます。あるいは、例外条件に AnyException 例外タイプを持つ単一の例外分岐で、すべて の例外タイプを処理できます。

コラボレーション・テンプレートで例外をキャッチして例外処理ノードに制御が渡されると、コラボレーション・テンプレートで、次の方法で例外を処理できます。

- シナリオのロジックに従って成功または失敗まで処理を進めます。
- アクティビティー・ダイアグラム内の次に高いレベルに例外を発生します。

## シナリオ・ロジックの続行

シナリオのロジックを続行するには、例外処理ノードで次のステップを実行します。

1. 例外を生成しない 方法で例外を処理します。

   例外分岐が指すノード内に、例外を処理するコードを組み込むことができます。177 ページの表 45 に、一部の使用できる処理ステップをリストします。
2. アクティビティー・ノード内の実行経路を終了成功ノードまたは終了障害ノードで終了します。

   例外処理ノードが例外を生成しない 限り (raiseException() メソッドを使用して)、コラボレーションの実行は通常状態のままです。したがって、コラボレーション実行の完了時に、コラボレーション・ランタイム環境で未解決のフローは

作成されません。コラボレーション・ランタイム環境によるこれらの終了ノードの処理方法の詳細については、 171 ページの『通常状態の処理』を参照してください。

表 45 に、例外処理ノードで実行できる使用可能な処理ステップの一部をリストします。これらのステップでは、コラボレーションの実行状態は変更しません。したがって、コラボレーションの実行は通常状態のままです。

*表 45. 例外処理に使用可能な処理ステップ*

| 例外処理ステップ | メソッド | 詳細 |
|---|---|---|
| メッセージをコラボレーションのログ宛先に記録 | `logError()`、`logWarning()`、`logInfo()` | 203 ページの『メッセージのロギング』 |
| 例外についての情報の取得 | `CollaborationException` クラスのメソッド | 170 ページの表 43 |

例えば、`logError()` メソッドはエラーをコラボレーションのログ宛先に記録します。この宛先には、標準出力 (STDOUT) またはログ・ファイル (そのように構成されている場合) が可能です。また、このメソッドはエラー・メッセージを E メール受信側に送信します。コラボレーション・テンプレートはこのメソッドを使用してエラーを記録し、管理者がこれを調査できます。次のコード・フラグメントは、`CollaborationException` クラスの `getMessage()` と `getMsgNumber()` メソッドを使用して、`currentException` 変数から例外情報を抽出します。その後この情報を `logError()` 呼び出しを使用してエラーをフォーマット設定し、コラボレーションのログ宛先に送信します。

```
// extract exception information
sMessage = currentException.getMessage();
imsgNumber = currentException.getMsgNumber();

// log message and send email (if configured)
logError(imsgNumber, sMessage, ...);
```

詳細については、 372 ページの『logError()、logInfo()、logWarning()』の `logError()` メソッドの説明を参照してください。エラーをログ宛先に送信するだけでは、明確な例外メッセージを未解決のフローと関連付けるには通常は不十分であることに注意してください。例えば、例外が発生して、例外分岐でキャッチし、例外処理ノードは単にエラーをログに記録して失敗で終了すると仮定します。この場合、不成功のコラボレーションの未解決のフローには失敗したイベントが含まれますが、その例外メッセージはコラボレーション・ランタイム環境のデフォルトのメッセージ (`Scenario failed.`) のみです。

## 例外の発生

`raiseException()` メソッドは、次に高い実行レベルにコラボレーション例外を発生します。コラボレーション・ランタイム環境で `raiseException()` 呼び出しが実行されると、コラボレーションの実行が例外状態に変更され、アクティビティー・ダイアグラムのロジックが続行されます。アクティビティー・ダイアグラム内の次に高いレベルに例外を発生するには、例外処理ノードで次のステップを実行します。

1. 現行の例外から例外情報を取得して、生成された例外に組み込みます。

   例外処理ノード内で、`CollaborationException` クラスのメソッドを使用して、`currentException` システム変数から例外情報を抽出できます。

**注:** currentException 変数からメッセージを抽出して生成された例外に組み込むことができるようにする必要があります。これにより、例外メッセージを未解決のフローと関連付けするときに、このメッセージをコラボレーション・ランタイム環境で使用できます。

2. raiseException() メソッドの呼び出しを組み込んで、発生する例外を生成します。

   コラボレーション・ランタイム環境で raiseException() 呼び出しが実行されると、コラボレーションの実行を例外状態に変更します。raiseException() 呼び出しにより、次に高い実行レベルに発生する例外が指定されます。

3. 例外処理コードを含む分岐の実行経路を、終了成功ノードかまたは終了障害ノードで終了します。
   - 実行経路を終了成功で終了すると、次に高い実行レベルに例外を発生し、そこで例外をキャッチするかまたは次に高いレベルに例外を発生できます。各実行レベルでこのメソッドを使用すると、コラボレーション・コードにより、エラー処理を最終的に決定できるトップレベル・ダイアグラムに、トラップされた例外を発生できます。
   - 実行経路を終了障害で終了すると、例外メッセージを未解決のフローの一部として組み込む、コラボレーション・ランタイム環境に例外を発生します。

   例外を発生したので、コラボレーション実行は例外状態です。各実行レベルで例外が発生される限り (raiseException() を使用)、コラボレーションの実行は例外状態のままです。したがって、コラボレーション実行の完了時に、コラボレーション・ランタイム環境で未解決のフローが作成されます。コラボレーション・ランタイム環境によるこれらの終了ノードの処理方法の詳細については、172 ページの『例外状態の処理』を参照してください。

コラボレーション・テンプレートで raiseException() メソッドと logError() メソッドを使用して例外を処理する方法を理解するには、次の例を考慮します。コラボレーションのメインダイアグラムが subdiagramA を呼び出し、それが次に subdiagramB を呼び出すと仮定します。subdiagramB は、例外を発生させる可能性があるサービスを呼び出します。したがって、このサブダイアグラムにはサービス呼び出しを起動するアクション・ノードが含まれます。このアクション・ノードはサービス呼び出し例外を検査する例外分岐を持つ決定ノードに接続します。サービス呼び出し例外が発生すると、例外処理コードを持つアクション・ノードに例外分岐が接続します。

例外が発生すると、コラボレーション・ランタイム環境によりコラボレーションの実行が例外状態に変更され、subdiagramB でのサービス呼び出しと関連した例外分岐の例外条件が評価されます。例外分岐の条件が true と評価されると、例外分岐により例外がキャッチされ、例外分岐が指す例外処理ノードに制御が渡されます。例外分岐が例外をキャッチすると、コラボレーションの実行は通常状態に戻ります。

179 ページの図 56 に、subdiagramB でのサービス呼び出し例外についての例外処理を実行するコード・フラグメントを示します。

```
// exception handling in subdiagramB
sMessage = currentException.getMessage();
sType = currentException.getType();

// raise the exception to subdiagramA
raiseException(sType, 2345, parameter1, parameter2, sMessage);
}
```

*図 56. subdiagramB でのサービス呼び出し例外の処理*

この例外処理ノードのコードは、次のステップを実行します。

1. currentException システム変数から例外の情報を調べます。

   コードで currentException から例外メッセージと例外タイプを取得し、2 つのストリング変数 (それぞれ sMessage と sType) に保管します。

2. 親ダイアグラム (この場合は subdiagramA) に例外を発生します。

   例外情報を収集したら、コードで raiseException() メソッドを呼び出して、subdiagramA に例外を発生します。この形式の raiseException() メソッドが、3 つのメッセージ・パラメーターを持つ例外タイプとエラー・メッセージ (2345) として、例外情報を受け取ります。これらのメッセージ・パラメーターには、currentException システム変数からコードで取得した例外メッセージが組み込まれています。その後 raiseException() 呼び出しにより、この例外情報を含む例外が作成されます。また、コラボレーションの実行を例外状態に変更します。

   **注:**

   a. raiseException() でメッセージに指定されるメッセージ・パラメーターの数は、コラボレーション・メッセージ・ファイル内の特定のメッセージ・フォーマットにより異なります。

   b. raiseException() メソッドを使用して例外処理コードで例外を発生する場合、例外分岐で例外条件を定義するときの場合と同じ例外タイプを指定できます。例外タイプのリストについては、170 ページの表 44 を参照してください。

raiseException() を実行後、コラボレーション・ランタイム環境で行うアクションは、実行経路を終了する終了ノードにより異なります。実行経路が終了成功ノードで終了すると、コラボレーション・ランタイム環境で次のステップが実行されます。

- subdiagramA に制御を渡します。
- この例外 (実行状態が例外のため) をキャッチする例外分岐で決定ノードを検査します。この決定ノードが subdiagramB のノードに接続し、その例外分岐が該当する例外処理ノードに接続します。

**注:** 実行経路が終了障害ノードで終了すると、コラボレーション・ランタイム環境によりコラボレーション全体が終了します。実行状態が例外であるため、ランタイム環境で、raiseException() 呼び出しを発生する例外の例外情報を使用して、未解決のフローが作成されます。

例外分岐を持つ決定ノードが subdiagramA に存在する場合、コラボレーション・ランタイム環境で各例外分岐の条件が評価されます。この例外条件が true に評価されると、subdiagramA は subdiagramB が発生した例外をキャッチします。コラボレー

ションの実行が通常状態に変更されて例外処理ノードに制御が渡され、このノードで次の raiseException() 呼び出しを使用して親ダイアグラム (メインダイアグラム) に例外を発生して例外を処理します。

```
// exception handling in subdiagramA: raise the exception to main diagram
raiseException(currentException);
```

この形式の raiseException() メソッドは、引き数として受け取る例外オブジェクトを発生するだけです。渡された情報から例外は作成しません。この場合、subdiagramB (179 ページの図 56) 内の例外処理コードで、該当する例外情報を使用して例外が作成済みなので、raiseException() で例外を作成する必要はありません。subdiagramA がその currentException 変数内に持つ例外は、subdiagramB が発生した例外と同じです。raiseException() が完了すると、subdiagramA のロジックに従ってコラボレーションの実行が続行します。subdiagramA の例外処理分岐が終了成功で終了すると、コラボレーション・ランタイム環境は subdiagramA を終了して、その親ダイアグラム (メインダイアグラム) に制御を渡します。したがって、raiseException() で生成される例外オブジェクト (subdiagramA の currentException 例外オブジェクト) は、メインダイアグラムに対して生成されます。

**注:** subdiagramA のこの例外処理分岐が終了障害で終了すると、コラボレーションが終了し、例外メッセージを未解決のフローの一部として含むコラボレーション・ランタイム環境に対して、例外オブジェクトが生成されます。

コラボレーションは、現在、メインダイアグラムの subdiagramA ノードで実行されています。subdiagramA の例外処理ノードで raiseException() を呼び出したので、コラボレーション実行は現在、例外状態です。したがって、コラボレーション・ランタイム環境で、発生した例外をキャッチする例外分岐がないかメインダイアグラムを検査します。これらの例外分岐は、subdiagramA の呼び出しを 1 つ以上の例外処理アクション・ノードに接続する決定ノード内にあります。例外分岐の条件が true と評価されると、メインダイアグラムにより subdiagramA が発生した例外がキャッチされます。コラボレーションの実行は通常状態に変化して、例外処理ノードに制御が渡され、このノードで該当する上位レベルの例外処理ステップを実行できます。

例として、メインダイアグラム内のこの例外処理ノードで次のステップを実行すると仮定します。

1. コラボレーションの SEND_EMAIL 構成プロパティーが、all またはコンマ区切りのメッセージ番号のリストに設定されているかどうかを検証します。

   すべてのコラボレーション・オブジェクトで、logError() がログ宛先に送信するエラーの E メールの受信側を指定できます。コラボレーションが CollaborationFoundation に基づいている場合、SEND_EMAIL コラボレーション・プロパティーに用意されている追加機能を利用できます。コラボレーション・オブジェクトが E メール送信に構成され、さらに SEND_EMAIL が all またはメッセージ番号のリストに設定されている場合、何らかのエラー (SEND_EMAIL が all の場合) または指定されたエラー (SEND_EMAIL でメッセージ番号リストが指定された場合) が発生すると、コラボレーションにより指定された受信側に E メールが送信されます。

これらの条件が一致すると、例外処理ノードにより logError() が呼び出されてエラーがログに記録され、E メールが指定された受信側に送信されます。したがって、コードでまず例外情報を取得して、現行の例外からエラー・メッセージに組み込む必要があります。

**注:** SEND_EMAIL 構成プロパティーは、CollaborationFoundation の機能です。コラボレーションが CollaborationFoundation に基づいている場合、この SEND_EMAIL プロパティーの検査を実行できます。それ以外の場合、この構成プロパティーは定義されません。

2. 例外メッセージをコラボレーションのログ宛先に送信し、必要であれば、E メール・メッセージとして送信します。

   コラボレーション・オブジェクトが E メール送信に構成されている場合、logError() メソッドでエラー・メッセージが指定された E メール受信側に自動的に送信されます。この分岐では logError() メソッドを使用して、コラボレーションのログ宛先 (標準出力またはログ・ファイル) に例外を送信します。

次のメインダイアグラムの例外処理ノードのコード・フラグメントは、これらのステップを実行します。

```
// exception handling in main diagram

// determine if SEND_EMAIL is set to "all" or a message-number list;
// if so, obtain exception information from the current exception
sMessage = currentException.getMessage();
imsgNumber = currentException.getMsgNumber();

// log message and send email
logError(imsgNumber, sMessage, ...);

// raise the exception to collaboration runtime environment
raiseException(currentException);
```

コラボレーション・ランタイム環境でこの raiseException() 呼び出しを実行すると、次のステップが実行されます。

- コラボレーションの実行を例外状態に設定します。
- 実行経路を続行してメインダイアグラムの実行の終了方法を判別します。
- コラボレーションの実行が例外状態なので、コラボレーションが終了すると未解決のフローを作成します。例外から例外メッセージを取得して、それを未解決のフローと関連付けします。
- 未解決のフローを未解決のフロー・キューに送信します。

管理者が未解決のフローを表示すると、Flow Manager ツールはこの未解決フローのメッセージ (subdiagramB で、最初にスローされたときに例外から取得) を表示します。

## 特定のサービス呼び出し例外の処理

コラボレーションがビジネス・オブジェクト要求を宛先アプリケーションに送信する場合、コラボレーション・ランタイム環境で、ServiceCallException の例外タイプでコラボレーション例外をスローして失敗が示されます。しかし、サービス呼び出しの失敗の原因はあいまいな場合があります。サービス呼び出しは、以下の理由により、失敗することがあります。

- アプリケーション関連またはロジック関連: 例えば、コラボレーションが取得しようとしたエンティティーがアプリケーションに存在しない場合、問題が発生することがあります。このような場合、アプリケーションは、要求されたエンティティーを作成または変更しません。
- トランスポート関連: 例えば、コラボレーションとアプリケーション間のビジネス・オブジェクトの転送時に問題が発生することがあります。このような場合、アプリケーションが要求を処理したが、戻り状況の転送がコラボレーションに到達できなかった可能性があります。

このセクションでは、コラボレーション・テンプレートでの次のサービス呼び出しのエラー条件の処理方法について説明します。

- 『サービス呼び出しおよび正確に 1 回のみの要求』
- 184 ページの『未送信サービス呼び出し要求』

## サービス呼び出しおよび正確に 1 回のみの要求

データの重複の可能性は、次のどの状態においても起こりえます。

- アプリケーションからコラボレーションへのビジネス・オブジェクトの転送時にサービス呼び出しの失敗が発生して、コラボレーション・オブジェクトが 1 つ以上のサービス呼び出しの処理を完了したか、まだ処理中である場合、リカバリー・プロセス中に要求が再サブミットされると、データが重複することがあります。
- コラボレーション・オブジェクトがサービス呼び出しを処理中に、InterChange Server ICS がクラッシュした場合。このような場合は、フローは「進行中」とみなされ、リカバリー・プロセスによってシナリオ全体が再実行されます。トランザクション・コラボレーションの場合、差し戻しステップが実行されないと、失敗したサービス呼び出しは実行されません。

トランスポート関連の例外によるデータの重複は、以下の両方の時点で回避する必要があります。

- コラボレーション実行時
- ブート時のリカバリー

次のセクションでは、トランスポート関連の例外の処理方法について説明します。

### 実行時のトランスポート関連例外の処理

コラボレーション実行時に発生するトランスポート障害によるデータの重複を回避するには、サービス呼び出しごとにトランスポート関連の例外とトランスポート関連ではない例外を区別するようにコラボレーション・テンプレートをコーディングします。トランスポート関連の例外を検査するには、ServiceCallException 例外タイプの ServiceCallTransportException サブタイプを使用します。この例外サブタイプは、トランスポートでエラーがあり、要求がアプリケーションに到達できたかどうかを正確に確認できなかったことを示します。

**要確認:** 以前は ServiceCallException の AppUnknown サブタイプによって表されていた例外タイプのサブセットは、ServiceCallTransportException サブタイプによって表されるようになりました。このため、特に AppUnknown サブタイプについてチェックするコラボレーション・テンプレートは、

ServiceCallTransportException についてもチェックする必要があります。AppUnknown サブタイプはトランスポート例外を処理しません。このため、コラボレーション・オブジェクトは、ServiceCallTransportException を除くすべてのサブタイプをチェックする場合、トランスポート例外をトラップしません。

例外分岐のすぐ後のノード (つまり、例外分岐が指すノード) でトランスポート関連例外を処理します。ServiceCallException 例外サブタイプを持つ例外をキャッチする例外処理ノードをコーディングして、宛先アプリケーションから要求ビジネス・オブジェクトを取得して、アプリケーションが正常に作成されたかまたはオブジェクトを変更したどうかを判別する、追加の検索サービス呼び出しを行います。オブジェクトが正常に作成または変更されていなかった 場合は、要求を再試行するようコラボレーションをコーディングします。

次のコード・フラグメントは、トランスポート関連例外についての例外処理を行います。ここでは、getSubType() メソッドを使用して currentException システム変数から例外サブタイプを取得します。この例外サブタイプが ServiceCallTransportException の場合、例外処理コードで検索を実行して、サービス呼び出し要求の結果としてデータがアプリケーションで変更されたかどうかを判別する必要があります。

```
if (currentException.getType().equals(ServiceCallException)) {
    if (currentException.getSubType().equals(
        ServiceCallTransportException))
    {
        //Perform a retrieve to determine whether data changed
    //in the application reflecting the ICS business object request
    }
    else
        raiseException(ServiceCallException, ...);
}
```

**注:** コラボレーションが不確実なネットワークを介して宛先アプリケーションに接続する場合は、トランスポート関連の例外をチェックすることが特に重要です。このような場合、この例外が発生するたびにチェックを行うことが重要です。コードは、トランスポート障害によって発生した例外を特定してチェックできるため、コードがすべての 例外に対してチェックを実行する場合よりもパフォーマンス上の影響は少なくなります。

## ブート時のリカバリー・トランスポート関連例外の処理

コラボレーションのサービス呼び出し処理中に、InterChange Server のクラッシュが原因で発生するデータの重複を回避するには、以下のいずれかの手順を実行します。

- すべてのサービス呼び出しに差し戻しを用意し、コラボレーションをトランザクションにします。
- 非トランザクション・コラボレーションについて「Service Call In-Transit」永続性を構成します。

***トランザクション・コラボレーション:*** InterChange Server がすべてのサービス呼び出しに対して差し戻しを指定するトランザクション・コラボレーションをリカバリーする場合、失敗した要求を再サブミットする前にコラボレーションをロールバ

ックします。このロールバックにより、サーバー・クラッシュの原因によるデータの重複が問題ではなくなります。詳細については、146 ページの『トランザクション機能の使用』を参照してください。

***非トランザクション・コラボレーション:*** 非トランザクション・コラボレーションのリカバリーでは、失敗した要求を再サブミットする前にコラボレーションはロールバックされません。したがって、データの重複が問題となります。非トランザクション・コラボレーションにおけるデータの重複を回避するには、コラボレーション・オブジェクトについて「Service Call In-Transit」永続性を構成します。「コラボレーション・プロパティー」ダイアログで、次のラベルのボックスにチェックマークを付けます。

`Persist Service Call In Transit State`

既存のコラボレーションとの後方互換性を維持する必要があること、およびトランザクション・コラボレーションで各サービス呼び出しの状態を永続的に保管することに利点がない ため、「Service Call In-Transit」永続性のデフォルト設定は、`off` になっています。つまり、コラボレーション・オブジェクトについて「`Persist Service Call In Transit State`」ボックスはチェックされていません。

永続的に構成されているコラボレーション・オブジェクトがサービス呼び出しを入力すると、ICS は、要求が完了するまでこの状態を維持します。コラボレーションによるサービス呼び出しの処理中にサーバーがクラッシュすると、この失敗したサービス呼び出しの状態が、次の状況で Flow Manager に表示されます。

- 「Service Call In Transit」のイベント状況
- 以下のメッセージ

  `Service Call In Transit`

この場合、管理者は、トランスポート障害前にコラボレーション要求が正常に処理されたかどうかを手動で確認する必要があります。要求が成功しなかった 場合、管理者は再サブミットする必要があります。詳細については、「*システム管理ガイド*」を参照してください。

**注:** コラボレーションが「Service Call In-Transit」永続性に構成されていない 場合、フローの障害の原因となるすべての実行時障害および例外は、「未解決のフロー」ビューアーで「失敗」のイベント状況に設定されています。

## 未送信サービス呼び出し要求

アプリケーションに送信されたサービス呼び出し要求を検証するには、コラボレーション・テンプレートで各サービス呼び出しを検査するようにコーディングします。要求が送信されたことを検証するには、`ServiceCallException` 例外タイプの `AppRequestNotYetSent` サブタイプを使用します。並列コネクター・エージェントの場合、この例外サブタイプは、要求がエージェント・マスター内にキューイングされたが、アプリケーションにディスパッチされないため、要求を再送信できることを示します。

例外分岐のすぐ後のノード (つまり、例外分岐が指すノード) で未送信サービス呼び出し例外を処理します。要求を再サブミットするようにコラボレーション・テンプレートをコーディングします。次のコード・フラグメントは、未送信サービス呼び出し要求の例外処理を行います。ここでは、`getSubType()` メソッドを使用して

currentException システム変数から例外サブタイプを取得します。この例外サブタイプが AppRequestNotYetSent の場合、コードでサービス呼び出しを使用してアクション・ノードに戻ってイベントを再サブミットする必要があります。

```
if (currentException.getType().equals(ServiceCallException))
{
    if (currentException.getSubType().equals(
        AppRequestNotYetSent))
    {
    // Resubmit the event by returning execution to the action node
    // with the service call.
    }
    else
        raiseException(ServiceCallException, ...);
}
```

**要確認:** コラボレーション・ランタイム環境では、宛先コネクターの ControllerStoreAndForward コネクター・プロパティーが true に設定されている場合は AppRequestNotYetSent サブタイプに対して値を設定しません。このコネクター・プロパティーが false に設定されている場合、コラボレーションはこのサブタイプを検査し、要求を再送信する必要があります。

## コラボレーション API からの例外

コラボレーション・テンプレートを設計する場合、例外分岐を持つ決定ノードを組み込んでコラボレーション API のメソッドからスローされる例外をキャッチできます。CollaborationException 例外をスローするメソッドの場合、参照説明には「例外」というタイトルのセクションがあり、例外がメソッドからスローされるとこのセクションにリストされます。

# 第 8 章 ワークスペースとレイアウトのオプション

この章では、アクティビティー・ダイアグラムを編集するときのシンボルの調整やワークスペースのカスタマイズで使用するオプションについて説明します。

Process Designer Express メインウィンドウのレイアウトのカスタマイズについては、27 ページの『メインウィンドウのカスタマイズ』 を参照してください。

## シンボルの位置合わせ

位置合わせツールバーの位置合わせ操作 (図 57 を参照) により、複数のシンボルの位置を変更し、指定した端または中央にラインアップします。位置合わせツールバーは、アクティビティー・ダイアグラムで複数のシンボルを選択するとアクティブになります。

位置合わせ操作の手順は、以下のとおりです。

1. 位置合わせの「ベース」 (またはアンカー) として使用するシンボルを選択します。
2. Shift キーを押したまま、最初に位置を合わせるその他のシンボルを選択します。
3. 位置合わせツールバーで、実行する操作をクリックします。図 57 を参照してください。

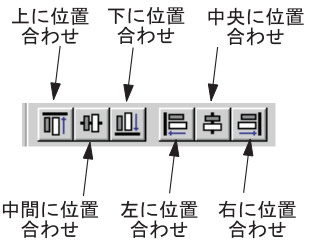



*図 57. 位置合わせツールバー*

### 端の位置合わせ

複数のシンボルの端の位置合わせにより、指定したモデル・シンボルの端に沿ったイメージ線に各シンボルの端の位置を合わせます。端の位置合わせ操作には、「上に位置合わせ」、「下に位置合わせ」、「左に位置合わせ」、および「右に位置合わせ」があります。

例えば、図 58 は、終了成功シンボルとアクション・シンボルの下の位置を合わせた結果を示します。


*図 58. 下部の位置合わせ*

終了成功シンボルの場合、ラベルとシンボルにより、下の位置がアクション・シンボルの下に合わせられたオブジェクトが形成されます。

シンボル集合の上、下、左、または右の端の位置を合わせる手順は、以下のとおりです。

1. その他のシンボルに位置を合わせるシンボル (ベースまたはアンカー) をクリックします。
2. Shift キーを押したまま、1 つ以上の追加シンボルまたはシンボルのグループをクリックします。
3. 位置合わせツールバーで、「上に位置合わせ」、「下に位置合わせ」、「左に位置合わせ」、および「右に位置合わせ」ボタンをクリックします。

   すべてのシンボルが目的の位置にラインアップされます。

## 中央の位置合わせ

選択する最初のシンボルの中央に引かれた水平または垂直のイメージ線に沿ってシンボルを中央にそろえることができます。これにより、各シンボルは、この線に沿って水平または垂直に中央にそろえられます。中央そろえ操作には、「中間に位置合わせ」および「中央に位置合わせ」が含まれます。

図 59 の破線は、「中間に位置合わせ」操作 (2 つのシンボルの垂直中央の位置合わせ) を示します。

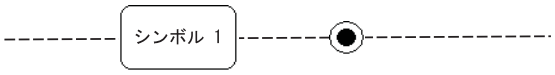



*図 59. 「中間に位置合わせ」操作*

図 60 の破線は、「中央に位置合わせ」操作 (2 つのシンボルの水平中央の位置合わせ) を示します。

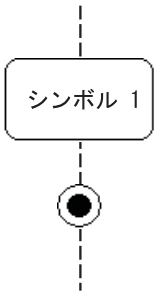



*図 60. 「中央に位置合わせ」操作*

中央をそろえる手順は、以下のとおりです。

1. 中央をベースまたはアンカーとして使用するシンボルまたは事前にグループ化したシンボルの集合を選択します。
2. Shift キーを押したまま、位置を合わせるその他のシンボルまたはシンボルのグループを選択します。
3. 位置合わせツールバーで、「中間に位置合わせ」または「中央に位置合わせ」をクリックします。

例えば、以下の図は、水平中央をそろえる前の 2 つのシンボルを示します。

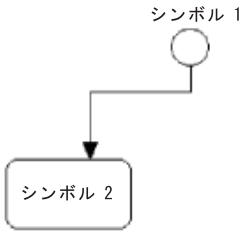



*図 61. 位置合わせされていないシンボル*

下図は、水平中央がそろえられた前図のシンボルを示します。

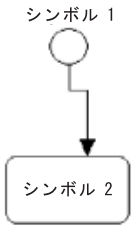



*図 62. 位置合わせされたシンボル*

## シンボルの微調整

微調整ツールバーの「微調整」操作 (図 63 を参照) を使用して、アクティビティー・ダイアグラムで選択したシンボルをわずかに移動できます。微調整ツールバーは、アクティビティー・ダイアグラムでシンボルを選択するとアクティブになります。

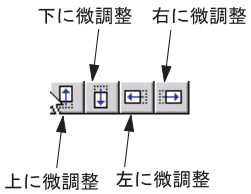



*図 63. 微調整ツールバー*

シンボルをわずかに移動する手順は、以下のとおりです。

1. 移動するシンボルをクリックします。
2. Shift キーを押したまま、移動するその他のシンボルまたはシンボルのグループを選択します。
3. 微調整ツールバーで、「上に微調整」、「下に微調整」、「左に微調整」、または「右に微調整」ボタンをクリックします。

   デフォルトでは、これらのコマンドを使用すると、選択したコンポーネントが指定した方向にピクセル単位で移動します。Shift キーを押したままで操作を行うと、選択したコンポーネントが 5 ピクセル単位で移動します。

## シンボルのズームまたはパン

ズーム/パン・ツールバーの操作 (図 64 を参照) を使用して、アクティビティー・ダイアグラムで選択したシンボルをズームまたはパンできます。ズーム/パン・ツールバーは、アクティビティー・ダイアグラムでシンボルを選択するとアクティブになります。

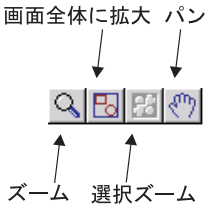



*図 64. ズーム/パン・ツールバー*

ズーム/パン・ツールバーには、以下の操作が用意されています。

- ズーム: この操作を使用すると、ダイアグラムで選択したシンボルにズームインできます。ズームを行うには、ズーム・モードで左マウス・ボタンをクリックして押したままにします。マウス・ボタンを押したままズーム・モードでドラッグすると、ズーム対象の領域を示す長方形が描画されます。ダイアグラムの領域内で位置を指定したら、マウス・ボタンを放してズーム対象の領域を選択します。
- 適合ズーム: この操作により、ダイアグラム内のすべてのシンボルが表示されるようにダイアグラムの拡大率が設定されます。
- 選択ズーム: この操作を使用すると、選択したシンボルをズームできます。シンボルを選択して「選択ズーム」をクリックし、このシンボルがフレームに合うように拡大します。
- パン: この操作を使用して、ダイアグラム内の別の領域に「パン」または移動できます。「パン」をクリックすると、ポインターが手の形に変わります。次に、マウスをクリックしてドラッグし、ダイアグラム内を移動します。

## ワークスペース・グリッドの使用

Process Designer Express では、シンボルの位置を合わせやすいようにダイアグラム・エディターのワークスペースにグリッドが表示されます。「グリッド・プロパティー」ダイアログ (図 65 を参照) では、以下のグリッド・オプションを設定できます。

- グリッドを可視または不可視に設定します。

  グリッドが可視に設定されている場合、シンボルをこのグリッドに手動でラインアップできます。デフォルトでは、グリッドは不可視です。
- 「グリッドに合わせる」をオンにします。
- グリッドの色を設定します。
- グリッドの間隔を設定します。

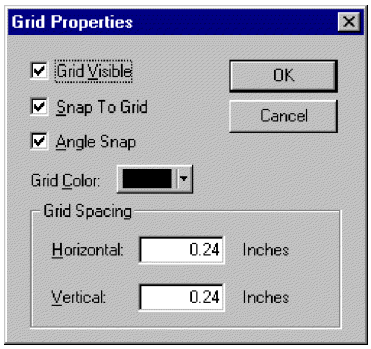



*図 65. 「グリッド・プロパティー」ダイアログ*

グリッド・プロパティーを変更する手順は、以下のとおりです。

1. ダイアグラム・エディター・ウィンドウが開いていることを確認します。
2. 「表示」—> 「グリッド・プロパティー」をクリックして、「グリッド・プロパティー」ダイアログ・ボックスを開きます。
3. 必要により、以下のグリッドの表示を調整します。
   - グリッドが可視と不可視のいずれであるか
   - シンボルがグリッド・ラインに合うようにシンボルの移動を設定するかどうか
   - グリッドの色
   - グリッド間隔

     グリッドの四角のサイズを目的の大きさに調整できます。グリッド・サイズは、グリッドが不可視の場合でも調整できます。「グリッド間隔」に、各グリッドの四角の幅および高さを示す数値をインチ単位で入力します。

   「角度合わせ」を選択しても、アクティビティー・ダイアグラムのシンボルには影響しません。
4. グリッド・オプションを保管する場合は、「OK」をクリックします。変更内容を取り消す場合は、「キャンセル」をクリックします。いずれのオプションを実行しても、ダイアログが閉じられます。

また、「表示」メニューから以下のグリッド・オプションを直接制御することもできます。

- 「グリッド」オプションにより、グリッドの可視性を制御します。
- 「グリッドに合わせる」オプションにより、シンボルをグリッド・ラインに移動するかどうかを制御します。

## 表示の変更: ユーザー設定

Process Designer Express には、表示方法を変更するための方法が複数用意されています。「表示」—> 「設定」をクリックするか、またはショートカット・キーの組み合わせ Ctrl+U を使用して、「ユーザー設定」ウィンドウにアクセスします。

「ユーザー設定」ウィンドウには、以下のカスタマイズを行うことができる 3 つのタブがあります。

- 『一般的な表示の変更』
- 193 ページの『ダイアグラムの表示の変更』
- 194 ページの『シンボルおよびリンクの色の変更』

### 一般的な表示の変更

「ユーザー設定」ウィンドウの「一般」タブを使用して、以下を行うことができます。

- Process Designer Express の作業域の下部にあるワークブック・タブの表示を使用可能または使用不可にします。 18 ページの図 5 に、3 つのワークブック・タブが表示された作業域を示します。

  **注:** この変更は、Process Designer Express を再始動しないと有効にはなりません。
- 内容の検証を使用可能または使用不可にします。テンプレートが破壊されてしまったが、いずれにしてもこれを検査したい場合は、内容の検証を使用不可にすると役に立ちます。ただし、内容の検証を使用不可にしても、Process Designer Express が破壊されたテンプレートを開くことができるとは限りません。
- テンプレートの内容の圧縮を使用可能または使用不可にします。このオプションは、ユーザーが構成することはできません。このオプションは、現在の圧縮設定を表示するために用意されています。内容を圧縮することにより、クライアントとサーバー間のデータ転送速度およびストレージ要件が向上します。
- 「テンプレート定義」ウィンドウの「宣言」タブの標準インポート・セクションおよび「ポートおよびトリガー・イベント」セクションを表示または非表示にします。
- 「テンプレート定義」ウィンドウの「宣言」タブの各セクションと「シナリオ定義」ダイアログ・ボックスの「シナリオ変数」ウィンドウの色を設定します。

図 66 に、「ユーザー設定」ウィンドウの「一般」タブを示します。

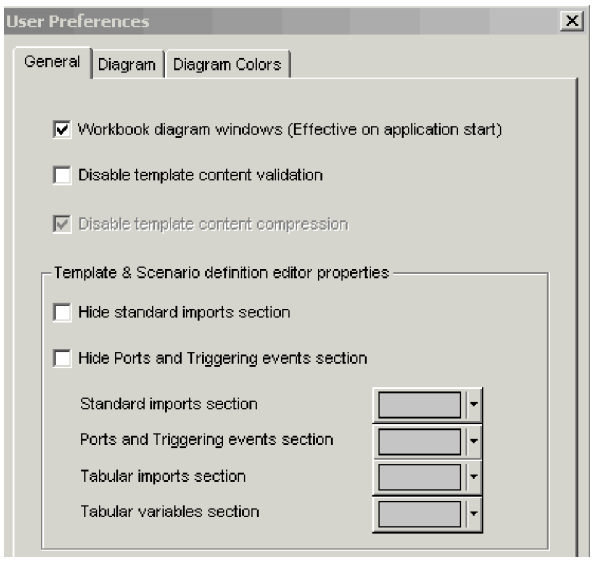



*図 66. 「ユーザー設定」ダイアログ・ボックス: 「一般」タブ*

## ダイアグラムの表示の変更

「ユーザー設定」ウィンドウの「ダイアグラム」タブを使用して、以下を行うことができます。

- ブランクの新しいダイアグラムでの開始ノードの自動表示を使用可能または使用不可にします。この表示を使用不可にすると、開発者が各ダイアグラムに開始ノードを手動で追加する必要があります。
- アクティビティー・ダイアグラムのノード上にある接続点の表示を使用可能または使用不可にします。接続点を表示することにより、2 つのノード間に遷移リンクを追加しやすくなります。
- 決定ノードの未使用分岐の表示を使用可能または使用不可にします。
- 「Action Properties」ダイアログ・ボックスでコード・フラグメントのインプレース編集を使用可能にします。これらのオプションが使用可能であると、Activity Editor を使用しないで、「Action Properties」ダイアログ・ボックスでコード・フラグメントを直接編集できます。

図 67 に、「ユーザー設定」ウィンドウの「ダイアグラム」タブを示します。

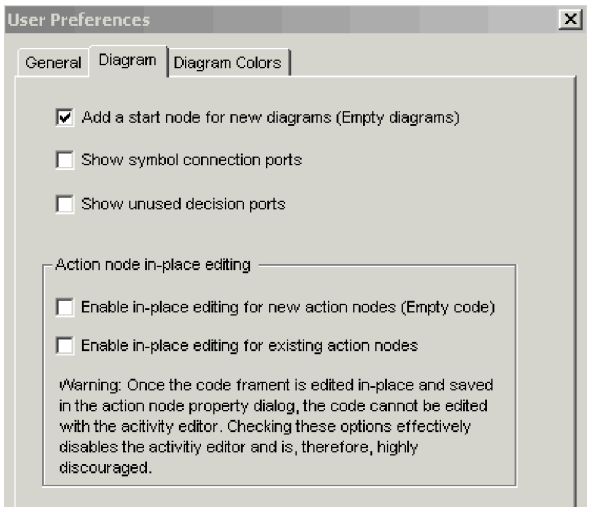



*図 67. 「ユーザー設定」ダイアログ・ボックス: 「ダイアグラム」タブ*

## シンボルおよびリンクの色の変更

「ユーザー設定」ウィンドウの「ダイアグラムの色」タブを使用して、アクティビティー・ダイアグラムのシンボルおよびリンクの表示色を変更できます。各シンボルまたはリンクについて、「塗りつぶし」または「行」色フィールドの矢印をクリックし、ドロップダウン・オプションから目的の色を選択します。図 65 に、「ダイアグラムの色」設定ウィンドウを示します。

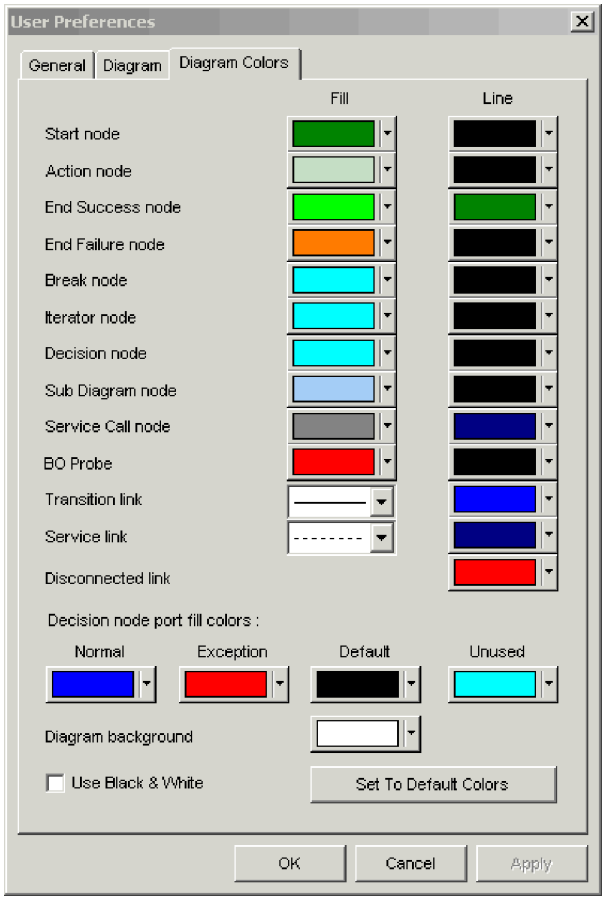



*図 68. 「ユーザー設定」ダイアログ・ボックス: 「ダイアグラムの色」タブ*

## 「Symbol Properties」ダイアログ・ボックスの非表示

シンボルをダブルクリックすると、そのプロパティーが小さいダイアログに表示されます。別のシンボルをクリックすると、新しく選択したシンボルのプロパティーがダイアログの内容として表示されます。

「Symbol Properties」ウィンドウを表示させたくない場合は、これを閉じます。

# 第 9 章 メッセージ・ファイルの作成

コラボレーションは、特定のメソッドを使用してメッセージを作成します。コラボレーションがユーザーに表示されるメッセージ・テキストを作成する方法には、以下の 2 通りがあります。

- コラボレーションは、メッセージ表示用のメソッドを呼び出し、メッセージ・テキストを呼び出しのパラメーターとして組み込みます。
- コラボレーションはメッセージング・メソッドを呼び出します。この呼び出しには、メッセージ・テキストを含む外部メッセージ・ファイルへの参照が含まれます。

通常、テキスト自体を生成するより、コラボレーションにメッセージ・ファイルを参照させる方が効率的です。メッセージを個別のコラボレーションではなく中央メッセージ・ファイルに保管することにより、メンテナンス、管理、および国際化対応が容易になります。

各コラボレーション・テンプレートにメッセージ・ファイルを作成することをお勧めします。Process Designer Express には、メッセージの作成を簡単に行うために「メッセージ」ビューが用意されています。InterChange Server Express は、コラボレーション・オブジェクトを始動するときに、関連メッセージ・ファイルをメモリーにロードしようとします。このメッセージ・ファイルがない場合は、警告がログに記録されます。

この章では、メッセージ・ファイルの機能と、その作成および保守方法について説明します。内容は、次のとおりです。



## メッセージ・ファイルを使用する操作

メッセージ・ファイルには、さまざまなタイプの操作で使用されるメッセージのテキストを格納できます。表 46 に、メッセージ・ファイルを使用する操作のタイプと、これらの操作を実行する BaseCollaboration クラスのメソッドを示します。

*表 46. メッセージ生成操作*

| 操作 | メソッド |
|---|---|
| ロギング | BaseCollaboration.logInfo()<br>BaseCollaboration.logError()<br>BaseCollaboration.logWarning() |
| トレース | BaseCollaboration.trace() |
| 例外の生成 | BaseCollaboration.raiseException() |

## メッセージ・ファイルの作成

メッセージ・ファイルを作成するには、次の手順で行います。

1. Process Designer Express が開かれていることを確認します。
2. 「テンプレート」—> 「テンプレート・メッセージを開く」をクリックします。図 69 に示すように、「テンプレート・メッセージ」ウィンドウが表示されます。

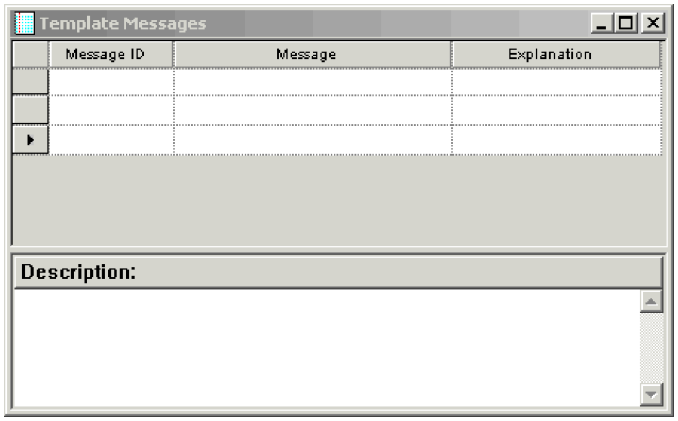



*図 69. 「テンプレート・メッセージ」ウィンドウ*

3. 作成する各メッセージについて、以下を実行します。
   a. 「メッセージ ID」列で、メッセージの固有 ID を指定します。
   b. 「メッセージ」列で、メッセージのテキストを入力します。これが、実行時にメッセージが表示される際に、ユーザーに表示されるテキストです。メッセージ・テキスト内にパラメーターを組み込んで、メッセージを再使用できます。 200 ページの『メッセージ・パラメーター』を参照してください。
   c. 必要により、「説明」列に説明を追加してメッセージを分かりやすくできます。 199 ページの『説明』を参照してください。
   d. 必要により、「説明」ペインにメッセージの説明を追加します。ここに入力されたテキストは、実行時にユーザーに表示されません。

## メッセージ・ファイル: 名前と位置

コラボレーション・メッセージ・ファイルの内容は、コラボレーション・テンプレートの一部として保管されます。コラボレーションをコンパイルし配置する場合、Process Designer Express によりメッセージの内容が抽出されて、実行時に使用するメッセージ・ファイルが作成または更新されます。コンパイル後、メッセージ・ファイルは System Manager の統合コンポーネント・ライブラリー・プロジェクトの ¥Templates¥messages ディレクトリーに格納されます。配置後、メッセージ・ファイルはコピーされて *productDir*¥collaborations¥messages ディレクトリーに配置されます。

メッセージ・ファイルの名前のフォーマットは、以下のとおりです。

*CollaborationName*.txt

例えば、SampleHello というコラボレーション・テンプレートをコンパイルし配置すると、Process Designer Express は SampleHello.txt というメッセージ・ファイルを作成して、それを collaborations¥messages サブディレクトリーに格納します。

**要確認:** コラボレーション・メッセージ・ファイルは直接変更しないでください。常に Process Designer Express を使用して、テンプレート・メッセージを編集してください。

コラボレーション・メッセージ・ファイルには、コラボレーションで使用されるすべてのテキスト・ストリングが含まれます。これらのストリングには、ロギング、例外処理、および E メール操作に関するストリングが含まれます。

**注:** InterChange Server Express 標準では、トレース・メッセージは通常エンド・ユーザーに表示されないので、コラボレーション・メッセージ・ファイルにトレース・メッセージを含めない ことをお勧めします。

国際化コラボレーションの場合、このメッセージ・ファイルを翻訳できるように、これらのテキスト・ストリングがコラボレーション・メッセージ・ファイル内で分離されている必要があります。翻訳されたコラボレーション・メッセージ・ファイルの名前に、関連するロケールの名前 (*CollaborationName_ll_TT*.txt) が含まれている必要があります。

前の行の *ll* は、ロケールの 2 文字の省略形 (小文字で表記) で、*TT* は 2 文字の地域の省略形 (大文字で表記) です。例えば、米国英語メッセージを含む SampleHello コラボレーションのメッセージ・ファイルのバージョンは、SampleHello_en_US.txt という名前です。

実行時に、コラボレーション・ランタイム環境が collaborations¥messages サブディレクトリーからコラボレーション・ロケール (InterChange Server から継承) に該当するメッセージ・ファイルを探します。例えば、コラボレーション・ロケールが米国英語 (en_US) の場合、コラボレーション・ランタイム環境により *CollaborationName*_en_US.txt ファイルからメッセージが検索されます。

コラボレーションのテキスト・ストリングの国際化の方法の詳細については、72 ページの『国際化対応コラボレーション』を参照してください。

## 説明

「テンプレート・メッセージ」ウィンドウの「説明」列を使用して各メッセージの詳細説明を追加し、分かりやすいメッセージを作成します。説明は任意ですが、コラボレーションの使用可能度を向上できます。

例えば、「`Update failed. Destination application missing entry for {1} {2}`」というテキストを持つメッセージがあると仮定します。このメッセージ・テキストは、ユーザーがエラーを修正するのに十分な詳細説明をしていません。この例では、次のようなメッセージの説明を追加すると、ユーザーにとってメッセージの価値が大幅に向上します。

```
An update request was sent to a connector, which
successfully contacted the application. However, the application
did not return data for the specified key attribute value.
```

分かりやすいメッセージのその他の例については、Business Integration Express と共にインストールされる InterchangeSystem.txt ファイルを参照してください。

### メッセージの説明の読み取り

説明は、実行時にメッセージと共に表示されません。代わりに、コラボレーションのメッセージ・ファイルで参照する必要があります。

InterChange Server Express ログがファイルに保管されている場合は、Log Viewer を使用してログ・ファイルに書き込まれているメッセージ説明を参照します。ログがファイルに保管されていない場は、メッセージ説明はコラボレーションのメッセージ・ファイルを直接参照する必要があります。

## メッセージ・パラメーター

考えられるすべての状況ごとに別のメッセージを作成する必要はありません。代わりに、パラメーターを使って実行時に変化する値を表します。パラメーターを使うことにより、1 つのメッセージで複数の状況に対処することが可能になり、メッセージ・ファイルの増大を防ぐことができます。

パラメーターは常に、中括弧で囲まれた数値 ({*number*}) として表示されます。メッセージに追加したいパラメーターごとに、中括弧で囲んだ数字を、メッセージのテキストに挿入してください。次に例を示します。

```
message text {number} more message text.
```

メッセージのロギングのために呼び出される API メソッドは、各パラメーターに値を設定する必要があります。例えば、以下のメッセージを例に取ります。

```
6
Update failed. Destination application missing entry for {1} {2}
```

このメッセージを送信するコード・フラグメントには、次のようなコードがあります。

```
logWarning(6, " CustomerID" , fromCust.getString("CustomerID"));
```

InterChange Server Express は、logWarning() メソッド呼び出しで指定されたパラメーター値をログ・ファイル内のメッセージと組み合わせ、メッセージを形成します。メッセージをログ・ファイルに書き込む前に、サーバーはメッセージ・パラメーターを以下の値で置換します。

- パラメーター 1 は、ストリング「Customer ID」になります。
- パラメーター 2 は、fromCust ビジネス・オブジェクトのカスタマー ID 属性の値になります。

メッセージは、以下のようにログ・ファイルに表示されます。

```
Update failed. Destination application missing entry for CustomerID 101961
```

メッセージ・テキストでは、欠落したエントリーの説明およびその ID がパラメーターとして使用されます。そのため、ハードコーディングされたストリングとしてこれらを組み込む代わりに、欠落した属性の任意のペアに対して同じメッセージを使用できます。

## ファイルの保守

ユーザー・サイトで、管理者は、コラボレーション・メッセージをフィルター操作して、E メールまたは E メール・ページャーを使って問題を解決する能力を持つ人に通知するためのプロシージャーを設定できます。そのため、エラー番号とその番号に対応する意味がコラボレーション・テンプレートのリリースの前後で変わらないようにすることが大切です。エラー番号に対応するテキストを変更することはできますが、テキストの意味を変更したりエラー番号を再度割り当てることは推奨できません。

エラー番号に対応する意味を変更する場合は、必ずその変更を文書化してコラボレーション・テンプレートのユーザーに通知してください。

コラボレーション・オブジェクトの実行中にコラボレーションのメッセージ・ファイルを変更することは可能です。ただし、この変更は、次にコラボレーション・オブジェクトが起動してそのメッセージ・ファイルがメモリーに読み込まれるまでは有効になりません。コラボレーションの実行中に InterChange Server Express に障害が発生すると、サーバーは自動的に、以前に実行中であったすべてのコラボレーションのメッセージ・ファイルをメモリーに読み込みます。

# 第 10 章 コーディングのヒントと例

この章では、特定のタイプのコラボレーション操作をプログラムする方法について説明します。

## コラボレーションに対する操作

このセクションでは、コラボレーション全体に影響する操作について説明します。これには、以下の操作が含まれます。

- 『メッセージのロギング』
- 205 ページの『トレース・メッセージの追加』
- 208 ページの『コラボレーション構成プロパティーの検索』
- 208 ページの『コラボレーション・オブジェクト・インスタンスの再利用』
- 210 ページの『ネイティブ・マップの呼び出し』

各コラボレーション・テンプレートには、メッセージ・ファイルが関連付けられている必要があります。メッセージ・ファイル には、コラボレーションの例外およびメッセージのロギングのためのテキストが含まれています。メッセージ・ファイル内の各メッセージは、固有 ID によって識別されます。また、メッセージのテキストには、プレースホルダー変数も含まれます。

コラボレーションは、特定のメッセージを表示するメソッドを呼び出すときに、メッセージの識別番号および (場合によっては) 追加パラメーターをこのメソッドに渡します。このメソッドはこの識別番号を使用して、メッセージ・ファイルから正しいメッセージを探し出し、追加パラメーターの値をメッセージ・テキストのプレースホルダー変数に挿入します。

例えば、あるコラボレーションのメッセージ・ファイルに番号 23 のメッセージが含まれ、このテキストには {1} および {2} と表示された 2 つのプレースホルダー変数が含まれるとします。

```
23
Customer ID {1} could not be changed: {2}
```

このコラボレーションは、このメッセージを表示または記録する場合、`raiseException()` など適切なメソッドに対して、メッセージ (23) の識別番号、2 つの追加パラメーター、顧客 ID 番号 (6701)、および `greater than maximum length` などの追加説明テキストを含む String 変数を渡します。このメソッドは、正しいメッセージを探し出し、メッセージのプレースホルダーのパラメーター値を置換し、以下のメッセージを表示または記録します。

```
Customer ID 6701 could not be changed: greater than maximum length
```

### メッセージのロギング

コラボレーション・テンプレートは、管理者に関係があると考えられる事態が発生するたびにメッセージをログに記録できます。メッセージをログに記録するには、コラボレーション・テンプレートの logInfo()、logWarning()、および logError() メソ

ッドを使用します。各メソッドには、さまざまなメッセージの重大度レベルが関連付けられています。表 47 に、重大度レベルおよびそれに関連したメソッドを示します。

*表 47. ログ・メソッドのメッセージ・レベル*

| メソッド | 重大度レベル | 説明 |
|---|---|---|
| `logInfo()` | 情報 | 情報のみ。ユーザーは何もする必要はありません。 |
| `logWarning()` | 警告 | 問題に関する情報を示します。このレベルは、ユーザーが解決する必要がある問題には使用しないでください。 |
| `logError()` | エラー | ユーザーが調べる必要がある重要な問題を示します。 |

これらのメソッドがログ宛先に送信するメッセージ・テキストは、接頭部に重大度レベルが付いています。

このセクションでは、ロギング・メッセージについて、以下の内容を説明します。

- 『メッセージ・ファイルの使用』
- 『適切なメッセージ・ロギングの原則』

## メッセージ・ファイルの使用

すべてのコラボレーション・テンプレートには、ログ・メッセージを保持するメッセージ・ファイルが必要です。コラボレーションがエラーをログに記録する場合、エラー・メッセージのテキストにはコラボレーションのメッセージ・ファイルが使用されます。次の例では、コラボレーションのメッセージ・ファイルにテキストが格納されているエラー・メッセージをログに記録します。

```
logError(10, customer.get("LName"), customer.get("FName");
```

エラー・メッセージ 10 のテキストには、2 つのメッセージ・パラメーターがあり、以下のようにメッセージ・ファイルに表示されます。

```
10
Credit report error for {1} {2}.
```

logError() メソッドが実行されると、メッセージ・ファイルからメッセージ 10 のテキストを取得し、メッセージ・パラメーター 1 および 2 で顧客のラストネームおよびファーストネームを置換して、メッセージの接頭部に重大度「Error」を付加します。その後、このエラー・メッセージがコラボレーションのログ宛先に書き込まれます。

例えば、顧客名 John Davidson のログに記録されたメッセージは、次のようになります。

```
Error: Credit report error for Davidson John.
```

コラボレーションが E メール通知に構成されていると、logError() はこのエラー・メッセージを指定された E メール受信側 (複数の場合あり) にも送信します。メッセージ・ファイルの設定方法の詳細は、197 ページの『第 9 章 メッセージ・ファイルの作成』を参照してください。

## 適切なメッセージ・ロギングの原則

メッセージを作成する場合、管理者がロギング機能を使用する方法に注意してください。

***重大度レベルの割り当て:*** メッセージに対するエラー・レベルの割り当ては、厳密に行う必要があります。 InterChange Server の E メール通知機能では、エラー・メッセージまたは致命エラー・メッセージの生成が検出されると、指定された人物 (通常は管理者) にメッセージが送信されます (`logError()`)。管理者は、この InterChange Server の E メール通知機能を使用すると共に、この機能を E メール・ページャーにリンクして、エラーが発生した場合にメッセージを送信することもできます。エラー・レベルをメッセージに正確に割り当てることにより、重大メッセージの数を減らすことができます。

***メッセージの変更:*** 例えば、テキストを明確にしたり、拡張するために、メッセージのテキストをいつでも変更できます。ただし、特定のタイプのエラーにメッセージ番号を割り当てたら、その番号を再度*割り当てない* ことが重要です。多くの管理者はスクリプトを使用してログ・メッセージをフィルター操作しています。ただし、これらのスクリプトはメッセージ番号に依存しています。このため、メッセージ・ファイルの番号の意味を変更しないことが重要です。番号が変更されると、ユーザーがメッセージを喪失したり、間違ったメッセージを受信することがあります。

***情報メッセージを使用する場合:*** `logInfo()` メソッドを使用して、ユーザー独自のデバッグ用一時メッセージを作成できます。ただし、開発が完了したら、これらのデバッグ・メソッド呼び出しは除去するようにしてください。

コラボレーションの通常操作を文書化する場合は、`logInfo()` メソッドを使用しないようにしてください。このメソッドを使用すると、管理者のログ・ファイルが関係のないメッセージでいっぱいになってしまいます。代わりに、`trace()` メソッドを使用して、管理者用のデバッグの詳細情報を記録してください。

## トレース・メッセージの追加

トレース・メッセージをコラボレーション・テンプレートに追加することにより、コラボレーション・オブジェクトの実行時にそのアクションの詳細な説明が生成されるようにすることができます。トレース・メッセージは、ユーザー独自のデバッグや、管理者によるオンサイトのトラブルシューティングに役立ちます。

トレース・メッセージは、そのトレース・メッセージに含まれるログ・メッセージとは異なり、デフォルトでは削除可能となっています。ただし、そのログ・メッセージを削除することはできません。通常、トレース・メッセージの方がより詳細で、一定の状況下でのみ表示されるようになっています。例えば、あるユーザーがコラボレーション・オブジェクトのトレース・レベルを意図的にゼロより大きい番号に構成した場合などがあります。トレース・メッセージとログ・メッセージは別のファイルに送信できます。

コラボレーションには、以下の 2 つのタイプのトレース・メッセージがあります。

- コラボレーション生成トレース・メッセージ。これは、コラボレーション・テンプレートにコーディングします。
- コラボレーション・ランタイム環境から出される InterChange Server Express 生成またはシステム生成のトレース・メッセージ

System Manager の「Collaboration Object Properties」ダイアログ・ボックスを使用して、両タイプのトレース・メッセージのトレース・レベルを設定します。

コラボレーション・テンプレート開発者は、コラボレーション生成トレースを要求できるレベルを作成します。これについては、次のセクションで説明します。システム生成トレース・レベルは、すべてのコラボレーション・オブジェクトに対して同じです。これについては、207 ページの『InterChange Server Express 生成トレース・メッセージ』を参照してください。

## コラボレーション生成トレース・メッセージ

トレース・メッセージをコラボレーション・テンプレートに追加して、そのコラボレーションに固有の操作を報告できます。以下は、コラボレーションがトレース・ファイルに書き込むことができる情報の例です。

- コラボレーションが特定のアクション・ノードに対して入出力されるときのビジネス・オブジェクトのキー値
- コラボレーションが更新オプションを検索するときのこのオプションの値
- 実行経路で特定の分岐が採用する決定内容
- サービス呼び出しの結果の例外コード
- ビジネス・オブジェクトが特定のアクション・ノード、イテレーター、またはサブダイアグラムに対して入出力されるときのビジネス・オブジェクトの各属性値

***トレース・レベルの割り当て:*** 各トレース・メッセージには、1 から 5 までの間の**トレース・レベル**を割り当てる必要があります。通常、トレース・レベルには、レベルとその詳細の度合いに相関関係があります。通常は、レベル 1 よりレベル 2 の方が詳細であり、レベル 2 よりレベル 3 の方が詳細というようになります。このため、レベル 1 のトレースをオンにした場合、レベル 5 のメッセージより詳細ではないメッセージが表示されます。ただし、レベルは必要なレベルに設定できます。以下は、レベルを割り当てる際の提案事項です。

- すべてのトレース・メッセージに対して同じレベルを割り当てることができます。
- トレース・レベルは、コラボレーション・ランタイム環境の場合と同じように、詳細のレベルに応じて割り当てることができます。
- 各メッセージ・レベルは、関連するビジネス・オブジェクトに応じて割り当てることができます。例えば、レベル 1 のトレース・メッセージはあるビジネス・オブジェクトに関連付け、レベル 2 のトレース・メッセージは別のビジネス・オブジェクトに関連付けて割り当てることができます。

特定のレベルでのトレースをオンにすると、特定のレベルに関するメッセージと、これ以下のレベルに関するメッセージが表示されます。例えば、レベル 2 のトレースの場合は、レベル 2 とレベル 1 に関するメッセージが表示されます。

**ヒント:** トレース・レベルは、文書に記録して、ユーザーがトレースを行うときに使用するレベルがわかるようにしてください。

***トレース・メッセージの生成:*** 以下は、メッセージと、メッセージを生成するメソッドの呼び出しの例です。メッセージは、以下のようにメッセージ・ファイルに表示されます。

```
20
Configuration property DO_VERIFICATION = {1}
```

このメソッドの呼び出しは、構成プロパティー DO_VERIFICATION の値を取得し、この値を使用してメッセージのパラメーターを置換します。コードは、以下のようにコラボレーションに表示され、このメッセージは、ユーザーがトレースをレベル 3 に設定したときに表示されます。

```
String validateProp = getConfigProperty("DO_VERIFICATION"); trace(3, 20,
validateProp);
```

以下の例では、Employee ビジネス・オブジェクトの Salary 属性が取得され、給与の額に基づいて分岐の決定が実行されます。メッセージ・ファイルのメッセージは、以下のとおりです。

```
15
Salary {1} {2}
```

この例では、給与の額と実行された経路が記述されたトレース・メッセージが送信されます。

```
int newsalary = employee.getInt("Salary"); String sal =
Integer.toString(newsalary); if (newsalary <150000)     {      trace (3, 15, sal,
"do extra check");     } else     {      trace (3, 15, sal, "take normal path");
    }
```

## InterChange Server Express 生成トレース・メッセージ

InterChange Server Express のコラボレーション・ランタイム環境には、コラボレーションの実行に関するメッセージを提供するトレース・コンポーネントが用意されています。

ランタイム環境のトレース・コンポーネントは、6 つの番号を使用してトレース・レベルを示します。第 1 レベルのゼロ (デフォルト設定) は、トレースが行われていないことを示します。レベル 1 から 5 は、それぞれ詳細のレベルを示します。レベル 1 のトレースは最小限の詳細レベルを示し、レベル 5 は最も高い詳細レベルを示します。

トレースをオンにするには、コラボレーション・オブジェクトのトレース・レベルをゼロより大きい値にします。各トレース・メッセージには、個別のレベルが関連付けられています。表 48 に、各レベルで表示されるメッセージのタイプを示します。

*表 48. システム生成トレースのトレース・レベル*

| レベル | InterChange Server Express システム生成トレース |
|---|---|
| 0 | なし。 |
| 1 | トリガー・イベントの受信およびシナリオの開始。 |
| 2 | シナリオの開始および完了 (順方向実行およびロールバックの報告)。 |
| 3 | アクション・ノードの実行。 |
| 4 | サービス呼び出しのビジネス・オブジェクトの送信および応答の受信。 |
| 5 | レベル 4 のトレースの詳細バージョン。このレベルのトレースでは、送受信されるビジネス・オブジェクトの各属性値が出力されます。 |

## コラボレーション構成プロパティーの検索

コラボレーションの構成プロパティーを検索するには、getConfigProperty() メソッドを使用します。

以下の例は、コラボレーションが構成プロパティーを使用してコード経路を決定する方法を示します。

```
if (getConfigProperty("CONVERT_NEGQTY").equals("true")) { // take this code path }
else { // take this code path }
```

構成値を特定の値と比較する場合は、常に equals() メソッドを使用してください (例を参照)。条件等価演算子 == は使用しないでください。この演算子を使用すると、同じ値に 2 つのオブジェクトが含まれるかどうかではなく、2 つの変数が同じオブジェクトを参照しているかどうかがテストされます。

値は大文字小文字が区別されるので注意してください。構成パラメーターの大文字小文字の区別は、等価性についてテストするコードと同じである必要があります。上記の例では、True の値により、比較が失敗します。

また、構成プロパティーには、セミコロンで区切った値の配列を使用することもできます。詳細については、364 ページの『getConfigPropertyArray()』を参照してください。

## コラボレーション・オブジェクト・インスタンスの再利用

通常、InterChange Server Express は、各トリガー・イベントを処理するためにコラボレーション・オブジェクトのインスタンスを作成します。インスタンスがトリガー・イベントの処理を完了すると、そのリソースは解放され、Java 空きプールに戻されます。ただし、JDK では、これらのインスタンスが必ずしも効率的にクリーンアップされるとは限らないため、メモリーが過度に使用される場合があります。

