WebSphere Business Integration Server
Express and Express Plus

ビジネス・オブジェクト開発ガイド

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、 315 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 Express バージョン 4.3、IBM 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 and Express Plus
Business Object 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

本書について

製品 IBM ®WebSphere Business Integration Server Express および IBM ®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 に含まれるツールは、ビジネス・