InterChange Server Express では、メモリーの使用量を減らすために、「Collaboration Instance Reuse」オプションが使用されます。これにより、コラボレーション・オブジェクトのインスタンスをキャッシュに格納し、同じタイプのコラボレーション・オブジェクトのインスタンスを後で生成するときにそれを再使用して、コラボレーション・オブジェクトのインスタンスを再生することができます。InterChange Server Express で既存のコラボレーション・インスタンスを再生すると、次のことを回避できます。

- コラボレーション・オブジェクトのインスタンス生成のオーバーヘッド
- メモリー管理用の JDK ガーベッジ・コレクターへの依存

「Collaboration Instance Reuse」オプションは、コラボレーション・テンプレートが以下の*両方の* 要件を満たす限り、自動的に使用されます。

- コラボレーションには、テンプレート (グローバル) 変数は含まれません。
- コラボレーションが、Process Designer Express のバージョン 3.0 *より後の* バージョンを使用してコンパイルされている。

これらの条件のいずれかが*満たされない* 場合、「Collaboration Instance Reuse」オプションは使用されません。このため、このオプションを利用するには、コラボレー

ション・テンプレート・コードでテンプレート (グローバル) 変数を使用しないようにしてください。テンプレート変数はユーザーが宣言し、その有効範囲はコラボレーション・テンプレート全体です。テンプレート変数は、「定義」ウィンドウの「宣言」タブにある「グローバル変数:」というラベルが付いた領域で宣言します。

コラボレーションにテンプレート変数が必要だが、「Collaboration Instance Reuse」オプションを使用したい場合は、コラボレーション・テンプレートが以下のプログラミング要件を満たすようにしてください。

- テンプレート変数は宣言時に初期化しないようにしてください。代わりに、テンプレート変数は常にコラボレーション・テンプレートの第 1 ノードで初期化してください。
- コラボレーション・テンプレートがテンプレート変数を使用して大きいオブジェクトを参照する場合は、以下の状況になる前にこの変数を `null` にリセットしてください。
  - コラボレーションが終了する。
  - 例外がスローされる。

**要確認**

第 1 ノードで初期化されていない テンプレート変数を含むコラボレーション・テンプレートは、安全に再生できません。これは、キャッシュに格納されているコラボレーション・オブジェクト・インスタンスの変数値が再使用される場合、このインスタンスが永続的であるためです。キャッシュに格納されているコラボレーション・インスタンスが再使用され、実行が開始されると、各テンプレート変数には、コラボレーション・インスタンスの前回の使用時の終わりの値が含まれます。

テンプレート変数が正しく初期化されるようコラボレーション・テンプレートをコーディングしたら、以下の操作を実行して「Collaboration Instance Reuse」オプションを使用可能にします。

1. `EnableInstanceReuse` というコラボレーション固有の構成プロパティーを定義し、そのデフォルト値を `true` または `false` に設定します。

   「テンプレート定義」ウィンドウで、コラボレーション固有の構成プロパティーを定義します。コラボレーション・オブジェクトに求める振る舞いに応じて、`EnableInstanceReuse` のデフォルト値を設定します。
   - コラボレーション・テンプレートの全 コラボレーション・オブジェクトのインスタンスを再生させるには、`EnableInstanceReuse` のデフォルト値を `true` に設定します。
   - コラボレーション・テンプレートの特定のコラボレーション・オブジェクトのみのインスタンスを再生させるには、`EnableInstanceReuse` のデフォルト値を `false` に設定します。
2. 再生する各コラボレーション・オブジェクトの `EnableInstanceReuse` コラボレーション・プロパティーが `true` に設定されていることを確認します。

コラボレーション固有の構成プロパティーの値は、System Manager の「Collaboration Object Properties」ウィンドウの「プロパティー」タブで設定します。詳細については、「*システム管理ガイド*」を参照してください。

前のプログラミング要件を満たすようコラボレーションをコーディングできない場合は、「Collaboration Instance Reuse」オプションを使用しない でください。このオプションを使用できないようにするには、コラボレーション・テンプレートに `EnableInstanceReuse` コラボレーション構成プロパティーを定義しないでください。

**注:** 「Collaboration Instance Reuse」オプションをアクティブにするには、コラボレーション・テンプレートを停止してから始動する必要があります。このオプションは、コラボレーションを再活動化しないとアクティブにはなりません。

ソフトウェアはコラボレーション・インスタンス・キャッシュ というキャッシュを使用して、コラボレーション・オブジェクトのインスタンスを保持します。コラボレーション・インスタンス・キャッシュのサイズは、「並行イベントの最大数」の値から算出されます。この値は、System Manager の「Collaboration Object Properties」ウィンドウの「一般」タブで構成します。コラボレーション・インスタンス・キャッシュのサイズは、コラボレーションを実行するフロー処理モデルがイベントと呼び出しのどちらを使用して起動されるかに応じて、変更する必要があることがあります。

コラボレーション・インスタンス・キャッシュのサイズの変更には、`CollaborationInstanceCacheSize` と呼ばれるコラボレーション構成プロパティーの定義が含まれます。このプロパティーはその他のコラボレーション・プロパティーと共に、「テンプレート定義」ウィンドウの「一般」タブにある「プロパティー」というラベルが付いた領域で定義します。 `CollaborationInstanceCacheSize` を定義したら、その値をコラボレーション・インスタンス数のデフォルト値に適した値に設定します。詳細については、「*システム管理ガイド*」の「Collaboration Instance Reuse」オプションに関する説明を参照してください。

## ネイティブ・マップの呼び出し

通常、マップおよびサブマップの呼び出しはマップ内でのみ行われます。ただし、場合によっては、コラボレーションは InterChange Server Express ネイティブ・マップを直接呼び出す必要がある場合があります。コラボレーションからネイティブ・マップを呼び出す場合、以下のようないくつかの利点があります。

- 汎用から汎用への簡単な変換
- 異なるアプリケーション構造間の便利な変換

コラボレーション・テンプレートからのネイティブ・マップの呼び出しには、以下のステップが含まれます。

- 211 ページの『マップ名用のコラボレーション・プロパティーの作成』
- 211 ページの『コラボレーションの初期化』
- 211 ページの『マップの呼び出し』
- 214 ページの『コラボレーション変数の取り込み』

## マップ名用のコラボレーション・プロパティーの作成

呼び出し対象のマップ名を含むコラボレーション・プロパティーを作成できます。このステップは必須ではありませんが、このステップにより、マップ名が変更された場合でもコラボレーション・コードを再コンパイルする必要がなくなります。代わりに、マップ名が変更された場合は、このコラボレーション・プロパティーの値を変更するだけで済みます。例えば、MAP_NAME と呼ばれるコラボレーション・プロパティーを定義することにより、呼び出す必要があるマップ名を保持できます。コラボレーション・プロパティーは、「テンプレート定義」ウィンドウで定義されます。詳細については、91 ページの『コラボレーション構成プロパティーの定義 (「プロパティー」タブ)』を参照してください。

**注:** getConfigProperty() メソッドを使用して、コラボレーション・テンプレートのコード内でこのコラボレーション・プロパティーの値を取得します。

## コラボレーションの初期化

マップを呼び出すためのコラボレーション・テンプレートの初期化には、コラボレーション・テンプレートに対するマッピング API の Java クラスのインポートが含まれます。InterChange Server Express マップには、実行すべき特定の Java クラスが必要です。これらのクラスの一部は、コラボレーション・テンプレートに自動的には組み込まれません。マップを実行するには、以下のマップ・クラスおよびパッケージを明示的にインポートする必要があります。

- CxCommon.CxExecutionContext クラス
- CxCommon.Exceptions パッケージ
- CxCommon.Dtp パッケージ
- CxCommon.BaseRunTimes パッケージ
- DLM パッケージ

これらの各項目は、「テンプレート定義」ウィンドウの「宣言」タブの「インポート」セクションでインポートします。例えば、インポート・テーブルに次のエントリーを追加して、マップ・クラスをコラボレーション・テンプレートにインポートします。

```
CxCommon.CxExecutionContext
CxCommon.Exceptions.*
CxCommon.Dtp.*
CxCommon.BaseRunTimes.*
DLM.*
```

**注:** パッケージ内のすべての クラスをインポートする場合は、パッケージ名の後ろに「.*」構文を付けるようにしてください。

Java クラスのインポート方法の詳細については、86 ページの『Java パッケージのインポート』を参照してください。

## マップの呼び出し

マップを呼び出すには、マッピング API クラスの runMap() メソッド、DtpMapService を使用します。runMap() メソッドには、以下の情報を引き数として渡す必要があります。

- 実行対象のマップの名前

- マップのタイプ (InterChange Server Express ネイティブ・マップの場合は必ず CWMAPTYPE で表されます)
- マップのソース・ビジネス・オブジェクトを含むビジネス・オブジェクトの入力配列
- マップの実行コンテキスト

したがって、runMap() を呼び出す*前に* コラボレーション内でこの情報を初期化する必要があります。

***マップ名の取得:*** 実行対象のマップの名前を渡す場合、runMap() の呼び出しでマップ名をハードコーディングできます。ただし、最も柔軟な設計は、コラボレーション・プロパティーにマップ名を含めるという方法です。この設計には、以下の手順が含まれます。

- 211 ページの『マップ名用のコラボレーション・プロパティーの作成』の説明に従ってコラボレーション・プロパティーを設定します。
- getConfigProperty() メソッドを使用して、コラボレーション・テンプレートのコード内でこのコラボレーション・プロパティーの値を取得します。

図 70 に、コラボレーション・プロパティー MAP_NAME に格納されているマップ名を取得するコード行を示します。

```
String map_name = getConfigProperty("MAP_NAME");
```

*図 70. 実行対象のマップの名前の取得*

**注:** この方法を使用するには、MAP_NAME コラボレーション・プロパティーが作成されており、実行対象の正しいマップ名がこのプロパティーに割り当てられていることが前提です。

***入力配列の初期化:*** runMap() メソッドには、マップのソース・ビジネス・オブジェクトを含む入力配列が必要です。通常、マップにより、単一のソース・ビジネス・オブジェクトが変換されます。このようなマップの場合、この入力配列には 1 つの要素のみがあります。通常、コラボレーションからマップを呼び出す場合、トリガー・ビジネス・オブジェクトのコピーを入力配列に入れます。この場合、この入力配列を 3 番目の引き数として runMap() に入力します。

図 71 に、コラボレーションのトリガー・ビジネス・オブジェクトのコピーを使用して入力配列を初期化するコード行を示します。

```
BusObj[] sourceBusObjs = { inputBusObj };
```

*図 71. トリガー・ビジネス・オブジェクトによる入力配列の初期化*

***マップ実行コンテキストの作成:*** マップ・インスタンスは、マップが必要とする以下のような情報を含む特定のマップ実行コンテキスト内で実行されます。

- 呼び出しコンテキストは、このマップの呼び出しを開始した条件を示します。呼び出しコンテキストは、MapExeContext クラスに事前定義されている定数として示されます。

- 元の要求ビジネス・オブジェクトは、このマップの呼び出しに関連付けられているビジネス・オブジェクトのコピーです。

マッピング API は、マッピング実行コンテキストを MapExeContext オブジェクトとして示します。マップ・コードでは、マップの実行コンテキストを常にシステム生成変数 cwExecCtx から取得できます。ただし、コラボレーション・テンプレートからはこのようなシステム生成変数にアクセスできません。代わりに、コラボレーションは以下の手順を実行する必要があります。

- MapExeContext() コンストラクターを使用して、MapExeContext オブジェクトのインスタンスを生成します。
- MapExeContext オブジェクトをグローバル実行コンテキストに割り当てます。

  CxExecutionContext クラスは、コラボレーションのグローバル実行コンテキストを表します。このため、マップ実行コンテキストを初期化するには、以下の手順を実行する必要があります。

  – CxExecutionContext() コンストラクターを使用して、CxExecutionContext オブジェクトのインスタンスを生成します。
  – setContext() メソッドを使用して、MapExeContext オブジェクトを CxExecutionContext オブジェクトに割り当てます。
- MapExeContext オブジェクトにこの呼び出しコンテキストを提供します。

  MapExeContext クラスの setInitiator() メソッドは 、呼び出しコンテキスト (「マップ・イニシエーター」とも呼ばれますが、この用語は推奨されません) を設定します。マップ実行コンテキストおよび MapExeContext クラスの各メソッドの詳細については、「*マップ開発ガイド*」を参照してください。

図 72 に、マップ実行コンテキストを初期化するコード・フラグメントを示します。これらのコード・フラグメントには EVENT_DELIVERY の呼び出しコンテキスト (アプリケーション固有のビジネス・オブジェクトから汎用ビジネス・オブジェクトへの変換) およびトリガー・ビジネス・オブジェクトの元の要求ビジネス・オブジェクトが使用されています。

```
// Instantiate objects for the map execution context and the global
// execution context
map_exe_context = new MapExeContext();
global_exe_context = new CxExecutionContext();

// Assign the map execution context to the global execution context
global_exe_context.setContext(
   CxExecutionContext.MAPCONTEXT,
   map_exe_context);

// Initialize the map execution context
map_exe_context.setInitiator(MapExeContext.EVENT_DELIVERY);
```

*図 72. マップ実行コンテキストの初期化*

***runMap() メソッドの呼び出し:*** コラボレーション・テンプレートは、マップ情報を初期化した後に、マップを呼び出すことができます。runMap() メソッドを呼び出すには、以下の 2 つの手順を実行します。

- runMap() メソッドを呼び出します。

このメソッドは、マッピング API クラス DtpMapService 内の静的メソッドです。このため、DtpMapService インスタンスのインスタンスを生成する必要はありません。

- runMap() 戻り値の出力配列を提供します。

  また、runMap() メソッドには、入力配列以外にも出力配列が必要です。これは、runMap() がマッピングの宛先ビジネス・オブジェクトを使用して取り込み、呼び出しコードに戻します。通常、マップにより、単一の宛先ビジネス・オブジェクトが生成されます。このようなマップの場合、この出力配列には 1 つの要素のみがあります。

図 73 に、次の特性でマップを実行するための runMap() への呼び出しを示します。

- マップ名は、MAP_NAME コラボレーション・プロパティーで示されます (図 70)。
- マップのソース宛先ビジネス・オブジェクトは、コラボレーションのトリガー・ビジネス・オブジェクトです (図 71)。
- マップの実行コンテキストは、元の要求ビジネス・オブジェクトとして EVENT_DELIVERY の呼び出しコンテキストおよびトリガー・ビジネス・オブジェクトに初期化されます (図 72)。

```
BusObj[] destinationBusObjs = DtpMapService.runMap(
   map_name,
   CWMAPTYPE,
   sourceBusObjs,
   global_exe_context);
```

*図 73. runMap() の呼び出しによるマップの実行*

## コラボレーション変数の取り込み

マップの宛先ビジネス・オブジェクトは、runMap() が戻す出力配列で使用できます。コラボレーションから宛先ビジネス・オブジェクトを送信するには、そのオブジェクトを出力配列から適切なコラボレーション変数にコピーする必要があります。通常、このコラボレーション変数は、コラボレーションの To ポートに関連付けられます。このため、通常、宛先ビジネス・オブジェクトは、ToBusObj コラボレーション変数にコピーされます。

図 74 は、BusObj.copy() メソッドを使用して、図 73 内の runMap() 呼び出しによって戻される単一宛先ビジネス・オブジェクトを ToBusObj コラボレーション変数へコピーします。

```
ToBusObj.copy(destinationBusObjs[0]);
```

*図 74. コラボレーション変数の取り込み*

# ビジネス・オブジェクトに対する操作

このセクションでは、ビジネス・オブジェクトおよびその値の操作について説明します。これには、以下の操作が含まれます。

- 『新規ビジネス・オブジェクトの作成』
- 216 ページの『新規ビジネス・オブジェクトでの子ビジネス・オブジェクトの作成』
- 217 ページの『トリガー・イベントのコピー』
- 218 ページの『ビジネス・オブジェクトのコピーまたは複製』
- 219 ページの『属性値の使用』
- 221 ページの『属性値の設定』
- 223 ページの『属性値を null に設定』

## 新規ビジネス・オブジェクトの作成

コンストラクター・メソッド new を使用して、新規ビジネス・オブジェクトを作成します。

### 構文

```
new BusObj(String busObjType)
```

### パラメーター

*busObjType*　　ビジネス・オブジェクト定義の名前

### 戻り値

BusObj タイプのオブジェクト。

### 例外

ObjectException - ビジネス・オブジェクト引き数が無効な場合に発生します。

### 注

このメソッドにより、値のないビジネス・オブジェクトが作成されます。この値は、属性を取り込むときに設定します。

新しいビジネス・オブジェクトの属性を子ビジネス・オブジェクトまたは子ビジネス・オブジェクト配列として定義する場合は、子ビジネス・オブジェクトを明示的に作成し、これに属性を関連付ける必要があります。詳細については、216 ページの『新規ビジネス・オブジェクトでの子ビジネス・オブジェクトの作成』を参照してください。

### 例

以下の例では、Customer ビジネス・オブジェクト定義を使用して、destinationCustomer と呼ばれる新しいビジネス・オブジェクトを作成します。新しいビジネス・オブジェクトが作成されますが、属性値はありません。

```
BusObj destinationCustomer = new BusObj("Customer");
```

## 新規ビジネス・オブジェクトでの子ビジネス・オブジェクトの作成

新しいビジネス・オブジェクトを作成する場合、カーディナリティー 1 または n の子ビジネス・オブジェクトを含むように定義されている属性には値がありません。この属性に値を設定するには、子ビジネス・オブジェクトまたは子ビジネス・オブジェクト配列を明示的に作成し、これに属性を関連付ける必要があります。このセクションでは、単一の子ビジネス・オブジェクトおよび配列に対してこの操作を行う方法について説明します。

### 単一の子ビジネス・オブジェクトの作成

以下の手順では、最初に新しいビジネス・オブジェクトを作成してから、その属性の 1 つに含まれる子ビジネス・オブジェクトを作成する方法を示します。

1. new メソッドを使用して、親ビジネス・オブジェクトを作成します。
2. new メソッドを使用して、属性を定義するタイプの子ビジネス・オブジェクトを 1 つ作成します。
3. BusObj.set() メソッドを使用して、親オブジェクトの属性値を新しい子ビジネス・オブジェクトに設定します。

以下の例は、SoldToAddressAttribute と呼ばれる属性を持つ新しい Invoice ビジネス・オブジェクトの作成を示します。この属性は、販売先顧客の住所を保持する Address タイプのビジネス・オブジェクトです。この例は、親ビジネス・オブジェクトと子ビジネス・オブジェクトの関係を示します。

```
// Declarations BusObj invoice = new BusObj("Invoice"); // Create child business
object in invoice invoice.set("SoldToAddressAttribute", new BusObj("Address"));
```

コラボレーションが子ビジネス・オブジェクトを操作する必要がある場合、この例は以下のようになります。

```
// Declarations BusObj invoice = new BusObj("Invoice"); BusObj soldToAddress = new
BusObj("Address"); // Manipulate child business object soldToAddress // Associate
child business object soldToAddress with parent invoice
invoice.set("SoldToAddressAttribute", soldToAddress);
```

### 子ビジネス・オブジェクト配列の作成

このセクションでは、以下の手順により、新しいビジネス・オブジェクトを作成してから、その属性の 1 つに含まれる子ビジネス・オブジェクト配列を作成する方法を示します。

1. new メソッドを使用して、ビジネス・オブジェクトを作成します。これは親ビジネス・オブジェクトです。
2. n と等しいカーディナリティーを持つビジネス・オブジェクトを含むように定義されている親ビジネス・オブジェクトの属性には、特定タイプの属性のビジネス・オブジェクトを 1 つ作成します。
3. 親の属性値を新しい単一のビジネス・オブジェクトに設定します。
4. BusObjArray オブジェクトを宣言し、属性の値を取得し、これを配列に割り当てます。

次に、BusObjArray クラスのメソッドを使用して、要素を追加するか、ビジネス・オブジェクト上でその他の操作を実行します。

以下の例は、新しい Bill of Materials ビジネス・オブジェクトの作成、その LineItems 属性の子ビジネス・オブジェクト配列の作成、そして配列上での追加ビジネス・オブジェクトの配置を示します。

```
// Declarations BusObj bom = new BusObj("Bill_Of_Materials"); BusObjArray
lineItemArray = null; BusObj singleLineItem = new BusObj ("LineItem"); // Create
first child item bom.set("LineItemsAttribute", singleLineItem); //If there are
additional line items, do this once lineItemArray =
bom.getBusObjArray("LineItemAttribute"); // Now do this for each additional child
item lineItemArray.addElement(new BusObj("singleLineItem"));
```

## トリガー・イベントのコピー

すべてのシナリオの第 1 アクションは、シナリオのフロー・トリガーを処理する必要があります。 Process Designer Express は、triggeringBusObj と呼ばれる BusObj タイプの変数を自動的に宣言します。この変数は、シナリオの実行を開始するフロー・トリガー (トリガー・イベントまたはトリガー・アクセス呼び出し) を保持します。

また、Process Designer Express は、定義されているポートごとにテンプレート BusObj 変数を自動的に宣言します。 BusObj 変数の名前は、ポート名に BusObj を付加したものです。

例えば、シナリオのトリガー・イベントが Customer.Create であるとします。トリガー・イベントを受け取るポートは SourceCust と呼ばれます。Process Designer Express は、ポート名に BusObj というサフィックスを付加した SourceCustBusObj という名前の変数を自動的に宣言します。この場合は、Process Designer Express に以下の変数宣言が表示されます。

```
BusObj SourceCustBusObj = new BusObj("Customer");
```

以下のコード・フラグメントを追加して、トリガー・イベントを SourceCustBusObj にコピーできます。

```
SourceCustBusObj.copy(triggeringBusObj);
```

多くのコラボレーションは、複数のタイプのビジネス・オブジェクトによって起動されます。このため、まずビジネス・オブジェクトのタイプを決定してから、ビジネス・オブジェクトを作成してフロー・トリガーを保持する必要があります。フロー・トリガーに対して BusObj.getType() メソッドを使用して最初にそのタイプを確認し、次に適切なタイプのビジネス・オブジェクトを作成し、最後に新しく作成したビジネス・オブジェクトにフロー・トリガーをコピーします。

```
sourceBusObj = new BusObj(triggeringBusObj.getType());
sourceBusObj.copy(triggeringBusObj);
```

**ヒント:** プログラミング上、最初に triggeringBusObj を別の変数にコピーしてから任意の操作を行うのが理想的です。

## ビジネス・オブジェクトのコピーまたは複製

ビジネス・オブジェクト間で値を移動するメソッドには、copy() と duplicate() という 2 つのメソッドあります。これらの両メソッドとも、親ビジネス・オブジェクトおよびすべての子ビジネスをコピーして、ビジネス・オブジェクトの階層全体を処理します。表 49 に、これら両メソッドの説明を示します。

*表 49. copy() と duplicate() メソッドの比較*

| メソッド | 説明 |
|---|---|
| copy() | 既存のビジネス・オブジェクトのすべての属性値を別のビジネス・オブジェクトの属性値にコピーします。 |
| duplicate() | 既存のビジネス・オブジェクトを複製して新しいビジネス・オブジェクトを作成します。属性値と動詞の両方を複製します。このメソッドの戻り値は、変数に割り当てる必要があります。 |

これら 2 つのメソッドの主な違いは、コピーした値を入力するビジネス・オブジェクトがあるかどうかです。

### コピー

copy() メソッドを使用する前に、値のコピー先のビジネス・オブジェクトが存在している必要があります。存在していない場合は、new メソッドを使用してビジネス・オブジェクトを作成します。

以下の例では、ソース・アプリケーションから受信した Customer ビジネス・オブジェクトに含まれる属性値を、宛先アプリケーションに送信されるビジネス・オブジェクトにコピーします。

```
BusObj destination = new BusObj("Customer"); destination.copy(sourceBusObj);
```

**注:** copy() メソッドは、すべての子ビジネス・オブジェクトおよび子ビジネス・オブジェクト配列を含むビジネス・オブジェクト全体をコピーします。このメソッドは、コピーされたオブジェクトへの参照は設定しません。代わりに、すべての属性を複製します。つまり、属性の別のコピーを作成します。

### 複製

duplicate() は、copy() メソッドとは異なり、既存のビジネス・オブジェクトの完全な複製を作成してそれを戻します。

以下の例では、ソース・ビジネス・オブジェクトを複製し、destination と呼ばれる変数にこれを割り当てます。

```
BusObj destination = sourceBusObj.duplicate();
```

## 属性値の使用

コラボレーションは、受信したビジネス・オブジェクトに含まれる属性値を何度も検索します。

コラボレーションが受信した属性値を使用してなんらかの処理を行う必要がある場合、コラボレーションは、これを使用する前に、属性値が null でないことを確認する必要があります (219 ページの『null の検査』を参照)。

### null の検査

null 属性値を検査するために、コラボレーションは BusObj.isNull(*attribute*) を呼び出します。通常、null 値は、以下のいずれかの理由によって表示されます。

- 属性が設定されていません。

  ビジネス・オブジェクトの作成時は、すべての属性は null で、明示的に設定されるまでは null のままです。これには、子ビジネス・オブジェクトと子ビジネス・オブジェクト配列が含まれます。
- 属性は BusObj.set() メソッドによって明示的に null に設定されました。
- マッピング処理時、この属性にマッピングする値が入力ビジネス・オブジェクトにありません。

以下のコード・サンプルでは、Order ビジネス・オブジェクトで受信したデータを使用して Billing ビジネス・オブジェクトが設定されます。受信したオーダー・データ自体が null またはブランクである場合、属性は「Unknown」に設定されます。

```
//Check whether ProductLine is null or blank; if so, default it if
(order.isNull("ProductLine") || order.isBlank("ProductLine"))      {
logInfo("Setting ProductLine to default Unknown");      order.set("ProductLine",
"Unknown");      } //Now set Billing object's equivalent attribute to the same
value billing.set("ProductLine", order.get("ProductLine");
```

isNull() メソッドは、子ビジネス・オブジェクトまたは子ビジネス・オブジェクト配列を含む属性を含め、すべてのタイプの属性に対して使用できます。

### 属性値と既知の値との比較

コラボレーションは、Java プログラム言語 equals() メソッドを使用して、特定の予測値に対して属性値をチェックできます。 equals() メソッドは、以下の例のように、予測値と検索した属性値を比較します。equals() を既知のオブジェクト上で呼び出し、これを既知の属性と比較します。

この例では、Smith は、LName 属性で予測される値です。

```
String name = "Smith"; boolean checkName=name.equals(CustBusObj.get("LName"));
```

### 属性の検索

このセクションでは、属性を検索する操作のタイプについて説明します。これらの操作は、簡単なものから複雑なものへと順に並んでいます。

- 属性値検索の基本型

- 子ビジネス・オブジェクトの属性の検索

***基本型の属性の検索:*** BusObj.get() メソッドは、基本型の属性値を検索します。属性の基本型は、サポートされているプリミティブです (表 50 を参照)。

*表 50. 基本データ型の属性の検索*

| 基本データ型 | 属性を設定するメソッド |
|---|---|
| boolean | getBoolean() |
| double | getDouble() |
| float | getFloat() |
| int | getInt() |
| long | getLong() |
| Object | get() |
| LongText | getLongText() |
| String | getString() |

以下の例では、Credit-Limit 属性の内容 (int) を検索します。

```
int creditLimit = customer.getInt("Credit-Limit");
```

属性のデータ型がわからない場合は、Java Object データ型を検索する get() メソッドの形式を使用します。

**注:** get() メソッドは、属性のコピーを戻します。このメソッドは、ソース・ビジネス・オブジェクトのこの属性に対するオブジェクト参照を戻しません。このため、ソース・ビジネス・オブジェクトの属性の変更内容は、get() が戻す値には適用されません。このメソッドは、呼び出されるたびに属性の新しいコピー (複製) を戻します。

***子ビジネス・オブジェクトからの属性の検索:*** 属性がビジネス・オブジェクト・タイプの場合、ビジネス・オブジェクト定義に定義されているカーディナリティーは、属性に単一のビジネス・オブジェクトあるいは配列のいずれが含まれているかを示します。カーディナリティーの値により、以下のようになります。

- 1 の場合、属性には、単一のビジネス・オブジェクトが含まれます。getBusObj() の戻り値を BusObj オブジェクトに割り当てます。
- n の場合、属性には、ビジネス・オブジェクトの配列が含まれます。getBusObjArray() の戻り値を BusObjArray オブジェクトに割り当てます。

以下の例では、Bill of Materials ビジネス・オブジェクトの SoldToAddress 属性に含まれる単一カーディナリティーの Address ビジネス・オブジェクトを検索します。

```
BusObj addr = new BusObj("Address"); addr = bom.getBusObj("SoldToAddress");
```

以下の例では、米国内の販売先住所を検索します。ビジネス・オブジェクト構造は、以下のとおりです。

- BusOrg はトップレベルのビジネス・オブジェクトです。
- BusOrg には、複数カーディナリティーのビジネス・オブジェクトである SoldToSite と呼ばれる属性があります。

- SoldToSite には、単一カーディナリティーのビジネス・オブジェクトである SoldToAddress と呼ばれる属性があります。
- SoldToAddress には、CountryName と呼ばれる属性が含まれています。

```
//Look for sold-to address in the US //Start with the busOrg business object //Get
the child business object array in the "SoldToSite" attribute if
(!busOrg.isNull("SoldToSite"))     {     BusObjArray siteAddArray =
busOrg.getBusObjArray("SoldToSite");     //     //String to compare with sold-to
country name     String countryName = "USA";     //Get size of child business
object array     int count = siteAddArray.size();     //     //For each business
object in the array get the SoldToAddress     //attribute, which is a business
object, and compare its     //SoldToCountryName attribute to the string "USA"
//     for (int i = 0; i < count ; i ++)         {          BusObj siteAddr =
siteAddArray.elementAt( i );          if (!siteAddr.isNull("SoldToAddress"))
            {             BusObj soldToAddress =
siteAddr.getBusObj("SoldToAddress");             if (countryName.equalsIgnoreCase(
                  soldToAddress.getString("SoldToCountryName")))               {
              //do something               }             }//end if
}//end for     }//end if
```

## 属性値の設定

このセクションでは、属性を設定する操作の 3 つのタイプについて説明します。これらの操作は、簡単なものから複雑なものへと順に並んでいます。

- 基本型の属性値の設定
- 単一の子ビジネス・オブジェクトでの属性の設定
- ビジネス・オブジェクト配列の一部である子ビジネス・オブジェクトの属性の設定

### 基本型の属性の設定

BusObj.set() メソッドは、基本型の属性値を設定します。属性の基本型は、サポートされているプリミティブの、boolean、double、float、int、long、Object、および String です。

各 set() メソッドにはすべて同じ名前が付けられていますが、シグニチャーは異なります。最初のパラメーターは常に、値を設定する属性の名前が含まれる String です。2 番目のパラメーターは、プリミティブ・タイプ String オブジェクトまたは Object タイプである変数または定数です。変数のタイプとは関係なく、同じ set() メソッドを呼び出します。このメソッドは、あらゆるタイプの変数を受け入れられるように多重定義されます。どのメソッドのバリエーションを使用するかはコンパイラーが決めます。

次の例では、FirstName と Salary の 2 つの属性の値を設定します。

```
Customer.set("FirstName", "Sue"); Customer.set("Salary", 30500);
```

BusObj.get() メソッドと BusObj.set() メソッドを使用すると、あるビジネス・オブジェクトから別のビジネス・オブジェクトへ属性値をコピーすることができます。次の例では、ソース・ビジネス・オブジェクトから String 変数 HeaderId と

ServiceId を取得し、宛先ビジネス・オブジェクト内の属性 HeaderId と SalesNum をそれらの値に設定します。

```
DestinationBusObj.set(     "HeaderId", SourceBusObj.getString("HeaderId"));
DestinationBusObj.set(     "SalesNum", SourceBusObj.getString("ServiceId"));
```

**注:** set() メソッドは、値を属性に割り当てるときに、値に対してオブジェクト参照を設定します。これにより、ソース・ビジネス・オブジェクトから属性値が複製されることはありません。このため、ソース・ビジネス・オブジェクトの値の変更内容は、set() を呼び出すビジネス・オブジェクトの属性にも適用されます。

## 単一の子ビジネス・オブジェクトでの属性の設定

カーディナリティーが 1 に等しい子ビジネス・オブジェクトの属性を設定するには、最初に子ビジネス・オブジェクトのハンドルを取得する必要があります。

BusObj.getBusObj() メソッドを使用して、子ビジネス・オブジェクトのハンドルまたは参照を取得し、その結果を BusObj タイプの変数に割り当てます。次に、BusObj.set() メソッドを呼び出します。これにより、属性のデータ型に一致するメソッドの多重定義バージョンが呼び出されます。

次の例では、図 75 を基にしています。Customer ビジネス・オブジェクトには、Address と呼ばれる属性があります。これは、CustAddress と呼ばれる子ビジネス・オブジェクトへの参照となります。

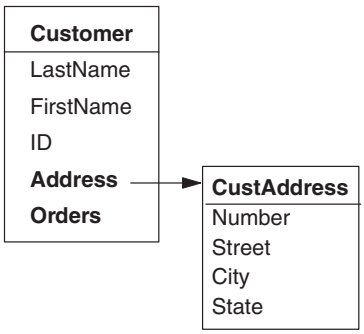



*図 75. 単一の子ビジネス・オブジェクト*

このコード例では、以下の処理が実行されます。

1. addr と呼ばれる変数を宣言します。
2. Customer ビジネス・オブジェクトの Address 属性を検索し、それを addr に割り当てます。
3. City 属性を設定します。

```
BusObj addr = Customer.getBusObj("Address"); addr.set("City", "SF");
```

### 子ビジネス・オブジェクトの配列での属性の設定

子ビジネス・オブジェクトの配列に属性 (カーディナリティーが n に等しい属性) を設定するには、BusObj.getBusObjArray() メソッドを使用し、その結果を BusObjArray タイプの変数に割り当てます。次に、その変数で BusObjArray メソッドを使用します。

下記のコードは、以下の構造に基づいています。

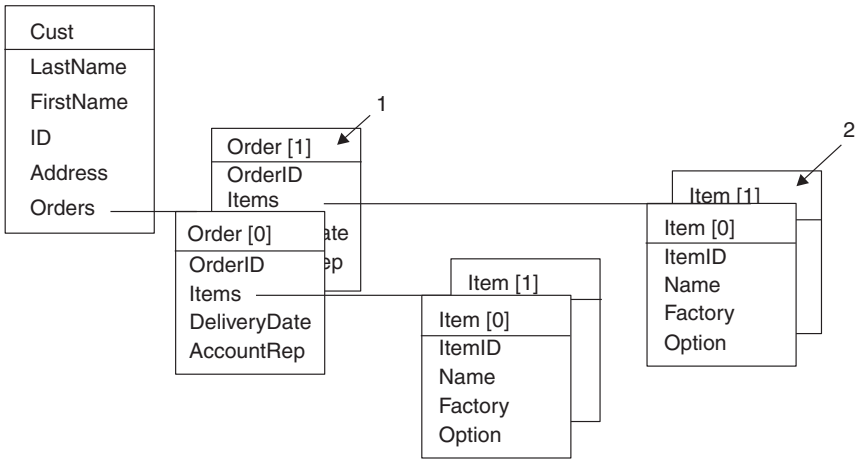



*図 76. 子ビジネス・オブジェクト配列*

次の例では、図 76 で 1 の印が付いているビジネス・オブジェクトの OrderID 属性を設定します。

```
BusObjArray[] orders =      cust.getBusObjArray("Orders").elementAt(1);
orders[1].set("OrderID", "x1234");
```

次の例では、図 76 で 2 の印が付いているビジネス・オブジェクトの Factory 属性を設定します。

```
BusObjArray items = orders[1].getBusObjArray("Items"); BusObj item =
items.elementAt(1); item.set("Factory", "MyCompany");
```

## 属性値を null に設定

以下の例では、order ビジネス・オブジェクトの Total 属性値を null に設定します。

```
order.set("Total", null);
```

この技法を使用して、属性値が基本型、BusObj タイプ、または BusObjArray タイプのいずれであるかに関係なく、あらゆるタイプの属性を null に設定できます。ただし、この技法を使用して、配列の子ビジネス・オブジェクトを null に設定することはできません。

# データベース照会の実行

コラボレーションの実行時に、データベース (関連データベースなど) から情報を取得する必要があることがあります。データベースの情報を取得または変更するには、その表を照会します。**照会**は、データベースに送信して実行させる、通常は SQL (Structured Query Language) ステートメント形式の要求です。表 51 に、データベースでの照会の実行に関する手順を示します。

**注:** WebSphere Business Integration Server Express がサポートする外部データベースには、Oracle シン・タイプ 4 ドライバーおよび WebSphere Business Integration Server Express ブランドの MS-SQL Server タイプ 4 ドライバーを介し、JDBC を使用してアクセスできます。

*表 51. 照会の実行作業*

| 照会の実行作業 | 詳細 |
|---|---|
| 1. データベースへの接続 (CwDBConnection オブジェクト) を取得します。 | 224 ページの『接続の取得』 |
| 2. CwDBConnection オブジェクトを使用して、データベース内で照会の送信とトランザクションの管理を行います。 | 225 ページの『照会の実行』237 ページの『トランザクションの管理』 |
| 3. 接続を解放します。 | 242 ページの『接続の解放』 |

**ヒント:** データベース照会の使用法の 1 つとして、待ち時間が長いサービス呼び出しの処理があります。コラボレーションは、処理時間が長いと予想されるサービス呼び出しを発行した後に、データベース接続を使用している実行コンテキストを保管してから終了します。サービス呼び出しからの実際の応答 (数時間または数日間後であることがあります) は、新しいイベントとして戻され、別のコラボレーションを起動します。これにより、データベースからの適切な実行コンテキストが復元され、ビジネス・プロセスの実行が再開されます。

## 接続の取得

データベースの照会を行うには、最初に BaseCollaboration クラスの getDBConnection() メソッドを使用してデータベースへの接続を取得する必要があります。取得する接続を識別するには、この接続が含まれる接続プールの名前を指定します。特定の接続プールのすべての接続は、同じデータベースに対して行われます。接続プール内の接続の数は、接続プール構成の一部として決定されます。照会対象のデータベースの接続が含まれる接続プールの名前を決定する必要があります。

**要確認:** 接続は、InterChange Server がブートされたとき、または新しい接続プールが動的に構成されたときに開かれます。このため、目的のデータベースに対する接続が含まれる接続プールは、接続を要求するコラボレーション・オブジェクトが実行される前に構成する必要があります。System Manager 内で接続プールを構成します。詳細については、「*WebSphere InterChange Server システム・インプリメンテーション・ガイド*」を参照してください。

図 77 では、getDBConnection() への呼び出しが、CustDBConnPool 接続プール内の接続に関連付けられたデータベースへの接続を取得します。

```
CwDBConnection connection = getDBConnection("CustDBConnPool");
```

*図 77. 接続プールからの接続の取得*

getDBConnection() 呼び出しは、connection 変数内の CwDBConnection オブジェクトを戻します。このオブジェクトを使用すると、接続に関連付けられたデータベースにアクセスできます。

**ヒント:** getDBConnection() メソッドは、接続のトランザクション・プログラミング・モデルを指定できる追加形式を提供します。詳細については、237 ページの『トランザクションの管理』を参照してください。

## 照会の実行

表 52 に、CwDBConnection クラスのメソッドを使用して SQL 照会を実行する方法を示します。

*表 52. CwDBConnection メソッドの使用による SQL 照会の実行*

| 照会のタイプ | 説明 | CwDBConnection メソッド |
|---|---|---|
| 静的照会 | SQL ステートメントは、テキストとしてデータベースに送信されます。 | executeSQL() |
| 準備済み照会 | 初回の実行後、SQL ステートメントは、コンパイルされた実行可能形式で保管されます。これにより、後続の実行時に、このプリコンパイル形式が使用できます。 | executePreparedSQL() |
| ストアード・プロシージャー | SQL ステートメントおよび条件ロジックが含まれるユーザー定義のプロシージャー | executeSQL()executePreparedSQL() executeStoredProcedure() |

### 静的照会の実行

executeSQL() メソッドは、静的照会を実行するためにデータベースへ送信します。**静的照会**は、ストリングとしてデータベースに送信される SQL ステートメントです。このデータベースがこのストリングを解析し、結果の SQL ステートメントを実行します。このセクションでは、executeSQL() を使用して以下の種類の SQL 照会をデータベースに送信する方法について説明します。

- データベースからデータを戻す照会 (SELECT)
- データベースのデータを変更する照会 (INSERT、UPDATE、DELETE)
- データベースに定義されているストアード・プロシージャーを実行する照会

***データを戻す静的照会の実行 (SELECT):*** SQL ステートメント SELECT は、1 つ以上の表のデータを照会します。SELECT ステートメントをデータベースに送信して実行させるには、SELECT のストリング表記を引き数として executeSQL() メソッドに指定します。例えば、executeSQL() への次の呼び出しは、customer 表から 1 列の値の SELECT を送信します。

```
connection.executeSQL(
   "select cust_id from customer where active_status = 1");
```

**注:** 前のコードで、connection 変数は、getDBConnection() メソッドへの前の呼び出しから取得した CwDBConnection オブジェクトです (図 77 を参照してください)。

また、executeSQL() メソッドの 2 番目の形式を使用して、パラメーターを持つ SELECT ステートメントを送信することもできます。例えば、executeSQL() に対する以下の呼び出しにより、前の例と同じ操作が行われます。ただし、この場合、アクティブ状況がパラメーターとして SELECT ステートメントに渡されます。

```
Vector argValues = new Vector();

String active_stat = "1";
argValues.add( active_stat );
connection.executeSQL(
   "select cust_id from customer where active_status = ?", argValues);
```

SELECT ステートメントは、データベースの表からデータを行として戻します。各行は、SELECT の WHERE 文節の条件と一致するデータの 1 行を示します。各行には、SELECT ステートメントが指定した列の値が含まれます。戻されたデータは、これらの行および列による 2 次元配列として表示できます。

**ヒント:** SELECT ステートメントの構文は、アクセスする特定のデータベースに対して有効である必要があります。SELECT ステートメントの正確な構文については、データベース文書を確認してください。

戻されたデータにアクセスする手順は、以下のとおりです。

1. 1 行のデータを取得します。
2. 列の値を 1 つずつ取得します。

表 53 に、戻された照会データの行にアクセスするための CwDBConnection クラスのメソッドを示します。

*表 53. 行にアクセスするための CwDBConnection メソッド*

| 列にアクセスするための操作 | CwDBConnection メソッド |
|---|---|
| 行があるかどうかのチェック | hasMoreRows() |
| 1 行のデータを取得 | nextRow() |

hasMoreRows() メソッドを使用して、戻された行のループを制御します。hasMoreRows() が false を戻したら、行のループを終了します。1 行のデータを取得するには、nextRow() メソッドを使用します。このメソッドは、選択された列の値を Java Vector オブジェクトの要素として戻します。これにより、Enumeration クラスを使用して、列の値に個別にアクセスできます。 Vector クラスと Enumeration クラスの両方が java.util パッケージ内にあります。

表 54 に、戻された照会行の列にアクセスするための Java メソッドを示します。

*表 54. 列の値にアクセスするための Java メソッド*

| 列にアクセスするための操作 | Java メソッド |
|---|---|
| 列の数の確認 | Vector.size() |
| Enumeration に対する Vector のキャスト | Vector.elements() |
| 列があるかどうかのチェック | Enumeration.hasMoreElements() |
| 1 列のデータの取得 | Enumeration.nextElement() |

hasMoreElements() メソッドを使用して、列の値のループを制御します。hasMoreElements() が false を戻したら、列のループを終了します。列の値を取得するには、nextElement() メソッドを使用します。

以下のコード・サンプルは、顧客情報が格納されたデータベースへの接続である CwDBConnection クラスのインスタンスを取得します。次に、SELECT ステートメントを実行します。これにより、顧客 ID 20987 の会社名「CrossWorlds」の単一列が含まれる 1 行が戻されます。

```
CwDBConnection connectn = null;
Vector theRow = null;
Enumeration theRowEnum = null;
String theColumn1 = null;

try
   {
   // Obtain a connection to the database
   connectn = getDBConnection("sampleConnectionPoolName");
   }

catch(CwDBConnectionFactoryException e)
   {
   System.out.println(e.getMessage());
   throw e;
   }

// Test for a resulting single-column, single-row, result set
try {
   // Send the SELECT statement to the database
   connectn.executeSQL(
      "select company_name from customer where cust_id = 20987");

   // Loop through each row
   while(connectn.hasMoreRows())
      {
      // Obtain one row
      theRow = connectn.nextRow();
      int length = 0;
      if ((length = theRow.size())!= 1)
         {
         return methodName + "Expected result set size = 1," +
            " Actual result state size = " + length;
         }

      // Get column values as an Enumeration object
      theRowEnum = theRow.elements();

      // Verify that column values exist
      if (theRowEnum.hasMoreElements())
         {
         // Get the column value
         theColumn1 = (String)theRowEnum.nextElement();
         if (theColumn1.equals("CrossWorlds")==false)
            {
            return "Expected result = CrossWorlds,"
               + " Resulting result = " + theColumn1;
            }
         }
      }
   }

// Handle any exceptions thrown by executeSQL()
```

```
catch(CwDBSQLException e)
   {
   System.out.println(e.getMessage());
   }
```

以下の例は、複数の行を戻す SELECT ステートメントのコード・フラグメントを示します。各行には、顧客 ID および関連会社名の 2 列が含まれます。

```
CwDBConnection connectn = null;
Vector theRow = null;
Enumeration theRowEnum = null;
Integer theColumn1 = 0;
String theColumn2 = null;

try
   {
   // Obtain a connection to the database
   connectn = getDBConnection("sampleConnectionPoolName");
   }

catch(CwDBConnectionFactoryException e)
   {
   System.out.println(e.getMessage());
   throw e;
   }

// Code fragment for multiple-row, multiple-column result set.
// Get all rows with the specified columns, where the
// specified condition is satisfied
try
   {
   connectn.executeSQL(
"select cust_id, company_name from customer where active_status = 1");

   // Loop through each row
   while(connectn.hasMoreRows())
      {
      // Obtain one row
      theRow = connectn.nextRow();

      // Obtain column values as an Enumeration object
      theRowEnum = theRow.elements();
      int length = 0;
      if ((length = theRow.size()) != 2)
      {
         return "Expected result set size = 2," +
            " Actual result state size = " + length;
      }
      // Verify that column values exist
      if (theRowEnum.hasMoreElements())
         {
         // Get the column values
         theColumn1 =
            ((Integer)theRowEnum.nextElement()).intValue();
         theColumn2 = (String)theRowEnum.nextElement();
         }
      }
   }
catch(CwDBSQLException e)
   {
   System.out.println(e.getMessage());
   }
```

**注:** SELECT ステートメントは、データベースの内容を変更しません。したがって、通常、SELECT ステートメントに対してトランザクション管理は実行しません。

***データを変更する静的照会の実行:*** データベースの表のデータを変更する SQL ステートメントには、以下が含まれます。

- INSERT は、データベースの表に新しい行を追加します。
- UPDATE は、データベースの表の既存の行を変更します。
- DELETE は、データベースの表から行を除去します。

これらのステートメントの 1 つを静的照会としてデータベースに送信して実行させるには、ステートメントのストリング表記を引き数として executeSQL() メソッドに指定します。例えば、executeSQL() に対する以下の呼び出しにより、現在の接続に関連付けられたデータベースの abc 表に対する 1 行の INSERT が送信されます。

```
connection.executeSQL("insert into abc values (1, 3, 6)");
```

**注:** 前のコードで、connection 変数は、getDBConnection() メソッドへの前の呼び出しから取得した CwDBConnection オブジェクトです。

UPDATE または INSERT ステートメントの場合、getUpdateCount() メソッドを使用して、変更または追加されたデータベースの表の行数を確認できます。

**要確認:** INSERT、UPDATE、および DELETE ステートメントはデータベースの内容を変更するため、これらのステートメントに対してトランザクション管理の必要性を評価することを推奨します。詳細については、237 ページの『トランザクションの管理』を参照してください。

***静的ストアード・プロシージャーの実行:*** 以下の*両方の* 条件が満たされていれば、executeSQL() メソッドを使用してストアード・プロシージャー呼び出しを実行できます。

- ストアード・プロシージャーは OUT パラメーターを使用しない。

  ストアード・プロシージャーが OUT パラメーターを使用する場合は、それを実行するために executeStoredProcedure() を使用する必要があります。
- ストアード・プロシージャーは 1 回のみ呼び出される。

  executeSQL() メソッドは、ストアード・プロシージャー呼び出しに対する準備済みステートメント保管しません。このため、例えば、ループ内で同じストアード・プロシージャーを複数回呼び出す場合、executeSQL() を使用すると、準備済みステートメントを保管するメソッド executePreparedSQL() または executeStoredProcedure() を呼び出す場合より処理が遅くなることがあります。

詳細については、232 ページの『ストアード・プロシージャーの実行』を参照してください。

## 準備済み照会の実行

executePreparedSQL() メソッドは、準備済み照会を実行するためにデータベースへ送信します。**準備済み照会**は、データベースによる実行可能形式にすでにプリコンパイルされている SQL ステートメントです。executePreparedSQL() は、初めてデータベースに照会を送信するときに照会をストリングとして送信します。データベースは、この照会を受信し、文字列を解析して実行可能形式にコンパイルし、その結果の SQL ステートメントを実行します (executeSQL() に対する操作と同様)。た

だし、データベースはこの SQL ステートメントのコンパイル済み形式を executePreparedSQL() に戻し、このメソッドがこのステートメントをメモリーに保管します。このコンパイル済み SQL ステートメントを、**準備済みステートメント**と呼びます。

これと同じ照会が後で実行されるときには、executePreparedSQL() は、この照会に対して準備済みステートメントがすでに存在するかどうかを最初にチェックします。準備済みステートメントが存在する場合、executePreparedSQL() は、照会ストリングの代わりにこれをデータベースに送信します。この後にこの照会が実行される場合、データベースはストリングを解析して準備済みステートメントを実行する必要がないため、処理がより効率的になります。

executePreparedSQL() を使用して、以下の種類の SQL 照会をデータベースに送信できます。

- データベースからデータを戻す照会 (SELECT)
- データベースのデータを変更する照会 (INSERT、UPDATE、DELETE)
- データベースに定義されているストアード・プロシージャーを実行する照会

***データを戻す準備済み照会の実行 (SELECT):*** 同じ SELECT ステートメントを複数回実行する必要がある場合は、executePreparedSQL() を使用してステートメントのプリコンパイル・バージョンを作成します。SELECT ステートメントを準備する場合は、以下の点に注意する必要があります。

- この SELECT ステートメントのパラメーターを使用して、準備済みステートメントの実行のたびに特定の情報を渡すことができます。準備済みステートメントに対してパラメーターを使用する方法の例については、図 78 を参照してください。
- SELECT ステートメントを executePreparedSQL() で実行した場合も、戻されたデータにアクセスするために同じメソッドを使用します (表 53 および 表 54)。詳細については、225 ページの『データを戻す静的照会の実行 (SELECT)』を参照してください。

***データを変更する準備済み照会の実行:*** 同じ INSERT、UPDATE、または DELETE ステートメントを複数回実行する必要がある場合は、executePreparedSQL() を使用してステートメントのプリコンパイル・バージョンを作成します。再実行する SQL ステートメントは、実行するたびに正確に同じでなくても、準備済みステートメントの利点を利用できます。SQL ステートメントのパラメーターを使用して、ステートメントの実行のたびに情報を動的に提供できます。

図 78 のコード・フラグメントは、employee 表に 50 行を挿入します。executePreparedSQL() は、初めて呼び出されるときに、INSERT ステートメントのストリング・バージョンをデータベースに送信します。これにより、このデータベースは、これを解析し、実行し、その実行可能形式である準備済みステートメントを戻します。INSERT ステートメントが実行される次の 49 回 (すべての INSERT の実行が成功であると仮定) では、executePreparedSQL() は、準備済みステートメントが存在することを認識し、この準備済みステートメントをデータベースに送信して実行させます。

```
CwDBConnection connection;
Vector argValues = new Vector();

argValues.setSize(2);

int emp_id = 1;
int emp_id = 2000;

for (int = 1; i < 50; i++)
   {
   argValues.set(0, new Integer(emp_id));
   argValues.set(1, new Integer(emp_num));

   try
      {
      // Send the INSERT statement to the database
      connection.executePreparedSQL(
"insert into employee (employee_id, employee_number) values (?, ?)",
          argValues);

      // Increment the argument values
      emp_id++;
      emp_num++
      }

   catch(CwDBSQLException e)
      {
      System.out.println(e.getMessage());
      }
   }
```

*図 78. 準備済みステートメントに対する引き数値の渡し*

**ヒント:** 通常、INSERT ステートメントの準備済みバージョンの実行により、アプリケーションのパフォーマンスが向上しますが、アプリケーションのメモリー・フットプリントは増加します。

データベースを変更する SQL ステートメントを再実行する場合、トランザクション・プログラミング・モデルに応じてトランザクションを処理する必要があります。詳細については、237 ページの『トランザクションの管理』を参照してください。

**注:** 単純化のため、図 78 のコードにはトランザクション管理が含まれていません。

***準備済みストアード・プロシージャーの実行:*** 以下の*両方の* 条件が満たされていれば、executePreparedSQL() メソッドを使用してストアード・プロシージャー呼び出しを実行できます。

- ストアード・プロシージャーの使用に OUT パラメーターが含まれていない。

  ストアード・プロシージャーが OUT パラメーターを使用する場合は、それを実行するために executeStoredProcedure() を使用する必要があります。
- ストアード・プロシージャーが複数回呼び出される。

  executePreparedSQL() メソッドは、ストアード・プロシージャー呼び出しに対する準備済みステートメントをメモリーに保管します。このため、ストアード・プロシージャーを 1 回のみ呼び出す場合、executePreparedSQL() を使用すると、

準備済みステートメントを保管しないメソッド executeSQL() を使用してストアード・プロシージャーを呼び出す場合より多くのメモリーが使用されることがあります。

詳細については、232 ページの『ストアード・プロシージャーの実行』を参照してください。

## ストアード・プロシージャーの実行

**ストアード・プロシージャー**は、SQL ステートメントおよび条件ロジックが含まれるユーザー定義のプロシージャーです。ストアード・プロシージャーは、データと共にデータベースに格納されます。

**注:** 新しい関係を作成する場合、Relationship Designer は、各関係表を保守するためにストアード・プロシージャーを作成します。

表 55 に、ストアード・プロシージャーを呼び出すための CwDBConnection クラスのメソッドを示します。

*表 55. ストアード・プロシージャーを呼び出すための CwDBConnection メソッド*

| ストアード・プロシージャーを呼び出す方法 | CwDBConnection メソッド | 使用 |
|---|---|---|
| データベースに CALL ステートメントを送信し、ストアード・プロシージャーを実行します。 | executeSQL() | OUT パラメーターがなく、*一度のみ* 実行するストアード・プロシージャーを呼び出す方法。 |
| | executePreparedSQL() | OUT パラメーターがなく、*複数回* 実行するストアード・プロシージャーを呼び出す方法。 |
| ストアード・プロシージャーの名前およびパラメーターの配列を指定し、プロシージャー呼び出しを作成し、これをデータベースに送信して実行させます。 | executeStoredProcedure() | OUT パラメーターがある任意のストアード・プロシージャーを呼び出す方法。 |

**注:** JDBC メソッドを使用して、ストアード・プロシージャーを直接実行できます。ただし、CwDBConnection クラスが提供するインターフェースはより簡単であり、また、データベース・リソースを再利用するので、実行効率が向上します。CwDBConnection クラス内のメソッドを使用して、ストアード・プロシージャーを実行することができます。

ストアード・プロシージャーは、1 行または複数行の形式でデータを戻すことができます。この場合、同じ Java メソッド (hasMoreRows() や nextRow() など) を使用して、SELECT ステートメントによって戻されたデータの場合と同じように、戻された照会結果の行にアクセスします。詳細については、225 ページの『データを戻す静的照会の実行 (SELECT)』を参照してください。

表 55 のように、どのメソッドを使用してストアード・プロシージャーを呼び出すかは、以下によって決まります。

- プロシージャーでの OUT パラメーターの有無。

OUT パラメーターは、ストアード・プロシージャーがこれを介して値を呼び出しコードに戻すパラメーターです。ストアード・プロシージャーが OUT パラメーターを使用する場合、executeStoredProcedure() を使用してストアード・プロシージャーを呼び出す必要があります 。

- ストアード・プロシージャーを呼び出す回数。

  executeStoredProcedure() メソッドは、ストアード・プロシージャーのコンパイル済みバージョンを保管します。このため、例えば、ループ内で同じストアード・プロシージャーを複数回呼び出す場合、executeStoredProcedure() を使用すると、データベースがプリコンパイル済みバージョンを再使用できるため、executeSQL() より処理が速くなることがあります。

以下のセクションでは、executeSQL() メソッドと executeStoredProcedure() メソッドを使用してストアード・プロシージャーを呼び出す方法を説明します。

***OUT パラメーターがないストアード・プロシージャーの呼び出し:*** OUT パラメーターがない ストアード・プロシージャーを呼び出すには、CwDBConnection の以下のメソッドのいずれかを使用できます。

- executeSQL() メソッドは、静的ストアード・プロシージャー呼び出しをデータベースに送信します。

  このプロシージャーは、ストリングとしてデータベースに送信され、ここで、実行される前に準備済みステートメントにコンパイルされます。この準備済みステートメントは、保管されません。このため、1 回のみ呼び出す必要があるストアード・プロシージャーの場合は executeSQL() が役に立ちます。

- executePreparedSQL() メソッドは、準備済みストアード・プロシージャー呼び出しをデータベースに送信します。

  このプロシージャー呼び出しは、初回の呼び出し時にデータベースに送信されます。これにより、準備済みステートメントが作成され実行されます。ただし、データベースはこの準備済みステートメントを executePreparedSQL() に送信し、このメソッドがこのステートメントをメモリーに保管します。このため、例えば、ループ内などで複数回呼び出す必要があるストアード・プロシージャーには executePreparedSQL() が役に立ちます。

これらいずれかのメソッドを使用してストアード・プロシージャーを呼び出すには、ストアード・プロシージャーおよび引き数を含む CALL ステートメントのストリング表記をメソッドの引き数として指定します。 図 79 では、executeSQL() への呼び出しが、setOrderCurrDate() ストアード・プロシージャーを実行するための CALL ステートメントを送信します。

```
connection.executeSQL("call setOrderCurrDate(345698)");
```

*図 79. executeSQL() でのストアード・プロシージャーの呼び出し*

図 79 で、connection 変数は、getDBConnection() メソッドへの前の呼び出しから取得した CwDBConnection オブジェクトです。executeSQL() を使用して、setOrderCurrDate() ストアード・プロシージャーを実行できます。これは、その単

一引き数が IN パラメーターである、つまり、その値がストアード・プロシージャーのみ に送信されるためです。このストアード・プロシージャーには OUT パラメーターがありません。

パラメーター配列を受け入れる executeSQL() または executePreparedSQL() の形式を使用して、その引き数値をストアード・プロシージャーに渡すことができます。ただし、これらのメソッドを使用して、OUT パラメーターを使用するストアード・プロシージャーを呼び出すことはできません。このようなストアード・プロシージャーを実行するには、executeStoredProcedure() を使用する必要があります。詳細については、234 ページの『executeStoredProcedure() でのストアード・プロシージャーの呼び出し』を参照してください。

**注:** CwDBConnection .executeSQL() メソッドを使用して ODBC を介して Oracle ストアード PL/SQL オブジェクトを呼び出す場合は、匿名 PL/SQL ブロックを使用します。以下は、受け入れ可能なフォーマットを示します (ストアード・プロシージャー名は myproc です)。

```
connection.executeSQL("begin myproc(...); end;");
```

***executeStoredProcedure() でのストアード・プロシージャーの呼び出し:***

executeStoredProcedure() メソッドは、OUT パラメーターを使用するストアード・プロシージャーを含む、任意のストアード・プロシージャーを実行できます。このメソッドは、executePreparedSQL() の場合と同じように、ストアード・プロシージャー呼び出しに対する準備済みステートメントを保管します。このため、executeStoredProcedure() は、複数回実行されるストアード・プロシージャー呼び出しのパフォーマンスを向上できます。

executeStoredProcedure() メソッドを使用してストアード・プロシージャーを呼び出す手順は、以下のとおりです。

1. 実行するストアード・プロシージャーの名前を String として指定します。
2. CwDBStoredProcedureParam オブジェクトの Vector パラメーター配列を構築します。このオブジェクトは、各ストアード・プロシージャー・パラメーターのイン/アウト・パラメーター・タイプおよび値を提供します。

**パラメーター**は、ストアード・プロシージャーに対して、またはストアード・プロシージャーから送信できる値です。パラメーターのイン/アウト・タイプにより、ストアード・プロシージャーがパラメーター値を使用する方法が決まります。

- IN パラメーターは*入力専用* です。つまり、ストアード・プロシージャーはその値を入力として受け入れますが、呼び出しコードへ値を戻す際にはそのパラメーターを使用しません。
- OUT パラメーターは*出力専用* です。つまり、ストアード・プロシージャーは、このパラメーター値を入力として*解釈しません* が、このパラメーターを使用して値を呼び出しコードに戻します。
- INOUT パラメーターは*入力と出力* の両方に使用されます。つまり、ストアード・プロシージャーはそのパラメーター値を入力として受け入れ、また値を戻すときにもそのパラメーターを使用します。

CwDBStoredProcedureParam オブジェクトは、ストアード・プロシージャーの単一パラメーターを示します。表 56 は、CwDBStoredProcedureParam オブジェクトに含まれるパラメーター情報と、このパラメーター情報を検索および設定するメソッドを

示します。

*表 56. CwDBStoredProcedureParam オブジェクトのパラメーター情報*

| パラメーター情報 | CwDBStoredProcedureParam メソッド |
|---|---|
| パラメーター値 | getValue() |
| パラメーターのイン/アウト・タイプ | getParamType() |

executeStoredProcedure() メソッドを使用してストアード・プロシージャーにパラメーターを渡す手順は、以下のとおりです。

1. パラメーター情報を保持するための CwDBStoredProcedureParam オブジェクトを作成します。

   CwDBStoredProcedureParam() コンストラクターを使用して、新規 CwDBStoredProcedureParam オブジェクトを作成します。このコンストラクターに以下のパラメーター情報を渡すことにより、オブジェクトを初期化します。
   - パラメーターのイン/アウト・タイプにより、パラメーターが IN、INOUT、または OUT のいずれであるかを指定します。
   - パラメーター値は、パラメーターに割り当てる値が含まれる Java データ型です。CwDBStoredProcedureParam クラスには、パラメーター値に関連付けることができるさまざまなデータ型をサポートする多くのバージョンのコンストラクターがあります。OUT パラメーターの場合、このパラメーター値はダミー値にすることができますが、データ型は、ストアード・プロシージャー宣言の OUT パラメーター・データ型と一致する必要があります。
2. 各ストアード・プロシージャー・パラメーターに対してステップ 1 を繰り返します。
3. すべてのストアード・プロシージャー・パラメーターを保持するために、十分な要素を備えた Vector オブジェクトを作成します。
4. 初期化した CwDBStoredProcedureParam オブジェクトをパラメーター Vector オブジェクトに追加します。

   Vector クラスの addElement() メソッドまたは add() メソッドを使用して、CwDBStoredProcedureParam オブジェクトを追加します。
5. すべての CwDBStoredProcedureParam オブジェクトを作成し、これらを Vector パラメーター配列に追加したら、このパラメーター配列を 2 番目の引き数として executeStoredProcedure() メソッドに渡します。

   executeStoredProcedure() メソッドは、ストアード・プロシージャーおよびそのパラメーターをデータベースに送信して実行させます。

例えば、以下のように、データベースに get_empno() ストアード・プロシージャーが定義されているとします。

```
create or replace procedure get_empno(emp_id IN number,
      emp_number OUT number) as
   begin
      select emp_no into emp_number
      from emp
      where emp_id = 1;
   end;
```

この get_empno() ストアード・プロシージャーには、次の 2 つのパラメーターがあります。

- 最初のパラメーター emp_id は、IN パラメーターです。

  このため、PARAM_IN のイン/アウト・タイプ、およびストアード・プロシージャーに送信する適切な値を使用して、その関連 CwDBStoredProcedureParam オブジェクトを初期化する必要があります。 emp_id は (整数値を保持する) SQL NUMBER 型として宣言されるので、パラメーターの値は、整数値 Integer を保持する Java Object になります。

- 2 番目のパラメーター emp_number は、OUT パラメーターです。

  このパラメーターに対しては、ストアード・プロシージャーへ送信する *empty* CwDBStoredProcedureParam オブジェクトを作成します。このオブジェクトは、PARAM_OUT のイン/アウト・タイプを使用して初期化します。ただし、このパラメーターにはダミーの Integer 値を入力します。ストアード・プロシージャーによる実行が終了したら、getValue() メソッドを使用してこの OUT パラメーターから戻された値を取得できます。

図 80 は、executeStoredProcedure() メソッドを使用して get_empno() ストアード・プロシージャーを実行し、65 という従業員 ID の従業員番号を取得します。

```
CwDBConnection connectn = null;

try
   {
   // Get database connection
   connectn = getDBConnection("CustomerDBPool");

   // Create parameter Vector
   Vector paramData = new Vector(2);

   // Create IN parameter for the employee id and add to parameter
   // vector
   paramData.add(
      new CwDBStoredProcedureParam(PARAM_IN, new Integer(65)));

   // Create dummy argument for OUT parameter and add to parameter
   // vector
   paramData.add(
      new CwDBStoredProcedureParam(PARAM_OUT, new Integer(0));

   // Call the get_empno() stored procedure
   connectn.executeStoredProcedure("get_empno", paramData);

   // Get the result from the OUT parameter
   CwDBStoredProcedureParam outParam =
      (CwDBStoredProcedureParam) paramData.get(1);
   int emp_number = ((Integer) outParam.getValue().Intvalue();
   }
```

*図 80. get_empno() ストアード・プロシージャーの実行*

**ヒント:** Java Vector オブジェクトは、ゼロ・ベースの配列です。前のコードでは、Vector 配列がゼロ・ベースであるため、Vector パラメーター配列からこの OUT パラメーターの値にアクセスするために、get() 呼び出しは、索引値 1 を指定します。

ストアード・プロシージャーは、そのパラメーターを SQL データ型として処理します。SQL および Java データ型は同一ではない ため、executeStoredProcedure() メソッドは、これら 2 つのデータ型間でパラメーター値を変換する必要があります。IN パラメーターの場合、executeStoredProcedure() は、パラメーター値を Java データ型から SQL データ型に変換します。OUT パラメーターの場合、executeStoredProcedure() は、パラメーター値を SQL データ型から Java データ型に変換します。

executeStoredProcedure() メソッドは、JDBC データ型を内部で使用し、ストアード・プロシージャーに送受信されるパラメーター値を保持します。JDBC は、java.sql.Types クラスで汎用 SQL タイプの ID の集合を定義します。これらのタイプは、最も一般的に使用される SQL タイプを示します。また、JDBC には、JDBC タイプから Java データ型への標準マッピングも用意されています。例えば、通常、JDBC INTEGER は Java int タイプにマッピングされます。executeStoredProcedure() メソッドは、表 57 に示されているマッピングを使用します。

*表 57. Java データ型と JDBC データ型の間のマッピング*

| Java データ型 | JDBC データ型 |
|---|---|
| String | CHAR、VARCHAR、または LONGVARCHAR |
| Integer、int | INTEGER |
| Long | BIGINT |
| Float、float | REAL |
| Double、double | DOUBLE |
| java.math.BigDecimal | NUMERIC |
| Boolean、boolean | BIT |
| java.sql.Date | DATE |
| java.sql.Time | TIME |
| java.sql.Timestamp | TIMESTAMP |
| java.sql.Clob | CLOB |
| java.sql.Blob | BLOB |
| byte[] | BINARY、VARBINARY、または LONGVARBINARY |
| Array | ARRAY |
| Struct | STRUCT |

# トランザクションの管理

**トランザクション**は、1 単位として実行される操作ステップの集合です。トランザクション内で実行されるすべての SQL ステートメントは、1 単位として成功または失敗します。このセクションでは、トランザクションの管理に関する以下の情報について説明します。

- 『トランザクション・プログラミング・モデルの決定』
- 238 ページの『トランザクション・スコープの指定』

## トランザクション・プログラミング・モデルの決定

トランザクションに対するデータベース操作の実行ステップのグループ化は、**トランザクション・ブラケット**と呼ばれます。各接続には、以下のトランザクション・プログラミング・モデルの 1 つが関連付けられます。

- 暗黙的なトランザクション・ブラケット: データベース操作は、**暗黙的なトランザクション**の一部です。これは、接続が獲得されると同時に開始され、接続が解放されると終了します。トランザクション・ブラケットは InterChange Server によって暗黙的に管理されます。
- 明示的なトランザクション・ブラケット: データベース操作は、**明示的なトランザクション**の一部です。各トランザクションの開始および終了はプログラマチックに決定されます。

コラボレーション・オブジェクトは実行時に、獲得する各接続に対して使用するトランザクション・プログラミング・モデルを決定します。デフォルトでは、コラボレーション・オブジェクトは、獲得するすべての 接続により、そのトランザクション・プログラミング・モデルとして暗黙的なトランザクション・ブラケットが使用されることを前提としています。デフォルトのトランザクション・プログラミング・モデルは、表 58 の任意の方法を使用してオーバーライドできます。

*表 58. 接続のトランザクション・プログラミング・モデルのオーバーライド*

| オーバーライドするトランザクション・プログラミング・モデル | 実行するアクション |
|---|---|
| 特定のコラボレーション・オブジェクトによって取得される すべての*接続に対する* 異なるトランザクション・プログラミング・モデルの指定 | System Manager の「Collaboration Properties」ダイアログの「暗黙的なデータベース・トランザクション」ボックスを選択または選択解除します。詳細については、「*WebSphere InterChange Server システム・インプリメンテーション・ガイド*」を参照してください。 |
| *特定の接続に対する* トランザクション・プログラミング・モデルの指定 | boolean 値を入力し、(この接続のみに対して) 目的のトランザクション・プログラミング・モデルを `getDBConnection()` メソッドのオプションの 2 番目の引き数として指定します。<br><br>次の `getDBConnection()` 呼び出しでは、`ConnPool` 接続プールから取得する接続に対して明示的なトランザクション・ブラケットを指定します。<br><br>`conn = getDBConnection("ConnPool", false);` |

`BaseCollaboration.implicitDBTransactionBracketing()` メソッドを使用して、接続が使用する現在のトランザクション・プログラミング・モデルを決定できます。このメソッドは、トランザクション・プログラミング・モデルが暗黙的なトランザクション・ブラケットであるかどうかを示す `boolean` 値を戻します。

## トランザクション・スコープの指定

接続のトランザクション・プログラミング・モデルにより、データベース・トランザクションのスコープの指定方法が決まります。このセクションの内容は次のとおりです。

- 239 ページの『暗黙的なトランザクション・ブラケットのあるトランザクション・スコープ』
- 240 ページの『明示的なトランザクション・ブラケットのあるトランザクション・スコープ』

***暗黙的なトランザクション・ブラケットのあるトランザクション・スコープ:***
InterChange Server は、すべてのコラボレーションのトランザクション管理を処理します。コラボレーションのビジネス・オブジェクトのすべてのアクションは、1 単位として完了するか、または完了しません。このため、InterChange Server は、ビジネス・プロセス全体を単一の暗黙的なトランザクションとして処理します。なんらかの操作が失敗した場合は、「未解決のフロー」ブラウザーを使用して、失敗したコラボレーションの処理方法を選択します。

接続が暗黙的なトランザクション・ブラケットを使用する場合、InterChange Server は、接続プールの接続に関連付けられた、外部データベースで実行された操作のトランザクション管理も処理します。コラボレーションがデータベース操作を実行する場合、これらのデータベース操作はコラボレーションのビジネス・オブジェクトの一部です。InterChange Server は、メイン・トランザクションのサブトランザクション (コラボレーションのビジネス・プロセス) である暗黙的なトランザクションとしてこれらのデータベース操作を処理します。このデータベース・サブトランザクションは、コラボレーションが接続を取得すると同時に開始されます。 ICS は、コラボレーションの実行が完了すると、サブトランザクションを暗黙的に終了します。

このデータベース・サブトランザクションの成功または失敗は、以下のように、メイン・トランザクションの成功または失敗によって決まります。

- コラボレーションが成功すると、InterChange Server はデータベース・サブトランザクションをコミットします。
- コラボレーションが失敗すると、InterChange Server はデータベース・サブトランザクションをロールバックします。このロールバックが失敗すると、InterChange Server は `CwDBTransactionException` 例外をスローしてエラーを記録します。

コラボレーションが別のコラボレーションを直接呼び出す場合、最初のコラボレーションは親と呼ばれ、2 番目のコラボレーションは子と呼ばれます。親コラボレーションが子コラボレーションを呼び出す場合、InterChange Server は、子コラボレーションのトランザクションを別個に管理します。この子コラボレーションの成功または失敗は、親コラボレーションの成功または失敗とは関係ありません。子コラボレーションが失敗した場合、親コラボレーションはこの失敗の処理方法を決定できます。例えば、親コラボレーションも失敗する必要があると決定することも、その状態を修正または無視して実行を続行すると決定することもできます。

ただし、子コラボレーションの成功または失敗が親コラボレーションの成功または失敗と関係ないとしても、子が実行する暗黙的データベース・トランザクションについてはこのことが当てはまりません。暗黙的なトランザクション・ブラケットを使用するデータベース接続を介して子コラボレーションがデータベース操作を実行する場合、この子コラボレーションは親コラボレーションのトランザクションを継承します。 InterChange Server は、これらのデータベース操作を親コラボレーションのサブトランザクションとして処理します。つまり、以下のように、子コラボレーションの暗黙的データベース・サブトランザクションの最終トランザクション状態は、親 (またはトップレベル) コラボレーションの成功または失敗によって決まります。

- 親コラボレーションが成功の場合、InterChange Server は、データベース・サブトランザクションをコミットします。

- 親コラボレーションが失敗の場合、InterChange Server は、データベース・サブトランザクションをロールバックします。

InterChange Server は、親コラボレーションの成功または失敗が判別するまでは、サブトランザクションをコミットまたはロールバックしません。

この振る舞いにより、子コラボレーションが失敗し、親コラボレーションが実行の続行を選択した場合、InterChange Server は、子コラボレーションの暗黙的データベース・トランザクションをコミットします。

**注:** InterChange Server は、子コラボレーションが実行し、まだアクティブであるデータベース・サブトランザクション (つまり、明示的なトランザクション・ブラケットを使用するデータベース接続を介して実行されているが、子のコラボレーション・テンプレートで明示的にコミットまたはロールバックされていないデータベース・トランザクション) を暗黙的データベース・サブトランザクションと同じように処理します。

子トランザクションを処理するこのメソッドにより、コラボレーションの開発者は、結合セマンティクスを明示的に使用せずに、子から親へのトランザクション結合を実行できます。また、子データベース・トランザクションが子レベルでコミットまたはロールバックされた場合、開発者は、子が (明示的または暗黙的に) 開始したトランザクションを、親が開始したグローバル・ビジネス・プロセス・トランザクションに関連付けることはできません。

**注:** トランザクション・コラボレーションは、その親および子データベース・トランザクションに対してこれと同じモデルを使用します。

***明示的なトランザクション・ブラケットのあるトランザクション・スコープ:*** 接続が明示的なトランザクション・ブラケットを使用する場合、ICS は、コラボレーション・テンプレートが各データベース・トランザクションのスコープを明示的に指定することを必要とします。コラボレーションの成功または失敗とは関係ないデータベース操作を実行する必要がある場合、明示的なトランザクション・ブラケットが役に立ちます。例えば、特定の表がアクセスされたことを示す監査を実行する必要がある場合、この監査は、表へのアクセスが成功と失敗のいずれであるかとは関係なく実行する必要があります。明示的なトランザクションにデータベース操作の監査が含まれる場合、これらの操作はコラボレーションの成功または失敗とは関係なく実行されます。

表 59 に、明示的なトランザクションのトランザクション境界を管理するための CwDBConnection クラスのメソッドを示します。

*表 59. 明示的にトランザクションを管理するための CwDBConnection メソッド*

| トランザクション管理操作 | CwDBConnection メソッド |
|---|---|
| 新しいトランザクションを開始します。 | `beginTransaction()` |
| トランザクションの実行時にデータベースに対して行われたすべての変更をコミット (保管) し、トランザクションを終了します。 | `commit()` |
| トランザクションが現在アクティブかどうかを確認します。 | `inTransaction()` |

*表 59. 明示的にトランザクションを管理するための CwDBConnection メソッド (続き)*

| トランザクション管理操作 | CwDBConnection メソッド |
|---|---|
| トランザクションの実行時に行われたすべての変更をロールバック (バックアウト) し、トランザクションを終了します。 | rollBack() |

明示的なトランザクションのトランザクション・スコープを指定する手順は、以下のとおりです。

1. beginTransaction() メソッドを呼び出して、トランザクションを開始します。
2. beginTransaction() の呼び出しとトランザクションの終了との*間の* 単位として成功または失敗する必要があるすべての SQL を実行します。
3. 以下のいずれかの方法により、トランザクションを終了します。
   - commit() を呼び出し、トランザクションを正常に終了します。SQL ステートメントが行ったすべての変更は、データベースに*保管* されます。
   - rollBack() を呼び出し、トランザクションを不成功に終了します。SQL ステートメントが行ったすべての変更は、データベースからバックアウト されます。

   トランザクションが失敗した原因となる条件を選択できます。失敗条件が満たされた場合は、条件をテストし、rollBack() を呼び出します。それ以外の場合は、commit() を呼び出してトランザクションを正常に終了します。

**要確認:** beginTransaction() を使用して明示的なトランザクションの開始を指定しない 場合、データベースは、各 SQL ステートメントを個別トランザクションとして実行します。beginTransaction() を組み込んだが、接続が解放される前に commit() または rollback() を使用してデータベース・トランザクションの終了を指定しない 場合は、コラボレーションが成功したかどうかに基づき、InterChange Server Express によって暗黙的にトランザクションが終了されます。コラボレーションが成功の場合、ICS は、このデータベース・トランザクションをコミットします。コラボレーションが成功では ない 場合、ICS は、このデータベース・トランザクションを暗黙的にロールバックします。コラボレーションが成功かどうかにかかわらず、ICS は警告を記録します。

以下のコード・フラグメントは、CustDBConnPool の接続に関連付けられたデータベースの 3 つの表を更新します。これらの更新がすべて 成功した場合は、コード・フラグメントによって commit() メソッドでこれらの変更がコミットされます。トランザクション・エラーが発生した場合は、CwDBTransactionException 例外が生じ、コード・フラグメントによって rollback() メソッドが呼び出されます。

```
CwDBConnection connection = getDBConnection("CustDBConnPool", false);

// Begin a transaction
connection.beginTransaction();

// Update several tables
try
   {
   connection.executeSQL("update table1....");
   connection.executeSQL("update table2....");
   connection.executeSQL("update table3....");
```

```
   // Commit the transaction
   connection.commit();
   }

catch (CwDBSQLException e)
   {
   // Roll back the transaction if an executeSQL() call throws
   // an exception
   connection.rollback();
   }

// Release the database connection
connection.release();
```

トランザクションが現在アクティブかどうかを判別するには、inTransaction() メソッドを使用します。

**重要:** beginTransaction()、commit()、および rollback() メソッドは、接続が明示的なトランザクション・ブラケットを使用する場合にのみ 使用します。接続が暗黙的なトランザクション・ブラケットを使用する場合は、これらのメソッドを使用すると、CwDBTransactionException 例外が発生します。

## 接続の解放

接続は、解放されると、その接続プールに戻され、ここでその他のコンポーネントが使用できるようになります。データベースへの接続が解放される方法は、トランザクション・プログラミング・モデルによって決まります。このセクションの内容は次のとおりです。

- 『暗黙的なトランザクション・ブラケットのある接続の解放』
- 『明示的なトランザクション・ブラケットのある接続の解放』

### 暗黙的なトランザクション・ブラケットのある接続の解放

ICS は、データベース・トランザクションを終了すると、暗黙的なトランザクション・ブラケットを使用する接続を自動的に解放します。ICS は、コラボレーション・オブジェクトが成功または失敗のいずれかを確認するまでデータベース・トランザクションを終了しません。つまり、ICS は、コラボレーションが実行を終了するときにこれらの接続を解放します。コラボレーションが正常に実行されると、ICS は、アクティブなデータベース・トランザクションを自動的にコミットします。コラボレーションの実行が失敗 (例えば、catch ステートメントによって処理されない例外がスローされる場合) すると、ICS は、アクティブなトランザクションを自動的にロールバックします。

### 明示的なトランザクション・ブラケットのある接続の解放

明示的なトランザクション・ブラケットを使用する接続の場合、接続は、以下のいずれかの状況で終了します。

- ICS は、明示的なトランザクション・ブラケットを使用する接続を自動的に解放します。
- CwDBConnection クラスの release() メソッドを使用して、接続を明示的に解放します。

CwDBConnection isActive() メソッドを使用すると、接続が解放されているかどうかを判別できます。次のコード・フラグメントが示しているように、接続が解放されている場合は、isActive() が false を戻します。

```
if (connection.isActive())
   connection.release();
```

**重要:** トランザクションが現在アクティブである場合、release() メソッドは使用しないでください。暗黙的なトランザクション・ブラケットの場合、ICS は、コラボレーションが成功または失敗のいずれかを確認するまでデータベース・トランザクションを終了しません。このため、暗黙的なトランザクション・ブラケットを使用する接続にこのメソッドを使用すると、CwDBTransactionException 例外が発生します。この例外を明示的に処理しないと、アクティブなトランザクションも自動的にロールバックされます。トランザクションがアクティブかどうかは、inTransaction() メソッドを使用して判別できます。ICS は、使用しているトランザクション・プログラミング・モデルとは関係なく、接続を自動的に解放します。多くの場合、接続は明示的に解放する必要はありません。

# 第 3 部 サポートされる機能ブロック

# 第 11 章 ビジネス・オブジェクトの機能ブロック

General¥APIs¥Business Object フォルダーおよびそのサブフォルダーにある機能ブロックは、ビジネス・オブジェクトの処理に関する基本機能を提供します。
General¥APIs¥Business Object フォルダーの機能ブロックは、コラボレーション API の BusObj クラスのメソッドを基にしています。General¥APIs¥Business Object¥Array フォルダーおよび General¥APIs¥Business Object¥Constants フォルダーにある機能ブロックは、クラス BusObj の Java 配列および定数に対応します。

以下のセクションでは、各機能ブロックの詳細を説明します。

*表 60. General¥APIs¥Business Object フォルダーの機能ブロックの要約*

| 機能ブロック | ページ |
|---|---|
| Copy | 248 |
| Duplicate | 249 |
| Equals | 250 |
| Equal Keys | 249 |
| Exists | 251 |
| Get Boolean | 251 |
| Get Business Object | 252 |
| Get Business Object Array | 253 |
| Get Business Object Type | 253 |
| Get Double | 254 |
| Get Float | 255 |
| Get Int | 256 |
| Get Locale | 256 |
| Get Long | 257 |
| Get Long Text | 258 |
| Get Object | 258 |
| Get String | 259 |
| Get Verb | 260 |
| Is Blank | 260 |
| Is Business Object | 261 |
| Is Key | 261 |
| Is Null | 262 |
| Is Required | 263 |
| Iterate Children | 263 |
| Keys to String | 263 |
| New Business Object | 264 |
| Set Content | 265 |
| Set Default Attribute Values | 265 |

表 60. General¥APIs¥Business Object フォルダーの機能ブロックの要約 (続き)

| 機能ブロック | ページ |
|---|---|
| Set Keys | 266 |
| Set Locale | 266 |
| Set Value | 267 |
| Set Value with Create | 268 |
| Set Verb | 269 |
| Set Verb with Create | 270 |
| Shallow Equals | 270 |
| To String | 271 |
| Valid Data | 272 |

表 61. General¥APIs¥Business Object¥Array フォルダーの機能ブロックの要約

| 機能ブロック | ページ |
|---|---|
| Get BusObj At | 254 |
| New Business Object Array | 264 |
| Set BusObj At | 265 |
| Size | 271 |

表 62. General¥APIs¥Business Object¥Constants フォルダーの機能ブロックの要約

| 機能ブロック | ページ |
|---|---|
| Verb:Create | 272 |
| Verb:Delete | 273 |
| Verb:Retrieve | 273 |
| Verb:Update | 273 |

# Copy

入力ビジネス・オブジェクトから、すべての属性値をコピーします。

## 入力

**Copy to** コピー操作の宛先オブジェクトを表す BusObj オブジェクト。

**Copy from** コピーするビジネス・オブジェクトを表す BusObj オブジェクト。

## 注

Copy 機能ブロックは、すべての子ビジネス・オブジェクトおよび子ビジネス・オブジェクト配列を含むビジネス・オブジェクト全体をコピーします。この機能ブロックは、コピーされたオブジェクトへの参照を設定しません。代わりに、すべての属性を複製します (つまり、属性の別個のコピーを作成します)。

### 関連情報

この機能ブロックは、BusObj.copy() メソッドを基にしています。詳細については、382 ページの『copy()』を参照してください。

## Duplicate

オリジナルのビジネス・オブジェクトと同一のビジネス・オブジェクトを作成します。

### 入力

**original** 複製するビジネス・オブジェクト (BusObj オブジェクト)。

### 出力

複製されたビジネス・オブジェクトを戻します。

### 例外

Duplicate 機能ブロックは、`CollaborationException` オブジェクトの例外タイプを `ObjectException` に設定できます。

### 注

この機能ブロックは、ビジネス・オブジェクトの複製を作成して戻します。この機能ブロックの戻り値は、`BusObj` タイプの宣言された変数に明示的に割り当てる必要があります。

### 関連情報

この機能ブロックは、BusObj.duplicate() メソッドを基にしています。詳細については、383 ページの『duplicate()』を参照してください。

## Equal Keys

現在のビジネス・オブジェクトのキー属性値を入力ビジネス・オブジェクトのキー属性値と比較して、等しいかどうかを判別します。

### 入力

**Business Object 1**
比較における第 1 ビジネス・オブジェクト (BusObj オブジェクト)。

**Business Object 2**
比較における第 2 ビジネス・オブジェクト (BusObj オブジェクト)。

### 出力

すべてのキー属性の値が同じ場合は `true` を戻し、同じでない場合は `false` を戻します。

### 例外

Equal Keys 機能ブロックは、CollaborationException 例外に対して以下の例外タイプを設定できます。

- ObjectException — ビジネス・オブジェクトの引き数が無効の場合に設定します。

### 注

この機能ブロックは、浅く比較を行います。つまり、子ビジネス・オブジェクトのキーは比較しません。

### 関連情報

この機能ブロックは、BusObj.equalKeys() メソッドを基にしています。詳細については、 383 ページの『equalKeys()』を参照してください。

## Equals

2 つのビジネス・オブジェクト (子ビジネス・オブジェクトを含む) の属性を比較して、それらが等しいかどうかを判別します。

### 入力

**Business Object 1**
比較における第 1 ビジネス・オブジェクト (BusObj オブジェクト)。

**Business Object 2**
比較における第 2 ビジネス・オブジェクト (BusObj オブジェクト)。

### 出力

すべての属性の値が同じ場合は true を戻し、同じでない場合は false を戻します。

### 例外

Equals 機能ブロックは、CollaborationException 例外に対して以下の例外タイプを設定できます。

- ObjectException — ビジネス・オブジェクトの引き数が無効の場合に設定します。

### 注

この機能ブロックは、このビジネス・オブジェクトの属性値を、入力ビジネス・オブジェクトの属性値と比較します。ビジネス・オブジェクトが階層的である場合は、子ビジネス・オブジェクトの属性もすべて比較されます。

比較時に、null 値は比較対象の任意の値と等価であるとみなされるため、null 値に起因して true が戻されないということはありません。

### 関連情報

この機能ブロックは、BusObj.equals() メソッドを基にしています。詳細については、 384 ページの『equals()』を参照してください。

## Exists

指定した名前のビジネス・オブジェクト属性が存在するかどうかを検査します。

### 入力

**Business Object**
ビジネス・オブジェクト (BusObj オブジェクト)。

**Attribute** 存在するかどうかを検査したい属性の名前を指定する String。

### 出力

属性が存在する場合は `true` を戻し、存在しない場合は `false` を戻します。

### 関連情報

この機能ブロックは、BusObj.exists() メソッドを基にしています。詳細については、386 ページの『exists()』を参照してください。

## Get Boolean

ビジネス・オブジェクトから、boolean 型の単一属性の値を検索します。

### 入力

**Business Object**
属性が存在するビジネス・オブジェクト (BusObj オブジェクト)。

**Attribute** 属性の名前を指定する String。

### 出力

指定された属性のブール値 (`true` または `false`) を戻します。

### 例外

Get Boolean 機能ブロックは、`CollaborationException` 例外に対して以下の例外タイプを設定できます。

- `AttributeException` — 属性でアクセスする際に問題が生じる場合に設定します。例えば、適切なデータを含まない属性でコラボレーションが Get Boolean を使用する場合に、この例外を発生させることができます。

### 注

Get Boolean 機能ブロックは、現在のビジネス・オブジェクトから属性値を検索し、属性値のコピーを戻します。このメソッドは、ソース・ビジネス・オブジェクトのこの属性に対するオブジェクト参照を戻しません。このため、ソース・ビジネス・

オブジェクトの属性値の変更内容は、機能ブロックによって戻される値には適用されません。この機能ブロックは、使用されるたびに属性の新しいコピー (複製) を戻します。

### 関連情報

この機能ブロックは、BusObj.getBoolean() メソッドを基にしています。詳細については、 386 ページの
『getBoolean()、getDouble()、getFloat()、getInt()、getLong()、get()、
getBusObj()、getBusObjArray()、getLongText()、getString()』を参照してください。

## Get Business Object

ビジネス・オブジェクトから、BusObj オブジェクトである単一属性の値を検索します。

### 入力

**Business Object**
ビジネス・オブジェクト (BusObj オブジェクト)。

**Attribute** 検索したい属性の名前を指定する String。

### 出力

指定された属性の値を BusObj オブジェクトとして戻します。

### 例外

Get Business Object 機能ブロックは、`CollaborationException` 例外に対して以下の例外タイプを設定できます。

- `AttributeException` — 属性でアクセスする際に問題が生じる場合に設定します。例えば、適切なデータを含まない属性でコラボレーションが Get Business Object を使用する場合に、この例外を発生させることができます。

### 注

Get Business Object 機能ブロックは、現在のビジネス・オブジェクトから属性値を検索し、属性値のコピーを戻します。このメソッドは、ソース・ビジネス・オブジェクトのこの属性に対するオブジェクト参照を戻しません。このため、ソース・ビジネス・オブジェクトの属性値の変更内容は、機能ブロックによって戻される値には適用されません。この機能ブロックは、使用されるたびに属性の新しいコピー (複製) を戻します。

### 関連情報

この機能ブロックは、BusObj.getBusObj() メソッドを基にしています。詳細については、 386 ページの
『getBoolean()、getDouble()、getFloat()、getInt()、getLong()、get()、
getBusObj()、getBusObjArray()、getLongText()、getString()』を参照してください。

## Get Business Object Array

ビジネス・オブジェクトから、ビジネス・オブジェクト配列である単一属性の値を検索します。

### 入力

**Business Object**
ビジネス・オブジェクト (BusObj オブジェクト)。

**Attribute** 検索したい属性の名前を指定する String。

### 出力

指定された属性の値をビジネス・オブジェクト配列として戻します。

### 例外

Get Business Object Array 機能ブロックは、`CollaborationException` 例外に対して以下の例外タイプを設定できます。

- `AttributeException` — 属性でアクセスする際に問題が生じる場合に設定します。例えば、配列を含まない属性でコラボレーションが Get Business Object Array を使用する場合に、この例外を発生させることができます。

### 注

Get Business Object Array 機能ブロックは、現在のビジネス・オブジェクトから属性値を検索し、属性値のコピーを戻します。このメソッドは、ソース・ビジネス・オブジェクトのこの属性に対するオブジェクト参照を戻しません。このため、ソース・ビジネス・オブジェクトの属性値の変更内容は、機能ブロックによって戻される値には適用されません。この機能ブロックは、使用されるたびに属性の新しいコピー (複製) を戻します。

### 関連情報

この機能ブロックは、BusObj.getBusObjArray() メソッドを基にしています。詳細については、386 ページの
『getBoolean()、getDouble()、getFloat()、getInt()、getLong()、get()、getBusObj()、getBusObjArray()、getLongText()、getString()』を参照してください。

## Get Business Object Type

現在のビジネス・オブジェクトのベースとなるビジネス・オブジェクト定義の名前を検索します。

### 入力

**Business Object**
現在のビジネス・オブジェクト (BusObj オブジェクト)。

### 出力

ビジネス・オブジェクト定義の名前を含む String を戻します。

### 注

ビジネス・オブジェクトのタイプは、ビジネス・オブジェクトの作成元となったビジネス・オブジェクト定義の名前です。

### 関連情報

この機能ブロックは、BusObj.getType() メソッドを基にしています。詳細については、 389 ページの『getType()』を参照してください。

---

## Get BusObj At

ビジネス・オブジェクト配列内の指定された指標にある要素を検索します。

**注:** この機能ブロックは、General¥APIs¥Business Object¥Array フォルダーにあります。

### 入力

**Business object array**
ビジネス・オブジェクト配列を表す BusObj[] オブジェクト。

**Index** 指標の位置を指定する整数。

### 出力

指定された指標にあるビジネス・オブジェクトを戻します。

---

## Get Double

ビジネス・オブジェクトから、double 型の単一属性の値を検索します。

### 入力

**Business Object**
現在のビジネス・オブジェクト (BusObj オブジェクト)。

**Attribute** 検索したい属性の名前を指定する String。

### 出力

指定された属性の値を double データ型として戻します。

### 例外

Get Double 機能ブロックは、`CollaborationException` 例外に対して以下の例外タイプを設定できます。

- `AttributeException` — 属性でアクセスする際に問題が生じる場合に設定します。例えば、数字で構成されない String 属性でコラボレーションが Get Double を使用する場合に、この例外を発生させることができます。

### 注

Get Double 機能ブロックは、現在のビジネス・オブジェクトから属性値を検索し、属性値のコピーを戻します。このメソッドは、ソース・ビジネス・オブジェクトのこの属性に対するオブジェクト参照を戻しません。このため、ソース・ビジネス・オブジェクトの属性値の変更内容は、機能ブロックによって戻される値には適用されません。この機能ブロックは、使用されるたびに属性の新しいコピー (複製) を戻します。

### 関連情報

この機能ブロックは、BusObj.getDouble() メソッドを基にしています。詳細については、 386 ページの
『getBoolean()、getDouble()、getFloat()、getInt()、getLong()、get()、
getBusObj()、getBusObjArray()、getLongText()、getString()』を参照してください。

---

## Get Float

ビジネス・オブジェクトから、float 型の単一属性の値を検索します。

### 入力

**Business Object**
現在のビジネス・オブジェクト (BusObj オブジェクト)。

**Attribute** 検索したい属性の名前を指定する String。

### 出力

指定された属性の値を float データ型として戻します。

### 例外

Get Float 機能ブロックは、`CollaborationException` 例外に対して以下の例外タイプを設定できます。

- `AttributeException` — 属性でアクセスする際に問題が生じる場合に設定します。例えば、数字で構成されない String 属性でコラボレーションが Get Float を使用する場合に、この例外を発生させることができます。

### 注

Get Float 機能ブロックは、現在のビジネス・オブジェクトから属性値を検索し、属性値のコピーを戻します。このメソッドは、ソース・ビジネス・オブジェクトのこの属性に対するオブジェクト参照を戻しません。このため、ソース・ビジネス・オブジェクトの属性値の変更内容は、機能ブロックによって戻される値には適用されません。この機能ブロックは、使用されるたびに属性の新しいコピー (複製) を戻します。

### 関連情報

この機能ブロックは、BusObj.getFloat() メソッドを基にしています。詳細については、 386 ページの

『getBoolean()、getDouble()、getFloat()、getInt()、getLong()、get()、getBusObj()、getBusObjArray()、getLongText()、getString()』を参照してください。

## Get Int

ビジネス・オブジェクトから、整数である単一属性の値を検索します。

### 入力

**Business Object**
現在のビジネス・オブジェクト (BusObj オブジェクト)。

**Attribute** 検索したい属性の名前を指定する String。

### 出力

指定された属性の値を整数として戻します。

### 例外

Get Int 機能ブロックは、`CollaborationException` 例外に対して以下の例外タイプを設定できます。

- `AttributeException` — 属性でアクセスする際に問題が生じる場合に設定します。例えば、数字で構成されない String 属性でコラボレーションが Get Int を使用する場合に、この例外を発生させることができます。

### 注

Get Int 機能ブロックは、現在のビジネス・オブジェクトから属性値を検索し、属性値のコピーを戻します。このメソッドは、ソース・ビジネス・オブジェクトのこの属性に対するオブジェクト参照を戻しません。このため、ソース・ビジネス・オブジェクトの属性値の変更内容は、機能ブロックによって戻される値には適用されません。この機能ブロックは、使用されるたびに属性の新しいコピー (複製) を戻します。

### 関連情報

この機能ブロックは、BusObj.getInt() メソッドを基にしています。詳細については、 386 ページの
『getBoolean()、getDouble()、getFloat()、getInt()、getLong()、get()、getBusObj()、getBusObjArray()、getLongText()、getString()』を参照してください。

## Get Locale

ビジネス・オブジェクトのデータに関連付けられているロケールを検索します。

### 入力

**Business Object**
ビジネス・オブジェクト (BusObj オブジェクト)。

### 出力

ビジネス・オブジェクトのロケールに関する情報を含んでいる Java Locale オブジェクトを戻します。この Locale オブジェクトは、java.util.Locale クラスのインスタンスでなければなりません。

### 注

Get Locale 機能ブロックは、ビジネス・オブジェクトのデータに関連付けられているロケールを戻します。このロケールは、コラボレーション実行時に使用されるコラボレーション・ロケールとは異なることがよくあります。

### 関連情報

この機能ブロックは、BusObj.getLocale() メソッドを基にしています。詳細については、388 ページの『getLocale()』を参照してください。

## Get Long

ビジネス・オブジェクトから、long データ型の単一属性の値を検索します。

### 入力

**Business Object**
現在のビジネス・オブジェクト (BusObj オブジェクト)。

**Attribute** 検索したい属性の名前を指定する String。

### 出力

指定された属性の値を long データ型として戻します。

### 例外

Get Long 機能ブロックは、`CollaborationException` 例外に対して以下の例外タイプを設定できます。

- `AttributeException` — 属性でアクセスする際に問題が生じる場合に設定します。例えば、数字で構成されない String 属性でコラボレーションが Get Long を使用する場合に、この例外を発生させることができます。

### 注

Get Long 機能ブロックは、現在のビジネス・オブジェクトから属性値を検索し、属性値のコピーを戻します。このメソッドは、ソース・ビジネス・オブジェクトのこの属性に対するオブジェクト参照を戻しません。このため、ソース・ビジネス・オブジェクトの属性値の変更内容は、機能ブロックによって戻される値には適用されません。この機能ブロックは、使用されるたびに属性の新しいコピー (複製) を戻します。

### 関連情報

この機能ブロックは、BusObj.getLong() メソッドを基にしています。詳細については、386 ページの

『getBoolean()、getDouble()、getFloat()、getInt()、getLong()、get()、getBusObj()、getBusObjArray()、getLongText()、getString()』を参照してください。

## Get Long Text

ビジネス・オブジェクトから、long text 型の単一属性の値を検索します。

### 入力

**Business Object**
現在のビジネス・オブジェクト (BusObj オブジェクト)。

**Attribute** 検索したい属性の名前を指定する String。

### 出力

指定された属性の値を String として戻します。

### 例外

Get Long Text 機能ブロックは、`CollaborationException` 例外に対して以下の例外タイプを設定できます。

- `AttributeException` — 属性でアクセスする際に問題が生じる場合に設定します。例えば、不適切なタイプのデータを含む属性でコラボレーションが Get Long Text を使用する場合に、この例外を発生させることができます。

### 注

Get Long Text 機能ブロックは、現在のビジネス・オブジェクトから属性値を検索し、属性値のコピーを戻します。このメソッドは、ソース・ビジネス・オブジェクトのこの属性に対するオブジェクト参照を戻しません。このため、ソース・ビジネス・オブジェクトの属性値の変更内容は、機能ブロックによって戻される値には適用されません。この機能ブロックは、使用されるたびに属性の新しいコピー (複製) を戻します。

この機能ブロックは String オブジェクトを戻します。これは、InterChange Server Express の longtext 型が、サイズに上限のない String オブジェクトだからです。

### 関連情報

この機能ブロックは、BusObj.getLongText() メソッドを基にしています。詳細については、 386 ページの
『getBoolean()、getDouble()、getFloat()、getInt()、getLong()、get()、getBusObj()、getBusObjArray()、getLongText()、getString()』を参照してください。

## Get Object

ビジネス・オブジェクトから、オブジェクトである単一属性の値を検索します。

## 入力

**Business Object**
現在のビジネス・オブジェクト (BusObj オブジェクト)。

**Attribute** 属性の名前を指定する String、またはビジネス・オブジェクトの属性リストにおける属性の順序を指定する整数。

## 出力

指定された属性の値をオブジェクトとして戻します。

## 例外

Get Object 機能ブロックは、`CollaborationException` 例外に対して以下の例外タイプを設定できます。

- `AttributeException` — 属性でアクセスする際に問題が生じる場合に設定します。例えば、不適切なタイプのデータを含む属性でコラボレーションが Get Object を使用する場合に、この例外を発生させることができます。

## 注

Get Object 機能ブロックは、現在のビジネス・オブジェクトから属性値を検索し、属性値のコピーを戻します。このメソッドは、ソース・ビジネス・オブジェクトのこの属性に対するオブジェクト参照を戻しません。このため、ソース・ビジネス・オブジェクトの属性値の変更内容は、機能ブロックによって戻される値には適用されません。この機能ブロックは、使用されるたびに属性の新しいコピー (複製) を戻します。

## 関連情報

この機能ブロックは、BusObj.get() メソッドを基にしています。詳細については、386 ページの『getBoolean()、getDouble()、getFloat()、getInt()、getLong()、get()、getBusObj()、getBusObjArray()、getLongText()、getString()』を参照してください。

---

# Get String

ビジネス・オブジェクトから、String 型の単一属性の値を検索します。

## 入力

**Business Object**
現在のビジネス・オブジェクト (BusObj オブジェクト)。

**Attribute** 検索したい属性の名前を指定する String。

## 出力

指定された属性の値を含む String を戻します。

## 例外

Get String 機能ブロックは、`CollaborationException` 例外に対して以下の例外タイプを設定できます。

- `AttributeException` — 属性でアクセスする際に問題が生じる場合に設定します。例えば、存在しない属性でコラボレーションが Get String を使用する場合に、この例外を発生させることができます。

### 注

Get String 機能ブロックは、現在のビジネス・オブジェクトから属性値を検索し、属性値のコピーを戻します。このメソッドは、ソース・ビジネス・オブジェクトのこの属性に対するオブジェクト参照を戻しません。このため、ソース・ビジネス・オブジェクトの属性値の変更内容は、機能ブロックによって戻される値には適用されません。この機能ブロックは、使用されるたびに属性の新しいコピー (複製) を戻します。

### 関連情報

この機能ブロックは、BusObj.getString() メソッドを基にしています。詳細については、386 ページの
『getBoolean()、getDouble()、getFloat()、getInt()、getLong()、get()、getBusObj()、getBusObjArray()、getLongText()、getString()』を参照してください。

## Get Verb

現在のビジネス・オブジェクトの動詞を検索します。

### 入力

**Business Object**
現在のビジネス・オブジェクト (BusObj オブジェクト)。

### 出力

ビジネス・オブジェクトの動詞の名前を含む String を戻します。

### 注

この機能ブロックは、複数タイプの受信イベントを扱うシナリオで役立ちます。シナリオの最初のアクション・ノードが Get Verb を呼び出します。次にそのアクション・ノードからの出力遷移リンクで、戻されたストリングの内容をテストし、それぞれの出力遷移リンクが、使用可能な動詞の 1 つを処理する実行経路の先頭になるようにします。

### 関連情報

この機能ブロックは、BusObj.getVerb() メソッドを基にしています。詳細については、389 ページの『getVerb()』を参照してください。

## Is Blank

属性の値がゼロ長ストリングに設定されているかどうかを判別します。

### 入力

**Business Object**
現在のビジネス・オブジェクト (BusObj オブジェクト)。

**Attribute** 照会する属性の名前を指定する String。

### 出力

属性値がゼロ長ストリングの場合は `true` を戻し、ゼロ長ストリングでない場合は `false` を戻します。

### 注

ゼロ長ストリングは、ストリング "" と同等です。ゼロ長ストリングは、Is Null 機能ブロックによって存在が検出される null とは異なります。

コラボレーションでは、属性値を検索し、その検索値に対してなんらかの操作を行う必要がある場合、値を検索する前に Is Blank および Is Null を呼び出して属性に値があることを検査することができます。

### 関連情報

この機能ブロックは、BusObj.isBlank() メソッドを基にしています。詳細については、390 ページの『isBlank()』を参照してください。

## Is Business Object

値がビジネス・オブジェクトかどうかを判別します。

### 入力

**Value** 照会したいオブジェクト。

### 出力

値がビジネス・オブジェクトの場合は `true` を戻し、ビジネス・オブジェクトでない場合は `false` を戻します。

## Is Key

ビジネス・オブジェクト属性がキー属性として定義されているかどうかを判別します。

### 入力

**Business Object**
現在のビジネス・オブジェクト (BusObj オブジェクト)。

**Attribute** 属性の名前を指定する String。

### 出力

属性がキー属性の場合は `true` を戻し、キー属性でない場合は `false` を戻します。

## 関連情報

この機能ブロックは、BusObj.isKey() メソッドを基にしています。詳細については、 391 ページの『isKey()』を参照してください。

---

# Is Null

ビジネス・オブジェクトの属性の値がヌルかどうかを判別します。

## 入力

**Business Object**
現在のビジネス・オブジェクト (BusObj オブジェクト)。

**Attribute** 属性の名前を指定する String。

## 出力

属性値がヌルの場合は `true` を戻し、ヌルでない場合は `false` を戻します。

## 注

null は、ゼロ長ストリング値とは対照的に、値がないことを示します。ゼロ長ストリング値は、Is Blank 機能ブロックを呼び出すことによって検出されます。オブジェクトが null だと操作に失敗する可能性があるので、オブジェクトを使用する前に Is Null 機能ブロックによってテストしてください

以下の状況では、属性値が null であることがあります。

- 属性値が明示的に null に設定された。

  属性値は、Set Value 機能ブロックを使用して null に設定できます。
- 属性値が設定されていない。

  コラボレーションが New Business Object 機能ブロックを使用して新規ビジネス・オブジェクトを作成する場合には、すべての属性値が null に初期化されます。属性値が、作成時刻と Is Null 機能ブロックの呼び出し時刻の間に設定されていない場合は、値は null のままです。
- マッピング中に null が挿入された。

  コネクターから受け取ったビジネス・オブジェクトをコラボレーションが処理するときに、マッピング処理で null 値が挿入されることがあります。マッピング処理では、コネクターから受け取ったアプリケーション固有のビジネス・オブジェクトは、コラボレーションによって処理される汎用ビジネス・オブジェクトに変換されます。汎用ビジネス・オブジェクトの属性のうち、アプリケーション固有のオブジェクトにおいて等価な属性のない属性については、それぞれマッピングによって null 値が挿入されます。

**ヒント:** Java では null オブジェクトで操作することができないので、子ビジネス・オブジェクトまたは子ビジネス・オブジェクト配列である属性に対して操作を実行する前に、必ず Is Null を呼び出してください。

### 関連情報

この機能ブロックは、BusObj.isNull() メソッドを基にしています。詳細については、 391 ページの『isNull()』を参照してください。

## Is Required

ビジネス・オブジェクトの属性が、必須属性として定義されているかどうかを判別します。

### 入力

**Business Object**
現在のビジネス・オブジェクト (BusObj オブジェクト)。

**Attribute** 属性の名前を指定する String。

### 出力

属性が必須の場合は `true` を戻し、必須でない場合は `false` を戻します。

### 注

属性が必須として定義されている場合、その属性は null でない値を持つ必要があります。

### 関連情報

この機能ブロックは、BusObj.isRequired() メソッドを基にしています。詳細については、 392 ページの『isRequired()』を参照してください。

## Iterate Children

子ビジネス・オブジェクト配列内で繰り返します。

### 入力

**Business Object**
現在のビジネス・オブジェクト (BusObj オブジェクト)。

**Attribute** 属性の名前を指定する String。

**Current index**
現在の指標を指定する整数。

**Current element**
現在の要素 (BusObj オブジェクト)。

## Keys to String

ビジネス・オブジェクトの基本キー属性の値を検索し、その値を String として戻します。

### 入力

**Business Object**
現在のビジネス・オブジェクト (BusObj オブジェクト)。

### 出力

ビジネス・オブジェクト内のすべてのキー値が、属性の序数値の順序に並べられて連結されたストリング・オブジェクト。

### 注

この機能ブロックの出力には、属性の名前と値が含まれています。複数値は、スペースで区切られて連結された基本キー属性値です。例えば、1 つの基本キー属性 SS# が存在する場合、出力は次のようになります。

```
SS#=100408394
```

基本キー属性が `FirstName` および `LastName` の場合の出力は、以下のようになります。

```
FirstName=Nina LastName=Silk
```

### 関連情報

この機能ブロックは、BusObj.keysToString() メソッドを基にしています。詳細については、 393 ページの『keysToString()』を参照してください。

## New Business Object

指定されたタイプの新規ビジネス・オブジェクト・インスタンス (BusObj) を作成します。

### 入力

**Type** 作成するビジネス・オブジェクトのタイプを指定する String。

### 出力

指定されたタイプの新規ビジネス・オブジェクトを戻します。

### 関連情報

この機能ブロックは、Collaboration.BusObj() コンストラクターを基にしています。

## New Business Object Array

新規ビジネス・オブジェクト配列を作成します。

**注:** この機能ブロックは、General¥APIs¥Business Object¥Array フォルダーにあります。

### 入力

**Size** 配列のサイズを指定する整数。

### 出力

指定されたサイズのビジネス・オブジェクト配列を戻します。

## Set BusObj At

ビジネス・オブジェクト配列内の指定された指標に要素を設定します。

**注:** この機能ブロックは、General¥APIs¥Business Object¥Array フォルダーにあります。

### 入力

**Business object array**
ビジネス・オブジェクト配列を表す BusObj[] オブジェクト。

**Index** 要素の位置を指定する整数。

**Business Object**
設定する要素を表す BusObj オブジェクト。

## Set Content

現在のビジネス・オブジェクトの内容を別のビジネス・オブジェクトに設定します。これにより、2 つのビジネス・オブジェクトは内容を共有します。一方のビジネス・オブジェクトに変更が加えられると、その変更内容はもう一方のビジネス・オブジェクトに反映されます。

### 入力

**Business Object**
現在のビジネス・オブジェクト (BusObj オブジェクト)。

**Content** 現在のビジネス・オブジェクトに使用したい内容を持つビジネス・オブジェクト (BusObj オブジェクト)。

### 例外

Set Content 機能ブロックは、`CollaborationException` 例外に対して以下の例外タイプのいずれかを設定できます。

- `AttributeException` — 属性でアクセスする際に問題が生じる場合に設定します。
- `ObjectException` — ビジネス・オブジェクトの引き数が無効の場合に設定します。

### 関連情報

この機能ブロックは、BusObj.setContent() メソッドを基にしています。詳細については、「*マップ開発ガイド*」の BusObj 参照ページを参照してください。

## Set Default Attribute Values

ビジネス・オブジェクトのすべての属性をデフォルト値に設定します。

### 入力

**Business Object**
現在のビジネス・オブジェクト (BusObj オブジェクト)。

### 注

ビジネス・オブジェクト定義には、属性のデフォルト値を含めることができます。この機能ブロックは、このビジネス・オブジェクトの属性値を、定義でデフォルトと指定された値に設定します。

### 関連情報

この機能ブロックは、BusObj.setDefaultAttrValues() メソッドを基にしています。詳細については、 395 ページの『setDefaultAttrValues()』を参照してください。

## Set Keys

現在のビジネス・オブジェクトのキー属性の値を、別のビジネス・オブジェクトのキー属性の値に設定します。

### 入力

**From business object**
使用したいキー属性値を持つビジネス・オブジェクト (BusObj オブジェクト)。

**To business object**
他のビジネス・オブジェクトのキー属性値を受け取る、現在のビジネス・オブジェクト (BusObjオブジェクト)。

### 例外

Set Keys 機能ブロックは、`CollaborationException` 例外に対して以下の例外タイプのいずれかを設定できます。

- `AttributeException` — 属性でアクセスする際に問題が生じる場合に設定します。
- `ObjectException` — ビジネス・オブジェクトの引き数が無効の場合に設定します。

### 関連情報

この機能ブロックは、BusObj.setKeys() メソッドを基にしています。詳細については、 395 ページの『setKeys()』を参照してください。

## Set Locale

現在のビジネス・オブジェクトのロケールを設定します。

### 入力

**Business Object**
ロケールの設定対象ビジネス・オブジェクト (BusObj オブジェクト)。

**Locale** ビジネス・オブジェクトへ割り当てるロケールに関する情報が含まれている Java Locale オブジェクト。この Locale オブジェクトは、java.util.Locale クラスのインスタンスでなければなりません。

### 注

Set Locale 機能ブロックは、ビジネス・オブジェクトに関連付けられたデータにロケールを割り当てます。このロケールは、コラボレーション実行時に使用されるコラボレーション・ロケールとは異なることがあります。

### 関連情報

この機能ブロックは、BusObj.setLocale() メソッドを基にしています。詳細については、396 ページの『setLocale()』を参照してください。

## Set Value

ビジネス・オブジェクトの属性を、特定のデータ型の指定値に設定します。

### 入力

**Business Object**
現在のビジネス・オブジェクト (BusObj オブジェクト)。

**Attribute** 設定したい属性の名前を指定する String。

**Value** 属性の値。その属性にとって適切な型 (boolean、double、float、int、long、Object、String、または BusObj) でなければなりません。

### 例外

Set Value 機能ブロックは、`CollaborationException` 例外に対して以下の例外タイプを設定できます。

- `AttributeException` — 属性でアクセスする際に問題が生じる場合に設定します。

### 注

Set Value 機能ブロックは、現在のビジネス・オブジェクトの属性値を設定します。属性に値を割り当てるときに、オブジェクト参照を設定します。これにより、ソース・ビジネス・オブジェクトから属性値が複製されることはありません。このため、ソース・ビジネス・オブジェクトの値の変更内容は、Set Value 機能ブロックを呼び出すビジネス・オブジェクトの属性にも適用されます。

Set Value 機能ブロックには、以下の形式があります。

- 1 番目の形式は、機能ブロックの Value 入力によって指定されるデータ型の値を設定します。特定の基本データ型または InterChange Server Express 定義のデータ型の属性を設定する場合は、この形式を使用してください。
- 2 番目の形式は、あらゆる データ型の属性値を設定します。Value 入力のデータ型は `Object` なので、属性値としてあらゆるデータ型を送信できます。例えば、`BusObj` または `LongText` オブジェクト・タイプの属性を設定するには、Value 入力に `BusObj` または `LongText` オブジェクトを使用します。

### 関連情報

この機能ブロックは、BusObj.set() メソッドを基にしています。詳細については、393 ページの『set()』を参照してください。

## Set Value By Position

ビジネス・オブジェクトの属性を、属性の位置に基づいた指定値に設定します。

### 入力

**Business Object**
現在のビジネス・オブジェクト (BusObj オブジェクト)。

**Position**
ビジネス・オブジェクトの属性リストの属性の順序を指定する整数値。

**Value**
属性の値。その属性にとって適切な型 (Object、String、または BusObj) にする必要があります。

### 例外

Set Value By Position 機能ブロックは、`CollaborationException` 例外に対して以下の例外タイプを設定できます。

- `AttributeException` — 属性でアクセスする際に問題が生じる場合に設定します。

### 注

この機能ブロックの許容値は、Object、String、および BusObj データ型に限定されます。

### 関連情報

この機能ブロックは、BusObj.set() メソッドを基にしています。詳細については、393 ページの『set()』を参照してください。

## Set Value with Create

ビジネス・オブジェクトの属性を、特定のデータ型の指定値に設定します。このときに、値のオブジェクトが存在しない場合は、そのオブジェクトを作成します。

### 入力

**Business Object**
現在のビジネス・オブジェクト (BusObj オブジェクト)。

**Attribute** 設定したい属性の名前を指定する String。

**Verb** 動詞 Create を指定する String。

### 例外

Set Value with Create 機能ブロックは、CollaborationException 例外に対して以下の例外タイプを設定できます。

- AttributeException — 属性でアクセスする際に問題が生じる場合に設定します。

### 注

指定するオブジェクトが BusObj であり、ターゲットの属性に複数カーディナリティーの子ビジネス・オブジェクトが含まれる場合、BusObj は BusObjArray に最後の要素として追加されます。ただし、ターゲットの属性に BusObj が含まれている場合は、このビジネス・オブジェクトが元の値に置き換わります。

### 関連情報

この機能ブロックは、BusObj.setWithCreate() メソッドを基にしています。詳細については、 397 ページの『setWithCreate()』を参照してください。

## Set Verb

ビジネス・オブジェクトの動詞を設定します。

### 入力

**Business Object**
現在のビジネス・オブジェクト (BusObj オブジェクト)。

**Verb** ビジネス・オブジェクトで使用する動詞を指定する String。

### 注

この機能ブロックは、ビジネス・オブジェクトのマッピングで使用されます。Set Verb 機能ブロックは、コラボレーション・テンプレートで使用しないでください。コラボレーション・テンプレートでは、サービス呼び出しのプロパティーを入力することにより、出力ビジネス・オブジェクトの動詞を対話式に設定する必要があります。

### 関連情報

この機能ブロックは、BusObj.setVerb() メソッドを基にしています。詳細については、 396 ページの『setVerb()』を参照してください。

## Set Verb with Create

子ビジネス・オブジェクトの動詞を設定します。このときに、子ビジネス・オブジェクトが存在しない場合は、その子ビジネス・オブジェクトを作成します。

### 入力

**Business Object**
現在のビジネス・オブジェクト (BusObj オブジェクト)。

**Attribute** 属性の名前を指定する String。

**Verb** 動詞 Create を指定する String。

### 例外

Set Verb with Create 機能ブロックは、`CollaborationException` 例外に対して以下の例外タイプを設定できます。

- `AttributeException` — 属性でアクセスする際に問題が生じる場合に設定します。

### 注

Attribute 入力によって指定される属性のデータ型が `BusObj` で、値が null の場合は、子ビジネス・オブジェクトの新規インスタンスが作成され、Verb 入力値にその動詞が設定されます。この子ビジネス・オブジェクトのインスタンスがすでに存在している場合は、その動詞だけが設定されます。子ビジネス・オブジェクトが複数のカーディナリティーを持つ場合は、Attribute 入力によって添え字を指定する必要があります。

### 関連情報

この機能ブロックは、BusObj.setVerbWithCreate() メソッドを基にしています。

## Shallow Equals

子ビジネス・オブジェクトを除いた 2 つのビジネス・オブジェクトの値を比較して、それらが等しいかどうかを判別します。

### 入力

**Business Object 1**
比較する第 1 ビジネス・オブジェクト (BusObj オブジェクト)。

**Business Object 2**
比較する第 2 ビジネス・オブジェクト (BusObj オブジェクト)。

### 出力

すべての属性の値が同じ場合は `true` を戻し、同じでない場合は `false` を戻します。

### 例外

Shallow Equals 機能ブロックは、CollaborationException 例外に対して以下の例外タイプを設定できます。

- ObjectException — ビジネス・オブジェクトの引き数が無効の場合に設定します。

### 関連情報

この機能ブロックは、BusObj.equalsShallow() メソッドを基にしています。詳細については、385 ページの『equalsShallow()』を参照してください。

## Size

ビジネス・オブジェクト配列のサイズを検索します。

**注:** この機能ブロックは、General¥APIs¥Business Object¥Array フォルダーにあります。

### 入力

**Business object array**
ビジネス・オブジェクト配列を表す BusObj[] オブジェクト。

### 出力

配列のサイズを指定する整数を戻します。

## To String

ビジネス・オブジェクトのすべての属性の値をストリングとして戻します。

### 入力

**Business Object**
現在のビジネス・オブジェクト (BusObj オブジェクト)。

### 出力

ビジネス・オブジェクトに含まれるすべての属性値を含む String オブジェクト。

### 注

このメソッドを呼び出した結果のストリングは、例えば以下のようになります。

```
Name: GenEmployee
Verb: Create
Type: AfterImage
Attributes: (Name, Type, Value)

LastName:String, Davis
FirstName:String, Miles
SS#:String, 041-33-8989
Salary:Float, 15.00
ObjectEventId:String, MyConnector_922323619411_1
```

### 関連情報

この機能ブロックは、BusObj.toString() メソッドを基にしています。詳細については、 398 ページの『toString()』を参照してください。

## Valid Data

指定した値が、指定した属性にとっての有効なデータ型であるかどうかを判別します。

### 入力

**Business Object**
現在のビジネス・オブジェクト (BusObj オブジェクト)。

**Attribute** 属性の名前を指定する String。

**Value** 属性の値。指定可能な型は、Object、BusObj、BusObjArray、String、long、int、double、float、または boolean です。

### 出力

指定した値が有効なデータ型の場合は `true` を戻し、有効でない場合は `false` を戻します。

### 注

ターゲットの属性 (Attribute 入力によって指定される) によって渡される値の互換性を検査します。基準は以下のとおりです。

| | |
|---|---|
| 基本型 (string、long、int、double、float、boolean) の場合 | 値は属性のデータ型に変換可能である必要があります。 |
| `BusObj` の場合 | 値のデータ型はターゲットの属性値のデータ型と同じである必要があります。 |
| `BusObjArray` の場合 | 値は、属性のタイプと同じ (ビジネス・オブジェクト定義) タイプである `BusObj` または `BusObjArray` を指している必要があります。 |
| `Object` の場合 | 値は、`String`、`BusObj`、または `BusObjArray` である必要があります。対応する検証規則が適用されます。 |

### 関連情報

この機能ブロックは、BusObj.validData() メソッドを基にしています。詳細については、 398 ページの『validData()』を参照してください。

## Verb:Create

ビジネス・オブジェクト動詞 Create。

**注:** この機能ブロックは、General¥APIs¥Business Object¥Constants フォルダーにあります。

### 出力

動詞 Create を含む String を戻します。

## Verb:Delete

ビジネス・オブジェクト動詞 Delete。

**注:** この機能ブロックは、General¥APIs¥Business Object¥Constants フォルダーにあります。

### 出力

動詞 Delete を含む String を戻します。

## Verb:Retrieve

ビジネス・オブジェクト動詞 Retrieve。

**注:** この機能ブロックは、General¥APIs¥Business Object¥Constants フォルダーにあります。

### 出力

動詞 Retrieve を含む String を戻します。

## Verb:Update

ビジネス・オブジェクト動詞 Update。

**注:** この機能ブロックは、General¥APIs¥Business Object¥Constants フォルダーにあります。

### 出力

動詞 Update を含む String を戻します。

# 第 12 章 ビジネス・オブジェクト配列の機能ブロック

General¥APIs¥Business Object Array フォルダーにある機能ブロックは、ビジネス・オブジェクト配列の処理に関する基本機能を提供します。これらの機能ブロックは、コラボレーション API の BusObjArray クラスを基にしています。

以下のセクションでは、各機能ブロックの詳細を説明します。

*表 63. General¥APIs¥Business Object Array フォルダーの機能ブロックの要約*



## Add Element

ビジネス・オブジェクトを現在のビジネス・オブジェクト配列に追加します。

### 入力

**Business object array**

ビジネス・オブジェクト配列 (BusObjArray オブジェクトとして指定する)。

**Element**　追加したいビジネス・オブジェクト (BusObj オブジェクトとして指定する)。

### 例外

Add Element 機能ブロックは、`CollaborationException` 例外に対して以下の例外タイプを設定できます。

- `AttributeException`—要素が無効の場合に設定します。

### 関連情報

この機能ブロックは、BusObjArray.addElement() メソッドを基にしています。詳細については、 402 ページの『addElement()』を参照してください。

## Duplicate

オリジナルのビジネス・オブジェクト配列と同一のビジネス・オブジェクト配列を作成します。

### 入力

**Original business object array**
複製したいビジネス・オブジェクト配列 (BusObjArray オブジェクトとして指定する)。

### 出力

複製されたビジネス・オブジェクト配列を戻します。

### 関連情報

この機能ブロックは、BusObjArray.duplicate() メソッドを基にしています。詳細については、 403 ページの『duplicate()』を参照してください。

## Equals

2 つのビジネス・オブジェクト配列を比較して、それらが等しいかどうかを判別します。

### 入力

**Business object array 1**
比較する第 1 ビジネス・オブジェクト配列 (BusObjArray オブジェクトとして指定する)。

**Business object array 2**
比較する第 2 ビジネス・オブジェクト配列 (BusObjArray オブジェクトとして指定する)。

### 出力

配列が等しい場合は `true` を戻し、等しくない場合は `false` を戻します。

### 注

2 つのビジネス・オブジェクト配列の比較とは、要素の数とそれらの属性値を検査することです。

### 関連情報

この機能ブロックは、BusObjArray.equals() メソッドを基にしています。詳細については、 403 ページの『equals()』を参照してください。

## Get Element At

配列でのオブジェクトの位置を指定して、配列から単一のビジネス・オブジェクトを検索します。

### 入力

**Business object array**
ビジネス・オブジェクト配列 (BusObjArray オブジェクトとして指定する)。

**Index** 指標の位置を指定する整数。

### 出力

指定されたビジネス・オブジェクトを戻します。

### 例外

Get Element At 機能ブロックは、`CollaborationException` 例外に対して以下の例外タイプを設定できます。

- `AttributeException`—要素が無効の場合に設定します。

### 関連情報

この機能ブロックは、BusObjArray.elementAt() メソッドを基にしています。詳細については、 403 ページの『elementAt()』を参照してください。

## Get Elements

ビジネス・オブジェクト配列の内容を検索します。

### 入力

**Business object array**
ビジネス・オブジェクト配列 (BusObjArray オブジェクトとして指定する)。

### 出力

配列内のビジネス・オブジェクトを戻します。

### 例外

Get Elements 機能ブロックは、CollaborationException 例外に対して以下の例外タイプを設定できます。

- ObjectException — 要素のいずれかが無効である場合に設定します。

### 関連情報

この機能ブロックは、BusObjArray.getElements() メソッドを基にしています。詳細については、 404 ページの『getElements()』を参照してください。

## Get Last Index

ビジネス・オブジェクト配列から、最後の有効な指標を検索します。

### 入力

**Business object array**
ビジネス・オブジェクト配列 (BusObjArray オブジェクトとして指定する)。

### 出力

最後の指標を整数として戻します。

### 注

Get Last Index 機能ブロックは、Size 機能ブロックと類似の機能を実行しますが、重要な相違点として、Size 機能ブロックは Get Last Index 機能ブロックよりも 1 大きい値を戻すことが挙げられます。この相違は、入力として使用される BusObjArray オブジェクトの値が相対配列ではゼロになることによって生じます。

### 関連情報

この機能ブロックは、BusObjArray.getLastIndex() メソッドを基にしています。詳細については、 404 ページの『getLastIndex()』を参照してください。

## Is Business Object Array

オブジェクトがビジネス・オブジェクト配列 (BusObjArray) かどうかを判別します。

### 入力

**Value** テストするオブジェクトの名前。

### 出力

値がビジネス・オブジェクト配列の場合は true を戻し、ビジネス・オブジェクト配列でない場合は false を戻します。

## Max Attribute Value

ビジネス・オブジェクト配列のすべての要素のうち、指定した属性の最大値を検索します。

### 入力

**Business object array**
ビジネス・オブジェクト配列 (BusObjArray オブジェクトとして指定する)。

**Attribute** 属性名を指定する String。

### 出力

指定された属性の最大値を含む String を戻します。

### 例外

Max Attribute Value 機能ブロックは、以下の例外をスローすることができ、どちらの例外も `CollaborationException` からサブクラス化されます。

- `UnknownAttributeException` — 指定した属性が、渡されたビジネス・オブジェクトの有効な属性ではない場合です。
- `UnsupportedAttributeTypeException`— 指定した属性のデータ型が、『注記』セクションにリストされているサポート対象の属性データ型ではない場合です。

Max Attribute Value 機能ブロックは、それぞれの例外に対して `AttributeException` 例外タイプを設定できます。

### 注

Max Attribute Value 機能ブロックは、この `BusObjArray` のビジネス・オブジェクトの中から指定された属性の最大値を検索します。例えば、3 つの Employee オブジェクトが使用されていて、その属性が Float 型の Salary である場合、最大の給与を表すストリングが戻されます。

`BusObjArray` 内の要素のうち、指定した属性の値が null である要素は無視されます。すべての要素について指定した属性値が null の場合、null が戻されます。

属性のデータ型が String の場合、Max Attribute Value は字句的に最長のストリングである属性値を戻します。

### 関連情報

この機能ブロックは、BusObjArray.max() メソッドを基にしています。詳細については、405 ページの『max()』を参照してください。

## Max Business Object Array

指定した属性の最大値を持つビジネス・オブジェクトをビジネス・オブジェクト配列 (BusObjArray オブジェクト) として戻します。

## 入力

**Business object array**
ビジネス・オブジェクト配列 (BusObjArray オブジェクトとして指定する)。

**Attribute** 属性名を指定する String。

## 出力

BusObjArray オブジェクト形式のビジネス・オブジェクトのリスト。

## 例外

Max Business Object Array 機能ブロックは、以下の例外をスローすることができ、どちらの例外も `CollaborationException` からサブクラス化されます。

- `UnknownAttributeException` — 指定した属性が、渡されたビジネス・オブジェクトの有効な属性ではない場合です。
- `UnsupportedAttributeTypeException`— 指定した属性のデータ型が、『注記』セクションにリストされているサポート対象の属性データ型ではない場合です。

Max Business Object Array 機能ブロックは、それぞれの例外に対して `AttributeException` 例外タイプを設定できます。

## 注

Max Business Object Array 機能ブロックは、指定した属性の最大値を持つ 1 つ以上のビジネス・オブジェクトを検索し、これらのビジネス・オブジェクトを `BusObjArray` オブジェクトとして戻します。

例えば、Employee ビジネス・オブジェクトを含むビジネス・オブジェクト配列があり、入力引き数が Float 型の属性 Salary であるとします。この機能ブロックは、すべての Employee ビジネス・オブジェクトの中で Salary の最大値を判別し、最大値を含むビジネス・オブジェクトを戻します。Salary の最大値を持つビジネス・オブジェクトが複数ある場合、機能ブロックはそれらのビジネス・オブジェクトをすべて戻します。

指定した属性が null を含むビジネス・オブジェクトは無視されます。配列内のすべてのビジネス・オブジェクトで値が null である場合、null が戻されます。

属性のデータ型が String の場合、機能ブロックは字句的に最長のストリングを戻します。

## 関連情報

この機能ブロックは、BusObjArray.maxBusObjArray() メソッドを基にしています。詳細については、406 ページの『maxBusObjArray()』を参照してください。

# Max Business Objects

指定した属性の最大値を持つビジネス・オブジェクトを BusObj オブジェクトの配列として戻します。

## 入力

**Business object array**
ビジネス・オブジェクト配列 (BusObjArray オブジェクトとして指定する)。

**Attribute** 属性名を指定する String。

## 出力

BusObj[] オブジェクト形式のビジネス・オブジェクトのリスト。

## 例外

Max Business Objects 機能ブロックは、以下の例外をスローすることができ、どちらの例外も `CollaborationException` からサブクラス化されます。

- `UnknownAttributeException` — 指定した属性が、渡されたビジネス・オブジェクトの有効な属性ではない場合です。
- `UnsupportedAttributeTypeException`— 指定した属性のデータ型が、『注記』セクションにリストされているサポート対象の属性データ型ではない場合です。

Max Business Objects 機能ブロックは、それぞれの例外に対して `AttributeException` 例外タイプを設定できます。

## 注

Max Business Objects 機能ブロックは、指定した属性の最大値を持つ 1 つ以上のビジネス・オブジェクトを検索し、これらのビジネス・オブジェクトを `BusObj` オブジェクトの配列として戻します。

例えば、Employee ビジネス・オブジェクトを含むビジネス・オブジェクト配列があり、入力引き数が Float 型の属性 Salary であるとします。この機能ブロックは、すべての Employee ビジネス・オブジェクトの中で Salary の最大値を判別し、最大値を含むビジネス・オブジェクトを戻します。Salary の最大値を持つビジネス・オブジェクトが複数ある場合、機能ブロックはそれらのビジネス・オブジェクトをすべて戻します。

指定した属性が null を含むビジネス・オブジェクトは無視されます。配列内のすべてのビジネス・オブジェクトで値が null である場合、null が戻されます。

属性のデータ型が String の場合、機能ブロックは字句的に最長のストリングを戻します。

## 関連情報

この機能ブロックは、BusObjArray.maxBusObjs() メソッドを基にしています。詳細については、 407 ページの『maxBusObjs()』を参照してください。

# Min Attribute Value

ビジネス・オブジェクト配列のすべての要素のうち、指定した属性の最小値を検索します。

## 入力

**Business object array**
ビジネス・オブジェクト配列 (BusObjArray オブジェクトとして指定する)。

**Attribute** 属性名を指定する String。

## 出力

指定された属性の最大値を含む String を戻します。

## 例外

Min Attribute Value 機能ブロックは、以下の例外をスローすることができ、どちらの例外も `CollaborationException` からサブクラス化されます。

- `UnknownAttributeException` — 指定した属性が、渡されたビジネス・オブジェクトの有効な属性ではない場合です。
- `UnsupportedAttributeTypeException`— 指定した属性のデータ型が、『注記』セクションにリストされているサポート対象の属性データ型ではない場合です。

Min Attribute Value 機能ブロックは、それぞれの例外に対して `AttributeException` 例外タイプを設定できます。

## 注

Min Attribute Value 機能ブロックは、このビジネス・オブジェクト配列のビジネス・オブジェクトの中から、指定された属性の最小値を検索します。

例えば、Employee ビジネス・オブジェクトを含むビジネス・オブジェクト配列があり、入力引き数が Float 型の属性 Salary であるとします。この機能ブロックは、すべての Employee ビジネス・オブジェクトの中で Salary の最小値を判別し、最小値を含むビジネス・オブジェクトを戻します。Salary の最小値を持つビジネス・オブジェクトが複数ある場合、機能ブロックはそれらのビジネス・オブジェクトをすべて戻します。

指定した属性が null を含むビジネス・オブジェクトは無視されます。配列内のすべてのビジネス・オブジェクトで値が null である場合、null が戻されます。

属性のデータ型が String の場合は、字句的に最短のストリングが戻されます。

## 関連情報

この機能ブロックは、BusObjArray.min() メソッドを基にしています。詳細については、408 ページの『min()』を参照してください。

# Min Business Object Array

指定した属性の最小値を持つビジネス・オブジェクトを、ビジネス・オブジェクト配列 (BusObjArray オブジェクト) として戻します。

## 入力

**Business object array**
ビジネス・オブジェクト配列 (BusObjArray オブジェクトとして指定する)。

**Attribute** 属性名を指定する String。

## 出力

BusObjArray オブジェクト形式のビジネス・オブジェクトのリスト。

## 例外

Min Business Object Array 機能ブロックは、以下の例外をスローすることができ、どちらの例外も CollaborationException からサブクラス化されます。

- UnknownAttributeException — 指定した属性が、渡されたビジネス・オブジェクトの有効な属性ではない場合です。
- UnsupportedAttributeTypeException— 指定した属性のデータ型が、『注記』セクションにリストされているサポート対象の属性データ型ではない場合です。

Min Business Object Array 機能ブロックは、それぞれの例外に対して AttributeException 例外タイプを設定できます。

## 注

Min Business Object Array 機能ブロックは、指定した属性の最小値を持つ 1 つ以上のビジネス・オブジェクトを検索し、これらのビジネス・オブジェクトを BusObjArray オブジェクトとして戻します。

例えば、Employee ビジネス・オブジェクトを含むビジネス・オブジェクト配列があり、入力引き数が Float 型の属性 Salary であるとします。この機能ブロックは、すべての Employee ビジネス・オブジェクトの中で Salary の最小値を判別し、最小値を含むビジネス・オブジェクトを戻します。Salary の最大値を持つビジネス・オブジェクトが複数ある場合、機能ブロックはそれらのビジネス・オブジェクトをすべて戻します。

指定した属性が null を含むビジネス・オブジェクトは無視されます。配列内のすべてのビジネス・オブジェクトで値が null である場合、null が戻されます。

属性のデータ型が String の場合は、字句的に最短のストリングが戻されます。

## 関連情報

この機能ブロックは、BusObjArray.minBusObjArray() メソッドを基にしています。詳細については、 409 ページの『minBusObjArray()』を参照してください。

# Min Business Objects

指定した属性の最小値を持つビジネス・オブジェクトを BusObj オブジェクトの配列として戻します。

## 入力

**Business object array**
ビジネス・オブジェクト配列 (BusObjArray オブジェクトとして指定する)。

**Attribute** 属性名を指定する String。

## 出力

BusObj[] オブジェクト形式のビジネス・オブジェクトのリスト。

## 例外

Min Business Objects 機能ブロックは、以下の例外をスローすることができ、どちらの例外も CollaborationException からサブクラス化されます。

- UnknownAttributeException — 指定した属性が、渡されたビジネス・オブジェクトの有効な属性ではない場合です。
- UnsupportedAttributeTypeException— 指定した属性のデータ型が、『注記』セクションにリストされているサポート対象の属性データ型ではない場合です。

Min Business Objects 機能ブロックは、それぞれの例外に対して AttributeException 例外タイプを設定できます。

## 注

Min Business Objects 機能ブロックは、指定した属性の最大値を持つ 1 つ以上のビジネス・オブジェクトを検索し、これらのビジネス・オブジェクトを BusObj オブジェクトの配列として戻します。

例えば、Employee ビジネス・オブジェクトを含むビジネス・オブジェクト配列があり、入力引き数が Float 型の属性 Salary であるとします。この機能ブロックは、すべての Employee ビジネス・オブジェクトの中で Salary の最小値を判別し、最小値を含むビジネス・オブジェクトを戻します。Salary の最大値を持つビジネス・オブジェクトが複数ある場合、機能ブロックはそれらのビジネス・オブジェクトをすべて戻します。

指定した属性が null を含むビジネス・オブジェクトは無視されます。配列内のすべてのビジネス・オブジェクトで値が null である場合、null が戻されます。

属性のデータ型が String の場合は、字句的に最短のストリングが戻されます。

## 関連情報

この機能ブロックは、BusObjArray.minBusObjs() メソッドを基にしています。詳細については、410 ページの『minBusObjs()』を参照してください。

---

# Remove All Elements

ビジネス・オブジェクト配列からすべての要素を除去します。

### 入力

**Business object array**
要素が除去されるビジネス・オブジェクト配列 (BusObjArray)。

### 関連情報

この機能ブロックは、BusObjArray.removeAllElements() メソッドを基にしています。詳細については、 411 ページの『removeAllElements()』を参照してください。

## Remove Element

ビジネス・オブジェクト配列から 1 つのビジネス・オブジェクト要素を除去します。

### 入力

**Business object array**
要素が除去されるビジネス・オブジェクト配列 (BusObjArray)。

**Element** 配列から除去されるビジネス・オブジェクト (BusObj) 要素。

### 例外

Remove Element 機能ブロックは、`CollaborationException` 例外に対して以下の例外タイプを設定できます。

- `AttributeException`—要素が無効の場合に設定します。

### 注

配列から要素を削除すると、配列のサイズが変わり、既存の要素の指標が変更されます。

### 関連情報

この機能ブロックは、BusObjArray.removeElement() メソッドを基にしています。詳細については、 411 ページの『removeElement()』を参照してください。

## Remove Element At

ビジネス・オブジェクト配列内の特定の位置からビジネス・オブジェクト要素を除去します。

### 入力

**Business object array**
要素が除去されるビジネス・オブジェクト配列 (BusObjArray)。

**Index** 除去される要素の指標位置 (整数として指定する)。

### 例外

Remove Element At 機能ブロックは、`CollaborationException` 例外に対して以下の例外タイプを設定できます。

- AttributeException—要素が無効の場合に設定します。

### 注

配列から要素を除去すると、配列のサイズが変わり、場合によっては既存の要素の指標が変更されます。

### 関連情報

この機能ブロックは、BusObjArray.removeElementAt() メソッドを基にしています。詳細については、411 ページの『removeElementAt()』を参照してください。

## Set Element At

ビジネス・オブジェクト配列内のビジネス・オブジェクトの値を設定します。

### 入力

**Business object array**
要素の値が設定されるビジネス・オブジェクト配列 (BusObjArray 型のオブジェクトとして指定する)。

**Element**
値が設定されるビジネス・オブジェクト要素 (BusObj 型のオブジェクトとして指定する)。

**Index**
設定されるビジネス・オブジェクト要素の指標位置 (整数として指定する)。

### 例外

Set Element At 機能ブロックは、CollaborationException 例外に対して以下の例外タイプを設定できます。

- AttributeException—要素が無効の場合に設定します。

### 注

この機能ブロックは、配列の指定された位置にあるビジネス・オブジェクトの値を、入力ビジネス・オブジェクトの値に設定します。

### 関連情報

この機能ブロックは、BusObjArray.setElementAt() メソッドを基にしています。詳細については、412 ページの『setElementAt()』を参照してください。

## Size

ビジネス・オブジェクト配列内の要素の数を判別します。

### 入力

**Business object array**
サイズを判断したいビジネス・オブジェクト配列 (BusObjArray 型のオブジェクトとして指定する)。

### 出力

配列の要素数を指定する整数を戻します。

### 関連情報

この機能ブロックは、BusObjArray.size() メソッドを基にしています。詳細については、 413 ページの『size()』を参照してください。

## Sum

ビジネス・オブジェクト配列内のすべてのビジネス・オブジェクトの、指定した属性の値を合計します。

### 入力

**Business object array**
ビジネス・オブジェクト配列 (BusObjArray 型のオブジェクトとして指定する)。

**Attribute** 属性の名前を指定する String。

### 出力

ビジネス・オブジェクトのリストから、指定された属性の合計を (double データ型として) 戻します。

### 例外

Sum 機能ブロックは、以下の例外をスローすることができ、どちらの例外も `CollaborationException` からサブクラス化されます。

- `UnknownAttributeException` — 指定した属性が、渡されたビジネス・オブジェクトの有効な属性ではない場合です。
- `UnsupportedAttributeTypeException` — 指定した属性のデータ型が、『注記』セクションにあるサポート対象の属性データ型でない場合です。

Sum 機能ブロックは、それぞれの例外に対して `AttributeException` 例外タイプを設定できます。

### 関連情報

この機能ブロックは、BusObjArray.sum() メソッドを基にしています。詳細については、 413 ページの『sum()』を参照してください。

## Swap

ビジネス・オブジェクト配列内の 2 つのビジネス・オブジェクトの位置を逆にします。

### 入力

**Business object array**
ビジネス・オブジェクト配列 (BusObjArray 型のオブジェクトとして指定する)。

**Index 1** 第 1 ビジネス・オブジェクトの位置を指定する整数。

**Index 2** 第 2 ビジネス・オブジェクトの位置を指定する整数。

### 関連情報

この機能ブロックは、BusObjArray.swap() メソッドを基にしています。詳細については、 413 ページの『swap()』を参照してください。

---

## To String

ビジネス・オブジェクト配列内の値を検索し、それらを単一の String として戻します。

### 入力

**Business object array**
ビジネス・オブジェクト配列 (BusObjArray 型のオブジェクトとして指定する)。

### 出力

ビジネス・オブジェクト配列内のすべての値を含む String を戻します。

### 関連情報

この機能ブロックは、BusObjArray.toString() メソッドを基にしています。詳細については、 414 ページの『toString()』を参照してください。

# 第 13 章 コラボレーション・テンプレートの機能ブロック

コラボレーション・テンプレートの機能ブロックには、コラボレーション・オブジェクトで操作するための基本的な機能があります。これらの機能ブロックは、以下のフォルダーで編成されています。

- General¥APIs¥Collaboration Template— コラボレーション・オブジェクトでの作業に使用。
- General¥APIs¥Collaboration Template¥Exception— コラボレーション・テンプレート内で新規例外オブジェクトを作成するために使用。
- General¥APIs¥Collaboration Template¥Exception¥Constants— コラボレーション例外オブジェクト内の特定の例外タイプを表すために使用。

以下のセクションでは、それぞれのコラボレーション・テンプレート機能ブロックについて詳しく説明します。

*表 64. コラボレーション・テンプレートの機能ブロックの要約*

| ロケーション | 機能ブロック | ページ |
|---|---|---|
| General¥APIs¥Collaboration Template | Get Locale | 290 |
| | Get Message | 291 |
| | Get Message with Parameter | 291 |
| | Get Name | 292 |
| | Get Property | 292 |
| | Get Property Array | 292 |
| | Implicit DB Bracketing | 293 |
| | Is Trace Enabled | 294 |
| | Property Exists | 295 |
| | Send Email | 300 |
| General¥APIs¥Collaboration Template¥Exception | Raise Collaboration Exception | 295 |
| | Raise Collaboration Exception 1 | 296 |
| | Raise Collaboration Exception 2 | 297 |
| | Raise Collaboration Exception 3 | 298 |
| | Raise Collaboration Exception 4 | 298 |
| | Raise Collaboration Exception 5 | 299 |
| | Raise Collaboration Exception with Parameter | 300 |

表 64. コラボレーション・テンプレートの機能ブロックの要約 (続き)

| ロケーション | 機能ブロック | ページ |
|---|---|---|
| General¥APIs¥Collaboration Template¥Exception¥Constants | AnyException | 290 |
| | AttributeException | 290 |
| | JavaException | 294 |
| | ObjectException | 294 |
| | OperationException | 295 |
| | ServiceCallException | 301 |
| | SystemException | 301 |
| | TransactionException | 302 |

## AnyException

例外のタイプを表す定数です。

**注:** この機能ブロックは、General¥APIs¥Collaboration Template¥Exception¥Constants フォルダーにあります。

### 出力

値が「AnyException」の String を戻します。

## AttributeException

属性のアクセス問題を表す定数 (例えば、コラボレーションが String ベースの属性に Get Double 機能ブロックを使用する場合や、存在しない属性で Get String 機能ブロックを使用した場合) です。

**注:** この機能ブロックは、General¥APIs¥Collaboration Template¥Exception¥Constants フォルダーにあります。

### 出力

値が「AttributeException」の String を戻します。

## Get Locale

現行コラボレーション・オブジェクトのコラボレーション・ロケールを検索します。

### 入力

**Collaboration** 現行コラボレーション・オブジェクト。

### 出力

コラボレーション・ロケールの言語コードと国別コードを含む Java Locale オブジェクトを戻します。この Locale オブジェクトは、java.util.Locale クラスのインスタンスでなければなりません。

### 注

Get Locale 機能ブロックは、現行フローのロケールを戻します。このフローのロケールは、コラボレーション・オブジェクトのトリガー・ビジネス・オブジェクトと関連付けられているロケールです。

### 関連情報

この機能ブロックは、BaseCollaboration.getLocale() メソッドを基にしています。詳細については、 368 ページの『getLocale()』を参照してください。

## Get Message

コラボレーション・メッセージ・ファイルからメッセージを取得します。

### 入力

**Collaboration** 現行コラボレーション・オブジェクト。

**ID** コラボレーションのメッセージ・ファイルにあるメッセージのメッセージ番号を指定する整数。メッセージ・ファイルはメッセージ番号によって索引付けされます。

### 出力

ID 入力によって指定されるメッセージのテキストを含む String オブジェクトを戻します。

### 関連情報

この機能ブロックは、BaseCollaboration.getMessage() メソッドを基にしています。詳細については、 368 ページの『getMessage()』を参照してください。

## Get Message with Parameter

コラボレーション・メッセージ・ファイルからメッセージを取得します。

### 入力

**Collaboration** 現行コラボレーション・オブジェクト。

**ID** コラボレーションのメッセージ・ファイルにあるメッセージのメッセージ番号を指定する整数。メッセージ・ファイルはメッセージ番号によって索引付けされます。

**Parameters** メッセージ・パラメーター値の配列。各パラメーターは順次的に解決され、メッセージ・テキスト内のパラメーターになります。メッセージ内 (コラボレーション・メッセージ・ファイル内) では、メッセージ・パラメーターは中括弧で囲まれた整数 (例えば、{1}) で示されます。

### 出力

ID 入力と Parameters 入力によって指定されるメッセージのテキストを含む String オブジェクトを戻します。

### 注

Get Message with Parameter 機能ブロックでは、メッセージ番号とメッセージ・パラメーター値の配列を使用します。コラボレーション・メッセージ・ファイルから関連するメッセージを取得して、パラメーター配列内のオブジェクトでメッセージ・パラメーターを置き換え、結果のメッセージを `String` オブジェクトとして戻します。

### 関連情報

この機能ブロックは、BaseCollaboration.getMessage() メソッドを基にしています。詳細については、 368 ページの『getMessage()』を参照してください。

## Get Name

コラボレーション・オブジェクトの名前を検索します。

### 入力

**Collaboration** 現行コラボレーション・オブジェクト。

### 出力

現行コラボレーション・オブジェクトの名前を含む String を戻します。

### 関連情報

この機能ブロックは、BaseCollaboration.getName() メソッドを基にしています。詳細については、 370 ページの『getName()』を参照してください。

## Get Property

コラボレーション構成プロパティーの値を検索します。

### 入力

**Collaboration** 現行コラボレーション・オブジェクト。

**Property name**
照会するコラボレーション構成プロパティーを指定する String。

### 出力

指定されたコラボレーション構成プロパティーの値を含む String を戻します。

## Get Property Array

複数要素のコラボレーション構成プロパティーの値を検索します。

### 入力

**Collaboration** 現行コラボレーション・オブジェクト。

**Property name**
照会するコラボレーション構成プロパティーを指定する String。

### 出力

String オブジェクトの配列を戻します。配列のそれぞれの String オブジェクトには、コラボレーション構成プロパティーの 1 つの要素の値が含まれています。

## Implicit DB Bracketing

コラボレーション・オブジェクトが取得する任意の接続に対して、コラボレーション・オブジェクトが使用するトランザクション・プログラミング・モデルを検索します。

### 入力

**Collaboration** 現行コラボレーション・オブジェクト。

### 出力

すべてのデータベース接続で使用されるトランザクション・プログラミング・モデルを示す boolean 値を戻します。

- 値が `true` の場合は、すべての接続が*暗黙的な* トランザクション・ブラケットを使用していることを示します。
- 値が `false` の場合は、すべての接続が*明示的な* トランザクション・ブラケットを使用していることを示します。

### 注

Implicit DB Bracketing 機能ブロックは、コラボレーション・オブジェクトが前提とし、コラボレーション・オブジェクトが取得するすべての 接続によって使用される、トランザクション・プログラミング・モデルを示す boolean 値を戻します。この機能ブロックは、接続を取得する前に現在のトランザクション・プログラミング・モデルが適切かどうかを確認するのに役立ちます。

**注:** Get Database Connection 機能ブロック (General¥APIs¥Database Connection フォルダーに配置) を使用して、特定の接続のトランザクション・プログラミング・モデルをオーバーライドできます。

### 関連情報

この機能ブロックは、BaseCollaboration.implicitDBTransactionBracketing() メソッドを基にしています。詳細については、 370 ページの
『implicitDBTransactionBracketing()』を参照してください。

## Is Trace Enabled

指定されたトレース・レベルとコラボレーションの現在のトレース・レベルを比較します。

### 入力

**Collaboration** 現行コラボレーション・オブジェクト。

**Trace level** 現在のトレース・レベルと比較するトレース・レベルを指定する整数。

### 出力

指定されたトレース・レベルに現行システムのトレース・レベルが設定されると、`true` を戻します。2 つのトレース・レベルが異なる場合、`false` を戻します。

### 注

Is Trace Enabled 機能ブロックは、トレース・メッセージを記録するかどうかを判断する場合に役立ちます。トレースを実行するとパフォーマンスが低下することがあるので、Is Trace Enabled 機能ブロックは特にプロジェクトの開発段階に関連して使用されます。

### 関連情報

この機能ブロックは、BaseCollaboration.isTraceEnabled() メソッドを基にしています。詳細については、371 ページの『isTraceEnabled()』を参照してください。

## JavaException

コラボレーション・テンプレートのビジネス・ロジックの Java コードに関係する問題を表す定数です。

**注:** この機能ブロックは、General¥APIs¥Collaboration Template¥Exception¥Constants フォルダーにあります。

### 出力

値が「JavaException」の String を戻します。

## ObjectException

無効なビジネス・オブジェクトを機能ブロックに渡すことによって、または null オブジェクトにアクセスすることによって発生するエラーを表す定数です。

**注:** この機能ブロックは、General¥APIs¥Collaboration Template¥Exception¥Constants フォルダーにあります。

### 出力

値が「ObjectException」の String を戻します。

## OperationException

不適切に構成され、送信不能なサービス呼び出しによって発生するエラーを表す定数です。

**注:** この機能ブロックは、General¥APIs¥Collaboration Template¥Exception¥Constants フォルダーにあります。

### 出力

値が「OperationException」の String を戻します。

## Property Exists

指定されたコラボレーション構成プロパティーが存在するかどうかを判別します。

### 入力

**Collaboration** 現行コラボレーション・オブジェクト。

**Property name**
照会するコラボレーション構成プロパティーを指定する String。

### 出力

コラボレーション構成プロパティーが存在する場合は `True` を、存在しない場合は `False` を戻します。

## Raise Collaboration Exception

1 つ上の実行レベルに例外を生成するためにコラボレーション例外を準備します。この機能ブロックは、指定された例外タイプとメッセージ・ストリングを持つ新規の例外オブジェクトを作成します。ストリングとして保管された例外メッセージを渡すには、この形式を使用します。

**注:** この機能ブロックは、General¥APIs¥Collaboration Template¥Exception フォルダーにあります。

### 入力

**Collaboration** 現行コラボレーション・オブジェクト。

**Exception type**
例外タイプを指定する String。

**messageNum** 例外オブジェクトに関連付けられているメッセージの番号を指定する整数。

### 注

Raise Collaboration Exception 機能ブロックは、1 つ上の実行レベルに例外を生成するために、コラボレーション例外を準備します。コラボレーション・ランタイム環境で Raise Collaboration Exception 機能ブロックを実行すると、コラボレーションの実行が例外状態に変更され、アクティビティー・ダイアグラムのロジックが続行さ

れます。生成された例外に対するアクティビティー・ダイアグラムの応答方法は、実行経路の終端ノードによって以下のように異なります。

- 実行経路が終了成功で終わる場合、1 つ上の実行レベルに制御が渡されます。

  この親ダイアグラムの次ノードが決定ノードの場合、コラボレーション・ランタイム環境で、生成された例外を処理するこの決定ノード内の実行分岐が検査されます。この親ダイアグラムは、currentException システム変数を使用して生成された例外にアクセスできます。
- 実行経路が終了障害で終わる場合、コラボレーション・ランタイム環境によりコラボレーションが終了され、コラボレーションのログにエントリーが作成されて、未解決のフローが作成されます。

  コラボレーション・ランタイム環境により未解決のフローが、生成された例外が含む任意の例外テキストと関連付けされます。この例外に例外テキストが含まれていない場合、コラボレーション・ランタイム環境はデフォルトのメッセージを使用します。

  ```
  Scenario failed.
  ```

  障害が発生したときには、単に失敗として終了させるのではなく、例外を明示的に生成させることを推奨します。コードによりコラボレーション・ランタイム環境に明示的に例外を生成すると、管理者は Flow Manager を使用して、未解決のフローの一部として例外テキストを表示できます。詳細については、 177 ページの『例外の発生』を参照してください。

  Raise Collaboration Exception 機能ブロックはシリーズで存在し、それぞれが若干異なるタスクを実行します。Raise Collaboration Exception 1、Raise Collaboration Exception 2、Raise Collaboration Exception 3、Raise Collaboration Exception 4、および Raise Collaboration Exception 5 機能ブロックにより、例外メッセージ・テキストに 5 つまでのメッセージ・パラメーター値を指定できます。Raise Collaboration Exception with Parameters 機能ブロックにより、メッセージ・パラメーター値の配列を指定できます。

### 関連情報

この機能ブロックは、BaseCollaboration.sendEmail() メソッドを基にしています。詳細については、 378 ページの『sendEmail()』を参照してください。

## Raise Collaboration Exception 1

1 つ上の実行レベルに例外を生成するためにコラボレーション例外を準備します。この機能ブロックは、指定された例外タイプと、コラボレーションのメッセージ・ファイルから取得した例外メッセージを持つ新規の例外オブジェクトを作成します。メッセージ・ファイル内のこのメッセージ番号で、メッセージを識別します。この機能ブロックは、メッセージ・テキストの 1 つのメッセージ・パラメーター値を渡すことができます。

**注:** この機能ブロックは、General¥APIs¥Collaboration Template¥Exception フォルダーにあります。

## 入力

**Collaboration** 現行コラボレーション・オブジェクト。

**Exception type**
例外タイプを指定する String。

**messageNum** 例外オブジェクトに関連付けられているメッセージの番号を指定する整数。

**Parameter 1** 1 つのメッセージ・パラメーターの値を指定する String。

## 関連情報

Raise Collaboration Exception 機能ブロックの使用法の詳細については、 295 ページの『Raise Collaboration Exception』を参照してください。

この機能ブロックは、BaseCollaboration.sendEmail() メソッドを基にしています。詳細については、 378 ページの『sendEmail()』を参照してください。

---

# Raise Collaboration Exception 2

1 つ上の実行レベルに例外を生成するためにコラボレーション例外を準備します。この機能ブロックは、指定された例外タイプと、コラボレーションのメッセージ・ファイルから取得した例外メッセージを持つ新規の例外オブジェクトを作成します。メッセージ・ファイル内のこのメッセージ番号で、メッセージを識別します。この機能ブロックは、メッセージ・テキストの 2 つのメッセージ・パラメーター値を渡すことができます。

**注:** この機能ブロックは、General¥APIs¥Collaboration Template¥Exception フォルダーにあります。

## 入力

**Collaboration** 現行コラボレーション・オブジェクト。

**Exception type**
例外タイプを指定する String。

**messageNum** 例外オブジェクトに関連付けられているメッセージの番号を指定する整数。

**Parameter 1** 1 つのメッセージ・パラメーターの値を指定する String。

**Parameter 2** 1 つのメッセージ・パラメーターの値を指定する String。

## 関連情報

Raise Collaboration Exception 機能ブロックの使用法の詳細については、 295 ページの『Raise Collaboration Exception』を参照してください。

この機能ブロックは、BaseCollaboration.sendEmail() メソッドを基にしています。詳細については、 378 ページの『sendEmail()』を参照してください。

## Raise Collaboration Exception 3

1 つ上の実行レベルに例外を生成するためにコラボレーション例外を準備します。この機能ブロックは、指定された例外タイプと、コラボレーションのメッセージ・ファイルから取得した例外メッセージを持つ新規の例外オブジェクトを作成します。メッセージ・ファイル内のこのメッセージ番号で、メッセージを識別します。この機能ブロックは、メッセージ・テキストの 3 つのメッセージ・パラメーター値を渡すことができます。

**注:** この機能ブロックは、General¥APIs¥Collaboration Template¥Exception フォルダーにあります。

### 入力

**Collaboration** 現行コラボレーション・オブジェクト。

**Exception type**
例外タイプを指定する String。

**messageNum** 例外オブジェクトに関連付けられているメッセージの番号を指定する整数。

**Parameter 1** 1 つのメッセージ・パラメーターの値を指定する String。

**Parameter 2** 1 つのメッセージ・パラメーターの値を指定する String。

**Parameter 3** 1 つのメッセージ・パラメーターの値を指定する String。

### 関連情報

Raise Collaboration Exception 機能ブロックの使用法の詳細については、295 ページの『Raise Collaboration Exception』を参照してください。

この機能ブロックは、BaseCollaboration.sendEmail() メソッドを基にしています。詳細については、378 ページの『sendEmail()』を参照してください。

## Raise Collaboration Exception 4

1 つ上の実行レベルに例外を生成するためにコラボレーション例外を準備します。この機能ブロックは、指定された例外タイプと、コラボレーションのメッセージ・ファイルから取得した例外メッセージを持つ新規の例外オブジェクトを作成します。メッセージ・ファイル内のこのメッセージ番号で、メッセージを識別します。この機能ブロックは、メッセージ・テキストの 4 つのメッセージ・パラメーター値を渡すことができます。

**注:** この機能ブロックは、General¥APIs¥Collaboration Template¥Exception フォルダーにあります。

### 入力

**Collaboration** 現行コラボレーション・オブジェクト。

**Exception type**
例外タイプを指定する String。

**messageNum** 例外オブジェクトに関連付けられているメッセージの番号を指定する整数。

**Parameter 1** 1 つのメッセージ・パラメーターの値を指定する String。

**Parameter 2** 1 つのメッセージ・パラメーターの値を指定する String。

**Parameter 3** 1 つのメッセージ・パラメーターの値を指定する String。

**Parameter 4** 1 つのメッセージ・パラメーターの値を指定する String。

## 関連情報

Raise Collaboration Exception 機能ブロックの使用法の詳細については、 295 ページの『Raise Collaboration Exception』を参照してください。

この機能ブロックは、BaseCollaboration.sendEmail() メソッドを基にしています。詳細については、 378 ページの『sendEmail()』を参照してください。

---

# Raise Collaboration Exception 5

1 つ上の実行レベルに例外を生成するためにコラボレーション例外を準備します。この機能ブロックは、指定された例外タイプと、コラボレーションのメッセージ・ファイルから取得した例外メッセージを持つ新規の例外オブジェクトを作成します。メッセージ・ファイル内のこのメッセージ番号で、メッセージを識別します。この機能ブロックは、メッセージ・テキストの 5 つのメッセージ・パラメーター値を渡すことができます。

**注:** この機能ブロックは、General¥APIs¥Collaboration Template¥Exception フォルダーにあります。

## 入力

**Collaboration** 現行コラボレーション・オブジェクト。

**Exception type**
例外タイプを指定する String。

**messageNum** 例外オブジェクトに関連付けられているメッセージの番号を指定する整数。

**Parameter 1** 1 つのメッセージ・パラメーターの値を指定する String。

**Parameter 2** 1 つのメッセージ・パラメーターの値を指定する String。

**Parameter 3** 1 つのメッセージ・パラメーターの値を指定する String。

**Parameter 4** 1 つのメッセージ・パラメーターの値を指定する String。

**Parameter 5** 1 つのメッセージ・パラメーターの値を指定する String。

## 関連情報

Raise Collaboration Exception 機能ブロックの使用法の詳細については、 295 ページの『Raise Collaboration Exception』を参照してください。

この機能ブロックは、BaseCollaboration.sendEmail() メソッドを基にしています。詳細については、 378 ページの『sendEmail()』を参照してください。

## Raise Collaboration Exception with Parameter

1 つ上の実行レベルに例外を生成するためにコラボレーション例外を準備します。この機能ブロックは、メッセージ・ファイルの指定されたメッセージを含む新規例外オブジェクトを作成するための別の方法として使用できます。パラメーター値は、すべて `Objects` の配列に配置されます。

**注:** この機能ブロックは、General¥APIs¥Collaboration Template¥Exception フォルダーにあります。

### 入力

**Collaboration** 現行コラボレーション・オブジェクト。

**Exception type**
例外タイプを指定する String。

**messageNum** 例外オブジェクトに関連付けられているメッセージの番号を指定する整数。

**Parameters** メッセージ・パラメーター値の配列。各パラメーターは順次的に解決され、メッセージ・テキスト内のパラメーターになります。メッセージ内 (コラボレーション・メッセージ・ファイル内) では、メッセージ・パラメーターは中括弧で囲まれた整数 (例えば、{1}) で示されます。

### 注

この機能ブロックは、次のような例外オブジェクトを生成する場合に使用できます。

- コラボレーションがすでに処理されている例外オブジェクト。例えば、1 つのシナリオが例外を取得し、それを変数に割り当ててから他の作業に移ります。
- メッセージ・パラメーターが 5 つを超える場合。他の Raise Collaboration Exception 機能ブロックで処理できるパラメーターは 5 つまでですが、このパラメーター配列には任意の数のパラメーターを格納できます。

### 関連情報

Raise Collaboration Exception 機能ブロックの使用法の詳細については、 295 ページの『Raise Collaboration Exception』を参照してください。

この機能ブロックは、BaseCollaboration.raiseException() メソッドを基にしています。詳細については、 374 ページの『raiseException()』を参照してください。

## Send Email

E メール・メッセージを非同期に送信します。

### 入力

**Collaboration** 現行コラボレーション・オブジェクト。

**Message** E メール・メッセージのテキストを含む String。

**Subject** E メール・メッセージの件名。

**Recipients** メッセージ宛先の E メール・アドレスを含む Vector。この Vector には String オブジェクトが含まれています。

### 注

Send Email 機能ブロックは、以下の場合に、入力の Recipients ベクトルで指定された宛先に、E メール・メッセージを送信できます。

- E メール・アドレスが「Collaboration Object Properties」ダイアログの**「E メール通知アドレス」**フィールドで指定されている場合。
- Email コラボレーションおよび Email アダプターが実行されている場合。(InterChange Server Express が始動するときにユーザーによる入力を必要としない場合、Email コラボレーションは自動的にインスタンス化および構成されます。)Email アダプターが実行されていない場合は、Send Email によってコラボレーションの実行が停止されることはありません。

**注:** Log Error 機能ブロックは、E メールの宛先にエラー・メッセージを自動送信します (Email コラボレーションと Email アダプターが実行していることを前提としています)。Send Email 機能ブロックにより、E メール・メッセージを明示的に送信できます。

### 関連情報

この機能ブロックは、BaseCollaboration.sendEmail() メソッドを基にしています。詳細については、378 ページの『sendEmail()』を参照してください。

## ServiceCallException

サービス呼び出しの障害によって発生するエラーを表す定数 (例えば、アダプターまたはアプリケーションが使用不可の場合)。

**注:** この機能ブロックは、General¥APIs¥Collaboration Template¥Exception¥Constants フォルダーにあります。

### 出力

値が「ServiceCallException」の String を戻します。

## SystemException

InterChange Server Express システムの内部エラーを表す定数。

**注:** この機能ブロックは、General¥APIs¥Collaboration Template¥Exception¥Constants フォルダーにあります。

### 出力

値が「SystemException」の String を戻します。

## TransactionException

トランザクション・コラボレーションのトランザクションの振る舞いに関連するエラーを表す定数 (例えば、ロールバックの失敗、またはコラボレーションで差し戻しを適用できなかったなど)。

**注:** この機能ブロックは、General¥APIs¥Collaboration Template¥Exception¥Constants フォルダーにあります。

### 出力

値が「TransactionException」の String を戻します。

# 第 14 章 データベース接続の機能ブロック

General¥APIs¥Database Connection フォルダーにある機能ブロックは、データベース接続の管理およびデータベースでの SQL 照会の実行に関する基本機能を提供します。以下のセクションでは、各機能ブロックの詳細を説明します。

*表 65. データベース接続の機能ブロックの要約*



## Begin Transaction

現行接続の明示的なトランザクションを開始します。

### 入力

**Database connection**

データベース接続を表す CwDBConnection オブジェクト。

### 例外

Begin Transaction 機能ブロックは、データベース・エラーが発生した場合に `CwDBConnectionException` 例外をスローできます。

### 注

Begin Transaction 機能ブロックは、現行接続における新規の明示的なトランザクションの開始をマークします。Begin Transaction、Commit、および Roll Back 機能ブロックは、明示的なトランザクションのトランザクション境界を共同で管理しま

す。このトランザクションには、SQL 照会 (SQL ステートメント INSERT、DELETE、または UPDATE を含む) と、これらの SQL ステートメントの 1 つを含むストアード・プロシージャーとが含まれます。

**要確認**

接続において明示的なトランザクション・ブラケットを使用している場合は、Begin Transaction のみを使用してください。接続で暗黙的なトランザクション・ブラケットを使用しているときに Begin Transaction を使用すると、`CwDBTransactionException` 例外が発生します。

明示的なトランザクションを開始する前に、Get Database Connection 機能ブロックによって `CwDBConnection` オブジェクトを作成する必要があります。この接続では、明示的なトランザクション・ブラケットを必ず使用してください。

### 関連情報

この機能ブロックは、CwDBConnection.beginTransaction() メソッドを基にしています。詳細については、 415 ページの『beginTransaction()』を参照してください。

## Commit

現行接続と関連付けられているアクティブなトランザクションをコミットします。

### 入力

**Database connection**

データベース接続を表す CwDBConnection オブジェクト。

### 例外

Commit 機能ブロックは、データベース・エラーが発生した場合に `CwDBConnectionException` 例外をスローできます。

### 注

Commit 機能ブロックは、現行接続に関連付けられているデータベースへの変更すべてをコミットすることにより、アクティブ・トランザクションを終了させます。Begin Transaction、Commit、および Roll Back 機能ブロックは、明示的なトランザクションのトランザクション境界を共同で管理します。このトランザクションには、SQL 照会 (SQL ステートメント INSERT、DELETE、または UPDATE を含む) と、これらの SQL ステートメントの 1 つを含むストアード・プロシージャーとが含まれます。

**要確認**

接続において明示的なトランザクション・ブラケットを使用している場合は、Commit のみを使用してください。接続で暗黙的なトランザクション・ブラケットを使用しているときに Commit を使用すると、CwDBTransactionException 例外が発生します。接続が解放される前に Commit (または Roll Back) を使用して明示的なトランザクションを終了しない場合は、InterChange Server がコラボレーションの成功に基づいてトランザクションを暗示的に終了させます。コラボレーションが成功の場合、ICS は、このデータベース・トランザクションをコミットします。コラボレーションが成功ではない 場合、ICS は、このデータベース・トランザクションを暗黙的にロールバックします。コラボレーションが成功かどうかにかかわらず、ICS は警告を記録します。

明示的なトランザクションを開始する前に、Get Database Connection 機能ブロックによって CwDBConnection オブジェクトを作成する必要があります。この接続では、明示的なトランザクション・ブラケットを必ず使用してください。

### 関連情報

この機能ブロックは、CwDBConnection.commit() メソッドを基にしています。詳細については、416 ページの『commit()』を参照してください。

## Execute Prepared SQL

構文を指定して、準備済み SQL 照会を実行します。

### 入力

**Database connection**
データベース接続を表す CwDBConnection オブジェクト。

**Query** データベースで実行する SQL 照会を表す String。

### 例外

Execute Prepared SQL 機能ブロックは、データベース・エラーが発生した場合に CwDBSQLException 例外をスローできます。

### 注

実行できる SQL ステートメントを以下に示します (必要なデータベース許可がある場合)。

- 1 つ以上のデータベース表からデータを要求する SELECT ステートメント

  Has More Rows 機能ブロックと Next Row 機能ブロックを使用して検索されたデータにアクセスします。
- データベース内のデータを変更する SQL ステートメント
  - INSERT
  - DELETE

– UPDATE

Execute Prepared SQL 機能ブロックは、準備済み SQL ステートメントとして、指定された*照会* ストリングを現行接続に関連付けられているデータベースに送信します。この照会は、最初の実行時にストリングとしてデータベースに送信されます。データベースは、受信したストリングを実行可能形式 (準備済みステートメントと呼ばれる) にコンパイルし、SQL ステートメントを実行して、準備済みステートメントを機能ブロックに戻します。次にこの機能ブロックは、準備済みステートメントをメモリー内に保管します。複数回実行する必要がある SQL ステートメントには、Execute Prepared SQL を使用してください。

**注:** Execute SQL 機能ブロックは準備済みステートメントを保管しない ので、1 回だけ実行する必要のある照会の場合に便利です。

**要確認**

Execute Prepared SQL を使用して照会を実行する前に、Get Database Connection 機能ブロックを使用して `CwDBConnection` オブジェクトを生成することにより、目的のデータベースへの接続を確立する必要があります。

接続で明示的なトランザクション・ブラケットを使用している場合は、Begin Transaction によって明示的に各トランザクションを開始し、Commit または Roll Back のいずれかによって終了する必要があります。

- 準備済みストアード・プロシージャーを実行する CALL ステートメント (ただしこのストアード・プロシージャーでは、OUT パラメーターを使用できません)

  OUT パラメーターを使用してストアード・プロシージャーを実行するには、Execute Stored Procedure 機能ブロックを使用します。詳細については、234 ページの『executeStoredProcedure() でのストアード・プロシージャーの呼び出し』を参照してください。

## 関連情報

この機能ブロックは、CwDBConnection.executePreparedSQL() メソッドを基にしています。詳細については、418 ページの『executePreparedSQL()』を参照してください。

# Execute Prepared SQL with Parameter

構文およびパラメーターを指定して準備済み SQL 照会を実行します。

## 入力

**Database connection**
データベース接続を表す CwDBConnection オブジェクト。

**Query** データベースで実行する SQL 照会を表す String。

**Parameters** SQL 照会でパラメーターに渡される引き数の Vector オブジェクト。

### 例外

Execute Prepared SQL with Parameter 機能ブロックは、データベース・エラーが発生した場合に CwDBSQLException 例外をスローできます。

### 注

Execute Prepared SQL 機能ブロックについては、この章の『注記』を参照してください。

### 関連情報

この機能ブロックは、CwDBConnection.executePreparedSQL() メソッドを基にしています。詳細については、 418 ページの『executePreparedSQL()』を参照してください。

## Execute SQL

構文を指定して静的 SQL 照会を実行します。

### 入力

**Database connection**
データベース接続を表す CwDBConnection オブジェクト。

**Query** データベースで実行する SQL 照会を表す String。

### 例外

Execute SQL 機能ブロックは、データベース・エラーが発生した場合に CwDBSQLException 例外をスローできます。

### 注

Execute SQL 機能ブロックは、静的 SQL ステートメントとして、指定された*照会*ストリングを現行接続に関連付けられているデータベースに送信します。この照会はストリングとしてデータベースに送信されます。データベースは、このストリングを実行可能形式にコンパイルし、SQL ステートメントを実行します。この場合に、実行可能形式は保管されません。1 回だけ実行する必要のある SQL ステートメントには、Execute SQL を使用してください。

**注:** Execute Prepared SQL 機能ブロックと Execute Prepared SQL with Parameter 機能ブロックは、実行可能形式 (準備済みステートメントと呼ばれる) を保管するので、何度も実行する必要がある照会の場合に便利です。

**要確認**

Execute SQL を使用して照会を実行する前に、Get Database Connection 機能ブロックを使用して CwDBConnection オブジェクトを生成することにより、目的のデータベースへの接続を確立する必要があります。

実行できる SQL ステートメントを以下に示します (必要なデータベース許可がある場合)。

- 1 つ以上のデータベース表からデータを要求する SELECT ステートメント

  Has More Rows 機能ブロックと Next Row 機能ブロックを使用して検索されたデータにアクセスします。
- データベース内のデータを変更する SQL ステートメント
  - INSERT
  - DELETE
  - UPDATE

  接続で明示的なトランザクション・ブラケットを使用している場合は、Begin Transaction 機能ブロックによって明示的に各トランザクションを開始し、Commit または Roll Back のいずれかの機能ブロックを使用して終了する必要があります。
- ストアード・プロシージャーを静的に実行する CALL ステートメント (ただしこのストアード・プロシージャーでは、OUT パラメーターを使用できません)

  OUT パラメーターを使用してストアード・プロシージャーを実行するには、Execute Stored Procedure 機能ブロックを使用します。詳細については、234 ページの『executeStoredProcedure() でのストアード・プロシージャーの呼び出し』を参照してください。

### 関連情報

この機能ブロックは、CwDBConnection.executeSQL() メソッドを基にしています。詳細については、419 ページの『executeSQL()』を参照してください。

## Execute SQL with Parameter

構文およびパラメーターを指定して静的 SQL 照会を実行します。

### 入力

**Database connection**
: データベース接続を表す CwDBConnection オブジェクト。

**Query**
: データベースで実行する SQL 照会を表す String。

**Parameters**
: SQL 照会でパラメーターに渡される引き数の Vector オブジェクト。

### 例外

Execute SQL with Parameter 機能ブロックは、データベース・エラーが発生した場合に `CwDBSQLException` 例外をスローできます。

### 注

Execute SQL 機能ブロックについては、この章の『注記』を参照してください。

## 関連情報

この機能ブロックは、CwDBConnection.executeSQL() メソッドを基にしています。詳細については、 419 ページの『executeSQL()』を参照してください。

# Execute Stored Procedure

名前とパラメーター配列を指定して SQL ストアード・プロシージャーを実行します。

## 入力

**Database connection**
データベース接続を表す CwDBConnection オブジェクト。

**Stored procedure**
データベースで実行するストアード・プロシージャーを表す String。

**Parameters** SQL 照会でパラメーターに渡される引き数の Vector オブジェクト。

## 例外

Execute Stored Procedure 機能ブロックは、データベース・エラーが発生した場合に `CwDBSQLException` 例外をスローできます。

## 注

Execute Stored Procedure 機能ブロックは、指定された *`storedProcedure`* への呼び出しを、現行接続に関連付けられているデータベースに送信します。この機能ブロックは、ストアード・プロシージャー呼び出しを準備済み SQL ステートメントとして送信します。つまり、このストアード・プロシージャー呼び出しは、最初に実行されるときにストリングとしてデータベースに送信されます。データベースでは、このストリングを実行可能形式 (準備済みステートメントと呼ばれる) にコンパイルし、SQL ステートメントを実行して、この準備済みステートメントを Execute Stored Procedure に戻します。次にこの機能ブロックは、準備済みステートメントをメモリー内に保管します。

**要確認**
この機能ブロックによってストアード・プロシージャーを実行する前に、Get Database Connection 機能ブロックによって `CwDBConnection` オブジェクトを作成する必要があります。

ストアード・プロシージャーによって戻されるデータを処理するには、Has More Rows 機能ブロックと Next Row 機能ブロックを使用します。

また、ストアード・プロシージャーに OUT パラメーターが含まれていない かぎり、Execute SQL、Execute SQL with Parameter、Execute Prepared SQL、または Execute Prepared SQL with Parameter の各機能ブロックを使用してそのストアー

ド・プロシージャーを実行できます。ストアード・プロシージャーで OUT パラメーターを使用する場合は、Execute Stored Procedure 機能ブロックを使用してプロシージャーを実行する必要があります。

Execute SQL または Execute Prepared SQL 機能ブロックのグループとは異なり、Execute Stored Procedure 機能ブロックでは、全 SQL ステートメントを渡してストアード・プロシージャーを実行する必要はありません。Execute Stored Procedure の場合は、ストアード・プロシージャーの名前と、`CwDBStoredProcedureParam` オブジェクトの `Vector` パラメーター配列のみを渡す必要があります。Execute Stored Procedure 機能ブロックは、*`storedProcParameters`* 配列からのパラメーター数を判別し、ストアード・プロシージャーの呼び出しステートメントを作成することができます。

### 関連情報

この機能ブロックは、CwDBConnection.executeStoredProcedure() メソッドを基にしています。詳細については、 421 ページの『executeStoredProcedure()』を参照してください。

## Get Database Connection

データベースへの接続を確立します。

### 入力

**Connection pool name**
有効な接続プールの名前を指定する String。

### 出力

CwDBConnection オブジェクトを戻します。

### 例外

Get Database Connection 機能ブロックは、データベース接続の確立動作中にエラーが発生した場合に、`CwDBConnectionFactoryException` 例外をスローできます。

### 注

Get Database Connection 機能ブロックは、Connection Pool Name 入力によって指定される接続プールから接続を確立します。この接続によって、接続に関連付けられたデータベースへの照会および更新の実行が可能になります。特定の接続プールのすべての接続は、同じデータベースに関連付けられています。この機能ブロックは、照会の実行とトランザクションの管理を可能にする `CwDBConnection` オブジェクトを戻します。

デフォルトでは、暗黙的なトランザクション・ブラケットをすべての接続でトランザクション・プログラミング・モデルとして使用します。*特定の接続に対する* トランザクション・プログラミング・モデルを指定するには、Get Database Connection with Transaction 機能ブロックを使用します。

コラボレーション・オブジェクトの実行が完了すると、接続が解除されます。この接続は、Release 機能ブロックによって明示的に閉じることができます。接続が解放されたかどうかは、Is Active 機能ブロックによって判断できます。詳細については、242 ページの『接続の解放』を参照してください。

### 関連情報

この機能ブロックは、BaseCollaboration.getDBConnection() メソッドを基にしています。詳細については、366 ページの『getDBConnection()』を参照してください。

## Get Database Connection with Transaction

特定のトランザクション・プログラミング・モデルを使用するデータベースへの接続を確立します。

### 入力

**Connection pool name**

有効な接続プールの名前を指定する String。

**Implicit transaction**

トランザクション・プログラミング・モデルを、接続に関連付けられたデータベースに対して使用することを指示する boolean 値です。次の値が有効です。

`true` データベースで使用される暗黙的なトランザクション・ブラケット

`false` データベースで使用される明示的なトランザクション・ブラケット

### 出力

CwDBConnection オブジェクトを戻します。

### 例外

Get Database Connection With Transaction 機能ブロックは、データベース接続の確立動作中にエラーが発生した場合に、`CwDBConnectionFactoryException` 例外をスローできます。

### 注

Get Database Connection 機能ブロックは、Connection Pool Name 入力によって指定される接続プールから接続を確立します。この接続によって、接続に関連付けられたデータベースへの照会および更新の実行が可能になります。特定の接続プールにおけるすべての接続は、同じデータベースに関連付けられています。この機能ブロックは、照会の実行とトランザクションの管理を可能にする `CwDBConnection` オブジェクトを戻します。

コラボレーション・オブジェクトの実行が完了すると、接続が解除されます。この接続は、Release 機能ブロックによって明示的に閉じることができます。接続が解放

されたかどうかは、Is Active 機能ブロックによって判断できます。詳細については、242 ページの『接続の解放』を参照してください。

### 関連情報

この機能ブロックは、BaseCollaboration.getDBConnection() メソッドを基にしています。詳細については、 366 ページの『getDBConnection()』を参照してください。

## Get Next Row

照会結果から、次の行を取得します。

### 入力

**Database connection**
データベース接続を表す CwDBConnection オブジェクト。

### 出力

現行接続に関連付けられている照会結果から、データの 1 行を戻します。このメソッドは、データを戻す照会から結果を検索するときに使用してください。これらの照会には、SELECT ステートメントとストアード・プロシージャーが含まれています。

### 例外

Get Next Row 機能ブロックは、データベース・エラーが発生した場合に `CwDBSQLException` 例外をスローできます。

### 注

Get Next Row 機能ブロックは、現行接続に関連付けられている照会結果から、データの 1 行を戻します。この機能ブロックは、データを戻す照会からの結果を検索するときに使用してください。これらの照会には、SELECT ステートメントとストアード・プロシージャーが含まれています。一度に接続に関連付けることのできる照会は 1 つのみです。このため、Get Next Row がデータの最後の行を戻す前に別の照会を実行すると、最初の照会結果が失われます。

### 関連情報

この機能ブロックは、CwDBConnection.nextRow() メソッドを基にしています。詳細については、 425 ページの『nextRow()』を参照してください。

## Get Update Count

データベースへの最後の書き込み操作によって影響された行の数を判別します。

### 入力

**Database connection**
データベース接続を表す CwDBConnection オブジェクト。

### 出力

最後の書き込み操作によって影響された行の数を示す整数を戻します。

### 例外

Get Update Count 機能ブロックは、データベース・エラーが発生した場合に `CwDBConnectionException` 例外をスローできます。

### 注

Get Update Count 機能ブロックは、現行接続に関連付けられているデータベースでの最新の更新操作によって変更された行数を示します。この機能ブロックは、UPDATE または INSERT ステートメントをデータベースに送信した後で、その SQL ステートメントによって変更された行数を判別する必要がある場合に役立ちます。

**要確認**

このメソッドを使用する前に、Get Database Connection 機能ブロックによって `CwDBConnection` オブジェクトを作成し、以下の機能ブロックの 1 つによってデータベースを更新する照会を送信する必要があります。

- Execute SQL
- Execute SQL with Parameter
- Execute Prepared SQL
- Execute Prepared SQL with Parameter

### 関連情報

この機能ブロックは、CwDBConnection.getUpdateCount() メソッドを基にしています。詳細については、422 ページの『getUpdateCount()』を参照してください。

## Has More Rows

現在の照会結果に、処理する必要のある行がさらに存在するかどうかを判別します。

### 入力

**Database connection**

データベース接続を表す CwDBConnection オブジェクト。

### 出力

処理すべき行が照会結果内にまだ存在する場合は `true` を戻し、存在しない場合は `false` を戻します。

### 例外

Has More Rows 機能ブロックは、データベース・エラーが発生した場合に CwDBSQLException 例外をスローできます。

### 注

Has More Rows 機能ブロックは、現行接続に関連付けられている照会結果に、処理対象の行がさらに存在するかどうかを判別します。この機能ブロックは、データを戻す照会からの結果を検索するときに使用してください。これらの照会には、SELECT ステートメントとストアード・プロシージャーが含まれています。一度に接続に関連付けることのできる照会は 1 つのみです。このため、Has More Rows によって false が戻される前に別の照会を実行すると、最初の照会によるデータが失われます。

### 関連情報

この機能ブロックは、CwDBConnection.hasMoreRows() メソッドを基にしています。詳細については、423 ページの『hasMoreRows()』を参照してください。

## In Transaction

トランザクションが、現行データベース接続で進行中かどうかを判別します。

### 入力

**Database connection**
データベース接続を表す CwDBConnection オブジェクト。

### 出力

進行中のトランザクションがある場合は true を戻し、ない場合は false を戻します。

### 例外

In Transaction 機能ブロックは、データベース・エラーが発生した場合に CwDBConnectionException 例外をスローできます。

### 注

In Transaction 機能ブロックは、現行接続にアクティブ・トランザクション (開始されたが終了していないトランザクション) があるかどうかを示す boolean 値を戻します。

**要確認**
トランザクションを開始する前に、Get Database Connection 機能ブロックを使用して CwDBConnection オブジェクトを作成する必要があります。

### 関連情報

この機能ブロックは、CwDBConnection.inTransaction() メソッドを基にしています。詳細については、 424 ページの『inTransaction()』を参照してください。

## Is Active

現行接続がアクティブかどうかを判別します。

### 入力

**Database connection**
データベース接続を表す CwDBConnection オブジェクト。

### 出力

現行接続がアクティブの場合は `true` を戻し、アクティブでない場合は `false` を戻します。

### 関連情報

この機能ブロックは、CwDBConnection.isActive() メソッドを基にしています。詳細については、 424 ページの『isActive()』を参照してください。

## Release

現行接続の使用を解放して、それを接続プールに戻します。

### 入力

**Database connection**
データベース接続を表す CwDBConnection オブジェクト。

### 例外

Release 機能ブロックは、`CwDBConnectionException` 例外をスローできます。

### 注

Release 機能ブロックは、コラボレーション・オブジェクトによる現行接続の使用を明示的に解放します。解放された接続はその接続プールに戻されます。関連付けられているデータベースへの接続を必要とする他のコンポーネント (マップまたはコラボレーション) は、この接続を使用できるようになります。接続を明示的に解放しない場合は、現行のコラボレーションの実行の最後にコラボレーション・オブジェクトによって暗黙的に接続が解放されます。このため、静的変数の中に接続を保管して再使用することはできません。

**要確認**

トランザクションが現在アクティブになっている場合は、Release を使用しないでください。暗黙的なトランザクション・ブラケットの場合、ICS は、コラボレーションが成功または失敗のいずれかを確認するまでデータベース・トランザクションを終了しません。したがって、暗黙的なトランザクション・ブラケットを使用する接続でこの機能ブロックを使用すると、CwDBTransactionException 例外が発生します。この例外を明示的に処理しないと、アクティブなトランザクションも自動的にロールバックされます。トランザクションがアクティブであるかどうかは、In Transaction 機能ブロックを使用して判別できます。

## 関連情報

この機能ブロックは、CwDBConnection.release() メソッドを基にしています。詳細については、425 ページの『release()』を参照してください。

# Roll Back

現行接続と関連付けられているアクティブなトランザクションをロールバックします。

## 入力

**Database connection**

データベース接続を表す CwDBConnection オブジェクト。

## 例外

Roll Back 機能ブロックは、CwDBTransactionException 例外をスローできます。

## 注

Roll Back 機能ブロックは、現行接続に関連づけられているデータベースに加えられた変更すべてをロールバックすることにより、アクティブ・トランザクションを終了させます。Begin Transaction、Commit、および Roll Back 機能ブロックは、明示的なトランザクションのトランザクション境界を共同で管理します。このトランザクションには、SQL 照会 (SQL ステートメント INSERT、DELETE、または UPDATE を含む) と、これらの SQL ステートメントの 1 つを含むストアード・プロシージャーとが含まれます。ロールバックが失敗すると、Roll Back は CwDBTransactionException 例外をスローして、エラーをログに記録します。

**要確認**

接続において明示的なトランザクション・ブラケットを使用している場合は、Roll Back のみを使用してください。接続で暗黙的なトランザクション・ブラケットを使用しているときに Roll Back を使用すると、CwDBTransactionException 例外が発生します。接続が解放される前に Roll Back (または Commit 機能ブロック) を使用して明示的なトランザクションを終了しない場合は、InterChange Server Express がコラボレーションの成功に基づいてトランザクションを暗示的に終了させます。コラボレーションが成功の場合、InterChange Server Express は、このデータベース・トランザクションをコミットします。コラボレーションが成功では ない 場合、InterChange Server Express は、このデータベース・トランザクションを暗黙的にロールバックします。コラボレーションが成功かどうかにかかわらず、InterChange Server Express は警告を記録します。

## 関連情報

この機能ブロックは、CwDBConnection.rollBack() メソッドを基にしています。詳細については、 426 ページの『rollBack()』を参照してください。

# 第 15 章 データベース・ストアード・プロシージャーの機能ブロック

データベース・ストアード・プロシージャーの機能ブロックには、ストアード・プロシージャーのパラメーターで作業するための基本的な機能があります。これらの機能ブロックは、¥General¥APIs¥DB Stored Procedure Param フォルダーに配置されています。

以下のセクションでは、各機能ブロックの詳細を説明します。

*表 66. データベース・ストアード・プロシージャーの機能ブロックの要約*

| 機能ブロック | ページ |
|---|---|
| Get Param Type | 319 |
| Get Param Value | 320 |
| New DB Stored Procedure Param | 321 |

## Get Param Type

現行ストアード・プロシージャー・パラメーターのイン/アウト・タイプを検索します。

### 入力

**CwDBStoredProcedureParam**
イン/アウトを検索する対象のストアード・プロシージャー・パラメーター (CwDBStoredProcedureParam オブジェクト)。

### 出力

関連付けられているストアード・プロシージャー・パラメーターのイン/アウト・タイプを整数定数として戻します。

### 注

Get Param Type 機能ブロックは、現行ストアード・プロシージャー・パラメーターのイン/アウト・パラメーター・タイプを戻します。イン/アウト・パラメーター・タイプは、ストアード・プロシージャーがパラメーターをどのように使用するかを示します。CwDBStoredProcedureParam クラスは、 320 ページの表 67 に示すとおり、それぞれのイン/アウト・タイプを定数として表します。

*表 67. パラメーターのイン/アウト・タイプ (Get Param Type 機能ブロック)*

| パラメーターのイン/アウト・タイプ | 説明 | イン/アウト・タイプ定数 |
|---|---|---|
| IN パラメーター | IN パラメーターは*入力専用* です。つまり、ストアード・プロシージャーはその値を入力として受け入れますが、値を戻すためにそのパラメーターを使用しません。 | PARAM_IN |
| OUT パラメーター | OUT パラメーターは*出力専用* です。つまり、ストアード・プロシージャーはその値を入力として読み取るのではなく、値を戻すためにそのパラメーターを使用します。 | PARAM_OUT |
| INOUT パラメーター | INOUT パラメーターは*入力および出力* です。つまり、ストアード・プロシージャーはその値を入力として受け入れ、また値を戻すときにもそのパラメーターを使用します。 | PARAM_INOUT |

### 関連情報

この機能ブロックは、CwDBStoredProcedureParam.getParamType() メソッドを基にしています。詳細については、431 ページの『getParamType()』を参照してください。

## Get Param Value

現行ストアード・プロシージャー・パラメーターの値を検索します。

### 入力

**CwDBStoredProcedureParam**
値を検索する対象のストアード・プロシージャー・パラメーター (CwDBStoredProcedureParam オブジェクト)。

### 出力

関連付けられているストアード・プロシージャー・パラメーターの値を Java Object として戻します。

### 注

Get Param Value 機能ブロックは、パラメーター値を Java Object (Integer、Double、または String など) として戻します。OUT パラメーターに戻された値が JDBC NULL の場合、Get Param Value は null 定数を戻します。

### 関連情報

この機能ブロックは、CwDBStoredProcedureParam.getValue() メソッドを基にしています。詳細については、432 ページの『getValue()』を参照してください。

## New DB Stored Procedure Param

ストアード・プロシージャーのパラメーターの引き数情報を持つ CwDBStoredProcedureParam の新しいインスタンスを作成します。

### 入力

**Param Type** ストアード・プロシージャー・パラメーターに関連付けられているイン/アウト・パラメーター・タイプ。

**Param Value** ストアード・プロシージャーに送信される引き数値。この値は、以下の Java データ型のいずれかになります。

- String
- int
- Integer
- Long
- double
- Double
- float
- Float
- BigDecimal
- boolean
- java.sql.Date
- java.sql.Time
- java.sql.Timestamp
- java.sql.Blob
- java.sql.Clob
- byte[]
- Array
- Struct

### 出力

ストアード・プロシージャーの宣言内の 1 つの引き数の引き数情報を保持するための、新規 CwDBStoredProcedureParam オブジェクトを戻します。

### 注

New DB Stored Procedure Param 機能ブロックは、ストアード・プロシージャーの 1 つのパラメーターを記述する CwDBStoredProcedureParam インスタンスを作成します。この場合のパラメーター情報は以下のとおりです。

- パラメーターのイン/アウト・タイプ

  機能ブロックの最初の入力により、このイン/アウト・パラメーター・タイプが初期化されます。パラメーターの有効なイン/アウト・タイプのリストについては、320 ページの表 67 を参照してください。

- パラメーター値

機能ブロックの 2 番目の入力により、このパラメーター値が初期化されます。CwDBStoredProcedureParam クラスには、サポートされるパラメーター値のデータ型ごとにコンストラクターの 1 つの形式が用意されています。ストアード・プロシージャー・パラメーターの場合の Java データ型と JDBC データ型の間のマッピングのリストについては、237 ページの表 57 を参照してください。

Execute Stored Procedure 機能ブロック (General¥APIs¥Database Connection フォルダー) にストアード・プロシージャー・パラメーターの Java Vector を指定します。この機能ブロックは、ストアード・プロシージャー名とパラメーター・ベクトルからストアード・プロシージャー呼び出しを作成し、その呼び出しを現行接続に関連付けられているデータベースに送信します。

## 関連情報

この機能ブロックは、CwDBStoredProcedureParam コンストラクターを基にしています。詳細については、429 ページの『CwDBStoredProcedureParam()』を参照してください。

# 第 16 章 例外機能ブロック

General¥APIs¥Collaboration Exception フォルダーの機能ブロックは、例外を処理するための基本的な機能を提供します。以下のセクションでは、各機能ブロックの詳細を説明します。

*表 68. 例外機能ブロックの要約*

| 機能ブロック | ページ |
|---|---|
| Catch Collaboration Exception | 323 |
| Get Message | 323 |
| Get Message Number | 324 |
| Get Subtype | 324 |
| Get Type | 326 |
| To String | 327 |

## Catch Collaboration Exception

現在のアクティビティーまたはそのサブアクティビティーでスローされたコラボレーション例外をキャッチします。

### 入力

**Collaboration exception**
機能ブロックがキャッチしようとするコラボレーション例外 (CollaborationException オブジェクト) です。

### 注

サブアクティビティーを定義するには、編集キャンバスで Catch Collaboration Exception 機能ブロックをダブルクリックします。

## Get Message

例外オブジェクトからメッセージ・テキストを取得します。

### 入力

**Collaboration exception**
コラボレーション例外 (CollaborationException オブジェクト) です。

### 出力

例外オブジェクトからメッセージ・テキストを含む String を戻します。

### 注

Get Message 機能ブロックは、currentException システム変数から例外テキストを抽出する場合に役立ちます。この例外テキストは、いずれかの Raise Collaboration Exception 機能ブロックへの呼び出しの中で指定して、例外発生の理由が確実に 1 つ上の実行レベルにエスカレートされるようにすることができます。

**注:** To String 機能ブロックを使用して、現在の例外から例外タイプと例外テキストをフォーマット済みストリングとして取得できます。

### 関連情報

この機能ブロックは、collaborationException.getMessage() メソッドを基にしています。詳細については、437 ページの『getMessage()』を参照してください。

## Get Message Number

例外オブジェクトに関連付けられているメッセージのメッセージ番号を取得します。

### 入力

**Collaboration exception**
コラボレーション例外 (CollaborationException オブジェクト) です。

### 出力

現在の例外に関連付けられている整数 (int) のメッセージ番号。例外のメッセージがメッセージ・ファイルのメッセージではない場合には、この機能ブロックはゼロ (0) を戻します。

### 注

Get Message Number 機能ブロックは、例外のメッセージに関連付けられたメッセージ番号を取得する場合に役立ちます。このメッセージ番号は、Raise Collaboration Exception 機能ブロックの 1 つ、または Log Error 機能ブロックに渡すことができます。

### 関連情報

この機能ブロックは、collaborationException.getMsgNumber() メソッドを基にしています。詳細については、438 ページの『getMsgNumber()』を参照してください。

## Get Subtype

例外オブジェクトから例外サブタイプを取得します。

## 入力

**Collaboration exception**
コラボレーション例外 (CollaborationException オブジェクト) です。

## 出力

現在の例外の例外サブタイプを含む String を戻します。有効な例外サブタイプの詳細については、『注記』を参照してください。

## 注

例外タイプにより原因が十分に示されていない例外の場合、例外サブタイプから詳細な情報を把握できることがあります。一般に例外サブタイプを使用する例外タイプを以下に示します。

- JavaException

  コラボレーション・ランタイム環境は Java 例外をキャッチし、Java 例外の関連するタイプと一緒にコラボレーション例外の中にラップします。コラボレーションでは、コラボレーション例外で Get Subtype 機能ブロックを使用して、Java 例外の元のタイプ (取り込まれた Java 例外のクラス名) を取得できます。ただし、通常このようにする必要はありません。

- ServiceCallException

  ServiceCallException 例外タイプは、サービス呼び出しが失敗すると発生します。より堅固なコラボレーションを開発するために、サービス呼び出し失敗の原因を示す例外サブタイプを使用できます。有効な例外サブタイプには以下のものがあります。

| | |
|---|---|
| AppTimeOut | コネクターがアプリケーションとの通信を完了できなかったことを示します。 |
| AppLogOnFailure | コネクターがアプリケーションへログインできなかったことを示します。 |
| AppRetrieveByContentFailed | アプリケーションに対して実行された非キー値による Retrieve 操作で、条件に一致するものを取得できなかったことを示します。 |
| AppMultipleHits | アプリケーションが Retrieve 要求に応答して複数のエンティティーを取得したことを示します。 |
| AppBusObjDoesNotExist | アプリケーションに対して Retrieve 操作が実行されたが、ビジネス・オブジェクトを表すエンティティーがそのアプリケーション・データベース内に存在しないことを示します。 |
| AppRequestNotYetSent | 並列コネクター・エージェントの場合に、要求がエージェント・マスターのキューに入っているが、アプリケーションにディスパッチされていないため、要求を再送信できることを示します。詳細については、184 ページの『未送信サービス呼び出し要求』を参照してください。 |

| | |
|---|---|
| ServiceCallTransportException | トランスポートにエラーがあり、要求がアプリケーションに到達できたかどうかを正確に確認できないことを示します。詳細については、182 ページの『実行時のトランスポート関連例外の処理』を参照してください。 |
| AppUnknown | その他のサブタイプのいずれでもないタイプのエラーです。この例外サブタイプが存在する場合、サービス呼び出しで要求されたアプリケーション操作は、終了している場合と終了していない場合があります。 |

**要確認**

例外サブタイプ AppTimeOut、AppLogOnFailure、AppRetrieveByContent、AppMultipleHits、および AppUnknown は、失敗の原因を示すためにアダプターが戻す結果状況値に対応しています。古いアダプターでは、対応する結果状況値の一部がサポートされていないことがあります。Test Connector ツールを使用して、コラボレーションへバインドされているアダプターを厳密にテストし、アダプターが戻す実際の結果状況値を確認してください。

### 関連情報

この機能ブロックは、collaborationException.getSubType() メソッドを基にしています。詳細については、438 ページの『getSubType()』を参照してください。

## Get Type

例外オブジェクトからコラボレーション例外サブタイプを取得します。例外タイプは、例外の原因を示す String です。

### 入力

**Collaboration exception**

コラボレーション例外 (CollaborationException オブジェクト) です。

### 出力

現在の例外の例外タイプを含む String を戻します。この String 値を、以下の例外タイプ静的変数のいずれかと比較します。

| | |
|---|---|
| AnyException | 任意のタイプの例外。2 つの例外リンクがあり、1 つは例外の特定タイプをテストし、もう 1 つは AnyException をテストする場合、例外の特定タイプを検査するリンクが先に検査されます。現在の例外が特定の例外と一致しない場合、AnyException についてテストするリンクが次にチェックされます。 |
| AttributeException | 属性でアクセスする場合の問題です。例えば、String 属性に対してコラボレーションが getDouble() を呼び出した場合や、存在しない属性に対して getString() を呼び出した場合です。 |

| | |
|---|---|
| JavaException | コラボレーション・ロジックにおける Java コードの問題です。 |
| ObjectException | メソッドに渡されたビジネス・オブジェクトが無効でした。または、null オブジェクトにアクセスされました。 |
| OperationException | サービス呼び出しが適切に設定されませんでした。あるいは送信されませんでした。 |
| ServiceCallException | サービス呼び出しに失敗しました。例えば、コネクターやアプリケーションは使用できません。 |
| SystemException | InterChange Server Express の内部エラーです。 |
| TransactionException | トランザクション・コラボレーションの、トランザクションの振る舞いに関連するエラーです。例えば、ロールバックに失敗しました。あるいは、コラボレーションが差し戻しを適用できませんでした。 |

### 注

Get Type 機能ブロックは、現在の例外から例外タイプを取得します。例外タイプは、例外の原因を示す String です。

### 関連情報

この機能ブロックは、collaborationException.getType() メソッドを基にしています。詳細については、 440 ページの『getType()』を参照してください。

## To String

例外タイプや例外テキストなどの例外情報を String にフォーマット設定します。

### 入力

**Collaboration exception**
コラボレーション例外 (CollaborationException オブジェクト) です。

### 出力

例外タイプと例外テキストを含む String を戻します。

### 注

To String 機能ブロックは、現在の例外に関する例外情報を以下のようにフォーマット設定します。

*exceptionType*: *messageText*

上記の行の *exceptionType* は例外オブジェクトの例外タイプであり、*messageText* は例外テキストです。

**注:** Get Message 機能ブロックを使用して、現在の例外から例外テキストだけを取得できます。

## 関連情報

この機能ブロックは、collaborationException.toString() メソッドを基にしています。詳細については、 441 ページの『toString()』を参照してください。

# 第 17 章 実行機能ブロック

General¥APIs¥Execution Context フォルダーの機能ブロックには、実行コンテキスト機能があります。この機能ブロックは、グローバル実行コンテキストで実行されます。グローバル実行コンテキストは、所定のフローに関連付けられたユーザー・アクセス可能なコンテキスト情報のホルダーです。以下のセクションでは、機能ブロックについて詳しく説明します。

*表 69. 実行コンテキスト機能ブロックの要約*

| 機能ブロック | ページ |
|---|---|
| Get Context | 329 |
| MAPCONTEXT | 329 |
| New Execution Context | 330 |
| Set Context | 330 |

## Get Context

グローバル実行コンテキストから、指定した実行コンテキストを検索します。

### 入力

**Execution context**
グローバル実行コンテキスト (CxExecutionContext オブジェクト)。

**Context name**
グローバル実行コンテキストから取得する実行コンテキストの名前を含む String オブジェクト。

### 出力

指定した実行コンテキストのインスタンスを戻します。

### 関連情報

この機能ブロックは、CwExecutionContext.getContext() メソッドを基にしています。詳細については、 434 ページの『getContext()』を参照してください。

## MAPCONTEXT

実行コンテキストがマップ特定であることを示すために使用される String 定数。

### 出力

MAPCONTEXT String を戻します。

## New Execution Context

グローバル実行コンテキストの新しいインスタンスを作成します。

### 出力

グローバル実行コンテキストの新しいインスタンスを戻します。

### 注

New Execution Context 機能ブロックは、グローバル実行コンテキストを戻します。グローバル実行コンテキストは、コラボレーションからマップを呼び出す前にマップ実行コンテキストを保持するために必要です。

### 関連情報

この機能ブロックは、CwExecutionContext() コンストラクターを基にしています。詳細については、433 ページの『CxExecutionContext()』を参照してください。

## Set Context

特定の実行コンテキストをグローバル実行コンテキストの一部となるように設定します。

### 入力

**Execution context**
グローバル実行コンテキスト (CxExecutionContext オブジェクト)。

**Context name**
グローバル実行コンテキストから取得する実行コンテキストの名前を含む String オブジェクト。

**Context**
実行コンテキストの情報を含むオブジェクト。マップ実行コンテキストの場合は、このオブジェクトのタイプは MapExeContext です。

### 関連情報

この機能ブロックは、CwExecutionContext.setContext() メソッドを基にしています。詳細については、434 ページの『setContext()』を参照してください。

# 第 18 章 日付の機能ブロック

General¥Date フォルダーおよびその ¥Formats サブフォルダーにある機能ブロックは、日付の処理に関する機能を提供します。

*表 70. 日付の機能ブロックの要約*

| フォルダー | 機能ブロック | ページ |
|---|---|---|
| General¥Date | Add Day | 331 |
| | Add Month | 331 |
| | Add Year | 332 |
| | Date After | 332 |
| | Date Before | 332 |
| | Date Equals | 333 |
| | Format Change | 333 |
| | Get Day | 333 |
| | Get Month | 334 |
| | Get Year | 334 |
| | Get Year Month Day | 334 |
| General¥Date¥Formats | yyyy-MM-dd | 335 |
| | yyyyMMdd | 335 |
| | yyyyMMdd HH:mm:ss | 335 |

## Add Day

オリジナルの日付 (From date 入力で指定する) にさらに日数を追加します。

### 入力

**From date** オリジナルの日付を表す String オブジェクト。

**Date format** 日付の形式を表す String オブジェクト。

**Day to add** オリジナルの日付に追加する日数を指定する整数。

### 出力

更新された日付を含む String オブジェクトを戻します。

## Add Month

オリジナルの日付にさらに月数を追加します。

### 入力

**From date** オリジナルの日付を表す String オブジェクト。

**Date format** 日付の形式を表す String オブジェクト。

**Month to add**
オリジナルの日付に追加する月数を指定する整数。

### 出力

更新された日付を含む String オブジェクトを戻します。

---

## Add Year

オリジナルの日付にさらに年数を追加します。

### 入力

**From date** オリジナルの日付を表す String オブジェクト。

**Date format** 日付の形式を表す String オブジェクト。

**Year to add** オリジナルの日付に追加する年数を指定する整数。

### 出力

更新された日付を含む String オブジェクトを戻します。

---

## Date After

2 つの日付を比較して、Date 1 が Date 2 よりも後かどうかを判別します。

### 入力

**Date 1** 比較する第 1 の日付を表す String オブジェクト。

**Date 1 format**
Date 1 の形式を表す String オブジェクト。

**Date 2** 比較における第 2 の日付を表す String オブジェクト。

**Date 2 format**
Date 2 の形式を表す String オブジェクト。

### 出力

Date 1 が Date 2 よりも後の場合は `True` を戻し、それ以外の場合は `False` を戻します。

---

## Date Before

2 つの日付を比較して、Date 1 が Date 2 よりも前かどうかを判別します。

### 入力

**Date 1** 比較する第 1 の日付を表す String オブジェクト。

**Date 1 format**
Date 1 の形式を表す String オブジェクト。

**Date 2** 比較における第 2 の日付を表す String オブジェクト。

**Date 2 format**
Date 2 の形式を表す String オブジェクト。

### 出力

Date 1 が Date 2 よりも前の場合は `True` を戻し、それ以外の場合は `False` を戻します。

---

## Date Equals

2 つの日付を比較して、それらが等しいかどうかを判別します。

### 入力

**Date 1** 比較する第 1 の日付を表す String オブジェクト。

**Date 1 format**
Date 1 の形式を表す String オブジェクト。

**Date 2** 比較における第 2 の日付を表す String オブジェクト。

**Date 2 format**
Date 2 の形式を表す String オブジェクト。

### 出力

両方の日付が等しい場合は `True` を戻し、等しくない場合は `False` を戻します。

---

## Format Change

日付の形式を変更します。

### 入力

**Date** 形式を再設定したい日付を表す String オブジェクト。

**Input format** 日付のオリジナル形式を表す String オブジェクト。

**Output format**
日付の新規形式を表す String オブジェクト。

### 出力

形式が再設定された日付を含む String オブジェクトを戻します。

---

## Get Day

日付式に基づいて、月のうちの日を示す数値を戻します。

### 入力

**Date** 日付を表す String オブジェクト。

**Format** 日付の形式を表す String オブジェクト。

### 出力

月のうちの日を示す整数を戻します。

---

## Get Month

日付式に基づいて、年のうちの月を示す数値を戻します。

### 入力

**Date** 日付を表す String オブジェクト。

**Format** 日付の形式を表す String オブジェクト。

### 出力

月の数値を示す整数を戻します。

---

## Get Year

日付式に基づいて、年を示す数値を戻します。

### 入力

**Date** 日付を表す String オブジェクト。

**Format** 日付の形式を表す String オブジェクト。

### 出力

年を示す整数を戻します。

---

## Get Year Month Day

入力された日付から年、月、および日の要素を抽出します。

### 入力

**Date** 日付を表す String オブジェクト。

**Format** 日付の形式を表す String オブジェクト。

### 出力

年を示す整数、月を示す整数、および日を示す整数、の 3 つを戻します。

---

## Now

今日の日付を検索します。

### 入力

**Format** 日付に使用する形式を表す String オブジェクト。

### 出力

今日の日付が Format 入力で指定された値に基づく形式になっている String オブジェクトを戻します。

## yyyy-MM-dd

日付形式 yyyy-MM-dd を表します (例: 2003-11-25)。

**注:** この機能ブロックは、General¥Date¥Formats フォルダーにあります。

### 出力

yyyy-MM-dd 形式に設定された日付を含む String オブジェクトを戻します。

### 注

この機能ブロックは、実際には日付の形式を設定しないので、独立の機能ブロックとして使用することはできません。この機能ブロックは、General¥Date フォルダーにある 1 つ以上の機能ブロック (例えば、Format Change または Add Day 機能ブロック) とともに使用する必要があります。

## yyyyMMdd

日付形式 yyyyMMdd を表します (例: 20031125)。

**注:** この機能ブロックは、General¥Date¥Formats フォルダーにあります。

### 出力

yyyyMMdd 形式に設定された日付を含む String オブジェクトを戻します。

### 注

この機能ブロックは、実際には日付の形式を設定しないので、独立の機能ブロックとして使用することはできません。この機能ブロックは、General¥Date フォルダーにある 1 つ以上の機能ブロック (例えば、Format Change または Add Day 機能ブロック) とともに使用する必要があります。

## yyyyMMdd HH:mm:ss

日付形式 yyyyMMdd HH:mm:ss を表します (例: 20031125 12:36:40)。

**注:** この機能ブロックは、General¥Date¥Formats フォルダーにあります。

### 出力

yyyyMMdd HH:mm:ss 形式に設定された日付を含む String オブジェクトを戻します。

### 注

この機能ブロックは、実際には日付の形式を設定しないので、独立の機能ブロックとして使用することはできません。この機能ブロックは、General¥Date フォルダー

にある 1 つ以上の機能ブロック (例えば、Format Change または Add Day 機能ブロック) とともに使用する必要があります。

# 第 19 章 ログおよびトレースの機能ブロック

¥General¥Logging and Tracing フォルダーおよびそのサブフォルダーにある機能ブロックは、エラー・メッセージ、情報メッセージ、警告メッセージ、およびトレース・メッセージの処理に関する機能を提供します。

以下のセクションでは、各機能ブロックの詳細を説明します。

*表 71. ログおよびトレースの機能ブロックの要約*

| フォルダー | 機能ブロック | ページ |
|---|---|---|
| General¥Logging and Tracing | Log error | 337 |
| | Log Error ID | 338 |
| | Log Information | 339 |
| | Log Information ID | 339 |
| | Log Warning | 341 |
| | Log Warning ID | 341 |
| | Trace | 342 |
| General¥Logging and Tracing¥Log Error | Log Error ID 1 | 338 |
| | Log Error ID 2 | 338 |
| | Log Error ID 3 | 339 |
| General¥Logging and Tracing¥Log Information | Log Information ID 1 | 339 |
| | Log Information ID 2 | 340 |
| | Log Information ID 3 | 340 |
| General¥Logging and Tracing¥Log Warning | Log Warning ID 1 | 341 |
| | Log Warning ID 2 | 341 |
| | Log Warning ID 3 | 342 |
| General¥Logging and Tracing¥Trace | Trace ID 1 | 343 |
| | Trace ID 2 | 343 |
| | Trace ID 3 | 344 |
| | Trace on Level | 344 |

## Log error

指定されたエラー・メッセージを InterChange Server Express ログ・ファイルに送信します。

### 入力

**Message** ログ・ファイルに送信するメッセージ。この入力に使用できる型は String、byte、short、int、long、float、または double です。

## Log Error ID

指定された ID に関連付けられたエラー・メッセージを InterChange Server Express ログ・ファイルに送信します。

### 入力

**ID** ログに記録するエラー・メッセージの ID。この入力に使用できる型は String、byte、short、int、long、float、または double です。

## Log Error ID 1

指定されたパラメーターを使って、ID に関連付けられたエラー・メッセージのフォーマットを設定し、そのメッセージを InterChange Server Express ログ・ファイルに送信します。

**注:** この機能ブロックは、General¥Logging and Tracing¥Log Error フォルダーにあります。

### 入力

**ID** ログに記録するエラー・メッセージの ID。この入力に使用できる型は String、byte、short、int、long、float、または double です。

**Parameter** エラー・メッセージのフォーマット設定に使用するパラメーター。この入力に使用できる型は String、byte、short、int、long、float、または double です。

## Log Error ID 2

指定された 2 つのパラメーターを使って、ID に関連付けられたエラー・メッセージのフォーマットを設定し、そのメッセージを InterChange Server Express ログ・ファイルに送信します。

**注:** この機能ブロックは、General¥Logging and Tracing¥Log Error フォルダーにあります。

### 入力

**ID** ログに記録するエラー・メッセージの ID。この入力に使用できる型は String、byte、short、int、long、float、または double です。

**Parameter 1** エラー・メッセージのフォーマット設定に使用する第 1 パラメーター。この入力に使用できる型は String、byte、short、int、long、float、または double です。

**Parameter 2** エラー・メッセージのフォーマット設定に使用する第 2 パラメーター。この入力に使用できる型は String、byte、short、int、long、float、または double です。

## Log Error ID 3

指定された 3 つのパラメーターを使って、ID に関連付けられたエラー・メッセージのフォーマットを設定し、そのメッセージを InterChange Server Express ログ・ファイルに送信します。

**注:** この機能ブロックは、General¥Logging and Tracing¥Log Error フォルダーにあります。

### 入力

**ID** ログに記録するエラー・メッセージの ID。この入力に使用できる型は String、byte、short、int、long、float、または double です。

**Parameter 1** エラー・メッセージのフォーマット設定に使用する第 1 パラメーター。この入力に使用できる型は String、byte、short、int、long、float、または double です。

**Parameter 2** エラー・メッセージのフォーマット設定に使用する第 2 パラメーター。この入力に使用できる型は String、byte、short、int、long、float、または double です。

**Parameter 3** エラー・メッセージのフォーマット設定に使用する第 3 パラメーター。この入力に使用できる型は String、byte、short、int、long、float、または double です。

## Log Information

指定された情報メッセージを InterChange Server Express ログ・ファイルに送信します。

### 入力

**Message** ログ・ファイルに送信するメッセージ。この入力に使用できる型は String、byte、short、int、long、float、または double です。

## Log Information ID

指定された ID に関連付けられた情報メッセージを InterChange Server Express ログ・ファイルに送信します。

### 入力

**ID** ログに記録する情報メッセージの ID。この入力に使用できる型は String、byte、short、int、long、float、または double です。

## Log Information ID 1

指定されたパラメーターを使って、ID に関連付けられた情報メッセージのフォーマットを設定し、そのメッセージを InterChange Server Express ログ・ファイルに送信します。

**注:** この機能ブロックは、General¥Logging and Tracing¥Log Information フォルダーにあります。

### 入力

**ID** ログに記録する情報メッセージの ID。この入力に使用できる型は String、byte、short、int、long、float、または double です。

**Parameter** メッセージのフォーマット設定に使用するパラメーター。この入力に使用できる型は String、byte、short、int、long、float、または double です。

## Log Information ID 2

指定された 2 つのパラメーターを使って、ID に関連付けられた情報メッセージのフォーマットを設定し、そのメッセージを InterChange Server Express ログ・ファイルに送信します。

**注:** この機能ブロックは、General¥Logging and Tracing¥Log Information フォルダーにあります。

### 入力

**ID** ログに記録する情報メッセージの ID。この入力に使用できる型は String、byte、short、int、long、float、または double です。

**Parameter 1** メッセージのフォーマット設定に使用する第 1 パラメーター。この入力に使用できる型は String、byte、short、int、long、float、または double です。

**Parameter 2** メッセージのフォーマット設定に使用する第 2 パラメーター。この入力に使用できる型は String、byte、short、int、long、float、または double です。

## Log Information ID 3

指定された 3 つのパラメーターを使って、ID に関連付けられた情報メッセージのフォーマットを設定し、そのメッセージを InterChange Server Express ログ・ファイルに送信します。

**注:** この機能ブロックは、General¥Logging and Tracing¥Log Information フォルダーにあります。

### 入力

**ID** ログに記録する情報メッセージの ID。この入力に使用できる型は String、byte、short、int、long、float、または double です。

**Parameter 1** メッセージのフォーマット設定に使用する第 1 パラメーター。この入力に使用できる型は String、byte、short、int、long、float、または double です。

**Parameter 2** メッセージのフォーマット設定に使用する第 2 パラメーター。この入力に使用できる型は String、byte、short、int、long、float、または double です。

**Parameter 3** メッセージのフォーマット設定に使用する第 3 パラメーター。この入力に使用できる型は String、byte、short、int、long、float、または double です。

## Log Warning

指定された警告メッセージを InterChange Server Express ログ・ファイルに送信します。

### 入力

**Message** ログ・ファイルに送信するメッセージ。この入力に使用できる型は String、byte、short、int、long、float、または double です。

## Log Warning ID

指定された ID に関連付けられた警告メッセージを InterChange Server Express ログ・ファイルに送信します。

### 入力

**ID** ログに記録する警告メッセージの ID。この入力に使用できる型は String、byte、short、int、long、float、または double です。

## Log Warning ID 1

指定されたパラメーターを使って、ID に関連付けられた警告メッセージのフォーマットを設定し、そのメッセージを InterChange Server Express ログ・ファイルに送信します。

**注:** この機能ブロックは、General¥Logging and Tracing¥Log Warning フォルダーにあります。

### 入力

**ID** ログに記録する警告メッセージの ID。この入力に使用できる型は String、byte、short、int、long、float、または double です。

**Parameter** メッセージのフォーマット設定に使用するパラメーター。この入力に使用できる型は String、byte、short、int、long、float、または double です。

## Log Warning ID 2

指定された 2 つのパラメーターを使って、ID に関連付けられた警告メッセージのフォーマットを設定し、そのメッセージを InterChange Server Express ログ・ファイルに送信します。

**注:** この機能ブロックは、General¥Logging and Tracing¥Log Warning フォルダーにあります。

### 入力

**ID** ログに記録する警告メッセージの ID。この入力に使用できる型は String、byte、short、int、long、float、または double です。

**Parameter 1** メッセージのフォーマット設定に使用する第 1 パラメーター。この入力に使用できる型は String、byte、short、int、long、float、または double です。

**Parameter 2** メッセージのフォーマット設定に使用する第 2 パラメーター。この入力に使用できる型は String、byte、short、int、long、float、または double です。

## Log Warning ID 3

指定された 3 つのパラメーターを使って、ID に関連付けられた警告メッセージのフォーマットを設定し、そのメッセージを InterChange Server Express ログ・ファイルに送信します。

**注:** この機能ブロックは、General¥Logging and Tracing¥Log Warning フォルダーにあります。

### 入力

**ID** ログに記録する警告メッセージの ID。この入力に使用できる型は String、byte、short、int、long、float、または double です。

**Parameter 1** メッセージのフォーマット設定に使用する第 1 パラメーター。この入力に使用できる型は String、byte、short、int、long、float、または double です。

**Parameter 2** メッセージのフォーマット設定に使用する第 2 パラメーター。この入力に使用できる型は String、byte、short、int、long、float、または double です。

**Parameter 3** メッセージのフォーマット設定に使用する第 3 パラメーター。この入力に使用できる型は String、byte、short、int、long、float、または double です。

## Trace

指定されたトレース・メッセージを InterChange Server Express ログ・ファイルに送信します。

### 入力

**Message** ログ・ファイルに送信するメッセージ。この入力に使用できる型は String、byte、short、int、long、float、または double です。

## Trace ID 1

指定されたパラメーターを使って、ID に関連付けられたトレース・メッセージのフォーマットを設定します。また、指定されたレベルに基づいて、トレース・メッセージを表示するかどうかを判断します。つまり、コラボレーションのトレースが指定レベル以上のレベルに設定されていれば、トレース・メッセージは表示されます。

**注:** この機能ブロックは、General¥Logging and Tracing¥Trace フォルダーにあります。

### 入力

**ID** ログに記録するトレース・メッセージの ID。この入力に使用できる型は String、byte、short、int、long、float、または double です。

**Level** メッセージが表示される最小トレース・レベル。この入力に使用できる型は String、byte、short、int、long、float、または double です。

**Parameter** メッセージのフォーマット設定に使用するパラメーター。この入力に使用できる型は String、byte、short、int、long、float、または double です。

## Trace ID 2

指定された 2 つのパラメーターを使って、ID に関連付けられたトレース・メッセージのフォーマットを設定します。また、指定されたレベルに基づいて、トレース・メッセージを表示するかどうかを判断します。つまり、コラボレーションのトレースが指定レベル以上のレベルに設定されていれば、トレース・メッセージは表示されます。

**注:** この機能ブロックは、General¥Logging and Tracing¥Trace フォルダーにあります。

### 入力

**ID** ログに記録するトレース・メッセージの ID。この入力に使用できる型は String、byte、short、int、long、float、または double です。

**Level** メッセージが表示される最小トレース・レベル。この入力に使用できる型は String、byte、short、int、long、float、または double です。

**Parameter 1** メッセージのフォーマット設定に使用する第 1 パラメーター。この入力に使用できる型は String、byte、short、int、long、float、または double です。

**Parameter 2** メッセージのフォーマット設定に使用する第 2 パラメーター。この入力に使用できる型は String、byte、short、int、long、float、または double です。

## Trace ID 3

指定された 3 つのパラメーターを使って、ID に関連付けられたトレース・メッセージのフォーマットを設定します。また、指定されたレベルに基づいて、トレース・メッセージを表示するかどうかを判断します。つまり、コラボレーションのトレースが指定レベル以上のレベルに設定されていれば、トレース・メッセージは表示されます。

**注:** この機能ブロックは、General¥Logging and Tracing¥Trace フォルダーにあります。

### 入力

**ID** ログに記録するトレース・メッセージの ID。この入力に使用できる型は String、byte、short、int、long、float、または double です。

**Level** メッセージが表示される最小トレース・レベル。この入力に使用できる型は String、byte、short、int、long、float、または double です。

**Parameter 1** メッセージのフォーマット設定に使用する第 1 パラメーター。この入力に使用できる型は String、byte、short、int、long、float、または double です。

**Parameter 2** メッセージのフォーマット設定に使用する第 2 パラメーター。この入力に使用できる型は String、byte、short、int、long、float、または double です。

**Parameter 3** メッセージのフォーマット設定に使用する第 3 パラメーター。この入力に使用できる型は String、byte、short、int、long、float、または double です。

## Trace on Level

コラボレーションのトレースが指定レベル以上のレベルに設定されている場合に、トレース・メッセージを表示します。

**注:** この機能ブロックは、General¥Logging and Tracing¥Trace フォルダーにあります。

### 入力

**Message** 表示されるメッセージ。この入力に使用できる型は String、byte、short、int、long、float、または double です。

**Level** メッセージが表示される最小トレース・レベル。この入力に使用できる型は String、byte、short、int、long、float、または double です。

# 第 20 章 ストリングの機能ブロック

General¥String フォルダーにある機能ブロックは、String オブジェクトの処理に関する機能を提供します。以下のセクションでは、各機能ブロックの詳細を説明します。

*表 72. ストリングの機能ブロックの要約*



## Append Text

In String 2 入力の値を In String 1 入力の値に付加します。

### 入力

**In String 1** String オブジェクト。

**In String 2** In String 1 で指定したストリングに付加する String オブジェクト。

### 出力

In String 1 のストリングに In String 2 のストリングが付加された String オブジェクト。例えば、In String 1 の値が「Hello world.」で、In String 2 の値が「How are you?」の場合、この機能ブロックの出力は、「Hello world. How are you?」というストリングになります。

---

## If

条件が真の場合は第 1 の値を戻し、条件が偽の場合は第 2 の値を戻します。

### 入力

**Condition** 出力の判別に使用する条件。この入力に使用できる型は、boolean または Boolean です。

**Value 1** String オブジェクト。

**Value 2** String オブジェクト。

### 出力

条件が満たされたかどうかによって、Value 1 または Value 2 に関連付けられた String オブジェクトを戻します。

---

## Is Empty

第 1 の値が空の場合に、第 2 の値を戻します。

### 入力

**Value 1** String オブジェクト。

**Value 2** String オブジェクト。

### 出力

Value 1 が空の場合は、Value 2 に関連付けられた String を戻します。それ以外の場合は、何も戻しません。

---

## Is NULL

第 1 の値がヌルの場合に、第 2 の値を戻します。

### 入力

**Value 1** String オブジェクト。

**Value 2** String オブジェクト。

### 出力

Value 1 がヌルの場合は、Value 2 に関連付けられた String を戻します。それ以外の場合は、何も戻しません。

## Left Fill

指定された長さの String オブジェクトを戻します。そのときに、左の部分を指定の値で充てんします。

### 入力

**String** String オブジェクト。

**Fill string** String オブジェクト。

**Length** 戻される String オブジェクトの長さを指定する整数。

### 出力

充てんされた String オブジェクトを戻します。

## Left String

ストリングの左から、指定された桁数に相当する部分を戻します。

### 入力

**String** 検索する String オブジェクト。

**Length** 戻す String オブジェクトの桁数を指定する整数。

### 出力

オリジナル String オブジェクトの左の部分を含む String オブジェクトを戻します。

## Lower Case

String オブジェクト内のすべての文字を小文字に変更します。

### 入力

**From string** オリジナルの String オブジェクト。

### 出力

すべての文字が小文字になった String オブジェクトを戻します。

## Object to String

オブジェクトのストリング表現を検索します。

### 入力

**Object** 検索するオブジェクト。

### 出力

指定されたオブジェクトを String オブジェクトとして戻します。

## Repeat

指定の回数繰り返された指定文字式を含む String を戻します。

### 入力

**Repeating string**
検索したい繰り返し String オブジェクト。

**Repeat count** 検索前に指定の文字ストリングを繰り返す回数を指定する整数。

### 出力

繰り返された文字ストリングを戻します。

## Replace

文字ストリングの一部を新しい文字ストリングに置き換えます。

### 入力

**String** String オブジェクト。

**Old string** 置換したい特定のサブストリング文字を含む String オブジェクト。

**New string** 代替として使用する新規文字ストリングを含む String オブジェクト。

### 出力

更新された文字ストリングを含む String オブジェクトを戻します。

## Right Fill

指定された長さの String オブジェクトを戻します。そのときに、右の部分を指定の値で充てんします。

### 入力

**String** String オブジェクト。

**Fill string** 右の部分の充てんに使用する String オブジェクト。

**Length** 戻される String オブジェクトの長さを指定する整数。

### 出力

指定された String オブジェクトを指定の値で充てんして戻します。

## Right String

String オブジェクトの右から、指定された桁数に相当する部分を戻します。

### 入力

**String** 戻す String オブジェクト。

**Length** 戻す文字ストリング内の桁数を指定する整数。

### 出力

指定されたストリングの右から、指定の桁数に相当する部分の String オブジェクトを戻します。

## Substring by Position

開始および終了パラメーターに基づいて、String オブジェクトの一部分を戻します。

### 入力

**String** String オブジェクト。

**Start position**
戻される String の部分の先頭を指定する整数。

**End position** 戻される String の部分の終了を指定する整数。

### 出力

指定されたサブストリングを含む String オブジェクトを戻します。

## Substring by Value

指定された開始値および終了値に基づいて、String オブジェクトの一部分を戻します。サブストリングには、開始値と終了値は含まれず、この 2 つの値の間にある部分がすべて含まれます。

### 入力

**String** String オブジェクト。

**Start value** サブストリングの開始値を指定する整数。

**End value** サブストリングの終了値を指定する整数。

### 出力

指定されたサブストリングを含む String オブジェクトを戻します。

## Text Equal

2 つの String オブジェクトの文字ストリングを比較して、それらが等しいかどうかを判別します。

### 入力

**In String 1** 比較における第 1 String オブジェクト。

**In String 2**　比較における第 2 String オブジェクト。

### 出力

両方の String オブジェクトの内容が等しい場合は `True` を戻し、等しくない場合は `False` を戻します。

## Text Equal Ignore Case

2 つの String オブジェクトの文字ストリングを辞書式に (大文字小文字の違いを無視して) 比較して、それらが等しいかどうかを判別します。

### 入力

**In String 1**　比較における第 1 String オブジェクト。

**In String 2**　比較における第 2 String オブジェクト。

### 出力

両方の String オブジェクトの内容が等しい場合は `True` を戻し、等しくない場合は `False` を戻します。

## Text Length

String オブジェクト内の文字の総数を検出します。

### 入力

**String**　String オブジェクト。

### 出力

String オブジェクトの長さを、byte、short、int、long、float、double のいずれかのデータ型で戻します。

## Trim Left

String オブジェクトの左側から、指定された文字数をトリムします。

### 入力

**String**　トリムする String オブジェクト。

**Trim length**　トリムする文字数を指定する整数。

### 出力

トリムされた String オブジェクトを戻します。

## Trim Right

String オブジェクトの右側から、指定された文字数をトリムします。

### 入力

**String** トリムする String オブジェクト。

**Trim length** トリムする文字数を指定する整数。

### 出力

トリムされた String オブジェクトを戻します。

---

## Trim Text

String オブジェクト内の文字の前後にある空白文字をトリムします。

### 入力

**In String** トリムする String オブジェクト。

### 出力

トリムされた String オブジェクトを戻します。

---

## Upper Case

String オブジェクト内のすべての文字を大文字に変更します。

### 入力

**From String** 文字を大文字に変換したいオリジナル String オブジェクト。

### 出力

すべての文字が大文字に変換された String オブジェクトを戻します。

# 第 21 章 ユーティリティーの機能ブロック

General¥Utilities フォルダーとそのサブフォルダーの機能ブロックには、アクティビティーとサブアクティビティーの例外を処理し、ロケール問題を管理する Vector オブジェクトで作業する機能があります。以下のセクションでは、各機能ブロックの詳細を説明します。

*表 73. ユーティリティーの機能ブロックの要約*

| フォルダー | 機能ブロック | ページ |
|---|---|---|
| General¥Utilities | Catch Error | 354 |
| | Catch Error Type | 354 |
| | Condition | 354 |
| | Loop | 357 |
| | Move Attribute in Child | 357 |
| | Raise Error | 359 |
| | Raise Error Type | 359 |
| General¥Utilities¥Locale | Get Country | 355 |
| | Get Language | 356 |
| | New Locale | 358 |
| | New Locale with Language | 358 |
| General¥Utilities¥Locale¥Constants | English | 354 |
| | French | 355 |
| | German | 355 |
| | Italian | 356 |
| | Japanese | 357 |
| | Korean | 357 |
| | Simplified Chinese | 359 |
| | Traditional Chinese | 360 |
| General¥Utilities¥Vector | Add Element | 353 |
| | Get Element | 355 |
| | Iterate Vector | 356 |
| | New Vector | 358 |
| | Size | 359 |
| | To Array | 359 |

## Add Element

指定された要素を Vector の最後に追加し、サイズを 1 つ分増やします。

**注:** この機能ブロックは、General¥Utilities¥Vector フォルダーにあります。

### 入力

**Vector** java.util.Vector オブジェクト。

### 出力

要素を戻します。

## Catch Error

現在のアクティビティーとそのサブアクティビティーでスローされたすべての例外をキャッチします。(サブアクティビティーを定義するには、編集キャンバスにある機能ブロックのアイコンをダブルクリックします。)

### 入力

**Error name** エラーの名前を指定する String オブジェクト。

**Error message**
エラー・メッセージの内容を指定する String オブジェクト。

## Catch Error Type

現在のアクティビティーとそのサブアクティビティーでスローされた指定の例外タイプをキャッチします。(サブアクティビティーを定義するには、編集キャンバスにある機能ブロックのアイコンをダブルクリックします。)

### 入力

**Error type** キャッチする例外のタイプを指定する String オブジェクト。

**Error message**
エラー・メッセージの内容を指定する String オブジェクト。

## Condition

指定された条件が存在する場合は、「真のアクション」に定義されたサブアクティビティーを実行します。それ以外の場合は、「偽のアクション」に定義されたサブアクティビティーを実行します。(サブアクティビティーを定義するには、編集キャンバスにある機能ブロックのアイコンをダブルクリックします。)

### 入力

**Condition** 満たすべき条件を指定する boolean。

## English

英語ロケールを表す定数です。

**注:** この機能ブロックは、General¥Utilities¥Locale¥Constants フォルダーにあります。

### 出力

英語の java.util.Locale オブジェクト。

## French

フランス語ロケールを表す定数です。

**注:** この機能ブロックは、General¥Utilities¥Locale¥Constants フォルダーにあります。

### 出力

フランス語の java.util.Locale オブジェクト。

## German

ドイツ語ロケールを表す定数です。

**注:** この機能ブロックは、General¥Utilities¥Locale¥Constants フォルダーにあります。

### 出力

ドイツ語の java.util.Locale オブジェクト。

## Get Country

現在のロケールの国/地域コードを判別します。

**注:** この機能ブロックは、General¥Utilities¥Locale フォルダーにあります。

### 入力

**Locale** 現在のロケールを表す java.util.Locale オブジェクトです。

### 出力

指定されたロケールの国/地域コードを含む String オブジェクトを戻します。通常は、空のストリングか、大文字の ISO 3166 による 2 文字コードです。

### 注

この機能ブロックは、java.util.Locale.getCountry() メソッドを基にしています。

## Get Element

Vector オブジェクト内の指定された指標にある要素を取得します。

**注:** この機能ブロックは、General¥Utilities¥Vector フォルダーにあります。

### 入力

**Vector** java.util.Vector オブジェクト。

**Index** 指標の位置を指定する整数。

### 出力

指定された指標にある要素を戻します。

---

## Get Language

現在のロケールの言語コードを判別します。

**注:** この機能ブロックは、General¥Utilities¥Locale フォルダーにあります。

### 入力

**Locale** 現在のロケールを表す java.util.Locale オブジェクトです。

### 出力

ロケールの言語コードを含む String オブジェクトを戻します。空のストリングか、小文字の ISO 639 コードです。

### 注

この機能ブロックは、java.util.Locale.getLanguage() メソッドを基にしています。

---

## Italian

イタリア語ロケールを表す定数です。

**注:** この機能ブロックは、General¥Utilities¥Locale¥Constants フォルダーにあります。

### 出力

イタリア語の java.util.Locale オブジェクト。

---

## Iterate Vector

Vector オブジェクト内で繰り返します。

**注:** この機能ブロックは、General¥Utilities¥Vector フォルダーにあります。

### 入力

**Vector** java.util.Vector オブジェクト。

**Current index**
指標の位置を指定する整数。

**Current element**
要素を指定するオブジェクト。

## Japanese

日本語ロケールを表す定数です。

**注:** この機能ブロックは、General¥Utilities¥Locale¥Constants フォルダーにあります。

### 出力

日本語の java.util.Locale オブジェクト。

## Korean

韓国語ロケールを表す定数です。

**注:** この機能ブロックは、General¥Utilities¥Locale¥Constants フォルダーにあります。

### 出力

韓国語の java.util.Locale オブジェクト。

## Loop

指定された条件が偽になるまで、サブアクティビティーを繰り返します。(サブアクティビティーを定義するには、編集キャンバスにある機能ブロックのアイコンをダブルクリックします。)

### 入力

**Condition** 満たすべき条件を指定する boolean。

## Move Attribute in Child

ある属性の値を別の属性に移動します。

### 入力

**Source parent**
移動する子ビジネス・オブジェクト属性が含まれるビジネス・オブジェクト (BusObj オブジェクト)。

**Source child BO attribute**
値を移動したい属性が含まれる子ビジネス・オブジェクトの名前を示す String。

**From attribute**
移動する属性の名前を示す String。

**Destination parent**
オリジナルの属性値の移動先となるビジネス・オブジェクト (BusObj オブジェクト)。

**Destination child BO attribute**
値を置き換えたい属性が含まれる子ビジネス・オブジェクトの名前を示す String。

**To attribute** 値を **From attribute** の値に置き換えたい属性の名前を示す String。

## New Locale

指定された言語と国に基づいて新しいロケールを作成します。

**注:** この機能ブロックは、General¥Utilities¥Locale フォルダーにあります。

### 入力

**Language** ロケールの現在の言語を指定する String オブジェクト。

**Country** ロケールの現在の国を指定する String オブジェクト。

### 出力

java.util.Locale オブジェクトを戻します。

### 注

この機能ブロックは、util.Locale() メソッドを基にしています。

## New Locale with Language

言語コードから新規ロケールを作成します。

**注:** この機能ブロックは、General¥Utilities¥Locale フォルダーにあります。

### 入力

**Language** ロケールの言語を指定する String オブジェクト。

### 出力

新規 java.util.Locale オブジェクトを戻します。

### 注

この機能ブロックは、util.Locale() メソッドを基にしています。

## New Vector

新規 Vector オブジェクトを作成します。

**注:** この機能ブロックは、General¥Utilities¥Vector フォルダーにあります。

### 出力

新規 java.util.Vector オブジェクトを戻します。

## Raise Error

指定されたメッセージが書き込まれている新規 Java 例外をスローします。

### 入力

**Message** Java 例外のメッセージを含む String オブジェクト。

## Raise Error Type

指定されたメッセージが書き込まれている指定の Java 例外をスローします。

### 入力

**Error type** スローする Java 例外のタイプを示す String オブジェクト。

**Message** Java 例外のメッセージを含む String オブジェクト。

## Simplified Chinese

中国語 (簡体字) ロケールを表す定数です。

**注:** この機能ブロックは、General¥Utilities¥Locale¥Constants フォルダーにあります。

### 出力

中国語 (簡体字) の java.util.Locale オブジェクトを戻します。

## Size

Vector オブジェクト内の要素の数を判断します。

**注:** この機能ブロックは、General¥Utilities¥Vector フォルダーにあります。

### 入力

**Vector** java.util.Vector オブジェクト。

### 出力

Vector オブジェクトに含まれる要素数を指定する整数を戻します。

## To Array

現在の Vector オブジェクト内のすべての要素が含まれた配列表現を取得します。

**注:** この機能ブロックは、General¥Utilities¥Vector フォルダーにあります。

### 入力

**Vector** java.util.Vector オブジェクト。

### 出力

Vector オブジェクト内のすべての要素を Object[] 型の配列として戻します。

---

## Traditional Chinese

中国語 (繁体字) ロケールを表す定数です。

**注:** この機能ブロックは、General¥Utilities¥Locale¥Constants フォルダーにあります。

### 出力

中国語 (繁体字) の java.util.Locale オブジェクト。

# 第 4 部 コラボレーション API リファレンス

# 第 22 章 BaseCollaboration クラス

この章では、コラボレーション・オブジェクトのメソッドについて説明します。これらは、InterChange Server Express 定義の BaseCollaboration クラスで定義されています。この BaseCollaboration クラスは、すべてのコラボレーションの基本クラスとなります。作成されるコラボレーションはすべて BaseCollaboration のサブクラスで、これらのメソッドを継承します。

表 74 に、BaseCollaboration クラスのメソッドの要約を示します。

*表 74. BaseCollaboration メソッドの要約*

| メソッド | 説明 | ページ |
|---|---|---|
| existsConfigProperty() | コラボレーション構成プロパティーが存在するかどうかを検査します。 | 363 |
| getConfigProperty() | コラボレーション構成プロパティーの値を検索します。 | 364 |
| getConfigPropertyArray() | 複数要素のコラボレーション構成プロパティーの値を検索します。 | 364 |
| getDBConnection() | データベースとの接続を確立して、CwDBConnection オブジェクトを戻します。 | 366 |
| getLocale() | コラボレーションのロケールを検索します。 | 370 |
| getMessage() | コラボレーション・メッセージ・ファイルから、メッセージ番号で識別されるメッセージを検索します。 | 368 |
| getName() | コラボレーション・オブジェクトの名前を検索します。 | 370 |
| implicitDBTransactionBracketing() | コラボレーション・オブジェクトが取得する任意の接続に対して、コラボレーション・オブジェクトが使用するトランザクション・プログラミング・モデルを検索します。 | 370 |
| isTraceEnabled() | 指定されたトレース・レベルとコラボレーションの現在のトレース・レベルを比較します。 | 371 |
| logError()、logInfo()、logWarning() | エラー、情報、または警告メッセージをログ・ファイルに送信します。 | 372 |
| raiseException() | コラボレーション例外を生成します。 | 374 |
| sendEmail() | E メール・メッセージを非同期に送信します。 | 378 |
| trace() | トレース・メッセージを生成します。 | 379 |

## existsConfigProperty()

コラボレーション構成プロパティーが存在するかどうかを検査します。

### 構文

```
boolean existsConfigProperty(String propertyName)
```

### パラメーター

*propertyName* コラボレーション・テンプレートで定義されているプロパティーの名前。

### 戻り値

プロパティーが存在する場合は `true` を戻し、存在しない場合は `false` を戻します。

### 例

次の例では、`VALIDATE_CUSTOMER` のプロパティーが存在するかどうかを調査します。

```
boolean validatePropExists =
   existsConfigProperty("VALIDATE_CUSTOMER");
```

## getConfigProperty()

コラボレーション構成プロパティーの値を検索します。

### 構文

```
String getConfigProperty(String propertyName)
```

### パラメーター

*propertyName* コラボレーション・テンプレートで定義されているプロパティーの名前。

### 戻り値

構成プロパティーの値を戻します。プロパティーが存在しない場合は、空ストリング (“”) を戻します。

### 例

次の例では、`VALIDATE_CUSTOMER` プロパティーの値を取得し、その値を *validateProp* 変数に割り当てます。

```
String validateProp = getConfigProperty("VALIDATE_CUSTOMER");
```

## getConfigPropertyArray()

複数要素のコラボレーション構成プロパティーの値を検索します。

### 構文

```
String[] getConfigPropertyArray(String propertyName)
```

## パラメーター

*propertyName* コラボレーション・テンプレートで定義されているプロパティーの名前。

## 戻り値

プロパティー値の配列。

## 注記

このメソッドを使用して、構成プロパティーの複数要素の値を検索します。複数の要素を持つ構成プロパティーは、セミコロンで区切られたいくつかの値から成ります。

プロパティーが存在しない場合、配列は空になります。プロパティーに 1 つ要素がある場合、配列の持つ要素は 1 つのみです。

複数要素構成プロパティーを使用して、ユーザーからの入力を取得します。ビジネス・オブジェクトのキー属性値が存在しなくても、コラボレーションは複数要素プロパティーを使用して Retrieve 要求を作成することができます。構成プロパティーの要素の値を指定することによって、ユーザーは、ビジネス・オブジェクトの検索に使用される属性を指定することができます。

## 例

次の例では、ATTR_LIST という名前に関連付けられたプロパティーのリストを検索します。

```
String[] list = getConfigPropertyArray("ATTR_LIST");
```

---

# getCurrentLoopIndex()

イテレーターがループとして構成されている場合に、索引付き変数の値を検索します。

## 構文

```
int getCurrentLoopIndex()
```

## パラメーター

なし。

## 戻り値

ループ索引付き変数の値を戻します。ループ外の場合は、ゼロ (0) を戻します。

## 例

次の例は、現在のループ指標を取得します。

```
int currentIndex = getCurrentLoopIndex();
```

## getDBConnection()

データベースとの接続を確立して、CwDBConnection オブジェクトを戻します。

### 構文

```
CwDBConnection getDBConnection(String connectionPoolName)
CwDBConnection getDBConnection(String connectionPoolName,
   boolean implicitTransaction)
```

### パラメーター

*connectionPoolName*
有効な接続プールの名前。接続がこの接続プールに指定されているデータベースに、メソッドは接続されます。

*implicitTransaction*
トランザクション・プログラミング・モデルを、接続に関連付けられたデータベースに対して使用することを指示する boolean 値です。次の値が有効です。

`true` データベースで使用される暗黙的なトランザクション・ブラケット

`false` データベースで使用される明示的なトランザクション・ブラケット

### 戻り値

CwDBConnection オブジェクトを戻します。

### 例外

CwDBConnectionFactoryException - データベース接続の確立中にエラーが発生した場合。

### 注記

getDBConnection() メソッドは、*connectionPoolName* で指定されている接続プールから接続を取得します。この接続によって、接続に関連付けられたデータベースへの照会および更新の実行が可能になります。特定の接続プールにおけるすべての接続は、同じデータベースに関連付けられています。メソッドは CwDBConnection オブジェクトを戻して、それによって照会の実行およびトランザクションの管理が可能になります。詳細については、CwDBConnection クラスのメソッドを参照してください。

デフォルトでは、暗黙的なトランザクション・ブラケットをすべての接続でトランザクション・プログラミング・モデルとして使用します。トランザクション・プログラミング・モデルを特定の*接続* に対して指定するには、boolean 値を付けて、getDBConnection() メソッドに対してオプションの *implicitTransaction* 引き数として目的のトランザクション・プログラミング・モデルを示します。次の getDBConnection() 呼び出しでは、ConnPool 接続プールから取得する接続に対して明示的なトランザクション・ブラケットを指定します。

```
conn = getDBConnection("ConnPool",false);
```

コラボレーション・オブジェクトの実行が完了すると、接続が解除されます。release() メソッドを使用して、接続を明示的に閉じることができます。接続が解除されたかどうかを、isActive() メソッドを使用して判断することができます。詳細については、242 ページの『接続の解放』を参照してください。

## 例

次の例では、CustConnPool 接続プールにおいて、接続に関連付けられたデータベースへの接続を確立します。その後で、接続は暗黙的なトランザクションを使用し、データベースの表に行を挿入および更新します。

```
CwDBConnection connection = getDBConnection("CustConnPool");

// Insert a row
connection.executeSQL("insert...");

// Update rows...
connection.executeSQL("update...");
```

getDBConnection() に先行する呼び出しに、オプションの第 2 引き数は含まれません。そのため、この接続は、トランザクション・プログラミング・モデルとして、暗黙的なトランザクション・ブラケットを使用します (トランザクション・プログラミング・モデルが System Manager の「コラボレーション・プロパティー」ダイアログでオーバーライドされる場合を除く)。その結果、この例では、beginTransaction()、commit()、および rollback() との明示的なトランザクション境界は指定されません。実際に、これらのトランザクション・メソッドの 1 つを暗黙的なトランザクション・ブラケットで呼び出そうとすると、CwDBTransactionException 例外が生成されます。

**注:** implicitDBTransactionBracketing() メソッドを使用して、現在のトランザクション・プログラミング・モデルをチェックすることができます。

次の例でも、CustConnPool 接続プールにおいて、接続に関連付けられたデータベースへの接続を確立します。しかし、この例では接続の明示的なトランザクション・ブラケットの使用が指定されます。その結果、次のような明示的なトランザクションを使用してデータベースの表に行を挿入する、あるいはデータベースの表の行を更新します。

```
CwDBConnection connection = getDBConnection("CustConnPool", false);

// Begin a transaction
connection.beginTransaction();

// Insert a row
connection.executeSQL("insert...");

// Update rows...
connection.executeSQL("update...");

// Commit the transaction
connection.commit();

// Release the connection
connection.release();
```

getDBConnection() に先行する呼び出しには、オプションの *implicitTransaction* 引き数が含まれています。そのため、トランザクション・プログラミング・モデルが明示的なトランザクション・ブラケットに設定されます。したがって、この例では明示的なトランザクション呼び出しを使用して、トランザクションの境界を示します。これらのトランザクション・メソッドが省略されると、InterChange Server Express はそのトランザクションを暗黙的なものとみなして処理します。

### 関連項目

『第 25 章 CwDBConnection クラス』、isActive()、release()

---

## getLocale()

現行コラボレーション・オブジェクトのコラボレーション・ロケールを検索します。

### 構文

```
java.util.Locale getLocale()
```

### パラメーター

なし。

### 戻り値

コラボレーション・ロケールの言語と国別コードを含む Java Locale オブジェクト。この Locale オブジェクトは、java.util.Locale クラスのインスタンスでなければなりません。

### 注記

getLocale() メソッドは、現行フローのロケールを戻します。このフローのロケールは、コラボレーション・オブジェクトのトリガー・ビジネス・オブジェクトと関連付けられているロケールです。

### 例

次の例では、コラボレーションのロケールを取得し、Locale オブジェクトから言語コードと国コードを検索した後、トレース・メッセージでその値を報告します。

```
Locale collaborationLocale = getLocale();
String collaborationCountry = collaborationLocale.getCountry();
String collaborationLanguage = collaborationLocale.getLanguage();

trace(3, "THE COUNTRY CODE FOR THE COLLABORATION IS "
+ collaborationCountry + ", AND THE LANGUAGE CODE FOR THE
COLLABORATION IS " + collaborationLanguage + ".");
```

---

## getMessage()

コラボレーション・メッセージ・ファイルからメッセージを取得します。

## 構文

```
public String getMessage(int messageNum)
public String getMessage(int messageNum, Object[] paramArray)
```

## パラメーター

**messageNum**

コラボレーションのメッセージ・ファイル内のメッセージのメッセージ番号で、メッセージ番号で索引付けされます。メッセージ・テキスト・ファイルの設定方法の詳細は、197 ページの『第 9 章 メッセージ・ファイルの作成』を参照してください。

**paramArray**

メッセージ・パラメーター値の配列。各パラメーターは順次的に解決され、メッセージ・テキスト内のパラメーターになります。メッセージ内 (コラボレーション・メッセージ・ファイル内) では、メッセージ・パラメーターは中括弧で囲まれた整数 (例えば、{1}) で示されます。

## 戻り値

*messageNum* で識別されるメッセージのメッセージ・テキストを含む String オブジェクト。

## 注記

getMessage() メソッドには、次の 2 つの形式があります。

- 最初の形式は、メッセージ番号を使用してコラボレーション・メッセージ・ファイルから関連するメッセージを String オブジェクトとして取得します。
- ２番目の形式は、メッセージ番号とメッセージ・パラメーター値の配列を使用します。コラボレーション・メッセージ・ファイルから関連するメッセージを取得して、パラメーター配列内のオブジェクトでメッセージ・パラメーターを置き換え、結果のメッセージを String オブジェクトとして戻します。

メッセージ・ファイルとメッセージ・パラメーターの詳細については、197 ページの『第 9 章 メッセージ・ファイルの作成』を参照してください。

## 例

コラボレーション・メッセージ・ファイルで、メッセージ番号が 8 と 9 の次の 2 つのメッセージを定義すると仮定します。

```
8
Error occurred during JDBC URL conversion. Reason:{1}
[EXPL]
An error, indicated by the reason, occurred during the
conversion of a JDBC URL string.
9
Invalid login encountered in command-line arguments. A valid
login must contain a login name and a password.
[EXPL]
A password has been specified but a user name has not. If no
login name is specified, the default login name "crossworlds" is assumed.
```

次の getMessage() の呼び出しで、メッセージ 9 に関連するテキストを取得します。

```
String invalidLogin = getMessage(9);
```

次の getMessage() の呼び出しで、メッセージ 8 に関連するテキストを取得し、メッセージの Reason パラメーターの値を組み込みます。

```
String reason = "Invalid database table.";
Object[] paramList = new Object[1];
paramList[0] = reason;
badConversion = getMessage(8, paramList);
```

前の getMessage() 呼び出しで取得したメッセージは、次のようになります。

```
Error occurred during JDBC URL conversion. Reason:Invalid database table.
```

## getName()

コラボレーション・オブジェクトの名前を検索します。

### 構文

```
String getName()
```

### 例

次の例では、現在のコラボレーション・オブジェクト名を取得して、情報メッセージを記録します。

```
String collabName = getname();
logInfo(collabName + " is starting");
```

## implicitDBTransactionBracketing()

コラボレーション・オブジェクトが取得する任意の接続に対して、コラボレーション・オブジェクトが使用するトランザクション・プログラミング・モデルを検索します。

### 構文

```
boolean implicitDBTransactionBracketing()
```

### パラメーター

なし。

### 戻り値

すべてのデータベース接続で使用されるトランザクション・プログラミング・モデルを示す boolean です。

### 注記

implicitDBTransactionBracketing() メソッドは、以下のような boolean 値を戻します。この値は、コラボレーション・オブジェクトが前提とし、コラボレーション・オブジェクトが取得するすべての 接続によって使用される、トランザクション・プログラミング・モデルを示します。

- 値が true の場合は、すべての接続が*暗黙的な* トランザクション・ブラケットを使用していることを示します。
- 値が false の場合は、すべての接続が*明示的な* トランザクション・ブラケットを使用していることを示します。

現在のトランザクション・プログラミング・モデルが適切かどうかを確認するには、接続を取得する前にこのメソッドを使用することです。

**注:** getDBConnection() メソッドを使用して、特定の接続のトランザクション・プログラミング・モデルをオーバーライドすることができます。

### 例

次の例では、コラボレーション・オブジェクトが、conn 接続と関連したデータベースに明示的なトランザクション・ブラケットを使用していることを示しています。

```
if (implicitDBTransactionBracketing())
   CwDBConnection conn = getDBConnection("ConnPool", false);
```

### 関連項目

237 ページの『トランザクションの管理』

getDBConnection()

## isTraceEnabled()

指定されたトレース・レベルとコラボレーションの現在のトレース・レベルを比較します。

### 構文

```
public Boolean isTraceEnabled(int traceLevel)
```

### パラメーター

*traceLevel* 指定されたトレース・レベルと現在のトレース・レベルを比較します。

### 戻り値

指定されたトレース・レベルに現行システムのトレース・レベルが設定されると、true を戻します。2 つのトレース・レベルが異なる場合、false を戻します。

### 注記

isTraceEnabled() メソッドは、トレース・メッセージを記録すべきかどうかを判断する際に役立ちます。トレースによってパフォーマンスが低下することがあるため、このメソッドはプロジェクトの開発段階において使用するようにしてください。

### 例

```
if ( isTraceEnabled(3) )
   {
   trace("Print this level-3 trace message");
   }
```

## logError()、logInfo()、logWarning()

エラー、情報、または警告メッセージをログ宛先に書き込みます。

### 構文

```
void logError(String message)
void logError(int messageNum)
void logError(int messageNum, String param [,...])
void logError(int messageNum, Object[] paramArray)

void logInfo(String message)
void logInfo(int messageNum)
void logInfo(int messageNum, String param [,...])
void logInfo(int messageNum, Object[] paramArray)

void logWarning(String message)
void logWarning(int messageNum)
void logWarning(int messageNum, String param [,...])
void logWarning(int messageNum, Object[] paramArray)
```

### パラメーター

*message* ログに記録されるメッセージ・テキスト。

*messageNum* コラボレーションのメッセージ・ファイル内のメッセージのメッセージ番号で、メッセージ番号で索引付けされます。メッセージ・テキスト・ファイルの設定方法の詳細は、197 ページの『第 9 章 メッセージ・ファイルの作成』を参照してください。

*param* 単一メッセージ・パラメーターの値。コンマで区切られた、最大 5 つのメッセージ・パラメーターが可能です。各パラメーターは順次的に解決され、メッセージ・テキスト内のパラメーターになります。

*paramArray* メッセージ・パラメーター値の配列。各パラメーターは順次的に解決され、メッセージ・テキスト内のパラメーターになります。

### 注記

logError()、logWarning()、および logInfo() メソッドは、コラボレーションのログ宛先にメッセージを送信します。デフォルトのログ宛先は、ファイル InterchangeSystem.log です。ユーザーは、InterChange Server 構成ファイル InterchangeSystem.cfg 内の LOG_FILE のパラメーター値を入力してログ宛先を変更することができます。パラメーター値がファイル名あるいは STDOUT になることがあります。STDOUT は、InterChange Server のコマンド・ウィンドウにログを記録します。

ロギングに関連するシステム構成パラメーターを、さらに 3 つ設定することもできます。すべてのパラメーターは、InterChange Server 構成ファイル InterchangeSystem.cfg 内に格納されます。

- MAX_LOG_FILE_SIZE パラメーターを使用して、ログ・ファイルの最大サイズを設定します。デフォルトのファイル・サイズは無制限なので、常に最大サイズに設定する必要があります。
- NUMBER_OF_ARCHIVE_LOGS パラメーターを使用して、1 つから 5 つの間でアーカイブ・ログ・ファイルを設定します。パラメーターが設定されていない場合、デフォルトは 5 つです。
- エラー・メッセージをログ・ファイルに書き込むのと同時に STDOUT に表示する場合は、MIRROR_LOG_TO_STDOUT パラメーターを設定します。

エラー・メッセージ、情報メッセージまたは警告メッセージを記録するメソッドを使用するかどうかを判断する際のヘルプとして、203 ページの『メッセージのロギング』を参照してください。ユーザーのログ・ファイルに表示されるメッセージ・テキストには、メッセージの記録に使用するメソッドに応じて、Error、Info または Warning という接頭部が付きます。

これらの各ロギング・メソッドには、いくつかの形式があります。

- 第 1 の形式は、メッセージの生成に必要なテキストをすべて組み込みます。このメッセージをログ宛先に送信します。
- 第 2 の形式は、コラボレーションのメッセージ・ファイルからパラメーターのない メッセージを取得し、そのメッセージをログ宛先に送信します。
- 第 3 の形式は、コラボレーションのメッセージ・ファイルからパラメーターのある メッセージを取得します。また、メッセージ・パラメーター値のリストも示します。
- 第 4 の形式も、コラボレーションのメッセージ・ファイルからパラメーターのあるメッセージを取得します。しかし、メッセージ・パラメーター値をパラメーター値の配列として示します。

*messageNum* パラメーターを取得するメソッドのすべての形式で、メッセージ・ファイルを使用する必要があります。このファイルはメッセージ番号で索引化されます。メッセージ・テキスト・ファイルの設定方法の詳細は、197 ページの『第 9 章 メッセージ・ファイルの作成』を参照してください。

logError() メソッドは、ログ宛先にメッセージを送信するのに加え、以下の場合にエラー・メッセージを E メール受信側に送信します。

- E メール・アドレスは「Collaboration Object Properties」ダイアログの「E メール通知アドレス」フィールドで指定されている場合
- Email コラボレーションおよび Email コネクターが実行されている場合 (InterChange Server が始動するときにユーザーによる入力が必要ない場合、Email コラボレーションは自動的にインスタンス化および構成されます)

**注:**

1. CollaborationFoundation テンプレートによって設定される SEND_EMAIL 構成プロパティーにより、メッセージ番号ごとに、E メール通知を生成するメッセージを指定できます。詳細については、34 ページの『CollaborationFoundation テンプレート』を参照してください。

2. logError() メソッドにより、エラー・メッセージが E メール受信側に自動的に送信されます (Email コラボレーションと Email アダプターが実行していることを前提としています)。sendEmail() メソッドは、E メール・メッセージを明示的に送信できるようにします。

### 例

次の例では、getString() を使用してメッセージ内の属性値を取得することにより、エラー・メッセージを記録します。

```
logError("Incorrect customer: CustomerID: "
   + fromCustomerBusObj.getString("CustomerID"));
```

次の例では、コラボレーションのメッセージ・ファイルにテキストが格納されているエラー・メッセージをログに記録します。メッセージ・ファイルの 10 番のメッセージは、顧客のラストネーム (LName 属性) および顧客のファーストネーム (FName 属性) を示す 2 つのパラメーターを取得します。

```
logError(10, customer.get("LName"), customer.get("FName");
```

次の例では、パラメーターの配列を使用してエラー・メッセージを記録します。説明用に、この例ではパラメーターが 2 つのみの配列を使用します。この例では、カスタマー ID とカスタマー名の 2 つの要素を持つ配列 args が宣言されます。それから、12 番のメッセージおよび args 配列内の値を使用して、logError() メソッドはエラーを記録します。

```
Object[] args =
   {
      fromCustomerBusObj.getString("CustomerID"),
      fromCustomerBusObj.getString("CustomerName");
   }
logError(12, args);
```

---

## raiseException()

1 つ上の実行レベルに例外を生成するためにコラボレーション例外を準備します。

### 構文

```
void raiseException(String exceptionType, String message)
void raiseException(String exceptionType, int messageNum,
      String parameter[,...])
void raiseException (String exceptionType, int messageNum,
      Object[] paramArray)
void raiseException(CollaborationException exceptionObject)
```

### パラメーター

*exceptionType* 生成される例外についての例外タイプ。コラボレーション例外の原因を識別する、次の例外タイプ静的変数の 1 つとして、この例外タイプを指定します。

| | |
|---|---|
| AnyException | 任意のタイプの例外。 |
| AttributeException | 属性でアクセスする場合の問題です。例えば、String 属性に対してコラボレーションが getDouble() を |

| | |
|---|---|
| | 呼び出した場合や、存在しない属性に対して getString() を呼び出した場合です。 |
| JavaException | コラボレーション・ロジック内の Java コードの問題です。 |
| ObjectException | メソッドに渡されたビジネス・オブジェクトが無効でした。または、null オブジェクトにアクセスされました。 |
| OperationException | サービス呼び出しが適切に設定されませんでした。あるいは送信されませんでした。 |
| ServiceCallException | サービス呼び出しに失敗しました。例えば、コネクターやアプリケーションは使用できません。 |
| SystemException | InterChange Server システムの内部エラー。 |
| TransactionException | トランザクション・コラボレーションの、トランザクションの振る舞いに関連するエラーです。例えば、ロールバックに失敗しました。あるいは、コラボレーションが差し戻しを適用できませんでした。 |

*message* 例外メッセージを含むテキスト・ストリング。

*messageNum* コラボレーションのメッセージ・ファイル内のメッセージのメッセージ番号で、メッセージ番号で索引付けされます。メッセージ・テキスト・ファイルの設定方法の詳細は、197 ページの『第 9 章 メッセージ・ファイルの作成』を参照してください。

*parameter* 単一メッセージ・パラメーターの値。コンマで区切られた、最大 5 つのメッセージ・パラメーターが可能です。各パラメーターは順次的に解決され、メッセージ・テキスト内のパラメーターになります。

*paramArray* メッセージ・パラメーター値の配列。各パラメーターは順次的に解決され、メッセージ・テキスト内のパラメーターになります。

*exceptionObject* CollaborationException 例外オブジェクト変数の名前。

次の説明とコード例は、コラボレーション API 内で提供される必要があります。

## 注記

raiseException() メソッドは、1 つ上の実行レベルに例外を生成するためにコラボレーション例外を準備します。コラボレーション・ランタイム環境で raiseException() 呼び出しが実行されると、コラボレーションの実行が例外状態に

変更され、アクティビティー・ダイアグラムのロジックが続行されます。生成された例外に対するアクティビティー・ダイアグラムの応答方法は、実行経路の終端ノードによって以下のように異なります。

- 実行経路が終了成功で終わる場合、1 つ上の実行レベルに制御が渡されます。

  この親ダイアグラムの次ノードが決定ノードの場合、コラボレーション・ランタイム環境で、生成された例外を処理するこの決定ノード内の実行分岐が検査されます。この親ダイアグラムは、currentException システム変数を使用して生成された例外にアクセスできます。
- 実行経路が終了障害で終わる場合、コラボレーション・ランタイム環境によりコラボレーションが終了され、コラボレーションのログにエントリーが作成されて、未解決のフローが作成されます。

  コラボレーション・ランタイム環境により未解決のフローが、生成された例外が含む任意の例外テキストと関連付けされます。この例外に例外テキストが含まれていない場合、コラボレーション・ランタイム環境はデフォルトのメッセージを使用します。

  ```
  Scenario failed.
  ```

障害が発生したときには、単に失敗として終了させるのではなく、例外を明示的に生成させることを推奨します。コードによりコラボレーション・ランタイム環境に明示的に例外を生成すると、管理者は Flow Manager を使用して、未解決のフローの一部として例外テキストを表示できます。詳細については、177 ページの『例外の発生』を参照してください。

raiseException() メソッドにはいくつかの形式があります。

- *第 1* の形式では、指定された例外タイプおよびメッセージ・ストリングで新規の例外オブジェクトが作成されます。ストリングとして保管された例外メッセージを渡すには、この形式を使用します。このストリング・メッセージをログ・メソッドの 1 つに送信して、ログに記録することもできます。
- *第 2* の形式では、指定された例外タイプと、コラボレーションのメッセージ・ファイルから取得した例外メッセージを使用して新規の例外オブジェクトが作成されます。メッセージ・ファイル内のこのメッセージ番号で、メッセージを識別します。この形式のメソッド呼び出しでは、メッセージ・テキストについて最大 5 つのメッセージ・パラメーター値を渡すことができます。これらのメッセージ・パラメーターはコンマで区切ります。メッセージ・ファイル内のメッセージ・テキストでは、パラメーターは {1} などの中括弧内の番号で指定されます。raiseException() メソッドは、メッセージ内の各メッセージ・パラメーターの値を指定します。
- *第 3* の形式では、メッセージ・ファイル内に指定されたメッセージを含む新規の例外オブジェクトを作成する別の方法が提供されます。第 3 の形式での Objects のメッセージ・パラメーター配列は、このメソッドの第 2 の形式の String パラメーター・リストと同じように動作します。しかし、String リストの各メッセージ・パラメーター値は別々に指定されますが、この形式ではパラメーター値はすべて Objects の配列に配置されます。

  この形式は、次のような例外オブジェクトを生成するのに便利です。

– コラボレーションがすでに処理されている例外オブジェクト。例えば、1 つのシナリオが例外を取得し、それを変数に割り当ててから他の作業に移ります。
– メッセージ・パラメーターが 5 つを超える場合。String リストに格納できるパラメーターは 5 つまでですが、このパラメーターの配列には任意の数のパラメーターを格納できます。

- *第 4 の形式は、例外を生成しません。*それに代わり、引き数として指定される指定された例外オブジェクト (CollaborationException オブジェクト) を生成します。

**注:** *messageNum* パラメーターを取得するメソッドのすべての形式において、メッセージ番号で索引化されるメッセージ・ファイルを使用する必要があります。メッセージ・テキスト・ファイルの設定方法の詳細は、197 ページの『第 9 章 メッセージ・ファイルの作成』を参照してください。

## 例

このセクションでは、raiseException() メソッドの各形式の例を示します。

1. 次の例では、メソッドの第 1 の形式を使用して ServiceCallException タイプの例外を生成します。テキストはメソッドの呼び出しに直接渡されます。

```
raiseException(ServiceCallException, "Attempt to validate
Customer failed.");
```

2. 次の例では、メソッドの第 2 の形式を使用して OperationException タイプの例外を生成し、そのメッセージがメッセージ・ファイルに次のように表示されます。

```
23
Customer update failed for CustomerID={1} CustomerName={2}
```

この raiseException() 呼び出しは、メッセージ 23 を検索し、fromCustomer 変数からメッセージの 2 つのパラメーター (カスタマー ID と名前) の値を検索して例外メッセージを生成します。

```
raiseException(OperationException, 23,
   fromCustomer.getString("CustomerID"),
   fromCustomer.getString("CustomerName"));
```

3. 次の例は、第 3 の形式を使用してメッセージ・パラメーター値を Object の配列として送信します。

例えば、メッセージ・ファイルに次のメッセージ・テキストが含まれるとします。

```
2000
Collaboration Message: BOName: {1} with Verb: {2} encountered
an undefined error.
```

次のコードは Objects のパラメーターの配列を作成し、そこに値をロードして raiseException() メソッドを呼び出します。

```
Object[] myParamArray = new Object[2];

myParamArray[0] = triggeringBusObj.getType();
myParamArray[1] = triggeringBusObj.getVerb();

raiseException(AnyException, 2000, myParamArray);
```

4. 最後の例は、メソッドの第 4 の形式を使用して、以前に処理された例外を生成します。システム定義の currentException 変数は、例外を含む例外オブジェクトです。

```
raiseException(currentException);
```

## sendEmail()

E メール・メッセージを非同期に送信します。

### 構文

```
void sendEmail(String message, String subject, Vector recipients)
```

### パラメーター

*message* E メール・メッセージの本文。

*subject* E メール・メッセージの件名。

*recipients* メッセージ宛先の E メール・アドレスを含む Vector。この Vector には String オブジェクトが含まれており、複数の String オブジェクトの場合はコンマで区切られています。

### 注記

sendEmail() メソッドは、以下の場合に、*recipients* ベクトルに指定された宛先に E メール・メッセージを送信できます。

- E メール・アドレスは「Collaboration Object Properties」ダイアログの「E メール通知アドレス」フィールドで指定されている場合
- Email コラボレーションおよび Email アダプターが実行されている場合。(InterChange Server が始動するときにユーザーによる入力が必要ない場合、Email コラボレーションは自動的にインスタンス化および構成されます。) Email アダプターが実行されていなければ、sendEmail() メソッドがコラボレーションの実行を阻止することはありません。

**注:** logError() メソッドは E メール宛先に自動的にエラー・メッセージを送信します (Email コラボレーションおよび Email アダプターは実行中と想定します)。sendEmail() メソッドは、E メール・メッセージを明示的に送信できるようにします。

### 例

```
// Initialize the Vector for the list of email addresses
Vector emailList = new Vector();

// Add as many email addresses as Strings to the Vector
emailList.add("dbadmin@us.ibm.com, netadmin@us.ibm.com,
     cwadmin@us.ibm.com");

// Initialize the message and subject as Strings
String message = "This is the body of the email";
```

```
String subject = "This is the subject of the email";

// Make the call to sendEmail()
sendEmail(message, subject, emailList);
```

---

# trace()

トレース・メッセージをログ宛先に書き込みます。

## 構文

```
void trace(String traceMsg)
void trace(int traceLevel, String traceMsg)
void trace(int traceLevel, int messageNum)
void trace(int traceLevel, int messageNum, String param [,...])
void trace(int traceLevel, int messageNum, Object[] paramArray)
```

## パラメーター

*traceLevel* 出力するトレース・メッセージを決定するために使用されるトレース・レベル。このメソッドは、コラボレーション・オブジェクトのトレース・レベルがこの *traceLevel* 値以上の場合に、トレース・メッセージを書き込みます。このコラボレーションのトレース・レベルを定義し文書化して、管理者が使用するコラボレーション・オブジェクトのレベルを理解できるようにする必要があります。

*traceMsg* トレース・ファイルに書き込まれるトレース・メッセージ・テキスト。

*messageNum* コラボレーションのメッセージ・ファイル内のメッセージのメッセージ番号で、メッセージ番号で索引付けされます。メッセージ・テキスト・ファイルの設定方法の詳細は、197 ページの『第 9 章 メッセージ・ファイルの作成』を参照してください。

*param* 単一メッセージ・パラメーターの値。コンマで区切られた、最大 5 つのメッセージ・パラメーターが可能です。各パラメーターは順次的に解決され、メッセージ・テキスト内のパラメーターになります。

*paramArray* メッセージ・パラメーター値の配列。各パラメーターは順次的に解決され、メッセージ・テキスト内のパラメーターになります。

## 注記

trace() メソッドは、コラボレーション・ログ宛先にトレース・メッセージを送信します。デフォルトのログ宛先は、ファイル InterchangeSystem.log です。ユーザーは、InterChange Server 構成ファイル InterchangeSystem.cfg 内の LOG_FILE のパラメーター値を入力してログ宛先を変更することができます。パラメーター値がファイル名あるいは STDOUT になることがあります。STDOUT は、InterChange Server のコマンド・ウィンドウにログを記録します。

トレース・ロギングに関連するシステム構成パラメーターを、さらに 3 つ設定することもできます。すべてのパラメーターは構成ファイル InterchangeSystem.cfg 内に格納されます。

- MAX_TRACE_FILE_SIZE パラメーターを使用して、トレース・ファイルの最大サイズを設定します。デフォルトのファイル・サイズは無制限なので、常に最大サイズに設定する必要があります。
- NUMBER_OF_ARCHIVE_TRACES パラメーターを使用して、1 つから 5 つの間でアーカイブ・トレース・ファイルを設定します。パラメーターが設定されていない場合、デフォルトは 5 つです。
- エラー・メッセージをトレース・ファイルに書き込むのと同時に STDOUT に表示する場合は、MIRROR_TRACE_TO_STDOUT パラメーターを設定します。デフォルト値は false です。メッセージは STDOUT に同時に書き込まれません。

trace() メソッドにはいくつかの形式があります。

- メソッドの第 1 の形式は、ストリング・メッセージを 1 つのみ取得します。このメッセージはトレースがレベル 1 以上に設定されるときに表示されます。
- 第 2 の形式は、トレース・レベルおよびストリング・メッセージを取得します。このメッセージはトレースが指定されたレベル以上に設定されるときに表示されます。
- 第 3 の形式は、トレース・レベルおよび番号を取得します。この番号は、コラボレーションのメッセージ・ファイル内のメッセージを表します。メッセージ・テキスト全体はメッセージ・ファイルに表示されます。トレースが指定されたレベル以上に設定されるときメッセージ・ファイルは、パラメーターなしで表示どおりに出力されます。
- 第 4 の形式は、トレース・レベル、コラボレーションのメッセージ・ファイル内のメッセージを表す番号およびメッセージで使用される 1 つ以上のパラメーターを取得します。パラメーター値を最大 5 つまで送信できます。このパラメーター値を、コンマで区切ってメッセージに使用することができます。
- 第 5 の形式は、トレース・レベル、コラボレーションのメッセージ・ファイル内のメッセージを表す番号およびパラメーター値の配列を取得します。

コラボレーション・オブジェクトを構成して、システムまたはコラボレーションでトレースを生成することができます。コラボレーション生成トレースを出力するようにコラボレーションが構成されている場合、trace() メソッドは、コラボレーション・オブジェクトが出力するメッセージを生成します。トレースを使用する条件の決定についてのヘルプは、205 ページの『トレース・メッセージの追加』を参照してください。

## 例

次の例では、メソッドの第 2 の形式を使用して、指定されたメッセージ・テキストでレベル 2 トレース・メッセージを生成します。

```
trace (2, "Starting to trace at Level 2");
```

次の例では、コラボレーション・オブジェクトのトレース・レベルが 2 以上の場合に、メソッドの第 4 の形式を使用して、コラボレーションのメッセージ・ファイル内にメッセージ 201 を書き込みます。メッセージには、名前と年の 2 つのパラメーターがあります。このメソッドの呼び出しは、これらのパラメーターの値を渡します。

```
trace(2, 201, "DAVID", "1961");
```

# 第 23 章 BusObj クラス

この章では、BusObj クラスのオブジェクトに対して操作を行うメソッドについて説明します。これらのオブジェクトは、InterChange Server Express ビジネス・オブジェクトを表します。

**注:** BusObj クラスは、コラボレーション開発とマッピングの両方で使用されます。各メソッドの『注記』セクションには、メソッドの使用に関する注意事項が示してあります。

表 75 に、BusObj クラスのメソッドをリストします。

*表 75. BusObj メソッドの要約*

| メソッド | 説明 | ページ |
|---|---|---|
| copy() | 入力ビジネス・オブジェクトから、すべての属性値をビジネス・オブジェクトにコピーします。 | 382 |
| duplicate() | ビジネス・オブジェクト (BusObj オブジェクト) とまったく同じビジネス・オブジェクトを作成します。 | 383 |
| equalKeys() | ビジネス・オブジェクトのキー属性値を、入力ビジネス・オブジェクトのキー属性値と比較します。 | 383 |
| equals() | ビジネス・オブジェクトの属性値を、入力ビジネス・オブジェクトの属性値と比較します。ただし、子ビジネス・オブジェクトは比較に含まれます。 | 384 |
| equalsShallow() | ビジネス・オブジェクトの属性値を、入力ビジネス・オブジェクトの属性値と比較します。ただし、子ビジネス・オブジェクトは比較から除外されます。 | 385 |
| exists() | 指定した名前のビジネス・オブジェクト属性が存在するかどうかを検査します。 | 386 |
| getBoolean()、getDouble()、getFloat()、getInt()、getLong()、get()、getBusObj()、getBusObjArray()、getLongText()、getString() | ビジネス・オブジェクトから単一属性の値を検索します。 | 386 |
| getLocale() | ビジネス・オブジェクトのデータのロケールを検索します。 | 388 |
| getType() | ビジネス・オブジェクトのベースとなるビジネス・オブジェクト定義の名前を検索します。 | 389 |
| getVerb() | ビジネス・オブジェクトの動詞を検索します。 | 389 |
| isBlank() | 属性の値が、ゼロ長ストリングに設定されているかどうかを確認します。 | 390 |
| isKey() | ビジネス・オブジェクトの属性が、キー属性として定義されているかどうかを確認します。 | 391 |
| isNull() | ビジネス・オブジェクトの属性値が null かどうかを確認します。 | 391 |
| isRequired() | ビジネス・オブジェクトの属性が、必須属性として定義されているかどうかを確認します。 | 392 |

表 75. *BusObj* メソッドの要約 (続き)

| メソッド | 説明 | ページ |
|---|---|---|
| keysToString() | ビジネス・オブジェクトの基本キー属性の値をストリングとして検索します。 | 393 |
| set() | ビジネス・オブジェクトの属性を、特定のデータ型の指定値に設定します。 | 393 |
| setDefaultAttrValues() | すべての属性をそれぞれのデフォルト値に設定します。 | 395 |
| setKeys() | ビジネス・オブジェクトのキー属性の値を、別のビジネス・オブジェクトのキー属性の値に設定します。 | 395 |
| setVerb() | ビジネス・オブジェクトの動詞を設定します。 | 396 |
| setWithCreate() | ビジネス・オブジェクトの属性を、指定データ型の値に設定します。 | 397 |
| toString() | ビジネス・オブジェクトのすべての属性の値をストリングとして戻します。 | 398 |
| validData() | 指定した値が、指定した属性にとっての有効なデータ型であるかどうかを検査します。 | 398 |

# copy()

入力ビジネス・オブジェクトから、すべての属性値をビジネス・オブジェクトにコピーします。

## 構文

```
void copy(BusObj inputBusObj)
```

## パラメーター

*inputBusObj* 属性値が現在のビジネス・オブジェクトにコピーされるビジネス・オブジェクトの名前。

## 注記

copy() メソッドは、すべての子ビジネス・オブジェクトおよび子ビジネス・オブジェクト配列を含むビジネス・オブジェクト全体をコピーします。このメソッドは、コピーされたオブジェクトへの参照は設定しません。代わりに、すべての属性を複製します。つまり、属性の別のコピーを作成します。

## 例

次の例では、sourceCustomer に含まれている値を destCustomer にコピーします。

```
destCustomer.copy(sourceCustomer);
```

次の例では、3 つのビジネス・オブジェクト (myBusObj、myBusObj2、および mysettingBusObj) を作成して、myBusObj の attr1 属性を mysettingBusObj 内の値に設定します。次に、myBusObj のすべての属性を myBusObj2 に複製します。

```
BusObj myBusObj = new BusObj();
BusObj myBusObj2 = new BusObj();
```

```
BusObj mySettingBusObj = new BusObj();

myBusObj.set("attr1", mySettingBusObj);
myBusObj2.copy(myBusObj);
```

このコード・フラグメントの実行後に、myBusObj.attr1 および myBusObj2.attr1 は、いずれも mySettingBusObj ビジネス・オブジェクトに対して設定されます。ただし、mySettingBusObj がなんらかの方法で変更された場合は、myBusObj.attr1 は変更されますが、myBusObj2.attr1 は変更されません。myBusObj2 の属性が copy() を使用して設定されているため、属性の値は複製されます。このため、myBusObj2 内の attr1 は、変更*前* のオリジナルの mySettingBusObj.attr1 値のままになります。

## duplicate()

ビジネス・オブジェクト (BusObj オブジェクト) とまったく同じビジネス・オブジェクトを作成します。

### 構文

```
BusObj duplicate()
```

### 戻り値

複製されたビジネス・オブジェクト。

### 例外

CollaborationException — duplicate() メソッドは、この例外に対して、ObjectException という例外タイプを設定できます。

### 注記

このメソッドはビジネス・オブジェクトの複製を作成し、それを戻します。このメソッド呼び出しの戻り値は、BusObj タイプの宣言された変数に明示的に割り当てる必要があります。

### 例

次の例では、sourceCustomer を複製して、destCustomer を作成します。

```
BusObj destCustomer = sourceCustomer.duplicate();
```

## equalKeys()

ビジネス・オブジェクトのキー属性値を、入力ビジネス・オブジェクトのキー属性値と比較します。

### 構文

```
boolean equalKeys(BusObjinputBusObj)
```

### パラメーター

*inputBusObj*　ビジネス・オブジェクトと比較するビジネス・オブジェクト。

### 戻り値

すべてのキー属性の値が同じ場合は `true` を戻し、同じでない場合は `false` を戻します。

### 例外

`CollaborationException` — `equalKeys()` メソッドは、この例外に対して次の例外タイプを設定できます。

- `ObjectException` - ビジネス・オブジェクトの引き数が無効の場合に設定します。

### 関連項目

`equals()`、`equalsShallow()`

### 注記

このメソッドは浅く比較を行います。つまり子ビジネス・オブジェクトのキーは比較されません。

### 例

次の例では、`order2` のキー値を `order1` のキー値と比較します。

```
boolean areEqual = order1.equalKeys(order2);
```

---

## equals()

ビジネス・オブジェクトの属性値を、入力ビジネス・オブジェクトの属性値と比較します。ただし、子ビジネス・オブジェクトは比較に含まれます。

### 構文

```
-boolean equals(Object inputBusObj)
```

### パラメーター

*inputBusObj*　ビジネス・オブジェクトと比較するビジネス・オブジェクト。

### 戻り値

すべての属性の値が同じ場合は `true` を戻し、同じでない場合は `false` を戻します。

### 例外

`CollaborationException` — `equals()` メソッドは、この例外に対して次の例外タイプを設定できます。

- ObjectException — ビジネス・オブジェクトの引き数が無効の場合に設定します。

## 注記

このメソッドは、ビジネス・オブジェクトの属性値を入力ビジネス・オブジェクトの属性値と比較します。ビジネス・オブジェクトが階層的である場合は、子ビジネス・オブジェクトの属性もすべて比較されます。

**注:** ビジネス・オブジェクトを Object として渡すと、この equals() メソッドにより Object.equals() メソッドがオーバーライドされます。

比較時に、null 値は比較対象の任意の値と等価であるとみなされるため、null 値に起因して true が戻されないということはありません。

## 関連項目

equalKeys()、equalsShallow()

## 例

次の例では、order2 のすべての属性を order1 のすべての属性と比較し、比較結果を areEqual 変数に割り当てます。子ビジネス・オブジェクトが存在する場合は、その属性も比較されます。

```
boolean areEqual = order1.equals(order2);
```

---

# equalsShallow()

ビジネス・オブジェクトの属性値を、入力ビジネス・オブジェクトの属性値と比較します。ただし、子ビジネス・オブジェクトは比較から除外されます。

## 構文

```
boolean equalsShallow(BusObj inputBusObj)
```

## パラメーター

*inputBusObj*　　ビジネス・オブジェクトと比較するビジネス・オブジェクト。

## 戻り値

すべての属性の値が同じ場合は true を戻し、同じでない場合は false を戻します。

## 例外

CollaborationException — equalsShallow() メソッドは、この例外に対して次の例外タイプを設定できます。

- ObjectException - ビジネス・オブジェクトの引き数が無効の場合に設定します。

### 関連項目

equalKeys()、equals()

### 例

次の例では、order2 の属性を order1 の属性と比較します。ただし子ビジネス・オブジェクトが存在する場合、その属性は比較から除外されます。

```
boolean areEqual = order1.equalsShallow(order2);
```

---

## exists()

指定した名前のビジネス・オブジェクト属性が存在するかどうかを検査します。

### 構文

```
boolean exists(String attribute)
```

### パラメーター

*attribute*　　属性の名前。

### 戻り値

属性が存在する場合は true を戻し、存在しない場合は false を戻します。

### 例

次の例では、order というビジネス・オブジェクトの属性が、Notes という属性を持っているかどうかを検査します。

```
boolean notesAreHere = order.exists("Notes");
```

---

## getBoolean()、getDouble()、getFloat()、getInt()、getLong()、get()、getBusObj()、getBusObjArray()、getLongText()、getString()

ビジネス・オブジェクトから単一属性の値を検索します。

### 構文

```
Object get(String attribute)
Object get(int position)

boolean getBoolean(String attribute)
double getDouble(String attribute)
float getFloat(String attribute)
int getInt(String attribute)
long getLong(String attribute)

Object get(String attribute)
BusObj getBusObj(String attribute)
BusObjArray getBusObjArray(String attribute)
String getLongText(String attribute)
String getString(String attribute)
```

## パラメーター

*attribute* 属性の名前。

*position* ビジネス・オブジェクトの属性リストの属性の順序を指定する整数値。

## 戻り値

指定された属性の値。

## 例外

`CollaborationException` — これらの `get` メソッドは、この例外に対して次の例外タイプを設定できます。

- `AttributeException` — 属性でアクセスする際に問題が生じる場合に設定します。例えば、数字で構成されていない String 属性に対してコラボレーションが `getDouble()` を呼び出した場合や、存在しない属性に対して `getString()` を呼び出した場合に、この例外を発生させることができます。

## 注記

「get」メソッドは、現在のビジネス・オブジェクトの属性値を検索し、属性値のコピーを戻します。これらのメソッドは、ソース・ビジネス・オブジェクトのこの属性に対するオブジェクト参照を*戻しません*。このため、ソース・ビジネス・オブジェクトの属性値の変更内容は、特定の get メソッドによって戻される値には適用*されません*。これらのメソッドは、呼び出されるたびに属性の新しいコピー (複製) を戻します。

get メソッドにはいくつかの形式があります。

- 1 番目の形式は、メソッド名に指定されているデータ型の値を戻します。例えば、getBoolean() は boolean 値を戻し、getBusObj() は BusObj 値を戻し、getDouble() は double 値を戻すなどです。ただし、getLongText() は String オブジェクトを戻します。これは、InterChange Server Express の longtext 型が、サイズに上限のない String オブジェクトだからです。特定の基本データ型、または InterChange Server Express 定義のデータ型の属性を検索するには、この形式を使用してください。

  これらのメソッドでは、属性の*名前* を指定することで属性の値へアクセスできます。
- 2 番目の形式 get() は、*あらゆる* データ型の属性値を検索します。戻り値を、属性のデータ型の適切な値へキャストできます。

  このメソッドでは、属性の*名前* またはビジネス・オブジェクトの属性リストにおける属性索引の*位置* *のいずれか* を指定することで、属性の値へアクセスできます。

## 例

次の例では、`get()` がオブジェクト参照の代わりに属性値のコピー (複製) を戻すメカニズムを示しています。

```
BusObj mySettingBusObj = new BusObj();
BusObj myBusObj = new BusObj();

myBusObj.set("attr1", mySettingBusObj);

BusObj Extract = myBusObj.get("attr1");
```

このコード・フラグメントの実行後に Extract ビジネス・オブジェクトを変更しても、mySettingBusObj は変更されません。これは、get() 呼び出しにより、attr1 属性のコピーが戻されるからです。

次の例では、getBusObj() を使用して、customer ビジネス・オブジェクトから顧客の住所を含む子ビジネス・オブジェクトを検索して address 変数に割り当てます。

```
BusObj address = customer.getBusObj("Address");
```

次の例では、getString() を使用して CustomerName 属性の値を検索します。ビジネス・オブジェクト変数は、sourceCustomer です。

```
String customerName = sourceCustomer.getString("CustomerName");
```

次の例では、getInt() を使用して、item1 および item2 変数を持つ 2 つのビジネス・オブジェクトから Quantity 値を検索します。次に両方の数量の合計を計算します。

```
int sumQuantity = item1.getInt("Quantity") + item2.getInt("Quantity");
```

次の例では、order ビジネス・オブジェクト変数から Item 属性を検索します。Item 属性は、ビジネス・オブジェクト配列です。

```
BusObjArray items = order.getBusObjArray("Item");
```

次の例では、ソース・ビジネス・オブジェクトから CustID 属性値を取得し、これに一致するように宛先ビジネス・オブジェクトの Customer の値を設定します。

```
destination.set("Customer", source.get("CustID"));
```

属性リスト内での属性の順序を使用して属性値へアクセスする例を以下に示します。

```
for(i=0; i<maxAttrCount; i++)
{
     String strValue = (String)myBusObj.get(i);
     ...
```

## getLocale()

ビジネス・オブジェクトのデータに関連付けられているロケールを検索します。

### 構文

```
java.util.Locale getLocale()
```

### パラメーター

なし。

### 戻り値

ビジネス・オブジェクトのロケールに関する情報が含まれている Java Locale オブジェクト。この Locale オブジェクトは、java.util.Locale クラスのインスタンスでなければなりません。

### 注記

getLocale() メソッドは、ビジネス・オブジェクトのデータに関連付けられているロケールを戻します。このロケールは、コラボレーション実行時に使用されるコラボレーション・ロケールとは異なることがよくあります。

### 関連項目

getLocale() (BaseCollaboration クラス)、setLocale()

---

## getType()

ビジネス・オブジェクトのベースとなるビジネス・オブジェクト定義の名前を検索します。

### 構文

```
String getType()
```

### 戻り値

ビジネス・オブジェクト定義の名前。

### 注記

このメソッドでは、ビジネス・オブジェクトのタイプは、ビジネス・オブジェクトの作成元であるビジネス・オブジェクト定義の名前です。

### 例

次の例では、sourceShipTo という名前のビジネス・オブジェクトのタイプを検索します。

```
String typeName = sourceShipTo.getType();
```

次の例では、トリガー・イベントを適切なタイプの新規ビジネス・オブジェクトにコピーします。

```
BusObj source = new BusObj(triggeringBusObj.getType());
```

---

## getVerb()

ビジネス・オブジェクトの動詞を検索します。

### 構文

```
String getVerb()
```

## 戻り値

動詞の名前。例えば、Create、Retrieve、Update、または Delete です。

## 注記

このメソッドは、複数のタイプの受信イベントを扱うシナリオの場合に役立ちます。シナリオの最初のアクション・ノードが getVerb() を呼び出します。次にそのアクション・ノードからの出力遷移リンクで、戻されたストリングの内容をテストし、それぞれの出力遷移リンクが、使用可能な動詞の 1 つを処理する実行経路の先頭になるようにします。

## 例

次の例では、orderEvent というビジネス・オブジェクトから動詞を取得し、それを orderVerb という変数に割り当てます。

```
String orderVerb = orderEvent.getVerb();
```

---

# isBlank()

属性の値が、ゼロ長ストリングに設定されているかどうかを確認します。

## 構文

```
boolean isBlank(String attribute)
```

## パラメーター

*attribute* 属性の名前。

## 戻り値

属性値がゼロ長ストリングの場合は true を戻し、ゼロ長ストリングでない場合は false を戻します。

## 注記

ゼロ長ストリングは、ストリング "" と同等です。ゼロ長ストリングは null とは異なります。null の存在は、isNull() メソッドで検出されます。

コラボレーションが属性値を検索してからその値に対してなんらかの操作を行う必要がある場合、値を検索する前に isBlank() と isNull() を呼び出すことで、その属性が値を持っているかどうかを検査できます。

## 例

次の例では、sourcePaperClip ビジネス・オブジェクトの Material 属性がゼロ長ストリングであるかどうかを検査します。

```
boolean key = sourcePaperClip.isBlank("Material");
```

## isKey()

ビジネス・オブジェクトの属性が、キー属性として定義されているかどうかを確認します。

### 構文

```
boolean isKey(String attribute)
```

### パラメーター

*attribute* 属性の名前。

### 戻り値

属性がキー属性である場合は true を戻し、キー属性でない場合は false を戻します。

### 例

次の例では、customer ビジネス・オブジェクトの CustID 属性がキー属性であるかどうかを判別します。

```
boolean keyAttr = (customer.isKey("CustID"));
```

## isNull()

ビジネス・オブジェクトの属性値が null かどうかを確認します。

### 構文

```
boolean isNull(String attribute)
```

### パラメーター

*attribute* 属性の名前。

### 戻り値

属性値が null の場合は true を戻し、null でない場合は false を戻します。

### 注記

null は、ゼロ長ストリングと異なり、値がないことを示します。ゼロ長ストリングは isBlank() を呼び出すことによって検出されます。オブジェクトが null であると操作が失敗することがあるため、オブジェクトは、使用する前に isNull() によってテストしてください。

以下の状況では、属性値が null であることがあります。

- 属性値が明示的に null に設定された。

  属性値は、set() メソッドを使用して null に設定できます。
- 属性値が設定されていない。

コラボレーションが new() メソッドを使用して新規ビジネス・オブジェクトを作成すると、すべての属性値は null に初期化されます。ビジネス・オブジェクトの作成後に isNull() を呼び出すまでに属性値が設定されないと、値は null のままです。

- マッピング中に null が挿入された。

  コネクターから受け取ったビジネス・オブジェクトをコラボレーションが処理するときに、マッピング処理で null 値が挿入されることがあります。マッピング処理では、コネクターから受け取ったアプリケーション固有のビジネス・オブジェクトは、コラボレーションによって処理される汎用ビジネス・オブジェクトに変換されます。汎用ビジネス・オブジェクトの属性のうち、アプリケーション固有のオブジェクトにおいて等価な属性のない属性については、それぞれマッピングによって null 値が挿入されます。

**ヒント:** Java では null オブジェクトに対して操作を行うことができないため、子ビジネス・オブジェクトまたは子ビジネス・オブジェクト配列である属性に対して操作を実行する前に、必ず isNull() を呼び出してください。

### 例

次の例では、sourcePaperClip ビジネス・オブジェクトの Material 属性に null 値があるかどうかを検査します。

```
boolean key = sourcePaperClip.isNull("Material");
```

次の例では、contract1 ビジネス・オブジェクトの CustAddr 属性が、検索の前の時点で null であるかどうかを検査します。isNull() 検査が false である、つまり属性が null でない場合にのみ属性の検索が行われます。

```
if (! contract1.isNull("CustAddr"))
   {
   BusObj customerAddress = contract1.getBusObj("CustAddr");
   // do something with the "customerAddress" business object
   }
```

## isRequired()

ビジネス・オブジェクトの属性が、必須属性として定義されているかどうかを確認します。

### 構文

```
boolean isRequired(String attribute)
```

### パラメーター

*attribute* 属性の名前。

### 戻り値

属性が必須の場合は true を戻し、必須でない場合は false を戻します。

### 注記

属性が必須として定義されている場合、その属性は null でない値を持つ必要があります。

### 例

次の例では、必須属性の値が null である場合に警告をログに記録します。

```
if ( (customer.isRequired("Address"))
      && (customerBusObj.isNull("Address")) )
   {
   logWarning(12, "Address is required and cannot be null.");
   }
else
   {
   // do something else
   }
```

## keysToString()

ビジネス・オブジェクトの基本キー属性の値をストリングとして検索します。

### 構文

```
String keysToString()
```

### 戻り値

ビジネス・オブジェクト内のすべてのキー値が、属性の序数値の順序に並べられて連結されたストリング・オブジェクト。

### 注記

このメソッドの出力には、属性の名前と値が含まれます。複数値は、スペースで区切られて連結された基本キー属性値です。例えば、1 つの基本キー属性 SS# が存在する場合、出力は次のようになります。

```
SS#=100408394
```

基本キー属性が FirstName および LastName の場合の出力は、以下のようになります。

```
FirstName=Nina LastName=Silk
```

### 例

次の例では、変数名 fromOrder によって表されるビジネス・オブジェクトのキー属性の値を戻します。

```
String keyValues = fromOrder.keysToString();
```

## set()

ビジネス・オブジェクトの属性を、特定のデータ型の指定値に設定します。

## 構文

```
void set(String attribute, Object value)
void set(int position, Object value)

void set(String attribute, boolean value)
void set(String attribute, double value)
void set(String attribute, float value)
void set(String attribute, int value)
void set(String attribute, long value)
void set(String attribute, Object value)
void set(String attribute, String value)
```

## パラメーター

*attribute* 設定する属性の名前。

*position* ビジネス・オブジェクトの属性リストの属性の順序を指定する整数値。

*value* 属性値。

## 例外

`CollaborationException` — `set()` メソッドは、この例外に対して次の例外タイプを設定できます。

- `AttributeException` — 属性でアクセスする際に問題が生じる場合に設定します。

## 注記

set() メソッドは、現在のビジネス・オブジェクトの属性値を設定します。これらのメソッドは、属性へ値を割り当てるときにオブジェクト参照を設定します。これにより、ソース・ビジネス・オブジェクトから属性値が複製されることはありません。このため、ソース・ビジネス・オブジェクトの*値* の変更内容は、set() を呼び出すビジネス・オブジェクトの属性にも適用されます。

set() メソッドにはいくつかの形式があります。

- 1 番目の形式は、メソッドの 2 番目のパラメーターのデータ型として指定されているデータ型の値を設定します。例えば set(String *attribute*, boolean *value*) はブール値の属性を設定し、set(String *attribute*, double *value*) は double 値の属性を設定します。特定の基本データ型または InterChange Server Express 定義のデータ型の属性を設定する場合は、この形式を使用してください。

  これらのメソッドでは、属性の*名前* を指定することで属性の値へアクセスできます。
- 2 番目の形式は、*あらゆる* データ型の属性値を設定します。属性値パラメーターのデータ型が Object であるため、属性値としてあらゆるデータ型を設定できます。例えば `BusObj` または `LongText` オブジェクトの属性を設定するには、この形式のメソッドを使用して `BusObj` または `LongText` オブジェクトを属性値として渡します。

  この形式の set () メソッドでは、属性の*名前* またはビジネス・オブジェクトの属性リストにおける属性索引の*位置* *のいずれか* を指定することで属性の値へアクセスできます。

### 例

次の例では、toCustomer の LName 属性を、Smith という値に設定します。

```
toCustomer.set("LName", "Smith");
```

次の例では、set() が値を複製するのではなくオブジェクト参照を割り当てるメカニズムを示します。

```
BusObj myBusObj = new BusObj();
BusObj mySettingBusObj = new BusObj();
myBusObj.set("attr1", mySettingBusObj);
```

このコード・フラグメントの実行後に、myBusObj の attr1 属性は、mySettingBusObj ビジネス・オブジェクトに対して設定されます。mySettingBusObj がなんらかの方法で変更されると、myBusObj.attr1 は、まったく同じように変更されます。これは、set() が、attr1 属性を設定するときに mySettingBusObj に対してオブジェクト参照を行い、mySettingBusObj の静的コピーを作成しない ためです。

属性リスト内での属性の順序を使用して属性値を設定する例を以下に示します。

```
for(i=0; i<maxAttrCount; i++)
{
     myBusObj.set(i, strValue);
     ...
```

## setDefaultAttrValues()

すべての属性をそれぞれのデフォルト値に設定します。

### 構文

```
void setDefaultAttrValues()
```

### 注記

ビジネス・オブジェクト定義には、属性のデフォルト値を含めることができます。このメソッドは、ビジネス・オブジェクトの属性値を、定義の中でデフォルトとして指定されている値に設定します。

### 例

次の例では、PaperClip ビジネス・オブジェクトの値をそのデフォルト値に設定します。

```
PaperClip.setDefaultAttrValues();
```

## setKeys()

ビジネス・オブジェクトのキー属性の値を、別のビジネス・オブジェクトのキー属性の値に設定します。

### 構文

```
void setKeys(BusObj inputBusObj)
```

### パラメーター

*inputBusObj* 別のビジネス・オブジェクトの値を設定するのに値が使用されるビジネス・オブジェクト。

### 例外

CollaborationException — setKeys() メソッドは、この例外に対して次の例外タイプのいずれかを設定できます。

- AttributeException - 属性でアクセスする際に問題が生じる場合に設定します。
- ObjectException - ビジネス・オブジェクトの引き数が無効の場合に設定します。

### 例

次の例では、ビジネス・オブジェクト helpdeskCustomer のキー値をビジネス・オブジェクト ERPCustomer のキー値に設定します。

```
helpdeskCustomer.setKeys(ERPCustomer);
```

## setLocale()

現在のビジネス・オブジェクトのロケールを設定します。

### 構文

```
void setLocale(java.util.Locale locale)
```

### パラメーター

*locale* ビジネス・オブジェクトへ割り当てるロケールに関する情報が含まれている Java Locale オブジェクト。この Locale オブジェクトは、java.util.Locale クラスのインスタンスでなければなりません。

### 戻り値

なし。

### 注記

setLocale() メソッドは、ビジネス・オブジェクトに関連するデータにロケールを割り当てます。このロケールは、コラボレーション実行時に使用されるコラボレーション・ロケールとは異なることがあります。

### 関連項目

getLocale()

## setVerb()

ビジネス・オブジェクトの動詞を設定します。

## 構文

```
void setVerb(String verb)
```

## パラメーター

*verb* ビジネス・オブジェクトの動詞。

## 注記

このメソッドは、ビジネス・オブジェクトのマッピングで使用されます。

このメソッドをコラボレーション・テンプレートで使用しないでください。コラボレーション・テンプレートでは、サービス呼び出しのプロパティーを入力することにより、出力ビジネス・オブジェクトの動詞を対話式に設定する必要があります。

## 例

次の例では、ビジネス・オブジェクト contactAddress に動詞 Delete を設定します。

```
contactAddress.setVerb("Delete");
```

---

# setWithCreate()

ビジネス・オブジェクトの属性を、指定データ型の値に設定します。

## 構文

```
void setWithCreate(String attributeName, BusObj busObj)
void setWithCreate(String attributeName, BusObjArray busObjArray)
void setWithCreate(String attributeName, Object value)
```

## パラメーター

*attributeName* 設定する属性の名前。

*busObj* 宛先属性に挿入するビジネス・オブジェクト。

*busObjArray* 宛先属性に挿入するビジネス・オブジェクト配列。

*value* 宛先属性に挿入するオブジェクト。このオブジェクトは、BusObj、BusObjArray、Object タイプのいずれかである必要があります。

## 例外

CollaborationException — setWithCreate() メソッドは、この例外に対して次の例外タイプを設定できます。

- AttributeException — 属性でアクセスする際に問題が生じる場合に設定します。

## 注記

指定するオブジェクトが BusObj であり、ターゲットの属性に複数カーディナリティーの子ビジネス・オブジェクトが含まれる場合、BusObj は BusObjArray に最後

の要素として追加されます。ただし、ターゲットの属性に BusObj が含まれている場合は、このビジネス・オブジェクトが元の値に置き換わります。

### 例

次の例では、ChildAttrAttr という属性を値 5 に設定します。この属性は、myBO の属性ChildAttr に含まれるビジネス・オブジェクトの中にあります。呼び出し時に childAttr ビジネス・オブジェクトが存在しないと、このメソッド呼び出しによってこのビジネス・オブジェクトが作成されます。

```
myBO.setWithCreate("childAttr.childAttrAttr", "5");
```

## toString()

ビジネス・オブジェクトのすべての属性の値をストリングとして戻します。

### 構文

```
String toString()
```

### 戻り値

ビジネス・オブジェクトに含まれるすべての属性値を含む String オブジェクト。

### 注記

このメソッドを呼び出した結果のストリングは、例えば以下のようになります。

```
Name: GenEmployee
Verb: Create
Type: AfterImage
Attributes: (Name, Type, Value)

LastName:String, Davis
FirstName:String, Miles
SS#:String, 041-33-8989
Salary:Float, 15.00
ObjectEventId:String, MyConnector_922323619411_1
```

### 例

次の例では、ビジネス・オブジェクト変数 fromOrder の属性値を含むストリングを戻します。

```
String values = fromOrder.toString();
```

## validData()

指定した値が、指定した属性にとっての有効なデータ型であるかどうかを検査します。

### 構文

```
boolean validData(String attributeName, Object value)
boolean validData(String attributeName, BusObj value)
boolean validData(String attributeName, BusObjArray value)
boolean validData(String attributeName, String value)
boolean validData(String attributeName, long value)
boolean validData(String attributeName, int value)
```

```
boolean validData(String attributeName, double value)
boolean validData(String attributeName, float value)
boolean validData(String attributeName, boolean value)
```

## パラメーター

*attributeName* 属性。

*value* 値。

## 戻り値

true または false (boolean 戻り値)。

## 注記

ターゲットの属性 (*attributeName* で指定される) によって渡される値の互換性を検査します。基準は以下のとおりです。

| | |
|---|---|
| 基本型 (string、long、int、double、float、boolean) の場合 | 値は属性のデータ型に変換可能である必要があります。 |
| `BusObj` の場合 | 値のデータ型はターゲットの属性値のデータ型と同じである必要があります。 |
| `BusObjArray` の場合 | 値は、属性のタイプと同じ (ビジネス・オブジェクト定義) タイプである `BusObj` または `BusObjArray` を指している必要があります。 |
| `Object` の場合 | 値は、`String`、`BusObj`、または `BusObjArray` である必要があります。対応する検証規則が適用されます。 |

# 推奨されないメソッド

`BusObj` クラスのメソッドには、以前のバージョンではサポートされていたが現在のバージョンではサポートされていないものがいくつかあります。これらの**推奨されないメソッド**を使用しても、エラーは発生しません。しかし、これらのメソッドは使用しないで、既存のコードを新しいメソッドに移行することを推奨します。推奨されないメソッドは、今後のリリースで除外されることがあります。

表 76 に、deprecated `BusObj` クラスの推奨されないメソッドおよび代替メソッドをリストします。過去 Process Designer Express を使用したことがない場合は、このセクションは無視してください。

*表 76. BusObj クラスの推奨されないメソッド*

| 以前のメソッド | 代替メソッド |
|---|---|
| `getCount()` | `BusObjArray.size()` |
| `getKeys()` | `keysToString()` |
| `getValues()` | `toString()` |
| `not` | 標準 Java NOT 演算子、「!」 |
| `set(BusObj inputBusObj)` | `copy()` |

*表 76. BusObj クラスの推奨されないメソッド (続き)*

| 以前のメソッド | 代替メソッド |
| --- | --- |
| 入力引き数として子ビジネス・オブジェクトまたは子ビジネス・オブジェクト配列をとるメソッドすべて | 子ビジネス・オブジェクトまたはビジネス・オブジェクト配列へのハンドルを取得し、BusObj または BusObjArray クラスのメソッドを使用してください。 |

setVerb() メソッドは、以前のバージョンでは推奨されないメソッドのリストに入っていましたが、現在ではマッピングでの使用のためリストからはずされています。このメソッドはコラボレーション内では使用しないでください。

# 第 24 章 BusObjArray クラス

この章では、BusObjArray クラスのオブジェクトに対して操作を行うメソッドについて説明します。これらのメソッドは、InterChange Server Express 定義の BusObjArray クラスで定義されています。BusObjArray クラスは、ビジネス・オブジェクトの配列をカプセル化します。階層ビジネス・オブジェクトでは、属性のカーディナリティーが n に等しいときに、属性は子ビジネス・オブジェクトの配列への参照になります。BusObjArray クラスに対して操作を実行すると、BusObjArray オブジェクトまたはビジネス・オブジェクトの実配列のいずれかが戻されることがあります。

表 77 に、この章で説明するメソッドをリストします。

*表 77. BusObjArray メソッドの要約*

| メソッド | 説明 | ページ |
|---|---|---|
| addElement() | ビジネス・オブジェクトをビジネス・オブジェクト配列に追加します。 | 402 |
| duplicate() | ビジネス・オブジェクト配列 (BusObjArray オブジェクト) とまったく同じビジネス・オブジェクト配列を作成します。 | 403 |
| elementAt() | ビジネス・オブジェクト配列内に位置を指定して単一ビジネス・オブジェクトを検索します。 | 403 |
| equals() | ビジネス・オブジェクト配列を別のビジネス・オブジェクト配列と比較します。 | 403 |
| getElements() | ビジネス・オブジェクト配列の内容を検索します。 | 404 |
| getLastIndex() | ビジネス・オブジェクト配列から、最後の有効な指標を検索します。 | 404 |
| max() | ビジネス・オブジェクト配列のすべての要素のうち、指定した属性の最大値を検索します。 | 405 |
| maxBusObjArray() | 指定した属性の最大値を持つビジネス・オブジェクトをビジネス・オブジェクト配列 (BusObjArray オブジェクト) として戻します。 | 406 |
| maxBusObjs() | 指定した属性の最大値を持つビジネス・オブジェクトを BusObj オブジェクトの配列として戻します。 | 407 |
| min() | 配列の中のビジネス・オブジェクトのうち、指定した属性の最小値を検索します。 | 408 |
| minBusObjArray() | 指定した属性の最小値を持つビジネス・オブジェクトを BusObjArray オブジェクトとして戻します。 | 409 |
| minBusObjs() | 指定した属性の最小値を持つビジネス・オブジェクトを BusObj オブジェクトの配列として戻します。 | 410 |

表 77. *BusObjArray* メソッドの要約 (続き)

| メソッド | 説明 | ページ |
|---|---|---|
| removeAllElements() | ビジネス・オブジェクト配列からすべての要素を除去します。 | 411 |
| removeElement() | ビジネス・オブジェクト配列から 1 つの要素を削除します。 | 411 |
| removeElementAt() | ビジネス・オブジェクト配列内の特定の位置にある要素を除去します。 | 411 |
| setElementAt() | ビジネス・オブジェクト配列内のビジネス・オブジェクトの値を設定します。 | 412 |
| size() | ビジネス・オブジェクト配列内の要素の数を戻します。 | 413 |
| sum() | ビジネス・オブジェクト配列内のすべてのビジネス・オブジェクトの、指定した属性の値を合計します。 | 413 |
| swap() | ビジネス・オブジェクト配列内の 2 つのビジネス・オブジェクトの位置を逆にします。この場合、配列の最初の要素が 0、2 番目が 1、3 番目が 2 であり、以下同様となることを念頭に置いてください。 | 413 |
| toString() | ビジネス・オブジェクト配列内の値を検索し、それらの値を単一ストリングで戻します。 | 414 |

## addElement()

ビジネス・オブジェクトをビジネス・オブジェクト配列に追加します。

### 構文

```
void addElement(BusObj element)
```

### パラメーター

*element* 配列に追加するビジネス・オブジェクト。

### 例外

CollaborationException — addElement() メソッドは、この例外に対して次の例外タイプを設定できます。

- AttributeException - 要素が無効の場合に設定します。

### 例

次の例では、getBusObArray() メソッドを使用して、order ビジネス・オブジェクトから itemList という名前のビジネス・オブジェクトの配列を検索します。この配列が items に割り当てられると、items に新しいビジネス・オブジェクトが追加されます。

```
BusObjArray items = order.getBusObjArray("itemList");
items.addElement(new BusObj("oneItem"));
```

## duplicate()

ビジネス・オブジェクト配列 (BusObjArray オブジェクト) とまったく同じビジネス・オブジェクト配列を作成します。

### 構文

```
BusObjArray duplicate()
```

### 戻り値

ビジネス・オブジェクト配列。

### 例

次の例では、items 配列を複製して newItems を作成します。

```
BusObjArray newItems = items.duplicate();
```

## elementAt()

ビジネス・オブジェクト配列内に位置を指定して単一ビジネス・オブジェクトを検索します。

### 構文

```
BusObj elementAt(int index)
```

### パラメーター

*index* 検索する配列の要素。配列の最初の要素が 0、2 番目が 1、3 番目が 2 であり、以下同様です。

### 例外

CollaborationException — elementAt() メソッドは、この例外に対して次の例外タイプを設定できます。

- AttributeException - 要素が無効の場合に設定します。

### 例

次の例では、items 配列内の 11 番目のビジネス・オブジェクトを検索して Item 変数に割り当てます。

```
BusObj Item = items.elementAt(10);
```

## equals()

ビジネス・オブジェクト配列を別のビジネス・オブジェクト配列と比較します。

### 構文

```
boolean equals(BusObjArray inputBusObjArray)
```

### パラメーター

*inputBusObjArray*
ビジネス・オブジェクト配列と比較するビジネス・オブジェクト配列。

### 注記

2 つのビジネス・オブジェクト配列の比較とは、要素の数とそれらの属性値を検査することです。

### 例

次の例では、equals() を使用して、条件付きループを設定します (ループの内側は示してありません)。

```
if (items.equals(newItems))
   {
...
   }
```

## getElements()

ビジネス・オブジェクト配列の内容を検索します。

### 構文

```
BusObj[] getElements()
```

### 例外

CollaborationException — getElements() メソッドは、この例外に対して次の例外タイプを設定できます。

- ObjectException — 要素のいずれかが無効である場合に設定します。

### 例

次の例では、items 配列の要素を出力します。

```
BusObj[] elements = items.getElements();
for (i=0, i<size; i++)
   {
   trace(1, elements[i].toString());
   }
```

## getLastIndex()

ビジネス・オブジェクト配列から、最後の有効な指標を検索します。

### 構文

```
int getLastIndex()
```

### 戻り値

BusObjArray の中の最後の要素を指す最後の指標。

## 注記

以前のバージョンでは、この操作を行うのに size() メソッドを使用していたことに注意してください。つまり、BusObjArray の中の最後の有効な指標を検索するときにはビジネス・オブジェクト配列の size() を使用していました。しかし、BusObjArray にギャップが含まれる場合、この方法では不正なデータが戻されます。

すべての Java 配列と同様に、BusObjArray はゼロ相対配列です。これは、size() メソッドが、getLastIndex() メソッドより 1 大きい値を戻すことを意味します。

## 例

次の例では、ビジネス・オブジェクト配列内の最後の指標を検索します。

```
int lastElementIndex = items.getLastIndex();
```

---

# max()

ビジネス・オブジェクト配列のすべての要素のうち、指定した属性の最大値を検索します。

## 構文

```
String max(String attr)
```

## パラメーター

*attr* ビジネス・オブジェクトの中の属性を参照する変数。この属性は、String 型、LongText 型、int 型、float 型、および double 型のいずれかでなければなりません。

## 戻り値

指定した属性の、ストリング形式での最大値。または、BusObjArray 内のすべての要素についてその属性の値が null である場合は null。

## 例外

UnknownAttributeException — 指定した属性が、渡されたビジネス・オブジェクトの有効な属性ではない場合です。

UnsupportedAttributeTypeException — 指定した属性のデータ型が、『注記』セクションにあるサポート対象の属性データ型でない場合です。

上記の例外すべては CollaborationException からサブクラス化されています。max() メソッドは、これらの例外に対して AttributeException 例外タイプを設定できます。

### 注記

max() メソッドは、BusObjArray の中のビジネス・オブジェクトのうち、指定した属性の最大値を検索します。例えば、3 つの Employee オブジェクトが使用されていて、その属性が「Float」型の「Salary」である場合、最大の給与を表すストリングが戻されます。

BusObjArray 内の要素のうち、指定した属性の値が null である要素は無視されます。すべての要素について指定した属性値が null の場合、null が戻されます。

属性のデータ型が String である場合、max() は字句が最長のストリングである属性値を戻します。

### 例

```
String maxSalary = items.max("Salary");
```

---

## maxBusObjArray()

指定した属性の最大値を持つビジネス・オブジェクトをビジネス・オブジェクト配列 (BusObjArray オブジェクト) として戻します。

### 構文

```
BusObjArray maxBusObjArray(String attr)
```

### パラメーター

*attr* ビジネス・オブジェクト配列内のビジネス・オブジェクト内の属性を参照する String 型、LongText 型、int 型、float 型、または double 型の変数。

### 戻り値

BusObjArray 形式または null のビジネス・オブジェクトのリスト。

### 例外

UnknownAttributeException - 指定した属性が、渡されたビジネス・オブジェクトの有効な属性ではない場合です。

UnsupportedAttributeTypeException - 指定した属性のデータ型が、『注記』セクションにあるサポート対象の属性データ型でない場合です。

上記の例外すべては CollaborationException からサブクラス化されています。maxBusObjArray() メソッドは、これらの例外に対して AttributeException 例外タイプを設定できます。

### 注記

maxBusObjArray() メソッドは、指定した属性の最大値を持つ 1 つ以上のビジネス・オブジェクトを検索し、これらのビジネス・オブジェクトを BusObjArray オブジェクトによって戻します。

例えば、Employee ビジネス・オブジェクトを含むビジネス・オブジェクト配列があり、入力引き数が Float 型の属性 Salary であるとします。このメソッドは、すべての Employee ビジネス・オブジェクトの中で Salary の最大値を調べ、その値を含むビジネス・オブジェクトを戻します。Salary の最大値を持つビジネス・オブジェクトが複数ある場合は、それらのビジネス・オブジェクトすべてが戻されます。

指定した属性が null を含むビジネス・オブジェクトは無視されます。配列内のすべてのビジネス・オブジェクトで値が null である場合、null が戻されます。

属性のデータ型が String のときは、字句が最長のストリングが戻されます。

### 例

```
BusObjArray boarrayWithMaxSalary = items.maxBusObjArray("Salary");
```

---

## maxBusObjs()

指定した属性の最大値を持つビジネス・オブジェクトを BusObj オブジェクトの配列として戻します。

### 構文

```
BusObj[] maxBusObjs(String attr)
```

### パラメーター

*attr* ビジネス・オブジェクト内の属性を参照する String 型、LongText 型、int 型、float 型、または double 型の変数。

### 戻り値

BusObj[] 形式または null のビジネス・オブジェクトのリスト。

### 例外

UnknownAttributeException - 指定した属性が、渡されたビジネス・オブジェクトの有効な属性ではない場合です。

UnsupportedAttributeTypeException - 指定した属性のデータ型が、『注記』セクションにあるサポート対象の属性データ型でない場合です。

上記の例外すべては CollaborationException からサブクラス化されています。maxBusObjs() メソッドは、これらの例外に対して AttributeException 例外タイプを設定できます。

### 注記

maxBusObjs() メソッドは、指定した属性の最大値を持つ 1 つ以上のビジネス・オブジェクトを検索し、これらのビジネス・オブジェクトを BusObj オブジェクトの配列として戻します。

例えば、Employee ビジネス・オブジェクトを含むビジネス・オブジェクト配列があり、入力引き数が Float 型の属性 Salary であるとします。このメソッドは、すべて

の Employee ビジネス・オブジェクトの中で Salary の最大値を調べ、その値を含むビジネス・オブジェクトを戻します。Salary の最大値を持つビジネス・オブジェクトが複数ある場合は、それらのビジネス・オブジェクトすべてが戻されます。

指定した属性が null を含むビジネス・オブジェクトは無視されます。配列内のすべてのビジネス・オブジェクトで値が null である場合、null が戻されます。

属性のデータ型が String のときは、字句が最長のストリングが戻されます。

### 例

```
BusObj[] bosWithMaxSalary = items.maxBusObjs("Salary");
```

---

## min()

配列の中のビジネス・オブジェクトのうち、指定した属性の最小値を検索します。

### 構文

```
String min(String attr)
```

### パラメーター

*attr* ビジネス・オブジェクト内の属性を参照する String 型、LongText 型、int 型、float 型、または double 型の変数。

### 戻り値

指定した属性の、ストリング形式での最小値。または、BusObjArray 内のすべての要素についてその属性の値が null である場合は null。

### 例外

UnknownAttributeException - 指定した属性が、渡されたビジネス・オブジェクトの有効な属性ではない場合です。

UnsupportedAttributeTypeException - 指定した属性のデータ型が、『注記』セクションにあるサポート対象の属性データ型でない場合です。

上記の例外すべては CollaborationException からサブクラス化されています。min() メソッドは、これらの例外に対して AttributeException 例外タイプを設定できます。

### 注記

min() メソッドは、ビジネス・オブジェクト配列内のビジネス・オブジェクトのうち、指定した属性の最小値を検索します。

例えば、Employee ビジネス・オブジェクトを含むビジネス・オブジェクト配列があり、入力引き数が Float 型の属性 Salary であるとします。このメソッドは、すべての Employee ビジネス・オブジェクトの中で Salary の最小値を調べ、その値を含むビジネス・オブジェクトを戻します。Salary の最小値を持つビジネス・オブジェクトが複数ある場合は、それらのビジネス・オブジェクトすべてが戻されます。

指定した属性が null を含むビジネス・オブジェクトは無視されます。配列内のすべてのビジネス・オブジェクトで値が null である場合、null が戻されます。

属性のデータ型が String のときは、字句が最短のストリングが戻されます。

### 例

```
String minSalary = items.min("Salary");
```

## minBusObjArray()

指定した属性の最小値を持つビジネス・オブジェクトを BusObjArray オブジェクトとして戻します。

### 構文

```
BusObjArray minBusObjArray(String attr)
```

### パラメーター

*attr* ビジネス・オブジェクト内の属性を参照する String 型、LongText 型、int 型、float 型、または double 型 の変数。

### 戻り値

BusObjArray 形式または null のビジネス・オブジェクトのリスト。

### 例外

UnknownAttributeException - 指定した属性が、渡されたビジネス・オブジェクトの有効な属性ではない場合です。

UnsupportedAttributeTypeException - 指定した属性のデータ型が、『注記』セクションにあるサポート対象の属性データ型でない場合です。

上記の例外すべては CollaborationException からサブクラス化されています。minBusObjArray() メソッドは、これらの例外に対して AttributeException 例外タイプを設定できます。

### 注記

minBusObjArray() メソッドは、指定した属性の最小値を持つ 1 つ以上のビジネス・オブジェクトを検索し、これらのビジネス・オブジェクトを BusObjArray オブジェクトによって戻します。

例えば、Employee ビジネス・オブジェクトを含むビジネス・オブジェクト配列があり、入力引き数が Float 型の属性 Salary であるとします。このメソッドは、すべての Employee ビジネス・オブジェクトの中で Salary の最小値を調べ、その値を含むビジネス・オブジェクトを戻します。Salary の最小値を持つビジネス・オブジェクトが複数ある場合は、これらのビジネス・オブジェクトすべてが戻されます。

指定した属性が null を含むビジネス・オブジェクトは無視されます。配列内のすべてのビジネス・オブジェクトで値が null である場合、null が戻されます。

属性のデータ型が String のときは、字句が最短のストリングが戻されます。

### 例

```
BusObjArray boarrayWithMinSalary = items.minBusObjArray("Salary");
```

## minBusObjs()

指定した属性の最小値を持つビジネス・オブジェクトを BusObj オブジェクトの配列として戻します。

### 構文

```
BusObj[] minBusObjs(String attr)
```

### パラメーター

*attr* ビジネス・オブジェクト内の属性を参照する String 型、LongText 型、int 型、float 型、または double 型の変数。

### 戻り値

BusObj[] 形式または null のビジネス・オブジェクトのリスト。

### 例外

UnknownAttributeException - 指定した属性が、渡されたビジネス・オブジェクトの有効な属性ではない場合です。

UnsupportedAttributeTypeException - 指定した属性のデータ型が、『注記』セクションにあるサポート対象の属性データ型でない場合です。

上記の例外すべては CollaborationException からサブクラス化されています。minBusObjs() メソッドは、これらの例外に対して AttributeException 例外タイプを設定できます。

### 注記

minBusObjs() メソッドは、指定した属性の最大値を持つ 1 つ以上のビジネス・オブジェクトを検索し、これらのビジネス・オブジェクトを BusObj オブジェクトの配列として戻します。

例えば、Employee ビジネス・オブジェクトを含むビジネス・オブジェクト配列があり、入力引き数が Float 型の属性 Salary であるとします。このメソッドは、すべての Employee ビジネス・オブジェクトの中で Salary の最小値を調べ、その値を含むビジネス・オブジェクトを戻します。Salary の最小値を持つビジネス・オブジェクトが複数ある場合は、これらのビジネス・オブジェクトすべてが戻されます。

指定した属性が null を含むビジネス・オブジェクトは無視されます。配列内のすべてのビジネス・オブジェクトで値が null である場合、null が戻されます。

属性のデータ型が String のときは、字句が最短のストリングが戻されます。

### 例

```
BusObj[] bosWithMinSalary = items.minBusObjs("Salary");
```

## removeAllElements()

ビジネス・オブジェクト配列からすべての要素を除去します。

### 構文

```
void removeAllElements()
```

### 例

次の例では、配列 items のすべての要素を除去します。

```
items.removeAllElements();
```

## removeElement()

ビジネス・オブジェクト配列から 1 つの要素を削除します。

### 構文

```
void removeElement(BusObj element)
```

### パラメーター

*elementReference*
配列の要素を参照する変数。

### 例外

CollaborationException — removeElement() メソッドは、この例外に対して次の例外タイプを設定できます。

- AttributeException - 要素が無効の場合に設定します。

### 注記

配列から要素を削除すると、配列のサイズが変わり、既存の要素の指標が変更されます。

### 例

次の例では、Child1 要素を items ビジネス・オブジェクト配列から削除します。

```
items.removeElement(Child1);
```

## removeElementAt()

ビジネス・オブジェクト配列内の特定の位置にある要素を除去します。

### 構文

```
void removeElementAt(int index)
```

### パラメーター

*index* 配列内の要素の位置を表す整数。配列の最初の要素の位置が 0、2 番目の要素の位置が 1、3 番目の位置が 2 であり、以下同様です。

### 例外

CollaborationException — removeElementAt() メソッドは、この例外に対して次の例外タイプを設定できます。

- AttributeException - 要素が無効の場合に設定します。

### 注記

配列から要素を除去すると、配列のサイズが変わり、場合によっては既存の要素の指標が変更されます。

### 例

次の例では、配列 items の 6 番目のビジネス・オブジェクトを削除します。

```
items.removeElementAt(5);
```

---

## setElementAt()

ビジネス・オブジェクト配列内のビジネス・オブジェクトの値を設定します。

### 構文

```
void setElementAt (int index, BusObj element)
```

### パラメーター

*index* 配列の位置を表す整数。配列の最初の要素が 0、2 番目が 1、3 番目が 2 であり、以下同様です。

*inputBusObj* 配列の要素を設定する値を含むビジネス・オブジェクト。

### 例外

CollaborationException — setElementAt() メソッドは、この例外に対して次の例外タイプを設定できます。

- AttributeException - 要素が無効の場合に設定します。

### 注記

このメソッドは、配列内の指定位置にあるビジネス・オブジェクトの値を、入力ビジネス・オブジェクトの値に設定します。

### 例

次の例では、タイプ Item の新規ビジネス・オブジェクトを作成し、これを配列 items の 4 番目の要素として追加します。

```
items.setElementAt(3, new BusObj("Item"));
```

## size()

ビジネス・オブジェクト配列内の要素の数を戻します。

### 構文

```
int size()
```

### 例

次の例では、配列 items の要素の数を戻します。

```
int size = items.size();
```

## sum()

ビジネス・オブジェクト配列内のすべてのビジネス・オブジェクトの、指定した属性の値を合計します。

### 構文

```
double sum(String attrName)
```

### パラメーター

*attr* ビジネス・オブジェクトの中の属性を参照する変数。指定できる属性の型は、int、float、または double です。

### 戻り値

ビジネス・オブジェクトのリストからの、指定した属性の合計。

### 例外

UnknownAttributeException - 指定した属性が、渡されたビジネス・オブジェクトの有効な属性ではない場合です。

UnsupportedAttributeTypeException - 指定した属性のデータ型が、『注記』セクションにあるサポート対象の属性データ型でない場合です。

上記の例外すべては CollaborationException からサブクラス化されています。sum() メソッドは、これらの例外に対して AttributeException 例外タイプを設定できます。

### 例

```
double sumSalary = items.sum("Salary");
```

## swap()

ビジネス・オブジェクト配列内の 2 つのビジネス・オブジェクトの位置を逆にします。この場合、配列の最初の要素が 0、2 番目が 1、3 番目が 2 であり、以下同様となることを念頭に置いてください。

### 構文

```
void swap(int index1, int index2)
```

### パラメーター

*index1* 交換する一方の要素の配列の位置。

*index2* 交換するもう一方の要素の配列の位置。

### 例

次の例では、swap() を使用して、次の配列内の BusObjA と BusObjC の位置を逆にします。

| BusObjA | BusObjB | BusObjC |
|---|---|---|

```
swap(0,2);
```

swap() を呼び出した結果の配列は、次のようになります。

| BusObjC | BusObjB | BusObjA |
|---|---|---|

## toString()

ビジネス・オブジェクト配列内の値を検索し、それらの値を単一ストリングで戻します。

### 構文

```
String toString()
```

### 例

次の例では、toString() を使用して items ビジネス・オブジェクト配列の内容を検索してから、logInfo() を使用してそれらの内容をログ・ファイルに書き込みます。

```
logInfo(items.toString());
```

# 第 25 章 CwDBConnection クラス

CwDBConnection クラスは、データベース内の SQL 照会を実行するメソッドを提供します。照会は、接続プールから取得される接続を通じて実行されます。このクラスをインスタンス化するには、BaseCollaboration クラス内の getDBConnection() を呼び出す必要があります。すべてのコラボレーションは BaseCollaboration から派生したか、またはサブクラス化されているので、getDBConnection() へのアクセスが可能です。

表 78 に、CwDBConnection クラスのメソッドの要約を示します。

*表 78. CwDBConnection メソッドの要約*

| メソッド | 説明 | ページ |
|---|---|---|
| beginTransaction() | 現行接続の明示的なトランザクションを開始します。 | 415 |
| commit() | 現行接続と関連付けられているアクティブなトランザクションをコミットします。 | 416 |
| executeSQL() | 構文およびオプショナル・パラメーター配列を指定して静的 SQL 照会を実行します。 | 419 |
| executePreparedSQL() | 構文およびオプショナル・パラメーター配列を指定して準備済み SQL 照会を実行します。 | 418 |
| executeStoredProcedure() | 名前とパラメーター配列を指定して SQL ストアード・プロシージャーを実行します。 | 421 |
| getUpdateCount() | データベースへの最後の書き込み操作によって影響された行の数を戻します。 | 422 |
| hasMoreRows() | 照会結果に、処理する必要のある行がさらに存在するかどうかを判別します。 | 423 |
| inTransaction() | トランザクションが、現行接続で進行中かどうかを判別します。 | 424 |
| isActive() | 現行接続がアクティブかどうかを判別します。 | 424 |
| nextRow() | 照会結果から、次の行を検索します。 | 425 |
| release() | 現行接続の使用を解放して、それを接続プールに戻します。 | 425 |
| rollBack() | 現行接続と関連付けられているアクティブなトランザクションをロールバックします。 | 426 |

## beginTransaction()

現行接続の明示的なトランザクションを開始します。

### 構文

```
void beginTransaction()
```

### パラメーター

なし。

## 戻り値

なし。

## 例外

CwDBConnectionException — データベース・エラーが発生した場合。

## 注記

beginTransaction() メソッドは、現行接続における新規の明示的なトランザクションの開始を示します。beginTransaction()、commit()、および rollBack() メソッドはともに、明示的なトランザクションに対するトランザクション境界の管理を提供します。このトランザクションには、SQL 照会 (SQL ステートメント INSERT、DELETE、または UPDATE を含む) と、これらの SQL ステートメントの 1 つを含むストアード・プロシージャーとが含まれます。

**要確認:** beginTransaction() は、接続が明示的なトランザクション・ブラケットを使用しているときにのみ使用してください。接続が暗黙的なトランザクション・ブラケットを使用しているときに beginTransaction() を使用すると、CwDBTransactionException 例外が発生します。

明示的なトランザクションを開始する前に、getDBConnection() メソッドを使用して BaseCollaboration クラスから CwDBConnection オブジェクトを作成しておく必要があります。この接続では、明示的なトランザクション・ブラケットを必ず使用してください。

## 例

次の例では、トランザクションを使用して、CustDBConnPool の中の接続に関連付けられているデータベース表に行を挿入する照会を実行します。

```
CwDBConnection connection = getDBConnection("CustDBConnPool", false);

// Begin a transaction
connection.beginTransaction();

// Insert a row
connection.executeSQL("insert...");

// Commit the transaction
connection.commit();

// Release the connection
connection.release();
```

## 関連項目

237 ページの『トランザクションの管理』

commit()、getDBConnection()、inTransaction()、rollBack()

---

# commit()

現行接続と関連付けられているアクティブなトランザクションをコミットします。

## 構文

```
void commit()
```

## パラメーター

なし。

## 戻り値

なし。

## 例外

CwDBConnectionException - データベース・エラーが発生した場合。

## 注記

commit() メソッドは、現行接続に関連付けられているデータベースに加えられた変更をコミットすることによって、アクティブ・トランザクションを終了します。beginTransaction()、commit()、および rollBack() メソッドはともに、明示的なトランザクションに対するトランザクション境界の管理を提供します。このトランザクションには、SQL 照会 (SQL ステートメント INSERT、DELETE、または UPDATE を含む) と、これらの SQL ステートメントの 1 つを含むストアード・プロシージャーとが含まれます。

**要確認:** commit() は、接続が明示的なトランザクション・ブラケットを使用しているときにのみ使用してください。接続が暗黙的なトランザクション・ブラケットを使用しているときに commit() を使用すると、CwDBTransactionException 例外が発生します。接続が解放される前に commit() (または rollback()) を使用して明示的なトランザクションを終了しない場合、コラボレーションが成功したかどうかに基づいて InterChange Server Express によって暗黙的にトランザクションが終了されます。コラボレーションが成功の場合、ICS は、このデータベース・トランザクションをコミットします。コラボレーションが成功ではない 場合、ICS は、このデータベース・トランザクションを暗黙的にロールバックします。コラボレーションが成功かどうかにかかわらず、ICS は警告を記録します。

明示的なトランザクションを開始する前に、getDBConnection() メソッドを使用して BaseCollaboration クラスから CwDBConnection オブジェクトを作成しておく必要があります。この接続では、明示的なトランザクション・ブラケットを必ず使用してください。

## 例

トランザクションのコミットの例については、beginTransaction() の例を参照してください。

## 関連項目

237 ページの『トランザクションの管理』

`beginTransaction()`、`getDBConnection()`、`inTransaction()`、`rollBack()`

---

# executePreparedSQL()

構文およびオプショナル・パラメーター配列を指定して準備済み SQL 照会を実行します。

## 構文

```
void executePreparedSQL(String query)
void executePreparedSQL(String query, Vector queryParameters)
```

## パラメーター

*query* データベースで実行する SQL 照会のストリング表現。

*queryParameters*
SQL 照会でパラメーターに渡される引き数の `Vector` オブジェクト。

## 戻り値

なし。

## 例外

`CwDBSQLException` - データベース・エラーが発生した場合。

## 注記

`executePreparedSQL()` メソッドは、指定されている*照会* ストリングを準備済み SQL ステートメントとして、現行接続に関連付けられているデータベースに送信します。この照会は、最初に実行されるときにストリングとしてデータベースに送信されます。データベースは、このストリングを実行可能形式 (準備済みステートメントと呼ばれる) にコンパイルし、SQL ステートメントを実行し、この準備済みステートメントを `executePreparedSQL()` に戻します。`executePreparedSQL()` メソッドは、準備済みステートメントをメモリー内に保管します。SQL ステートメントを複数回実行する必要がある場合は、`executePreparedSQL()` を使用します。これに対して `executeSQL()` メソッドは、準備済みステートメントを保管しない ため、一度のみ実行すればよい照会の場合に便利です。

**要確認:** `executePreparedSQL()` を使用して照会を実行する前に、`getDBConnection()` メソッドを使用して `BaseDLM` クラスから `CwDBConnection` を生成し、目的のデータベースへの接続を取得しておく必要があります。

実行できる SQL ステートメントを以下に示します (必要なデータベース許可がある場合)。

- 1 つ以上のデータベース表からデータを要求する SELECT ステートメント

検索したデータにアクセスするには、hasMoreRows() および nextRow() メソッドを使用します。

- データベース内のデータを変更する SQL ステートメント
  - INSERT
  - DELETE
  - UPDATE

  接続が明示的なトランザクション・ブラケットを使用している場合は、beginTransaction() を使用して各トランザクションを明示的に開始して、commit() または rollback() のいずれかを使用して終了する必要があります。
- 準備済みストアード・プロシージャーを実行する CALL ステートメント (ただしこのストアード・プロシージャーでは OUT パラメーターを使用できません)

  OUT パラメーターを使用してストアード・プロシージャーを実行するには、executeStoredProcedure() メソッドを使用してください。詳細については、234 ページの『executeStoredProcedure() でのストアード・プロシージャーの呼び出し』を参照してください。

## 関連項目

229 ページの『準備済み照会の実行』

beginTransaction()、commit()、executeSQL()、executeStoredProcedure()、getDBConnection()、hasMoreRows()、nextRow()、rollBack()

---

# executeSQL()

構文およびオプショナル・パラメーター配列を指定して静的 SQL 照会を実行します。

## 構文

```
void executeSQL(String query)
void executeSQL(String query, Vector queryParameters)
```

## パラメーター

*query* データベースで実行する SQL 照会のストリング表現。

*queryParameters*
SQL 照会でパラメーターに渡される引き数の Vector オブジェクト。

## 戻り値

なし。

## 例外

CwDBSQLException - データベース・エラーが発生した場合。

## 注記

executeSQL() メソッドは、指定されている*照会* ストリングを静的 SQL ステートメントとして、現行接続に関係付けられているデータベースに送信します。この照会はストリングとしてデータベースに送信されます。データベースは、このストリングを実行可能形式にコンパイルし、SQL ステートメントを実行します。この場合に、実行可能形式は保管されません。SQL ステートメントを一度のみ実行する必要がある場合は、executeSQL() を使用します。これに対して executePreparedSQL() メソッドは、実行可能形式 (準備済みステートメントと呼ばれる) を保管するため、何度も実行する必要がある照会の場合に便利です。

**要確認:** executeSQL() を使用して照会を実行する前に、getDBConnection() メソッドを使用して BaseDLM クラスから CwDBConnection オブジェクトを生成し、目的のデータベースへの接続を取得しておく必要があります。

実行できる SQL ステートメントを以下に示します (必要なデータベース許可がある場合)。

- 1 つ以上のデータベース表からデータを要求する SELECT ステートメント

  検索したデータにアクセスするには、hasMoreRows() および nextRow() メソッドを使用します。
- データベース内のデータを変更する SQL ステートメント
  - INSERT
  - DELETE
  - UPDATE

  接続が明示的なトランザクション・ブラケットを使用している場合は、beginTransaction() を使用して各トランザクションを明示的に開始して、commit() または rollback() のいずれかを使用して終了する必要があります。
- ストアード・プロシージャーを静的に実行する CALL ステートメント (ただしこのストアード・プロシージャーでは、OUT パラメーターを使用できません)

  OUT パラメーターを使用してストアード・プロシージャーを実行するには、executeStoredProcedure() メソッドを使用してください。詳細については、234 ページの『executeStoredProcedure() でのストアード・プロシージャーの呼び出し』を参照してください。

## 例

次の例では、接続が AccntConnPool 接続プール内にあるアカウンティング・データベースに、行を挿入する照会を実行します。

```
CwDBConnection connection = getDBConnection("AccntConnPool");

// Begin a transaction
connection.beginTransaction();

// Insert a row
connection.executeSQL("insert...");

// Commit the transaction
```

```
connection.commit();

// Release the database connection
connection.release();
```

データベース表からデータを選択する、より完全なコード・サンプルは、225 ページの『データを戻す静的照会の実行 (SELECT)』を参照してください。

## 関連項目

225 ページの『静的照会の実行』

executePreparedSQL()、executeStoredProcedure()、getDBConnection()、hasMoreRows()、nextRow()

---

# executeStoredProcedure()

名前とパラメーター配列を指定して SQL ストアード・プロシージャーを実行します。

## 構文

```
void executeStoredProcedure(String storedProcedure,
                            Vector storedProcParameters)
```

## パラメーター

*storedProcedure*
: データベースで実行する SQL ストアード・プロシージャーの名前。

*storedProcParameters*
: ストアード・プロシージャーに渡されるパラメーターの Vector オブジェクト。各パラメーターは、CwDBStoredProcedureParam クラスのインスタンスです。この配列を通じてパラメーターを渡す方法の詳細については、234 ページの『executeStoredProcedure() でのストアード・プロシージャーの呼び出し』を参照してください。

## 戻り値

なし。

## 例外

CwDBSQLException - データベース・エラーが発生した場合。

## 注記

executeStoredProcedure() メソッドは、指定した *storedProcedure* への呼び出しを、現行接続に関連付けられているデータベースに送信します。このメソッドは、ストアード・プロシージャー呼び出しを準備済み SQL ステートメントとして送信します。つまり、このストアード・プロシージャー呼び出しは、最初に実行されるときにストリングとしてデータベースに送信されます。データベースは、このストリングを実行可能形式 (準備済みステートメントと呼ばれる) にコンパイルし、SQL

ステートメントを実行し、この準備済みステートメントを executeStoredProcedure() に戻します。executeStoredProcedure() メソッドは、準備済みステートメントをメモリー内に保管します。

**要確認:** executeStoredProcedure() を使用してストアード・プロシージャーを実行する前に、getDBConnection() メソッドを使用して BaseDLM クラスから CwDBConnection オブジェクトを作成しておく必要があります。

ストアード・プロシージャーが戻すデータを処理するには、hasMoreRows() および nextRow() メソッドを使用します。

ストアード・プロシージャーの処理は、ストアード・プロシージャーに OUT パラメーターが含まれていない 場合は、executeSQL() または executePreparedSQL() メソッドを使用して行うことも可能です。ストアード・プロシージャーが OUT パラメーターを使用する場合、executeStoredProcedure() を使用してそれを実行する必要があります。executeSQL() または executePreparedSQL() の場合と異なり、ストアード・プロシージャーを実行するのに全 SQL ステートメントを渡す必要はありません。executeStoredProcedure() の場合には、ストアード・プロシージャーの名前と CwDBStoredProcedureParam オブジェクトの Vector パラメーター配列のみを渡す必要があります。executeStoredProcedure() メソッドは、*storedProcParameters* 配列からパラメーターの数を判断し、ストアード・プロシージャーの呼び出しステートメントを構築することができます。

## 関連項目

234 ページの『executeStoredProcedure() でのストアード・プロシージャーの呼び出し』

executePreparedSQL()、executeSQL()、getDBConnection()、hasMoreRows()、nextRow()

---

# getUpdateCount()

データベースへの最後の書き込み操作によって影響された行の数を戻します。

## 構文

```
int getUpdateCount()
```

## パラメーター

なし。

## 戻り値

最後の書き込み操作によって影響された行の数を表す int を戻します。

## 例外

CwDBConnectionException - データベース・エラーが発生した場合。

## 注記

getUpdateCount() メソッドは、現行接続に関連付けられているデータベース内で、直近の更新操作によって変更された行の数を示します。このメソッドは、UPDATE または INSERT ステートメントをデータベースに送信した後に、その SQL ステートメントによって影響された行の数を判別する必要がある場合に役立ちます。

**要確認:** このメソッドを使用する前に、BaseDLM クラスから getDBConnection() メソッドを使用して CwDBConnection オブジェクトを作成し、CwDBConnection クラスから executeSQL() または executePreparedSQL() メソッドのいずれかを使用してデータベースを更新する照会を送信しておく必要があります。

## 関連項目

executePreparedSQL()、executeSQL()、getDBConnection()

---

# hasMoreRows()

照会結果に、処理する必要のある行がさらに存在するかどうかを判別します。

## 構文

```
boolean hasMoreRows()
```

## パラメーター

なし。

## 戻り値

行がさらに存在する場合は true を戻します。

## 例外

CwDBSQLException - データベース・エラーが発生した場合。

## 注記

hasMoreRows() メソッドは、現行接続に関連付けられている照会結果に、処理する必要のある行がさらに存在するかどうかを判別します。このメソッドは、データを戻す照会から結果を検索するときに使用してください。これらの照会には、SELECT ステートメントとストアード・プロシージャーが含まれています。一度に接続に関連付けることのできる照会は 1 つのみです。このため、hasMoreRows() が false を戻す前に別の照会を実行すると、最初の照会のデータが失われます。

## 関連項目

225 ページの『データを戻す静的照会の実行 (SELECT)』

executePreparedSQL()、executeSQL()、nextRow()

## inTransaction()

トランザクションが、現行接続で進行中かどうかを判別します。

### 構文

```
boolean inTransaction()
```

### パラメーター

なし。

### 戻り値

トランザクションが現行接続で現在アクティブである場合は `true` を戻し、アクティブでない場合は `false` を戻します。

### 例外

`CwDBConnectionException` - データベース・エラーが発生した場合。

### 注記

`inTransaction()` メソッドは、現行接続にアクティブ・トランザクション (開始されたが終了していないトランザクション) が存在するかどうかを示す boolean 値を戻します。

**要確認:** トランザクションを開始する前に、`BaseDLM` クラスから `getDBConnection()` メソッドを使用して `CwDBConnection` オブジェクトを作成しておく必要があります。

### 関連項目

237 ページの『トランザクションの管理』

`beginTransaction()`、`commit()`、`getDBConnection()`、`rollBack()`

## isActive()

現行接続がアクティブかどうかを判別します。

### 構文

```
boolean isActive()
```

### パラメーター

なし。

### 戻り値

現行接続がアクティブである場合は `true` を戻し、この接続が解放されている場合は `false` を戻します。

### 例外

なし。

### 関連項目

getDBConnection()、release()

## nextRow()

照会結果から、次の行を検索します。

### 構文

```
Vector nextRow()
```

### パラメーター

なし。

### 戻り値

照会結果の次の行を Vector オブジェクトとして戻します。

### 例外

CwDBSQLException - データベース・エラーが発生した場合。

### 注記

nextRow() メソッドは、現行接続に関連付けられている照会結果からデータの 1 行を戻します。このメソッドは、データを戻す照会から結果を検索するときに使用してください。これらの照会には、SELECT ステートメントとストアード・プロシージャーが含まれています。一度に接続に関連付けることのできる照会は 1 つのみです。このため、nextRow() がデータの最後の行を戻す前に別の照会を実行すると、最初の照会の結果が失われます。

### 関連項目

225 ページの『データを戻す静的照会の実行 (SELECT)』

hasMoreRows()、executePreparedSQL()、executeSQL()、executeStoredProcedure()

## release()

現行接続の使用を解放して、それを接続プールに戻します。

### 構文

```
void release()
```

### パラメーター

なし。

## 戻り値

なし。

## 例外

CwDBConnectionException

## 注記

release() メソッドは、コラボレーション・オブジェクトによる現行接続の使用を明示的に解放します。解放された接続はその接続プールに戻されます。関連付けられているデータベースへの接続を必要とする他のコンポーネント (マップまたはコラボレーション) は、この接続を使用できるようになります。接続を明示的に解放しない場合は、現行のコラボレーションの実行の最後にコラボレーション・オブジェクトによって暗黙的に接続が解放されます。このため、静的変数の中に接続を保管して再使用することはできません。

**重要:** トランザクションが現在アクティブである場合、release() メソッドは使用しないでください。暗黙的なトランザクション・ブラケットの場合、ICS は、コラボレーションが成功または失敗のいずれかを確認するまでデータベース・トランザクションを終了しません。このため、暗黙的なトランザクション・ブラケットを使用する接続にこのメソッドを使用すると、CwDBTransactionException 例外が発生します。この例外を明示的に処理しないと、アクティブなトランザクションも自動的にロールバックされます。トランザクションがアクティブかどうかは、inTransaction() メソッドを使用して判別できます。

## 関連項目

242 ページの『接続の解放』

getDBConnection()、inTransaction()、isActive()

---

# rollBack()

現行接続と関連付けられているアクティブなトランザクションをロールバックします。

## 構文

```
void rollBack()
```

## パラメーター

なし。

## 戻り値

なし。

## 例外

CwDBTransactionException

## 注記

rollback() メソッドは、現行接続に関連付けられているデータベースに加えられた変更をロールバックすることによって、アクティブ・トランザクションを終了します。beginTransaction()、commit()、および rollBack() メソッドはともに、明示的なトランザクションに対するトランザクション境界の管理を提供します。このトランザクションには、SQL 照会 (SQL ステートメント INSERT、DELETE、または UPDATE を含む) と、これらの SQL ステートメントの 1 つを含むストアード・プロシージャーとが含まれます。ロールバックが失敗すると、rollback() は CwDBTransactionException 例外をスローして、エラーを記録します。

**要確認:** rollback() は、接続が明示的なトランザクション・ブラケットを使用しているときにのみ使用してください。接続が暗黙的なトランザクション・ブラケットを使用しているときに rollback() を使用すると、CwDBTransactionException 例外が発生します。接続が解放される前に rollback() (または commit()) を使用して明示的なトランザクションを終了しないと、コラボレーションが成功したかどうかに基づいて InterChange Server Express によって暗黙的にトランザクションが終了されます。コラボレーションが成功の場合、ICS は、このデータベース・トランザクションをコミットします。コラボレーションが成功ではない 場合、ICS は、このデータベース・トランザクションを暗黙的にロールバックします。コラボレーションが成功かどうかにかかわらず、ICS は警告を記録します。

明示的なトランザクションを開始する前に、getDBConnection() メソッドを使用して BaseCollaboration クラスから CwDBConnection オブジェクトを作成しておく必要があります。この接続では、明示的なトランザクション・ブラケットを必ず使用してください。

## 例

rollback() を使用したトランザクションの管理の例は、240 ページの『明示的なトランザクション・ブラケットのあるトランザクション・スコープ』の例を参照してください。

## 関連項目

237 ページの『トランザクションの管理』

beginTransaction()、commit()、getDBConnection()、inTransaction()

# 第 26 章 CwDBStoredProcedureParam クラス

CwDBStoredProcedureParam オブジェクトは、ストアード・プロシージャーの単一パラメーターについて説明します。表 79 に、CwDBStoredProcedureParam クラスのメソッドの要約を示します。

*表 79. CwDBStoredProcedureParam メソッドの要約*

| メソッド | 説明 | ページ |
|---|---|---|
| CwDBStoredProcedureParam() | ストアード・プロシージャーのパラメーターの引き数情報を持つ CwDBStoredProcedureParam の新しいインスタンスを作成します。 | 429 |
| getParamType() | 現行ストアード・プロシージャー・パラメーターのイン/アウト・タイプを整数定数として検索します。 | 431 |
| getValue() | 現行ストアード・プロシージャー・パラメーターの値を検索します。 | 432 |

## CwDBStoredProcedureParam()

ストアード・プロシージャーのパラメーターの引き数情報を持つ CwDBStoredProcedureParam の新しいインスタンスを作成します。

### 構文

```
CwDBStoredProcedureParam(int paramType, String paramValue);

CwDBStoredProcedureParam(int paramType, int paramValue);
CwDBStoredProcedureParam(int paramType, Integer paramValue);
CwDBStoredProcedureParam(int paramType, Long paramValue);

CwDBStoredProcedureParam(int paramType, double paramValue);
CwDBStoredProcedureParam(int paramType, Double paramValue);
CwDBStoredProcedureParam(int paramType, float paramValue);
CwDBStoredProcedureParam(int paramType, Float paramValue);
CwDBStoredProcedureParam(int paramType, BigDecimal paramValue);

CwDBStoredProcedureParam(int paramType, boolean paramValue);
CwDBStoredProcedureParam(int paramType, Boolean paramValue);

CwDBStoredProcedureParam(int paramType, java.sql.Date paramValue);
CwDBStoredProcedureParam(int paramType, java.sql.Time paramValue);
CwDBStoredProcedureParam(int paramType, java.sql.Timestamp paramValue);

CwDBStoredProcedureParam(int paramType, java.sql.Blob paramValue);
CwDBStoredProcedureParam(int paramType, java.sql.Clob paramValue);

CwDBStoredProcedureParam(int paramType, byte[] paramValue);
CwDBStoredProcedureParam(int paramType, Array paramValue);
CwDBStoredProcedureParam(int paramType, Struct paramValue);
```

### パラメーター

*paramType* ストアード・プロシージャー・パラメーターに関連付けられているイン/アウト・パラメーター・タイプ。

*paramValue* ストアード・プロシージャーに送信される引き数値。この値は、以下の Java データ型のいずれかになります。

- String
- int
- Integer
- Long
- double
- Double
- float
- Float
- BigDecimal
- boolean
- java.sql.Date
- java.sql.Time
- java.sql.Timestamp
- java.sql.Blob
- java.sql.Clob
- byte[]
- Array
- Struct

## 戻り値

ストアード・プロシージャーの宣言内の 1 つの引き数の引き数情報を保持するための、新規 CwDBStoredProcedureParam オブジェクトを戻します。

## 例外

なし。

## 注記

CwDBStoredProcedureParam() コンストラクターは、ストアード・プロシージャーのパラメーターを説明する CwDBStoredProcedureParam インスタンスを作成します。この場合のパラメーター情報は以下のとおりです。

- パラメーターのイン/アウト・タイプ

  パラメーターのイン/アウト・タイプは、コンストラクターの最初の引き数によって初期化されます。パラメーターの有効なイン/アウト・タイプのリストについては、431 ページの表 80 を参照してください。

- パラメーター値

  このパラメーター値は、コンストラクターの 2 番目の引き数によって初期化されます。CwDBStoredProcedureParam クラスには、サポートされるパラメーター値のデータ型ごとにコンストラクターの 1 つの形式が用意されています。ストアー

ド・プロシージャー・パラメーターの場合の Java データ型と JDBC データ型の間のマッピングのリストについては、237 ページの表 57 を参照してください。

executeStoredProcedure() メソッドに渡すストアード・プロシージャー・パラメーターの Java Vector は、開発者が用意する必要があります。このメソッドは、ストアード・プロシージャーの名前とパラメーター・ベクターからストアード・プロシージャー呼び出しを作成し、現行接続に関連付けられているデータベースにこの呼び出しを送信します。

### 関連項目

234 ページの『executeStoredProcedure() でのストアード・プロシージャーの呼び出し』

executeStoredProcedure()

---

## getParamType()

現行ストアード・プロシージャー・パラメーターのイン/アウト・タイプを整数定数として検索します。

### 構文

```
int getParamType()
```

### パラメーター

なし。

### 戻り値

関連付けられている CwDBStoredProcedureParam パラメーターのイン/アウト・タイプを戻します。

### 例外

なし。

### 注記

getParamType() メソッドは、現行ストアード・プロシージャー・パラメーターのイン/アウト・パラメーター・タイプを戻します。イン/アウト・パラメーター・タイプは、ストアード・プロシージャーがパラメーターをどのように使用するかを示します。CwDBStoredProcedureParam クラスは、表 80 に示すとおり、それぞれのイン/アウト・タイプを定数として表します。

*表 80. パラメーターのイン/アウト・タイプ*

| パラメーターのイン/アウト・タイプ | 説明 | イン/アウト・タイプ定数 |
|---|---|---|
| IN パラメーター | IN パラメーターは*入力専用* です。つまり、ストアード・プロシージャーはその値を入力として受け入れますが、値を戻すためにそのパラメーターを使用しません。 | PARAM_IN |

*表 80. パラメーターのイン/アウト・タイプ (続き)*

| パラメーターのイン/アウト・タイプ | 説明 | イン/アウト・タイプ定数 |
|---|---|---|
| OUT パラメーター | OUT パラメーターは*出力専用* です。つまり、ストアード・プロシージャーはその値を入力として読み取るのではなく、値を戻すためにそのパラメーターを使用します。 | PARAM_OUT |
| INOUT パラメーター | INOUT パラメーターは*入力および出力* です。つまり、ストアード・プロシージャーはその値を入力として受け入れ、また値を戻すときにもそのパラメーターを使用します。 | PARAM_INOUT |

### 関連項目

234 ページの『executeStoredProcedure() でのストアード・プロシージャーの呼び出し』

CwDBStoredProcedureParam()、getValue()

## getValue()

現行ストアード・プロシージャー・パラメーターの値を検索します。

### 構文

```
Object getValue()
```

### パラメーター

なし。

### 戻り値

関連付けられている CwDBStoredProcedureParam パラメーターの値を Java Object として戻します。

### 例外

なし。

### 注記

getParamValue() メソッドは、パラメーター値を Java Object (Integer、Double、または String など) として戻します。OUT パラメーターに戻された値が JDBC NULL の場合は、getParamValue() は null 定数を戻します。

### 関連項目

CwDBStoredProcedureParam()、getParamType()

# 第 27 章 CxExecutionContext クラス

この章では、グローバル実行コンテキストに対して操作を行うメソッドについて説明します。グローバル実行コンテキストとは、特定のフローに関連したユーザー・アクセス可能なコンテキスト情報のホルダーです。これは、InterChange Server Express 定義の CxExecutionContext クラスによって表されます。現在のバージョンでは、マップ実行コンテキストとしてのマップ特定の実行情報のみが表示されます。 Map Designer は、マップ実行コンテキストにアクセスするための特殊な変数 (cwExecCtx) をマップのコード内に自動的に宣言します。しかし、コラボレーション内からマップを呼び出すときには、ユーザーが自身のグローバル実行コンテキストとマップ実行コンテキストをインスタンス化する必要があります。詳細については、210 ページの『ネイティブ・マップの呼び出し』を参照してください。

**注:** マップ実行コンテキストの詳細については、「*マップ開発ガイド*」を参照してください。

表 81 に、CxExecutionContext クラスのメソッドの要約を示します。

*表 81. CxExecutionContext メソッドの要約*

| メソッド | 説明 | ページ |
|---|---|---|
| CxExecutionContext() | グローバル実行コンテキストの新しいインスタンスを作成します。 | 433 |
| getContext() | グローバル実行コンテキストから、指定した実行コンテキストを検索します。 | 434 |
| setContext() | 特定の実行コンテキストをグローバル実行コンテキストの一部となるように設定します。 | 434 |

## 静的定数

CxExecutionContext クラスは、表 82 に示している静的定数を定義します。

*表 82. CxExecutionContext クラスで定義される静的定数*

| 定数名 | 意味 |
|---|---|
| MAPCONTEXT | 実行コンテキストがマップ特定であることを示すストリング定数。 |

## CxExecutionContext()

グローバル実行コンテキストの新しいインスタンスを作成します。

### 構文

```
CxExecutionContext()
```

### パラメーター

なし。

### 戻り値

グローバル実行コンテキストの新しいインスタンスを戻します。

### 注記

`CxExecutionContext()` コンストラクターはグローバル実行コンテキストを戻します。グローバル実行コンテキストは、コラボレーションからマップを呼び出す前にマップ実行コンテキストを保持しておくために必要です。

### 関連項目

210 ページの『ネイティブ・マップの呼び出し』

---

## getContext()

グローバル実行コンテキストから、指定した実行コンテキストを検索します。

### 構文

```
Object getContext(String contextName)
```

### パラメーター

*contextName* グローバル実行コンテキストから取得する実行コンテキストの名前。

### 戻り値

指定した実行コンテキストのインスタンスを戻します。

### 例

次の例では、グローバル実行コンテキストからマップ実行コンテキストを取得します。

```
(MapExeContext) mapExeContext = globalExeContext.getContext(
   CxExecutionContext.MAPCONTEXT);
```

### 関連項目

210 ページの『ネイティブ・マップの呼び出し』

---

## setContext()

特定の実行コンテキストをグローバル実行コンテキストの一部となるように設定します。

### 構文

```
void setContext(String contextName, Object context)
```

## パラメーター

*contextName* グローバル実行コンテキストに割り当てる特定の実行コンテキストの名前。現在は、MAPCONTEXT のみが有効な値です。

*context* 実行コンテキストの情報を含むオブジェクト。マップ実行コンテキストの場合は、この Object がタイプ MapExeContext を示します。

## 戻り値

なし。

## 例

次の例では、グローバル実行コンテキストの中にマップ実行コンテキストを保管します。

```
globalExeContext.setContext(CxExecutionContext.MAPCONTEXT,
   mapExeContext);
```

## 関連項目

210 ページの『ネイティブ・マップの呼び出し』

# 第 28 章 CollaborationException クラス

この章では、CollaborationException クラスのオブジェクトに対して操作を行うメソッドについて説明します。これらのオブジェクトは、コラボレーションの例外を表します。例外は、コラボレーション・オブジェクトの実行中に発生することがあります。シナリオで、これらの例外をキャッチおよび処理できます。コラボレーションが処理できる例外には 2 つのカテゴリーがあります。

- ビジネス・プロセス例外

  ビジネス・プロセス例外は、コラボレーション API メソッドを使用するコードから発生します。例えば、シナリオがビジネス・オブジェクト属性の値を設定し、コネクターに要求を送信する場合などに、ビジネス・プロセス例外が発生することがあります。
- ネイティブ Java 例外

  Java 例外は、ネイティブな Java メソッドを使用する開発者自身のコードが原因で発生します。コラボレーション・ランタイム環境は、開発者自身のコードから発生する Java 例外をキャッチし処理します。

表 83 に、この章で説明するメソッドをリストします。

*表 83. CollaborationException メソッドの要約*

| メソッド | 説明 | ページ |
|---|---|---|
| getMessage() | 現在の例外からメッセージ・テキストを取得します。 | 437 |
| getMsgNumber() | 現在の例外に関連付けられているテキストのメッセージ番号を取得します。 | 438 |
| getSubType() | 例外のサブタイプを取得します。 | 438 |
| getType() | コラボレーション例外タイプを取得します。 | 440 |
| toString() | 例外情報をストリングに書き込みます。 | 441 |

## getMessage()

例外オブジェクトのメッセージ・テキストを取得します。

### 構文

```
String getMessage()
```

### 戻り値

例外オブジェクトに関連付けられているメッセージが含まれている String。

### 注記

getMessage() メソッドは、currentException システム変数から例外テキストを抽出するために使用できるメソッドです。この例外テキストは、raiseException() の呼び出しに指定できます。これにより、例外発生の理由が 1 つ上の実行レベルにエスカレートされます。

**注:** toString() メソッドを使用して、現在の例外から例外タイプと例外テキストをフォーマット済みストリングとして取得できます。

## getMsgNumber()

例外オブジェクトに関連付けられているメッセージのメッセージ番号を取得します。

### 構文

```
int getMsgNumber()
```

### 戻り値

現在の例外に関連付けられている整数 (int) のメッセージ番号。例外のメッセージがメッセージ・ファイルのメッセージでない場合には、このメソッドはゼロ (0) を戻します。

### 注記

getMsgNumber() メソッドは、例外メッセージのメッセージ番号を取得するために使用できるメソッドです。raiseException() または logError() の呼び出しにこのメッセージ番号を渡すことができます。

## getSubType()

例外オブジェクトから例外サブタイプを取得します。

### 構文

```
String getSubType()
```

### 戻り値

現在の例外についての例外サブタイプが含まれている String。有効な例外サブタイプの詳細については、『注記』を参照してください。

### 注記

getSubType() メソッドは、現在の例外についての例外サブタイプを取得します。例外タイプにより原因が十分に示されていない例外の場合、例外サブタイプから詳細な情報を把握できることがあります。一般に例外サブタイプを使用する例外タイプを以下に示します。

- JavaException

コラボレーション・ランタイム環境は Java 例外をキャッチし、Java 例外の関連するタイプと一緒にコラボレーション例外の中にラップします。コラボレーションはコラボレーション例外に対して getSubType() を使用することで、Java 例外の元のタイプ (キャッチされた Java 例外のクラス名) を取得できます。ただし、これは通常は必要ありません。

- ServiceCallException

  ServiceCallException 例外タイプは、サービス呼び出しが失敗すると発生します。より堅固なコラボレーションを開発するために、サービス呼び出し失敗の原因を示す例外サブタイプを使用できます。有効な例外サブタイプには以下のものがあります。

| | |
|---|---|
| AppTimeOut | コネクターがアプリケーションとの通信を完了できなかったことを示します。 |
| AppLogOnFailure | コネクターがアプリケーションへログインできなかったことを示します。 |
| AppRetrieveByContentFailed | アプリケーションに対して実行された非キー値による Retrieve 操作で、条件に一致するものを取得できなかったことを示します。 |
| AppMultipleHits | アプリケーションが Retrieve 要求に応答して複数のエンティティーを取得したことを示します。 |
| AppBusObjDoesNotExist | アプリケーションに対して Retrieve 操作が実行されたが、ビジネス・オブジェクトを表すエンティティーがそのアプリケーション・データベース内に存在しないことを示します。 |
| AppRequestNotYetSent | 並列コネクター・エージェントの場合に、要求がエージェント・マスターのキューに入っているが、アプリケーションにディスパッチされていないため、要求を再送信できることを示します。詳細については、184 ページの『未送信サービス呼び出し要求』を参照してください。 |
| ServiceCallTransportException | トランスポートにエラーがあり、要求がアプリケーションに到達できたかどうかを正確に確認できないことを示します。詳細については、182 ページの『実行時のトランスポート関連例外の処理』を参照してください。 |
| AppUnknown | その他のサブタイプのいずれでもないタイプのエラーです。この例外サブタイプが存在する場合、サービス呼び出しで要求されたアプリケーション操作は、終了している場合と終了していない場合があります。 |

詳細については、181 ページの『特定のサービス呼び出し例外の処理』を参照してください。

**要確認:** 例外サブタイプ AppTimeOut、AppLogOnFailure、AppRetrieveByContent、AppMultipleHits、および AppUnknown は、失敗の原因を示すためにアダプターが戻す結果状況値に対応しています。古いアダプターでは、対応する結果状況値の一部がサポートされていないことがあります。Test Connector ツールを使用して、コラボレーションへバインドされているアダプターを厳密にテストし、アダプターが戻す結果状況値を確認してください。

## 例

このセクションでは、以下の例外タイプの例外サブタイプを取得する例を示します。

- JavaException

  次のコード例は、数学関数によってスローされた Java 例外を取得します。

```
//
// If the preceding division operation threw an exception,
// set the result to 0.
//
if (currentException.getType().equals("JavaException"))
   {
   String subType = currentException.getSubType();
   logWarning("Caught exception " + subType +
      ".  Setting result to 0.");
   result = 0;
   }
```

- ServiceCallException

  次の 2 つの ServiceCallException サブタイプの処理方法の例については、以下のセクションを参照してください。

  – AppRequestNotYetSent の例については、 184 ページの『未送信サービス呼び出し要求』を参照してください。

  – ServiceCallTransportException の例については、 182 ページの『実行時のトランスポート関連例外の処理』を参照してください。

# getType()

例外オブジェクトからコラボレーション例外サブタイプを取得します。

## 構文

```
String getType()
```

## 戻り値

現在の例外についての例外タイプが含まれている String。このストリング値を、以下のいずれかの例外タイプ静的変数の値と比較します。

| | |
|---|---|
| AnyException | 任意のタイプの例外。2 つの例外リンクがあり、1 つは例外の特定タイプをテストし、もう 1 つは AnyException をテストする場合、例外の特定タイプを検査するリンクが先に検査されます。現在の例外が特定の例外と一致しない場合、AnyException についてテストするリンクが次にチェックされます。 |
| AttributeException | 属性でアクセスする場合の問題です。例えば、String 属性に対してコラボレーションが getDouble() を呼び出した場合や、存在しない属性に対して getString() を呼び出した場合です。 |
| JavaException | コラボレーション・ロジックにおける Java コードの問題です。 |

| | |
|---|---|
| ObjectException | メソッドに渡されたビジネス・オブジェクトが無効でした。または、null オブジェクトにアクセスされました。 |
| OperationException | サービス呼び出しが適切に設定されませんでした。あるいは送信されませんでした。 |
| ServiceCallException | サービス呼び出しに失敗しました。例えば、コネクターやアプリケーションは使用できません。 |
| SystemException | InterChange Server Express の内部エラーです。 |
| TransactionException | トランザクション・コラボレーションの、トランザクションの振る舞いに関連するエラーです。例えば、ロールバックに失敗しました。あるいは、コラボレーションが差し戻しを適用できませんでした。 |

### 注記

getType() メソッドは、現在の例外から例外タイプを取得します。例外タイプは、例外の原因を示す String です。

### 例

コラボレーション例外タイプを取得し、raiseException() メソッドの呼び出しでこの例外タイプを使用する例を以下に示します。

```
String problem currentException.getType();
raiseException(problem, 1234);
```

## toString()

例外情報 (例外タイプやテキストなど) のフォーマットをストリングに設定します。

### 構文

```
String toString()
```

### パラメーター

*exception* 例外オブジェクトを保持している変数。

### 注記

toString() メソッドは、現在の例外についての例外情報を以下のようにフォーマット設定します。

*exceptionType*: *messageText*

上記の行の *exceptionType* は例外オブジェクトの例外タイプであり、*messageText* は例外テキストです。

**注:** getMessage() メソッドを使用して、現在の例外から例外テキストだけを取得できます。

### 例

次の例では、現行例外情報を String 変数である newmessage に書き込みます。

```
String newmessage = currentException.toString();
```

## 推奨されないメソッド

表 84 に、Process Designer Express のプレリリース・バージョンで使用可能であったが、現在は推奨されないメソッドを示します。これまでに Process Designer Express を使用したことがない場合は、このセクションは無視してください。詳細については、 399 ページの『推奨されないメソッド』を参照してください。

*表 84. 例外クラスの推奨されないメソッド*

| 以前のメソッド | 代替メソッド |
| --- | --- |
| `getText()` | `toString()` |

# 第 29 章 Filter クラス

この章では、Filter クラスのオブジェクトに対して操作を行うメソッドについて説明します。データのフィルター操作は、コラボレーション処理における共通のタスクです。これらのメソッドは、ビジネス・オブジェクトの特定の属性に含まれるデータをフィルター操作します。フィルター操作の結果を使用して、コラボレーションが特定のデータを持つビジネス・オブジェクトと同期する必要があるかを判断することができます。

表 85 に、この章で説明するメソッドをリストします。

*表 85. Filter メソッドの要約*

| メソッド | 説明 | ページ |
|---|---|---|
| Filter() | Filter クラスの新しいインスタンスを作成します。 | 444 |
| filterExcludes() | 指定された属性値が、指定された単一または複数の排他値と等しいかどうかを判断します。 | 445 |
| filterIncludes() | 指定された属性値が、指定された単一または複数の包含値と等しいかどうかを判断します。 | 446 |
| recurseFilter() | 指定された属性値が、指定された単一または複数の包含値または排他値と等しいかどうかを判断します。 | 447 |
| recursePreReqs() | 固有のビジネス・オブジェクトの指定された Vector 内で、指定されたビジネス・オブジェクト・タイプの Vector 位置を再帰的に検索します。 | 448 |

## Filter()

Filter の新しいインスタンスを作成します。

### 構文

```
Filter(BaseCollaboration baseCollab)
```

### パラメーター

*baseCollab*
現在のコラボレーション・インスタンスを指定します。

### 戻り値

新しくインスタンスを生成された Filter オブジェクトを戻します。

## filterExcludes()

指定された属性値が、排他値と等しいかどうかを判断します。

### 構文

```
boolean filterExcludes(String FilterAttributeValue,
                       String ExcludeValues)
```

### パラメーター

*FilterAttributeValue*
フィルター操作中の属性の値。

*ExcludesValues*
ビジネス・オブジェクトの同期を回避するために、コラボレーションがフィルター操作用として使用する値。

### 戻り値

*FilterAttributeValue* の値が *ExcludesValues* にリストされた値のいずれかに一致する場合、`False` を戻します。それ以外の場合、メソッドは `True` を戻します。

## filterIncludes()

指定された属性値が、包含値と等しいかどうかを判断します。

### 構文

```
boolean filterIncludes(String FilterAttributeValue, String IncludesValues)
```

### パラメーター

*FilterAttributeValue*
フィルター操作中の属性の値。

*IncludesValues*
ビジネス・オブジェクトの同期を許可するために、コラボレーションがフィルター操作用として使用する値。

### 戻り値

*FilterAttributeValue* の値が *IncludesValues* にリストされた値のいずれかに一致する場合、`True` を戻します。それ以外の場合、メソッドは `False` を戻します。

## recurseFilter()

指定された属性値が、排他値または包含値と等しいかどうかを判断します。

### 構文

```
boolean recurseFilter(BusObj busObj,
                      String filterAttribute,
                      boolean stopOnFail,
                      String includesValues,
                      String excludesValues)
```

### パラメーター

*busObj* フィルター操作対象のビジネス・オブジェクト・インスタンス。

*filterAttribute*

*includeValues* および *excludeValues* によって指定された値を比較するときに使用される、ビジネス・オブジェクト属性の名前。コラボレーションは、フィルター属性内の値を指定された包含値または排他値と比較して、ビジネス・オブジェクトの同期を回避または可能にします。

*stopOnFail*

*filterAttribute* 属性の値がフィルター操作基準を満たさない場合の、値の処理方法を指定します。

*includesValues*

ビジネス・オブジェクトの同期を許可するために、コラボレーションがフィルター操作用として使用する値。

*excludesValues*

ビジネス・オブジェクトの同期を回避するために、コラボレーションがフィルター操作用として使用する値。

### 戻り値

*filterAttribute* には、*includesValues* に指定された値が含まれるか、*excludesValues* に指定されていない値が含まれる場合、`True` を戻します。それ以外の場合、メソッドは `False` を戻します。

### 例外

CollaborationException — *filterAttribute* の属性値が包含値ではなく排他値として指定され、*stopOnFail* パラメーターが `True` に設定されている場合、この例外がスローされます。

## recursePreReqs()

固有のビジネス・オブジェクトの指定された Vector 内で、指定されたビジネス・オブジェクト・タイプの Vector 位置を再帰的に検索します。

### 構文

```
int recursePreReqs(String Type, Vector busObjs)
```

### パラメーター

*Type* *busObj* Vector 内で検索するビジネス・オブジェクトのタイプ。

*busObjs*
固有のビジネス・オブジェクト・タイプのビジネス・オブジェクトの Vector。

### 戻り値

*Type* で指定されたビジネス・オブジェクトが含まれる、*busObjs* Vector 内の位置を戻します。

### 例外

CollaborationException — コラボレーション構成プロパティー PREQ_*Type* が欠落している場合、この例外がスローされます。

# 第 30 章 Globals クラス

この章では、Globals クラスのオブジェクトに対して操作を行うメソッドについて説明します。

Globals クラスは、グローバル・ハッシュ・テーブルを保持して非同期イベント処理をサポートします。コラボレーションでは、別のコラボレーションやコネクターからの通信データを送受信し、指定された時間適切な応答を待ってから、同じスレッド内のイベント処理を続行します。

表 86 に、Globals クラスのメソッドをリストします。

*表 86. Globals クラスのメソッドの要約*

| メソッド | 説明 | ページ |
|---|---|---|
| Globals() | Globals クラスの新しいインスタンスを作成します。 | 450 |
| callMap() | DtpMapService.runMap API にラッパーを提供して、コラボレーション内からのマップの呼び出しを容易にします。 | 451 |

## Globals()

Globals クラスの新しいインスタンスを作成します。

### 構文

```
Globals(BaseCollaboration baseCollab)
```

### パラメーター

*baseCollab*
現在のコラボレーション・インスタンスを指定します。

### 戻り値

新しくインスタンスを生成された Globals オブジェクトを戻します。

## callMap()

DtpMapService.runMap API にラッパーを提供して、コラボレーション内からのマップの呼び出しを容易にします。

### 構文

```
BusObj callMap(String mapName, BusObj srcBusObj)
```

### パラメーター

*mapName*
実行するマップの名前を指定します。

*srcBusObj*
マップのソース・ビジネス・オブジェクトを指定します。

### 戻り値

*mapName* で指定されたマップの宛先ビジネス・オブジェクトを戻します。

### 例外

CollaborationException — *mapName* マップの実行中にエラーが発生すると、スローされます。

# 第 31 章 SmartCollabService クラス

この章では、SmartCollabService クラスのオブジェクトに対して操作を行うメソッドについて説明します。このクラスは、ビジネス・オブジェクト内の配列属性の分割、マージ、および集約を単純化するための一連のメソッドを提供します。

表 87 に、SmartCollabService クラスで提供されるメソッドをリストします。

*表 87. SmartCollabService メソッドの要約*

| メソッド | 説明 | ページ |
|---|---|---|
| SmartCollabService() | SmartCollabService クラスの新しいインスタンスを作成します。 | 453 |
| doAgg() | ユーザー指定の基準および属性に基づいて、類似のコンテナー属性を 1 つのコンテナー属性に集約します。 | 454 |
| doMergeHash() | ビジネス・オブジェクトを収集し、分割レベルで指定された新規の親ビジネス・オブジェクトの下にグループ化します。ビジネス・オブジェクトは、キー属性 (複数可) で指定された類似内容によりグループ化されます。 | 454 |
| doRecursiveAgg() | ユーザー指定の基準および属性に基づいて、類似の階層コンテナー属性を 1 つのコンテナー属性に再帰的に集約します。 | 455 |
| doRecursiveSplit() | 特定レベルのビジネス・オブジェクト階層からコンテナー・ビジネス・オブジェクトを検索し、オプションでトップレベルのビジネス・オブジェクト内に戻します。 | 456 |
| getKeyValues() | キー属性 (複数可) に指定されたコンマ区切り値に基づいて、ハッシュ・テーブルで使用されるビジネス・オブジェクトのキー値を計算します。 | 456 |
| merge() | ビジネス・オブジェクトの集合を 1 つのトップレベルのビジネス・オブジェクトの下にマージします。 | 457 |
| split() | 分割レベルで指定されたように、ビジネス・オブジェクトを複数のコンテナー・ビジネス・オブジェクトに分割します。 | 457 |

## SmartCollabService()

SmartCollabService の新しいインスタンスを作成します。

### 構文

```
SmartCollabService()
SmartCollabService(com.crossworlds.BaseCollaboration baseCollab)
```

### パラメーター

*baseCollab*
現在のコラボレーション・インスタンスを指定します。

### 戻り値

新しくインスタンスを生成された SmartCollabService オブジェクトを戻します。

---

## doAgg()

ユーザー指定の基準および集約する属性のリストに基づいて、同類のコンテナー属性を単一のコンテナー属性に集約します。

### 構文

```
BusObj doAgg(BusObj inBusObj,
          String Level,
          String KeyAttr,
          String Attr)
```

### パラメーター

*inBusObj*
集約するビジネス・オブジェクトを指定します。

*Level* ビジネス・オブジェクトのタイプ (ビジネス・オブジェクト定義) および集約する配列属性の名前を構成する、集約レベルを指定します。名前はピリオド (.) で区切ります。

*KeyAttr*
集約に使用されるビジネス・オブジェクトのキー属性を指定します。

*Attr* 集約される属性を指定します。複数の名前はコンマ (,) で区切ります。

### 戻り値

集約されたビジネス・オブジェクトを戻します。

### 例外

CollaborationException — ビジネス・オブジェクト属性の集約中にエラーが発生すると、スローされます。

---

## doMergeHash()

ビジネス・オブジェクトの集合を、分割レベルで指定されたように新規の親ビジネス・オブジェクトの下にグループ化します。ビジネス・オブジェクトは、キー属性 (複数可) の類似内容によりグループ化されます。

### 構文

```
java.util.Vector doMergeHash(java.util.Vector BusObj,
          String Level,
          String KeyAttr)
```

## パラメーター

*BusObj* マージするビジネス・オブジェクトの集合を含む Vector を指定します。

*Level* 親ビジネス・オブジェクトのタイプ (ビジネス・オブジェクト定義) 、および子ビジネス・オブジェクトを保持するよう指定された属性を構成する、マージ・レベルを指定します。名前はピリオド (.) で区切ります。

*KeyAttr*
マージ基準として使用されるビジネス・オブジェクト属性を指定します。値はコンマ (,) で区切ります。

## 戻り値

マージされたビジネス・オブジェクトの Vector を戻します。

## 例外

CollaborationException — ビジネス・オブジェクトのマージ中にエラーが発生すると、スローされます。

---

# doRecursiveAgg()

ユーザー指定の基準および集約する属性のリストに基づいて、同類の階層配列属性を単一のコンテナー属性に再帰的に集約します。

## 構文

```
BusObj doRecursiveAgg(BusObj inBusObj,
          String Level,
          String KeyAttr,
          String Attr)
```

## パラメーター

*inBusObj*
集約するビジネス・オブジェクトを指定します。

*Level* ビジネス・オブジェクトのタイプ (ビジネス・オブジェクト定義) および集約する配列属性の名前を構成する、集約レベルを指定します。名前はピリオド (.) で区切ります。

*KeyAttr*
集約に使用されるビジネス・オブジェクトのキー属性を指定します。

*Attr* 集約される属性を指定します。複数の名前はコンマ (,) で区切ります。

## 戻り値

集約されたビジネス・オブジェクトを戻します。

## 例外

CollaborationException — ビジネス・オブジェクト属性の集約中にエラーが発生すると、スローされます。

## doRecursiveSplit()

ビジネス・オブジェクト階層内の特定レベルからコンテナー・ビジネス・オブジェクトを検索します。

### 構文

```
java.util.Vector doRecursiveSplit(BusObj inBusObj,
          String Level)
java.util.Vector doRecursiveSplit(BusObj inBusObj,
          String Level,
          boolean KeepParents)
```

### パラメーター

*inBusObj*
メソッドによって配列属性を分割される、トップレベル・ビジネス・オブジェクトを指定します。

*Level* ビジネス・オブジェクトの分割先配列属性へのパスを指定します。値はピリオド (.) で区切ります。

*KeepParents*
分割されたビジネス・オブジェクトを親ビジネス・オブジェクト内に戻すか、スタンドアロン・オブジェクトとして戻すかを指定します。分割されたオブジェクトを親ビジネス・オブジェクト内に戻す場合は、パラメーターを `True` に設定します。

### 戻り値

ビジネス・オブジェクトの Vector を戻します。

### 例外

CollaborationException — ビジネス・オブジェクトの分割中にエラーが発生すると、スローされます。

## getKeyValues()

ハッシュ・テーブルで使用するビジネス・オブジェクトのキー値を計算します。

### 構文

```
String getKeyValues(BusObj inBusObj,
              String KeyAttr)
```

### パラメーター

*inBusObj*
キー値を計算するビジネス・オブジェクトを指定します。

*KeyAttr*
メソッドが操作する属性名を指定します。複数の値はコンマ (,) で区切ります。

### 戻り値

Java ハッシュ・テーブルで使用されるキー値を戻します。

### 例外

CollaborationException — ビジネス・オブジェクトのキー値の計算中にエラーが発生すると、スローされます。

## merge()

ビジネス・オブジェクトの集合を 1 つのトップレベルのビジネス・オブジェクトの下にマージします。

### 構文

```
BusObj merge(java.util.Vector BusObjs,
          String Level)
BusObj merge(java.util.Vector BusObjs,
          String Attr,
          BusObj mergeBusObj)
```

### パラメーター

*BusObjs*
マージする子ビジネス・オブジェクトの集合を指定します。

*Level* 子ビジネス・オブジェクトがマージされるビジネス・オブジェクト・タイプおよびその配列属性名を構成する、マージ・レベルを指定します。名前はピリオド (.) で区切ります。

*Attr* 子ビジネス・オブジェクトのマージ先となる *mergeBusObj* ビジネス・オブジェクト内の配列属性名を指定します。

*mergeBusObj*
マージされた子ビジネス・オブジェクトの集合を保持するトップレベルのビジネス・オブジェクトを指定します。

### 戻り値

マージされた子ビジネス・オブジェクトの集合が格納される、(新規または指定された) トップレベルのビジネス・オブジェクトを戻します。

### 例外

CollaborationException — ビジネス・オブジェクトのマージ中にエラーが発生すると、スローされます。

## split()

1 つのビジネス・オブジェクトを、分割レベルで指定された数のコンテナー・ビジネス・オブジェクトに分割します。

## 構文

```
Vector split(BusObj inBusObj,
          String Attr)
```

## パラメーター

*inBusObj*

split() メソッドが操作する親レベルのビジネス・オブジェクトを指定します。

*Attr* ビジネス・オブジェクトの分割先配列属性の名前を指定します。

## 戻り値

ビジネス・オブジェクトの Vector、親ビジネス・オブジェクトの配列属性内の各子ビジネス・オブジェクトに対して 1 つのビジネス・オブジェクトを戻します。

## 例外

CollaborationException — ビジネス・オブジェクトの分割中にエラーが発生すると、スローされます。

# 第 32 章 StateManagement クラス

StateManagement クラスにより、コラボレーションの状態およびビジネス・オブジェクトの永続性を管理できます。状態の管理とビジネス・オブジェクトの永続性は、長期存続ビジネス・プロセスのインプリメントに必要です。

コラボレーションの状態は、CxCollabState データベース表で保管、検索、更新、削除などの操作を実行することによって管理します。この表には、以下の属性が含まれます。

- Id
- Verb
- CollabObjName
- BusinessObjectType
- PropDocID
- State
- Retry
- BeginTime
- TimeOut

ビジネス・オブジェクトの永続性は、CxCollabStateBO データベース表で保管、検索、更新、削除などの操作を実行することによって管理します。この表には、以下の属性が含まれます。

- CollabObjName
- Verb
- PropDocID
- BusinessObjectType
- BusObj

表 88 で、StateManagement クラスのメソッドについて説明します。

*表 88. StateManagement メソッドの要約*

| メソッド | 説明 | ページ |
|---|---|---|
| beginTransaction() | トランザクションの開始をマークします。 | 460 |
| commit() | トランザクションを確定します。 | 460 |
| deleteBO() | CxCollabStateBO データベース表から存続するビジネス・オブジェクトを削除します。 | 460 |
| deleteState() | CxCollabState データベース表からエントリーを削除します。 | 461 |
| persistBO() | CxCollabStateBO データベース表内にビジネス・オブジェクトを存続させます。 | 462 |
| recoverBO() | CxCollabStateBO データベース表内に存続しているビジネス・オブジェクトをリカバリーします。 | 462 |
| releaseDBConnection() | データベース接続を解放します。 | 463 |

表 88. *StateManagement* メソッドの要約 (続き)

| メソッド | 説明 | ページ |
|---|---|---|
| resetData() | ブール変数 *bTranStarted* の値をリセットします。 | 463 |
| retrieveState() | CxCollabState データベース表に格納されている最新の再試行回数値を検索します。 | 464 |
| saveState() | CxCollabState データベース表にコラボレーションのプロセス・パラメーターを保管します。 | 464 |
| setDBConnection() | データベース接続を設定します。 | 465 |
| StateManagement() | StateManagement オブジェクトを作成および初期化します。 | 465 |
| updateBO() | CxCollabStateBO データベース表に存続するビジネス・オブジェクトを更新します。 | 466 |
| updateState() | CxCollabState データベース表内の再試行回数値を更新します。 | 466 |

## beginTransaction()

トランザクションの開始をマークします。

### 構文

```
public void beginTransaction()
```

### 例外

CwDBConnectionException — StateManagement クラスがトランザクションを開始できない場合、この例外がスローされます。

## commit()

トランザクションを確定します。

### 構文

```
pubic void commit()
```

### 例外

以下の例外をスローします。

- CwDBTransactionException — StateManagement クラスがトランザクションを確定できない場合、この例外が発生します。
- CwDBConnectionException — StateManagement クラスがトランザクションを開始できない場合、この例外が発生します。

## deleteBO()

CxCollabStateBO データベース表から存続するビジネス・オブジェクトを削除します。

### 構文

```
public void deleteBO(java.lang.String Verb,
                     java.lang.String PropDocID,
                     java.lang.String BusinessObjectType)
```

### パラメーター

*Verb*　使用する動詞を指定します。ここでは、値を Delete にする必要があります。

*BusinessObjectType*
削除するビジネス・オブジェクトのタイプを指定します。

*PropDocID*
削除するビジネス・オブジェクトの ID を指定します。

### 例外

以下の例外をスローします。

- CwDBSQLException — StateManagement クラスが SQL 照会を実行できない場合にスローされます。
- CollaborationException — 渡されたパラメーターの値が null であるか、メソッド内でその他の不明な例外が発生した場合にスローされます。

---

## deleteState()

CxCollabState データベース表からエントリーを削除します。

### 構文

```
public void deleteState(java.lang.String Verb,
                        java.lang.String BusinessObjectType,
                        java.lang.String PropDocID,
                        int stateValue)
```

### パラメーター

*Verb*　使用する動詞を指定します。ここでは、値を Delete にする必要があります。

*BusinessObjectType*
ビジネス・オブジェクトのタイプを指定します。

*PropDocID*
ビジネス・オブジェクトの ID を指定します。

*stateValue*
状態の数値タグを指定します。

### 例外

以下の例外をスローします。

- CwDBSQLException — StateManagement クラスが SQL 照会を実行できない場合にスローされます。

- CollaborationException — 渡されたパラメーターの値が null であるか、メソッド内でその他の不明な例外が発生した場合にスローされます。

## persistBO()

CxCollabStateBO データベース表内にビジネス・オブジェクトを存続させます。

### 構文

```
public void persistBO(java.lang.String CollabObjName,
                      java.lang.String Verb,
                      java.lang.String PropDocID,
                      java.lang.String BusinessObjectType,
                      com.crossworlds.BusObj BOtoSave)
```

### パラメーター

*CollabObjName*
コラボレーションの名前を指定します。

*Verb* 使用する動詞を指定します。ここでは、値を `Create` にする必要があります。

*BusinessObjectType*
ビジネス・オブジェクトのタイプを指定します。

*PropDocID*
ビジネス・オブジェクトの ID を指定します。

*BOtoSave*
存続させるビジネス・オブジェクトの名前を指定します。

### 例外

以下の例外をスローします。

- CwDBSQLException — StateManagement クラスが SQL 照会を実行できない場合にスローされます。
- CollaborationException — 渡されたパラメーターの値が null であるか、メソッド内でその他の不明な例外が発生した場合にスローされます。
- BOFormatException — ビジネス・オブジェクトをリカバリー可能ストリングに変換中に、データ・ハンドラーにエラーが発生した場合にスローされます。

### 注記

persistBO() メソッドは、国際化対応コラボレーションで使用することができます。DBCS 文字と MBCS 文字の両方を、CxCollabStateBO データベース表に正確に格納できます。また、ロケール情報を格納するため、CxCollabStateBO の表に Locale 列が追加されました。

## recoverBO()

CxCollabStateBO データベース表内に存続していたビジネス・オブジェクトをリカバリーします。

### 構文

```
public com.crossworlds.BusObj recoverBO(java.lang.String Verb,
                    java.lang.String PropDocID,
                    java.lang.String BusinessObjectType)
```

### パラメーター

*Verb*　使用する動詞を指定します。ここでは、値を Retrieve にする必要があります。

*BusinessObjectType*
ビジネス・オブジェクトのタイプを指定します。

*PropDocID*
ビジネス・オブジェクトの ID を指定します。

### 戻り値

CxCollabStateBO データベース表に保管されているビジネス・オブジェクトを戻します。

### 例外

以下の例外をスローします。

- CwDBSQLException — StateManagement クラスが SQL 照会を実行できない場合にスローされます。
- CollaborationException — 渡されたパラメーターの値が null であるか、メソッド内でその他の不明な例外が発生した場合にスローされます。
- BOFormatException — リカバリーされたストリングを InterChange Server Express ビジネス・オブジェクトに変換中に、データ・ハンドラーにエラーが発生した場合にスローされます。

## releaseDBConnection()

データベース接続を解放します。

### 構文

```
public void releaseDBConnection()
```

### 注記

getDBConnection() メソッドを使用して確立された接続は、再利用のために静的変数に保管することはできません。接続の漏えいを回避するため、接続は常に、プロセスの終了時に暗黙的に解放されます。これが実行されるように、StateManagement クラス内で使用される接続オブジェクトは、まず releaseDBConnection() メソッドを使用して解放する必要があります。

## resetData()

ブール変数 *bTranStarted* の値をリセットします。

### 構文

```
public void resetData()
```

## retrieveState()

CxCollabState データベース表に格納されている最新の再試行回数値を検索します。

### 構文

```
public int retrieveState(java.lang.String Verb,
                         java.lang.String BusinessObjectType,
                         java.lang.String PropDocID,
                         int State)
```

### パラメーター

*Verb* 使用する動詞を指定します。ここでは、値を Retrieve にする必要があります。

*BusinessObjectType*
ビジネス・オブジェクトのタイプを指定します。

*PropDocID*
ビジネス・オブジェクトの ID を指定します。

*State* 状態値を指定します。

### 例外

以下の例外をスローします。

- CwDBSQLException — StateManagement クラスがストアード・プロシージャーを実行できない場合にスローされます。
- CollaborationException — 渡されたパラメーターの値が null であるか、メソッド内でその他の不明な例外が発生した場合にスローされます。

## saveState()

CxCollabState データベース表にコラボレーションのプロセス・パラメーターを保管します。

### 構文

```
public void saveState(java.lang.String CollabObjName,
                      java.lang.String Verb,
                      java.lang.String BusinessObjectType,
                      java.lang.String PropDocID,
                      int stateValue,
                      int retry,
                      double hours_to_timeout)
```

### パラメーター

*CollabObjName*
コラボレーションの名前を指定します。

*Verb*　使用する動詞を指定します。ここでは、値を `Create` にする必要があります。

*BusinessObjectType*
ビジネス・オブジェクトのタイプを指定します。

*PropDocID*
ビジネス・オブジェクトの ID を指定します。

*stateValue*
状態の数値タグを指定します。

*retry*　ビジネス・オブジェクトの再試行値を指定します。

*hours_to_timeout*
プロセスがタイムアウトになるまでの時間数を指定します。データは Date データ型として CxCollabState 表に保管されます。

### 例外

以下の例外をスローします。

- CwDBSQLException — StateManagement クラスが SQL 照会を実行できない場合にスローされます。
- CollaborationException — 渡されたパラメーターの値が null であるか、メソッド内でその他の不明な例外が発生した場合にスローされます。

## setDBConnection()

データベース接続を設定します。

### 構文

```
public void setDBConnection(com.crossworlds.CwDBConnection DBConn)
```

### パラメーター

*DBConn*
データベース接続を指定します。

### 例外

*DBConn* の値が null の場合、CwDBConnectionFactoryException がスローされます。

## StateManagement()

StateManagement オブジェクトを作成し、初期化します。

### 構文

```
StateManagement()
```

## updateBO()

CxCollabStateBO データベース表に存続するビジネス・オブジェクトを更新します。ビジネス・オブジェクトは区切りデータ・ハンドラーをパススルーして、ストリングに変換されます。結果のストリングは、データベース表に保管されます。

### 構文

```
public void updateBO(java.lang.String CollabObjName,
                     java.lang.String Verb,
                     java.lang.String PropDocID,
                     java.lang.String BusinessObjectType,
                     com.crossworlds.BusObj BOtoUpdate)
```

### パラメーター

*CollabObjName*
コラボレーションの名前を指定します。

*Verb* 使用する動詞を指定します。ここでは、値を `Update` にする必要があります。

*BusinessObjectType*
ビジネス・オブジェクトのタイプを指定します。

*PropDocID*
ビジネス・オブジェクトの ID を指定します。

*BOtoUpdate*
更新するビジネス・オブジェクトの名前を指定します。

### 例外

以下の例外をスローします。

- CwDBSQLException — StateManagement クラスが SQL 照会を実行できない場合にスローされます。
- CollaborationException — 渡されたパラメーターの値が null であるか、メソッド内でその他の不明な例外が発生した場合にスローされます。
- BOFormatException — ビジネス・オブジェクトをストリングに変換中に、データ・ハンドラーにエラーが発生した場合にスローされます。

### 注記

updateBO() メソッドは、国際化対応コラボレーションで使用することができます。CxCollabStateBO データベース表には DBCS 文字および MBCS 文字が正しく保管されます。また、ロケール情報を保管するための Locale 列が含まれます。

## updateState()

CxCollabState データベース表内の再試行回数値を更新します。2 時間のタイムアウトが発生し、再試行が実行される場合、このメソッドは、表の 24 時間のタイムアウト列にある再試行回数を更新します。

## 構文

```
public void updateState(java.lang.String CollabObjName,
                java.lang.String Verb,
                java.lang.String BusinessObjectType,
                java.lang.String PropDocID,
                int stateValue,
                int retry)
```

## パラメーター

*CollabObjName*
コラボレーションの名前を指定します。

*Verb* 使用する動詞を指定します。ここでは、値を `Update` にする必要があります。

*BusinessObjectType*
ビジネス・オブジェクトのタイプを指定します。

*PropDocID*
ビジネス・オブジェクトの ID を指定します。

*stateValue*
状態の数値タグを指定します。

*retry* ビジネス・オブジェクトの再試行値を指定します。

## 例外

以下の例外をスローします。

- CwDBSQLException — StateManagement クラスが SQL 照会を実行できない場合にスローされます。
- CollaborationException — 渡されたパラメーターの値が null であるか、メソッド内でその他の不明な例外が発生した場合にスローされます。

# 第 5 部 付録

# 付録. 標準的なコラボレーションに関する情報

CrossWorlds が開発したコラボレーションのほとんどは CollaborationFoundation テンプレートからビジネス・プロセスを継承しているため、この付録では、すべてのコラボレーションに共通するビジネス・プロセスについて説明します。コラボレーションの各ユーザー・マニュアルでは、これらのプロセスに関する説明の参照先として本書を指定しています。CollaborationFoundation については、 34 ページの『CollaborationFoundation テンプレート』を参照してください。

この付録では、コラボレーションの設計上の指針について説明します。この指針に従うことで、動作が良好で再使用可能なコラボレーションを設計することができます。内容は、次のとおりです。

- 『コラボレーション・テンプレートの標準的なプロセス』
- 478 ページの『コラボレーション・テンプレートの標準プロパティー』

## コラボレーション・テンプレートの標準的なプロセス

このセクションでは、標準的な WebSphere Business Integration Server Express コラボレーションのビジネス・プロセス・フローを説明します。標準のプロセスは、以下のようになります。

- Retrieve プロセス
- USE_RETRIEVE プロセス
- フィルター操作プロセス
- ADDITIONAL_RETRIEVE プロセス
- エラー処理の E メール・プロセス

ビジネス・プロセスのフローは、トリガーとなる動詞およびコラボレーション・オブジェクトの構成プロパティーの値によって異なります。

### Retrieve プロセス

コラボレーションは、Retrieve 動詞を使用するトリガー・ビジネス・オブジェクトを受信できます。ソース・アプリケーションのビジネス・オブジェクトは、属性に相互参照のキー値が含まれている場合にこの動詞を使用すると想定されています。ビジネス・オブジェクトのマップおよび宛先アプリケーションのコネクターは、基本キーを使用して宛先アプリケーションからビジネス・オブジェクトを戻します。

472 ページの図 81 に、トリガー動詞が Retrieve である場合にビジネス・オブジェクトを検索するコラボレーションのプロセスを示します。

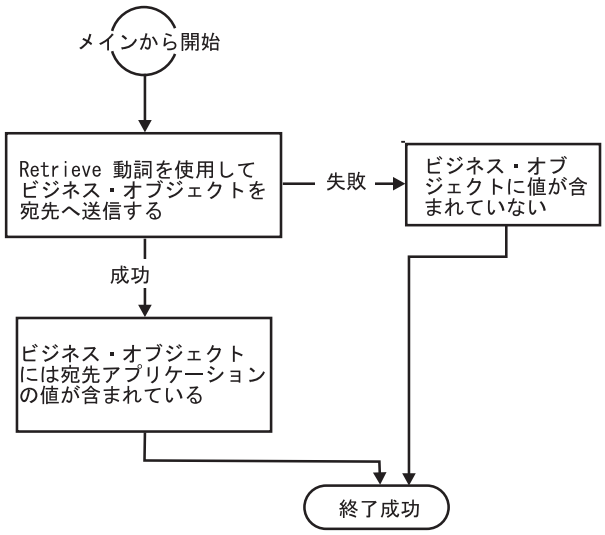



*図 81. Retrieve プロセス*

## Use_Retrieve プロセス

473 ページの図 82 に、USE_RETRIEVE 構成プロパティーが true と評価される場合のコラボレーションの Use_Retrieve Process を示します。構成プロパティーについては、TABLEA3 のエントリーを参照してください。

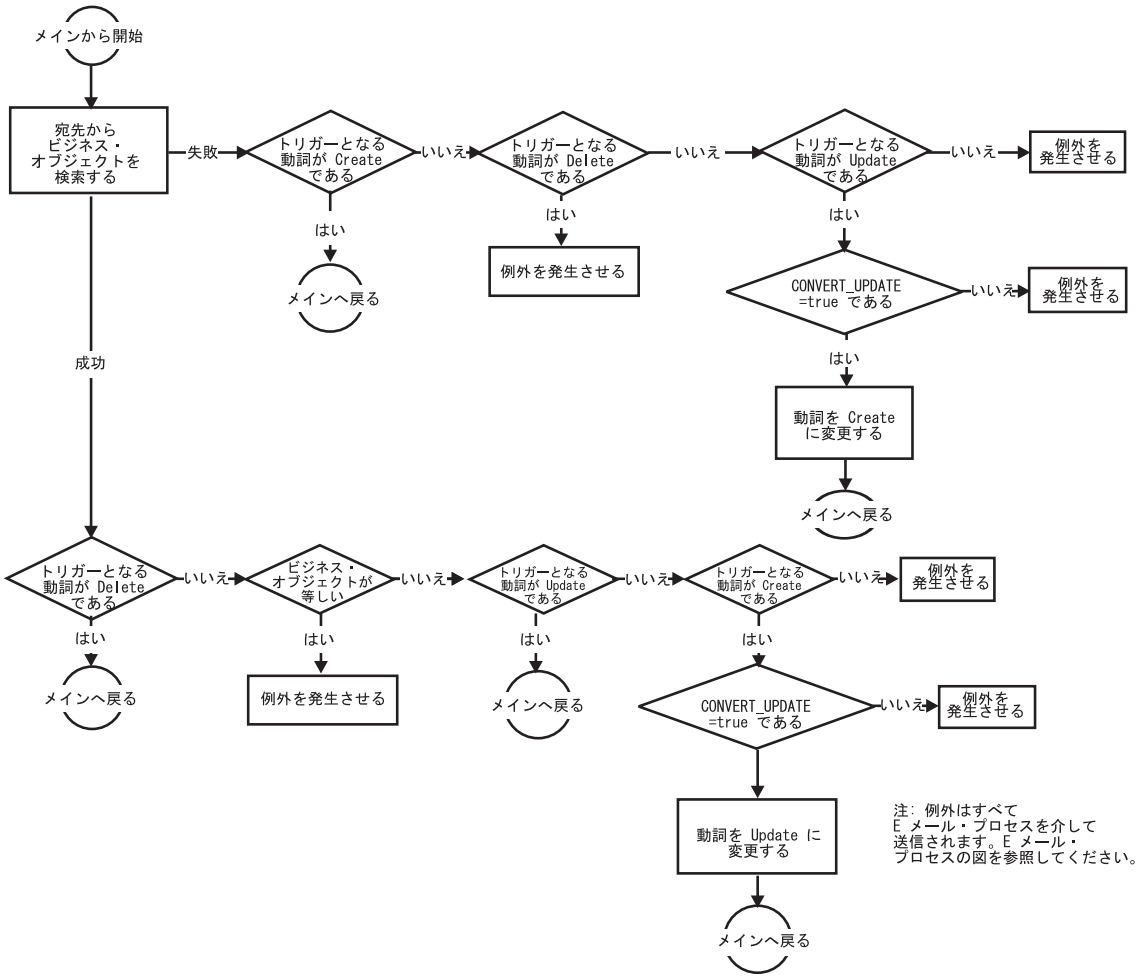



*図 82. Use_Retrieve プロセス*

474 ページの表 89 に、トリガー動詞が Create である場合の USE_RETRIEVE プロパティーと CONVERT_CREATE プロパティーとの設定の関係を示します。USE_RETRIEVE プロパティーと CONVERT_UPDATE プロパティーの関係もこれと同様です。

表 89. *USE_RETRIEVE プロパティーと CONVERT_CREATE プロパティーとの関係*

| USE_RETRIEVE の値 | CONVERT_CREATE の値 | アクション |
|---|---|---|
| true | True | 宛先アプリケーションからビジネス・オブジェクトを検索します。<br><br>宛先アプリケーションがビジネス・オブジェクトを戻さない場合は、ビジネス・オブジェクトを Create 動詞と共に宛先アプリケーションへ送信します。<br><br>宛先アプリケーションがビジネス・オブジェクトを戻した場合は、ソース・ビジネス・オブジェクトと宛先ビジネス・オブジェクトが同一であるかどうかを確認します。<br>• 同一である場合は、ビジネス・オブジェクトが同一であり処理が停止されることを示す例外を生成します。<br>• 同一でない場合は、Create 動詞を Update 動詞に変換し、ビジネス・オブジェクトを宛先に送信します。 |
| false | True | ビジネス・オブジェクトを Create 動詞と共に宛先アプリケーションへ送信します。<br>• 宛先アプリケーションがビジネス・オブジェクトの作成に失敗した場合は、Create 動詞を Update 動詞に変換してビジネス・オブジェクトを再送します。<br>• 宛先アプリケーションがビジネス・オブジェクトの作成に成功した場合は、処理を継続します。 |

表 89. *USE_RETRIEVE プロパティーと CONVERT_CREATE プロパティーとの関係 (続き)*

| USE_RETRIEVE の値 | CONVERT_CREATE の値 | アクション |
|---|---|---|
| false | False | ビジネス・オブジェクトを Create 動詞と共に宛先アプリケーションへ送信します。<br>• 宛先アプリケーションがビジネス・オブジェクトの作成に失敗した場合は、例外を生成して処理を停止します。<br>• 宛先アプリケーションがビジネス・オブジェクトの作成に成功した場合は、処理を継続します。 |
| true | False | 宛先アプリケーションからビジネス・オブジェクトを検索します。<br><br>宛先アプリケーションがビジネス・オブジェクトを戻さない場合は、ビジネス・オブジェクトを Create 動詞と共に宛先アプリケーションへ送信します。<br><br>宛先アプリケーションがビジネス・オブジェクトを戻した場合は、ソース・ビジネス・オブジェクトと宛先ビジネス・オブジェクトが同一であるかどうかを判別します。<br>• 同一である場合は、CONVERT_CREATE プロパティーが false と評価され、処理が停止されることを示す例外を生成します。<br>• 同一でない場合は、例外が生成され、処理が停止されます。 |

## フィルター操作プロセス

476 ページの図 83 に、指定された属性で値に基づいてビジネス・オブジェクトをフィルター操作するコラボレーションのプロセスを示します。

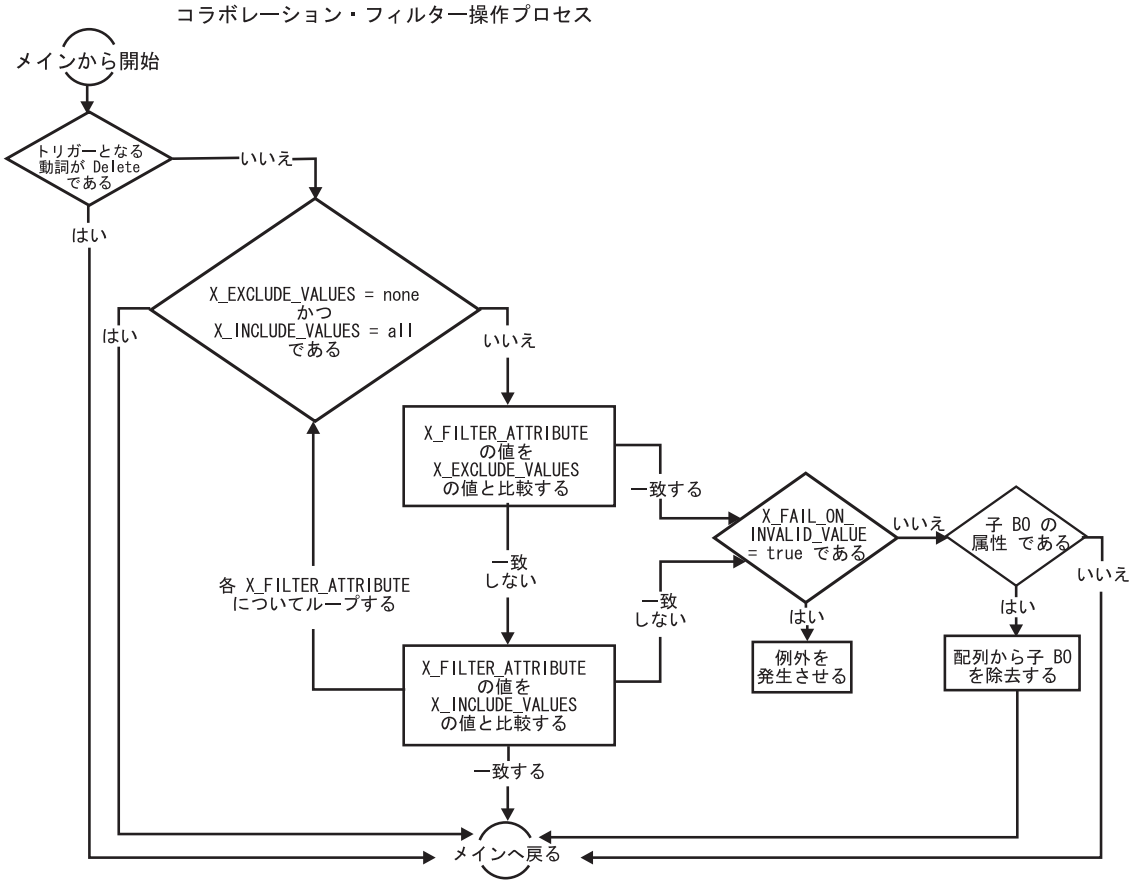



*図 83. フィルター操作プロセス*

上の図は、「1_」ではなく「X_」を持つフィルター構成プロパティーを示しています。これは、複数の属性でフィルター操作を行うために、フィルター構成プロパティーのセットをコラボレーションに追加できることを示しています。フィルター・プロパティーの追加の詳細については、37 ページの『トリガー・ビジネス・オブジェクトでのデータのフィルター操作』および 478 ページの表 90 を参照してください。

1_FAIL_ON_INVALID_VALUE が false と評価された場合のコラボレーションの振る舞いを理解するため、BusObjA というトップレベルのビジネス・オブジェクトに BusObjB という子ビジネス・オブジェクトの配列が含まれていると想定します。また、各 BusObjB 子ビジネス・オブジェクトに BusObjC という子ビジネス・オブジェクトの配列が含まれていると想定します。BusObjB ビジネス・オブジェクト内の Type という属性でフィルター操作するには、1_FILTER_ATTRIBUTE の値を BusObjA.BusObjB.Type に設定します。Type に「Non-standard」の値が含まれているビジネス・オブジェクトをすべて除外するには、1_EXCLUDE_VALUES プロパティーの値を「Non-standard」に設定します。

次に、BusObjB の配列内でビジネス・オブジェクトのいずれかの Type 属性に「Non-standard」の値があり、1_FAIL_ON_INVALID_VALUE が false に評価されたとします。コラボレーションは、このビジネス・オブジェクトおよびその子ビジネス・オブジェクトを処理から除去します。

**注:** フィルター操作に失敗した属性がトップレベルのビジネス・オブジェクトに含まれている場合、コラボレーションはビジネス・オブジェクト全体の処理を続行します。

## Additional Retrieve プロセス

コラボレーションの ADDITIONAL_RETRIEVE プロパティーが true に評価されると、コラボレーションは、データとの同期が成功した後に、宛先アプリケーションからビジネス・オブジェクトを検索します。このプロパティーは、ソース・アプリケーションが宛先アプリケーションから完全な値を持つビジネス・オブジェクトを受け取る必要があるが、宛先アプリケーションのコネクターがデータの作成または更新後に完全なビジネス・オブジェクトを戻さないという場合に役立ちます。ADDITIONAL_RETRIEVE 構成プロパティーの詳細については、 478 ページの『コラボレーション・テンプレートの標準プロパティー』を参照してください。

図 84 に、コラボレーションの Additional Retrieve プロセスを示します。

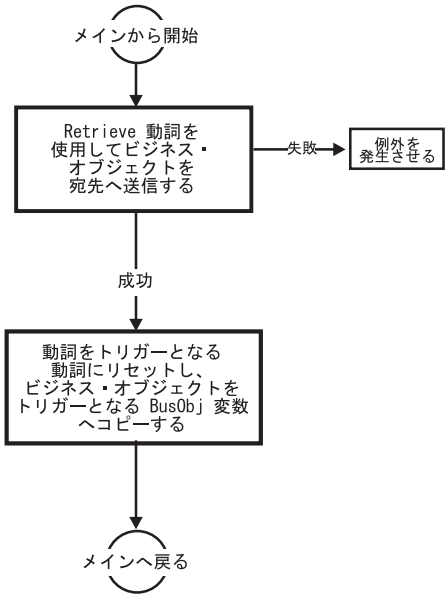



*図 84. Additional Retrieve プロセス*

## エラー処理の E メール・プロセス

コラボレーションの SEND_EMAIL プロパティーおよび INFORMATIONAL_EXCEPTIONS プロパティーの設定は、コラボレーションが次のいずれかを実行するかどうかを決定します。

- エラーが発生した場合に、指定されたアドレスに E メールを送信する
- 特定の例外が発生した場合に、失敗で終了する

図 85 に、コラボレーションで E メールを送信するプロセスおよび特定の例外によってコラボレーションを失敗で終了するかどうかを決定するプロセスを示します。

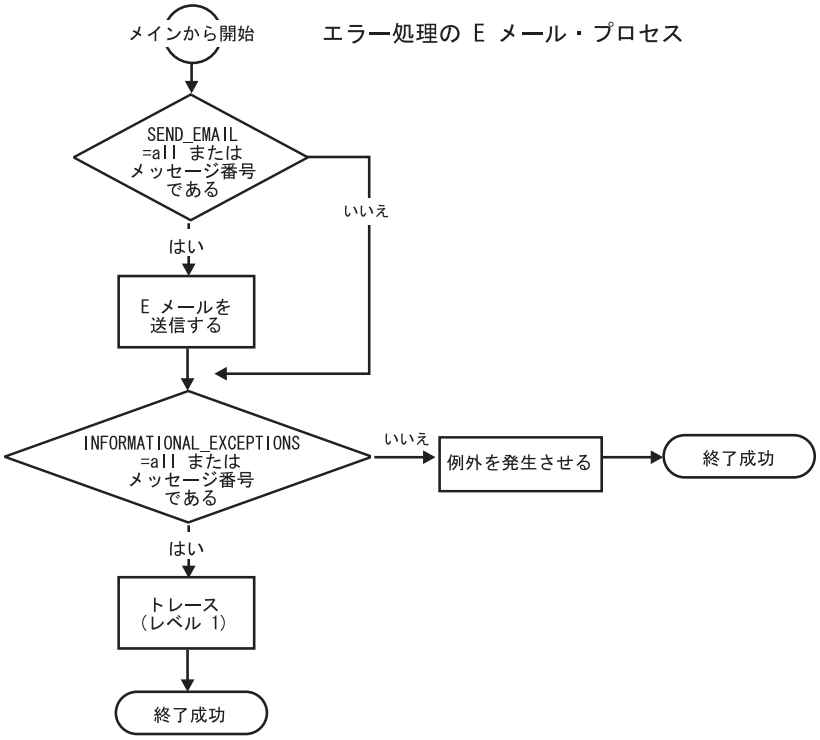



*図 85. エラー処理の E メール・プロセス*

## コラボレーション・テンプレートの標準プロパティー

表 90 に、コラボレーション・テンプレートのための標準的な構成プロパティーを示します。これらのプロパティーは CollaborationFoundation テンプレートを基にしています。

*表 90. コラボレーション・テンプレートの標準プロパティー*

| 標準構成プロパティー | 説明 |
|---|---|
| 1_EXCLUDE_VALUES | トリガー・ビジネス・オブジェクトの同期を回避するためにフィルター操作用として使用するコラボレーションの値を指定します。 |
| 1_FAIL_ON_INVALID_VALUE | フィルター操作の基準に合わない、1_FILTER_ATTRIBUTE で指定された属性の値をコラボレーションがどう処理するかを示します。 |
| 1_FILTER_ATTRIBUTE | 1_EXCLUDE_VALUES プロパティーと 1_INCLUDE_VALUES プロパティーで指定された値を比較する際に使用するコラボレーションのビジネス・オブジェクト属性を指定します。 |

*表 90. コラボレーション・テンプレートの標準プロパティー (続き)*

| 標準構成プロパティー | 説明 |
|---|---|
| 1_INCLUDE_VALUES | トリガー・ビジネス・オブジェクトの同期を許可するためにフィルター操作用として使用するコラボレーションの値を指定します。 |
| ADDITIONAL_RETRIEVE | データとの同期が成功した後に、コラボレーションが宛先アプリケーションからビジネス・オブジェクトを検索するかどうかを指定します。 |
| CONVERT_CREATE | ソース・アプリケーションで作成されたトリガー・ビジネス・オブジェクトが宛先アプリケーションにすでに存在する場合の Create 要求へのコラボレーションの対応方法を示します。 |
| CONVERT_UPDATE | ソース・アプリケーションで更新されたトリガー・ビジネス・オブジェクトが宛先アプリケーションに存在しない場合の Update 要求へのコラボレーションの対応方法を示します。 |
| INFORMATIONAL_EXCEPTIONS | コラボレーションが例外にどう対応するかを指定します。 |
| SEND_EMAIL | 例外を受け取った場合にコラボレーションが E メールを送信するかどうかを示します。 |
| USE_RETRIEVE | データとの同期を取る前に、コラボレーションが宛先アプリケーションからトリガー・ビジネス・オブジェクトを検索するかどうかを指定します。 |

## 1_EXCLUDE_VALUES

1_EXCLUDE_VALUES コラボレーション構成プロパティーは、トリガー・ビジネス・オブジェクトの同期を回避するためにフィルター操作用として使用するコラボレーションの値を指定します。1_FILTER_ATTRIBUTE で指定された属性の値がこのプロパティーにリストされている値のいずれかと一致する場合、コラボレーションはビジネス・オブジェクトを同期から除外します。

1_EXCLUDE_VALUES で使用される値は以下のとおりです。

- 除外する値のコンマ区切りリスト (「in-stock, at-customer」など)。
- オブジェクトを処理から除外しないことを示す値 none。
- 値なし。これは、null (CxIgnore) または空ストリング (CxBlank) の値を持つ属性を除外する場合に便利です。

デフォルト値は none です。

このプロパティーは、コラボレーション・テンプレートに使用できる構成プロパティーのリストから削除することができます。1_EXCLUDE_VALUES プロパティーは、削除されると、フィルター操作プロセスで使用されなくなります。

コラボレーションの指定値に大文字小文字の区別はありません。また、値のリスト内の余分なスペースに関する区別もありません。

排他値のリストが包含値のリストよりも短い 場合はこのプロパティーの値を指定します。それ以外の場合は、1_INCLUDE_VALUES の値を指定します。1_INCLUDE_VALUES 内で値を指定すると、1_EXCLUDE_VALUES の値は none に設定されます。

## 1_FAIL_ON_INVALID_VALUE

1_FAIL_ON_INVALID_VALUE コラボレーション構成プロパティーは、フィルター操作の基準に合わない、1_FILTER_ATTRIBUTE で指定された属性の値をコラボレーションがどう処理するかを示します。1_FAIL_ON_INVALID_VALUE プロパティーは、次の値のいずれかに設定してください。

- true: true に設定すると、コラボレーションは、1_FILTER_ATTRIBUTE で指定された属性の値が 1_EXCLUDE_VALUES または 1_INCLUDE_VALUES で指定されたフィルター操作の基準に合わない場合に、例外を生成して処理を停止します。
- false: false に設定すると、コラボレーションは、指定された属性の値がフィルター操作の基準に合わない場合に、次のいずれかの処理を実行します。
  - 1_FILTER_ATTRIBUTE で指定された属性が親ビジネス・オブジェクトに含まれている場合、コラボレーションは処理を継続します。
  - 1_FILTER_ATTRIBUTE で指定された属性が子ビジネス・オブジェクトに含まれている場合、コラボレーションは、その属性を含む子ビジネス・オブジェクトおよびその子をすべて除去します。その後、コラボレーションは処理を継続します。

デフォルト値は true です。値に大文字小文字の区別はありません。

このプロパティーは、コラボレーション・テンプレートに使用できる構成プロパティーのリストから削除することができます。プロパティーを削除すると、それはフィルター操作プロセスで使用されなくなります。

## 1_FILTER_ATTRIBUTE

1_FILTER_ATTRIBUTE コラボレーション構成プロパティーは、1_EXCLUDE_VALUES プロパティーと 1_INCLUDE_VALUES プロパティーで指定された値を比較する際に使用するコラボレーションのビジネス・オブジェクト属性を指定します。コラボレーションは、特定の値を持つビジネス・オブジェクトの同期を回避または保証するために、指定された属性の値を排他または包含用に指定された値と比較します。例えば、BusObjA というトップレベルのビジネス・オブジェクトに BusObjB という子ビジネス・オブジェクトの配列が含まれていると想定します。BusObjB ビジネス・オブジェクト内の Type という属性でフィルター操作するには、1_FILTER_ATTRIBUTE の値を BusObjA.BusObjB.Type に設定します。

**注:** コラボレーションがこのプロパティーを評価するのは、その 1_EXCLUDE_VALUES が、none 以外の値に評価される場合か、その 1_INCLUDE_VALUES が all 以外の値に評価される場合のみです。

このコラボレーション構成プロパティーの有効な値は、以下のとおりです。

- InterChange Server Express で定義されているビジネス・オブジェクトの正確な名前と属性 (InstallShipment.ShipmentType など)。属性の名前を確認するには、以下のいずれかを実行します。
  - System Manager でビジネス・オブジェクトを表示します。
  - repository¥BO_*BusinessObjectName*.txt というビジネス・オブジェクトのリポジトリー・ファイルを表示します。

  **注:** 本書では、ディレクトリー・パスに円記号 (¥) を使用します。UNIX システムの場合は、円記号をスラッシュ (/) に置き換えてください。ファイルのパス名はすべて、使用システムで製品がインストールされたディレクトリーを基準とした相対パス名です。
- *BusinessObjectName*.ObjectEventId。これは、各ビジネス・オブジェクトに固有の内部 ID を含む属性です。これは、1_FILTER_ATTRIBUTE のデフォルト値です。

コラボレーション・テンプレートは、任意の数のビジネス・オブジェクト属性でフィルター操作を行うように設計されています。フィルターを追加するには、構成プロパティーのセット
(*X*_FILTER_ATTRIBUTE、*X*_FAIL_ON_INVALID_VALUE、*X*_EXCLUDE_VALUES または *X*_INCLUDE_VALUES) をコラボレーションに追加します。*X* は、2 から N の整数です。その他のプロパティーは、2 から N の順に追加し、前記の正確な名前を指定する必要があります。

## 1_INCLUDE_VALUES

1_INCLUDE_VALUES コラボレーション構成プロパティーは、トリガー・ビジネス・オブジェクトの同期を許可するためにフィルターとして使用するコラボレーションの値を指定します。1_FILTER_ATTRIBUTE で指定された属性の値がこのプロパティーにリストされている値のいずれかと一致する場合、コラボレーションは、指定された値を持つビジネス・オブジェクトのみを同期します。

1_INCLUDE_VALUES プロパティーの有効な値は、以下のとおりです。

- 包含する値のコンマ区切りリスト (「standard, in-stock, at-customer」など)。
- 値 `all`。これは、すべてのオブジェクトを処理に含めることを示します。

デフォルト値は all です。値に大文字小文字の区別はありません。また、コラボレーションは、値のリスト内にある余分なスペースを無視します。

このプロパティーをフィルター操作プロセスで使用しない場合は、コラボレーションに使用できる構成プロパティーのリストから削除することができます。

包含値のリストが排他値のリストよりも短い 場合はこのプロパティーの値を指定します。それ以外の場合は、1_EXCLUDE_VALUES の値を指定します。
1_EXCLUDE_VALUES 内で値を指定すると、1_INCLUDE_VALUES の値は all に設定されます。

## ADDITIONAL_RETRIEVE

ADDITIONAL_RETRIEVE コラボレーション構成プロパティーは、データとの同期が成功した後に、コラボレーションが宛先アプリケーションからビジネス・オブジェクトを検索するかどうかを指定します。ADDITIONAL_RETRIEVE プロパティーは、以下のように設定します。

- true に設定すると、コラボレーションは、データとの同期が成功した後に、宛先アプリケーションからビジネス・オブジェクトを検索します。この設定は、ソース・アプリケーションが宛先アプリケーションから完全な値を持つビジネス・オブジェクトを受け取る必要があるが、宛先アプリケーションのコネクターがデータの作成または更新後に完全なビジネス・オブジェクトを戻さない場合に役立ちます。
  - コラボレーションは、完全な値を持つビジネス・オブジェクトを正常に取り出すと、ソース・アプリケーションに戻すため、それを triggeringBusObj 変数に保存します。同期要求を行うソース・コネクターにコラボレーションがバインドされている場合は、コラボレーションが処理を完了するとすぐに、トリガー・ビジネス・オブジェクトの値がソース・コネクターに戻されます。値は、ビジネス・オブジェクトの値が参照によって受け渡された場合と同じように戻されます。

    **注:** コラボレーションは、triggeringBusObj 変数の動詞を Retrieve から元の動詞にリセットします。動詞をリセットすると、コラボレーションは、Additional Processing 5 サブプロセスで必要になった場合に元の動詞を使用できます。Additional Retrieve プロセスの詳細については、 477 ページの『Additional Retrieve プロセス』を参照してください。
  - 完全な値を持つビジネス・オブジェクトの検索に成功しなかった場合、コラボレーションは例外を生成します。例外処理は、INFORMATIONAL_EXCEPTIONS プロパティーの設定によって決まります。
- false に設定すると、コラボレーションは、データとの同期が成功した後に、宛先アプリケーションからビジネス・オブジェクトを検索しません。

デフォルト値は false です。

## CONVERT_CREATE

CONVERT_CREATE コラボレーション構成プロパティーは、ソース・アプリケーションで作成されたトリガー・ビジネス・オブジェクトが宛先アプリケーションにすでに存在する場合にコラボレーションが Create 要求にどう対応するかを示します。CONVERT_CREATE プロパティーは、以下のように設定します。

- true に設定すると、ソース・アプリケーションで作成されたトリガー・ビジネス・オブジェクトが宛先にすでに存在する場合に、コラボレーションは Create 要求を Update 要求に変換します。
- false に設定すると、コラボレーションは、例外を生成します。例外処理は、INFORMATIONAL_EXCEPTIONS プロパティーの設定によって決まります。

デフォルト値は false です。

CONVERT_CREATE に対するコラボレーションのアクションは、USE_RETRIEVE コラボレーション構成プロパティーの値によって決まります。

- USE_RETRIEVE が true に評価された場合、コラボレーションが CONVERT_CREATE プロパティーを評価するのは、トリガー動詞が Create で、コラボレーションがビジネス・オブジェクトの検索に成功した場合のみです。
- USE_RETRIEVE が false に評価された場合、コラボレーションが CONVERT_CREATE プロパティーを評価するのは、宛先でのトリガー・ビジネス・オブジェクトの作成に失敗した後のみです。

USE_RETRIEVE 構成プロパティーと CONVERT_CREATE 構成プロパティーの関係については、472 ページの『Use_Retrieve プロセス』の表を参照してください。

## CONVERT_UPDATE

CONVERT_UPDATE コラボレーション構成プロパティーは、ソース・アプリケーションで更新されたトリガー・ビジネス・オブジェクトが宛先アプリケーションに存在しない場合にコラボレーションが Update 要求にどう対応するかを示します。CONVERT_UPDATE プロパティーは、以下のように設定します。

- true に設定すると、ソース・アプリケーションで作成されたトリガー・ビジネス・オブジェクトが宛先に存在しない場合に、コラボレーションは Update 要求を Create 要求に変換します。
- false に設定すると、コラボレーションは、例外を生成します。例外処理は、INFORMATIONAL_EXCEPTIONS プロパティーの設定によって決まります。

デフォルト値は false です。

CONVERT_UPDATE に対するコラボレーションのアクションは、USE_RETRIEVE コラボレーション構成プロパティーの値によって決まります。

- USE_RETRIEVE が true に評価された場合、コラボレーションが CONVERT_UPDATE プロパティーを評価するのは、トリガー動詞が Update で、ビジネス・オブジェクトが宛先にないことが検索によって確認された場合のみです。
- USE_RETRIEVE が false に評価された場合、コラボレーションが CONVERT_UPDATE プロパティーを評価するのは、宛先でのトリガー・ビジネス・オブジェクトの更新に失敗した後のみです。

## INFORMATIONAL_EXCEPTIONS

INFORMATIONAL_EXCEPTIONS コラボレーション構成プロパティーは、コラボレーションが例外にどう対応するかを指定します。プロパティーには、以下の値を入れることができます。

- `All` — 例外が発生するたびに、コラボレーションは例外をトレースに送信し、正常に終了します。
- `None` — 例外が発生するたびに、コラボレーションは例外を上げ、処理が失敗します。
- コンマで区切ったメッセージ番号のリスト — リストされた例外が発生するたびに、コラボレーションは例外をトレースに送信し、正常に終了します。例外メッセージ番号は、コラボレーション・メッセージ・ファイル (collaborations¥messages¥*CollaborationName*.txt) 内の例外メッセージ番号に対応しています。

デフォルト値は 1000、2000、2005、2010、2015、2020、3000、3010、3020 です。

## SEND_EMAIL

SEND_EMAIL コラボレーション構成プロパティーは、例外を受け取った場合にコラボレーションが E メールを送信するかどうかを示します。SEND_EMAIL プロパティーは、以下のように設定します。

- All — 例外が発生するたびに、コラボレーションは E メールを送信します。
- None — 例外が発生しても、コラボレーションは E メールを送信しません。
- コンマで区切ったメッセージ番号のリスト — リストされた例外が発生するたびに、コラボレーションは E メールを送信します。例外メッセージ番号は、コラボレーション・メッセージ・ファイル (collaborations¥messages¥*CollaborationName*.txt) 内の例外メッセージ番号に対応しています。
- メッセージ番号の範囲 — 指定された範囲にある例外が発生するたびに、コラボレーションは E メールを送信します。

デフォルト値は none です。

**注:** System Manager の「Collaboration Object Properties」ダイアログ・ボックスの **「E メール通知アドレス」**フィールドでアドレスを指定します。コラボレーションに対して E メール・アドレスが指定されている場合、コラボレーションは例外が発生したときに Email コラボレーションを呼び出します。Email コラボレーションは、WebSphere Business Integration Express の一部としてインストールし、構成することができます。

## USE_RETRIEVE

USE_RETRIEVE コラボレーション構成プロパティーは、データとの同期の前に、コラボレーションが宛先アプリケーションからトリガー・ビジネス・オブジェクトを検索するかどうかを指定します。USE_RETRIEVE プロパティーは、以下のように設定します。

- true に設定すると、コラボレーションは、データとの同期の前に、宛先アプリケーションからトリガー・ビジネス・オブジェクトを検索します。この設定は、以下のような場合に役立ちます。
  - コラボレーションが別のコラボレーションによって起動され、グループ化されたコラボレーションでなんらかのステップが失敗したときに差し戻し処理を実行する必要がある場合。変更前にコラボレーションで値を検索することにより、コラボレーションが Update プロセスまたは Delete プロセスで元の値を復元することが可能になります。
  - コラボレーションは Wrapper Collaboration によって起動された場合。Wrapper Collaboration は常に、トリガー・ビジネス・オブジェクトを Create 動詞で送信します。このプロパティーを true に設定すると、ビジネス・オブジェクトがすでに宛先に存在する場合にコラボレーションの失敗を回避することができます。USE_RETRIEVE が true に評価されると、コラボレーションは、Create または Update の実行前にソース・アプリケーションの値と宛先アプリケーションの値を比較します。オブジェクトが同一であれば、正常に終了します。

- false に設定すると、コラボレーションは、宛先アプリケーションからトリガー・ビジネス・オブジェクトを検索せずに処理を開始します。この設定は、ソース・ビジネス・オブジェクトの値と宛先ビジネス・オブジェクトの値との比較が重要ではない場合に役立ちます。

デフォルト値は false です。

**注:** このプロパティーの使用の詳細については、 482 ページの『CONVERT_CREATE』および 483 ページの『CONVERT_UPDATE』を参照してください。

# 特記事項

本書に記載の製品、サービス、または機能が日本においては提供されていない場合があります。日本で利用可能な製品、サービス、および機能については、日本 IBM の営業担当員にお尋ねください。本書で IBM 製品、プログラム、またはサービスに言及していても、その IBM 製品、プログラム、またはサービスのみが使用可能であることを意味するものではありません。これらに代えて、IBM の知的所有権を侵害することのない、機能的に同等の製品、プログラム、またはサービスを使用することができます。ただし、IBM 以外の製品とプログラムの操作またはサービスの評価および検証は、お客様の責任で行っていただきます。

IBM は、本書に記載されている内容に関して特許権 (特許出願中のものを含む) を保有している場合があります。本書の提供は、お客様にこれらの特許権について実施権を許諾することを意味するものではありません。実施権についてのお問い合わせは、書面にて下記宛先にお送りください。

〒106-0032
東京都港区六本木 3-2-31
IBM World Trade Asia Corporation
Licensing

以下の保証は、国または地域の法律に沿わない場合は、適用されません。

IBM およびその直接または間接の子会社は、本書を特定物として現存するままの状態で提供し、商品性の保証、特定目的適合性の保証および法律上の瑕疵担保責任を含むすべての明示もしくは黙示の保証責任を負わないものとします。国または地域によっては、法律の強行規定により、保証責任の制限が禁じられる場合、強行規定の制限を受けるものとします。

この情報には、技術的に不適切な記述や誤植を含む場合があります。本書は定期的に見直され、必要な変更は本書の次版に組み込まれます。IBM は予告なしに、随時、この文書に記載されている製品またはプログラムに対して、改良または変更を行うことがあります。

本書において IBM 以外の Web サイトに言及している場合がありますが、便宜のため記載しただけであり、決してそれらの Web サイトを推奨するものではありません。それらの Web サイトにある資料は、この IBM 製品の資料の一部ではありません。それらの Web サイトは、お客様の責任でご使用ください。

IBM は、お客様が提供するいかなる情報も、お客様に対してなんら義務も負うことのない、自ら適切と信ずる方法で、使用もしくは配布することができるものとします。

本プログラムのライセンス保持者で、(i) 独自に作成したプログラムとその他のプログラム（本プログラムを含む）との間での情報交換、および (ii) 交換された情報の相互利用を可能にすることを目的として、本プログラムに関する情報を必要とする方は、下記に連絡してください。

IBM Burlingame Laboratory Director

IBM Burlingame Laboratory
577 Airport Blvd., Suite 800
Burlingame, CA 94010
U.S.A

本プログラムに関する上記の情報は、適切な使用条件の下で使用することができますが、有償の場合もあります。

本書で説明されているライセンス・プログラムまたはその他のライセンス資料は、IBM 所定のプログラム契約の契約条項、IBM プログラムのご使用条件、またはそれと同等の条項に基づいて、IBM より提供されます。

この文書に含まれるいかなるパフォーマンス・データも、管理環境下で決定されたものです。そのため、他の操作環境で得られた結果は、異なる可能性があります。一部の測定が、開発レベルのシステムで行われた可能性がありますが、その測定値が、一般に利用可能なシステムのものと同じである保証はありません。さらに、一部の測定値が、推定値である可能性があります。実際の結果は、異なる可能性があります。お客様は、お客様の特定の環境に適したデータを確かめる必要があります。

IBM 以外の製品に関する情報は、その製品の供給者、出版物、もしくはその他の公に利用可能なソースから入手したものです。IBM は、それらの製品のテストは行っておりません。したがって、他社製品に関する実行性、互換性、またはその他の要求については確証できません。IBM 以外の製品の性能に関する質問は、それらの製品の供給者にお願いします。

本書には、日常の業務処理で用いられるデータや報告書の例が含まれています。より具体性を与えるために、それらの例には、個人、企業、ブランド、あるいは製品などの名前が含まれている場合があります。これらの名称はすべて架空のものであり、名称や住所が類似する企業が実在しているとしても、それは偶然にすぎません。

IBM の将来の方向または意向に関する記述については、予告なしに変更または撤回される場合があり、単に目標を示しているものです。

## プログラミング・インターフェース情報

プログラミング・インターフェース情報は、プログラムを使用してアプリケーション・ソフトウェアを作成する際に役立ちます。

一般使用プログラミング・インターフェースにより、お客様はこのプログラム・ツール・サービスを含むアプリケーション・ソフトウェアを書くことができます。

ただし、この情報には、診断、修正、および調整情報が含まれている場合があります。診断、修正、調整情報は、お客様のアプリケーション・ソフトウェアのデバッグ支援のために提供されています。

**警告:** 診断、修正、調整情報は、変更される場合がありますので、プログラミング・インターフェースとしては使用しないでください。

## 商標

以下は、IBM Corporation の商標です。

IBM
IBM ロゴ
AIX
CrossWorlds
DB2
DB2 Universal Database
Lotus
Lotus Domino
Lotus Notes
MQIntegrator
MQSeries
Tivoli
WebSphere

Microsoft、Windows、Windows NT および Windows ロゴは、Microsoft Corporation の米国およびその他の国における商標です。

MMX および Pentium は、Intel Corporation の米国およびその他の国における商標です。

Java およびすべての Java 関連の商標およびロゴは、Sun Microsystems, Inc. の米国およびその他の国における商標または登録商標です。

他の会社名、製品名およびサービス名等はそれぞれ各社の商標です。

System Manager には、Eclipse Project (http://www.eclipse.org) により開発されたソフトウェアが含まれています。


WebSphere Business Integration Server Express V4.3 および WebSphere Business Integration Server Express Plus V4.3

# 用語集

## ［ア行］

**アクション (action).** ビジネス・プロセスでの単一ステップを示す、アクティビティー・ダイアグラムのシンボル。アクション・ノードにはコード・フラグメントが含まれる。

**アクティビティー・ダイアグラム (activity diagram).** コラボレーションのシナリオ用の制御フローを定義する。アクティビティー・ダイアグラムは、ビジネス・プロセスに必要なアクションを指定する一連のシンボルと、アクションの実行順序を決定するロジックから構成される。

**イテレーター (iterator).** ループ操作を実装するネストされたダイアグラムへの参照を組み込んだアクティビティー・ダイアグラムのシンボル、およびループ動作を含むダイアグラムでもある。イテレーターは、ビジネス・オブジェクトのすべての属性、またはビジネス・オブジェクト配列のすべての要素を通じてループできる。

**イベント順序付け (event sequencing).** 同一のビジネス・オブジェクトに関連する複数のイベントは、イベントが到着した順序で一度に 1 つずつコラボレーションによって処理されるという InterChange Server の保証。

**イベント分離 (event isolation).** 複数のコラボレーションが、同一のビジネス・オブジェクト・データに関連する複数のイベントを並行処理しないという保証。

## ［カ行］

**完全な値を持つビジネス・オブジェクト (full-valued business object).** 基本キー属性のみでなく、それ以外の属性のデータ値を持つビジネス・オブジェクト。

**キー値 (key values).** 一般にはビジネス・オブジェクトまたは関連するアプリケーション・エンティティーの固有の識別を含む属性の値。

**決定ノード (decision node).** シナリオ内の決定分岐を処理するノード。決定ノードは、複数の可能性のある結果 (アクション、サブダイアグラム、またはイテレーター・ノード) が存在する場合に使用する。決定ノードの各分岐には条件があり、制御フローは条件が true と評価される分岐に移動する。

**コード・フラグメント (code fragment).** コラボレーション API または他の Java コードを使用するコード・ステートメントのセットを通じてのアクションの指定。

**コラボレーション (collaboration).** 汎用分散ビジネス・プロセスを説明するビジネス・ロジック。コラボレーションは個々のアプリケーションと対話し、これらの異なるアプリケーションのイベントとデータを 1 つにまとめて機能を拡張する。

**コラボレーション構成プロパティー (collaboration configuration property).** InterChange Server Express オブジェクトに関する構成可能な情報。コラボレーション・テンプレートには、標準プロパティーとコラボレーション固有プロパティーが含まれる。コラボレーション開発者は、コラボレーション固有プロパティーを作成することで、コラボレーション・オブジェクトの実行時動作のいくつかの性質を管理者が指定できるようにする。

**コラボレーション・オブジェクト (collaboration object).** コラボレーション・テンプレートから作成されるオブジェクト。コラボレーション・オブジェクトは、構成およびアプリケーション (コネクターまたは他のコラボレーションによって表される) にバインド後実行可能になる。

**コラボレーション・グループ (collaboration group).** コラボレーション・オブジェクトの実行可能セット。コラボレーション・オブジェクトをバインドすることによって形成される。

**コラボレーション・テンプレート (collaboration template).** コラボレーションのロジックおよびフレームワーク。コラボレーション・テンプレートはコラボレーションの定義を提供する。このテンプレートからコラボレーション・オブジェクトをインスタンス化できるが、コラボレーション・テンプレート自体は実行できない。

## ［サ行］

**サービス呼び出し (service call).** コネクターや別のコラボレーションなど、コラボレーション外の InterChange Server Express オブジェクトへの要求を表す、アクティビティー・ダイアグラムのシンボル。

**最小トランザクション・レベル (minimum transaction level).** コラボレーション・テンプレートの開発者により設定されたトランザクション・レベル。そのテンプレートから作成されたコラボレーション・オブジェクトを実行するために必要なトランザクション・サービスのレベルを示す。

**差し戻し (compensation).** コラボレーションがトランザクションのロールバック中に、直前に実行済みのサービス呼び出しを取り消すアクション。

**サブダイアグラム (subdiagram).** 別のネストされたアクティビティー・ダイアグラムを表すアクティビティー・ダイアグラムのシンボル、およびネストされたダイアグラム自身。

**参照値を持つビジネス・オブジェクト (reference-valued business object).** キー属性のみの値を持つビジネス・オブジェクト。このビジネス・オブジェクトには、非キー属性の値は含まれない。

**失敗イベント (failed event).** この用語は変更済み。『未解決のフロー (unresolved flow)』を参照。

**シナリオ (scenario).** 1 つ以上の受信イベントを処理するコード。シナリオを使用することで、コラボレーションのロジックを区画に分割することができる。

**シナリオ変数 (scenario variable).** シナリオ内のすべてのダイアグラムのすべての部分を有効範囲とする変数。

**シナリオ・ツリー (scenario tree).** 階層的に表示されるシナリオのセット。複合シナリオ、サブダイアグラム、およびイテレーターを含む。

**制御フロー (control flow).** ビジネス・プロセス・ロジックのフロー。コラボレーションにおいて、アクティビティー・ダイアグラムは特定のシナリオの制御フローを定義し、ビジネス・プロセスに必要なアクションを指定する。決定ノードとイテレーターをアクティビティー・ダイアグラム内で使用して、アクション・ノードの実行順序を指定する。

**遷移リンク (transition link).** アクティビティー・ダイアグラムの他のシンボル間の制御フローを示す、アクティビティー・ダイアグラムのシンボル。

**宣言 (declaration).** 使用する変数の名前と型。コンパイラーは使用される各変数の宣言を必要とする。

**相関属性 (correlation attribute).** コラボレーションが長期存続ビジネス・プロセスとして使用される場合に、2 つのビジネス・プロセス間の会話を示す。相関属性は、開始ノードまたはアウトバウンド・サービス呼び出しによって初期化される。次に、会話の参加者がその相関属性を使用して、外部呼び出しを行ったり外部ソースからマッチング・イベントを受け取ったりすることができる。

**属性 (attribute).** ビジネス・オブジェクト内のデータ項目。

## ［タ行］

**長期存続ビジネス・プロセス (long-lived business process).** 複数のビジネス・プロセス間の非同期通信を可能にする、コラボレーションの構成および配置メソッド。長期存続ビジネス・プロセスでは、サービス呼び出しの期間を通してイベント・フロー・コンテキストが存続する。

**テンプレート変数 (template variable).** コラボレーション・テンプレート内のすべてのシナリオを有効範囲とする変数。

**テンプレート・ツリー・ビュー (template tree view).** コラボレーション・テンプレートのテンプレート定義、シナリオ・ツリー、およびメッセージ・ファイルを表示するツリー・ビューアー。テンプレート・ツリー・ビューの表示はオプション。

**トランザクション動詞 (transactional verb).** Create、Update、または Delete など、データ変更を示すビジネス・オブジェクト動詞。Retrieve はデータを変更しないので、トランザクション動詞ではない。

**トランザクション・コラボレーション (transactional collaboration).** データベース・トランザクション・モデルに従い、ビジネス・プロセスのデータ整合性を提供するコラボレーション。トランザクション・コラボレーションは、実行時エラーが原因でコラボレーション・オブジェクトが失敗する場合にロールバックすることができる。トランザクション・コラボレーションでは、サービス呼び出しには差し戻しが定義される。

**トリガー・イベント (triggering event).** アプリケーション・イベントが発生したときに、コネクターがサブスクライブしているコラボレーションに送信するビジネス・オブジェクト。

## ［ハ行］

**バインディング (binding).** ビジネス・オブジェクトを提供できる、またはビジネス・オブジェクトを受け取ることができるオブジェクトに、コラボレーション・オブジェクトを付加すること。コラボレーションを付加でき

るオブジェクトは、コネクターまたはその他のコラボレーション・オブジェクトである。

**パッケージ (package).** 関連する Java クラスのグループ。コラボレーション・テンプレートをパッケージの一部とすることができる。また、他のパッケージをインポートすることもできる。

**ビジネス・オブジェクト (business object).** データへのアクションを示す動詞と、ビジネス・エンティティーを表すデータのセット。

**ビジネス・オブジェクト定義 (business object definition).** ビジネス・オブジェクトに含まれるフォーマットおよびデータの説明。ビジネス・オブジェクト定義には、名前、バージョン、サポートされる動詞のセット、および順序付き属性のセットが含まれる。

**ビジネス・オブジェクト・プローブ (business object probe).** 指定されたビジネス・オブジェクトの属性の値を、実行時にモニターおよび報告する。ビジネス・オブジェクト・プローブは、任意の遷移リンクに置くことができる (決定ノードの受信遷移リンクおよびサービス呼び出しリンクを除く)。

**ブレーク (break).** 反復を強制的に早期終了するために、イテレーターのアクティビティー・ダイアグラムに置かれるシンボル。

**ポート (port).** コラボレーションと、InterChange Server Express システム内の他のオブジェクトとの間のインターフェース。コラボレーション・オブジェクトとコネクターまたは別のコラボレーション・オブジェクトとのバインドは、ポートを介して行われる。

## ［マ行］

**未解決のフロー (unresolved flow).** それを受信すると、コラボレーションが不成功に終了するシナリオを実行することになるビジネス・オブジェクト。未解決のフローには、失敗したフロー (アプリケーションまたはロジックの問題が原因で失敗したフロー)、据え置きフロー (リカバリーが遅延されたフロー)、転送中フロー (Service Call In-Transit 永続性に対応するように構成されたコラボレーション内で、サービス呼び出し伝送中にサーバーが破損したときに作成されるフロー)、または重複の可能性があるフロー (コラボレーションによって受信されている可能性があるフロー) などがある。

## ［ラ行］

**例外 (exception).** エラーを処理できる他のエンティティーに実行時エラーを渡すために使用されるオブジェクト。アクティビティー・ダイアグラムでは、例外は例外遷移リンク上でキャッチされる。

## B

**BaseCollaboration.** 他のすべてのコラボレーション・クラスの派生元となる InterChange Server Express 定義クラス。BaseCollaboration クラスには、コラボレーションを操作するメソッドが含まれる。

**BusObjArray.** ビジネス・オブジェクトの配列を表す InterChange Server Express 定義クラス。BusObjArray は、値が子ビジネス・オブジェクトの配列への参照となるビジネス・オブジェクト属性に使用される。

**BusObj.** ビジネス・オブジェクトを表す InterChange Server Express 定義クラス。

## C

**CollaborationException.** InterChange Server Express 定義の例外オブジェクト。

**currentException.** 先行する例外の値を保持する InterChange Server Express 定義変数。`currentException` の有効範囲は、前のアクション、サブアクティビティー、またはイテレーター内に設定される。

## I

**import ステートメント (import statement).** クラスまたはクラスのパッケージをコラボレーション・クラスに含める Java ステートメント。

## T

**triggeringBusObj.** シナリオのトリガー・イベントを含む、Designer 宣言変数で、シナリオが開始されると実行される。

## U

**UID.** シナリオのアクティビティー・ダイアグラム内の各シンボルの固有 ID。

# 索引

日本語, 数字, 英字, 特殊文字の順に配列されています。なお, 濁音と半濁音は清音と同等に扱われています。

## [ア行]



## [カ行]

# [サ行]

# ［タ行］

## [ナ行]



## [ハ行]

# [マ行]

## [ヤ行]



## [ラ行]



## [ワ行]



## A



## B

# C

# D



# E



# F



# G

# H



# I



# J



# K



# L



# M



# N

# O



# P



# R



# S



# T



# U

# V



# W



# ［特殊文字］

Printed in Japan

日本アイ・ビー・エム株式会社
〒106-8711　東京都港区六本木3-2-12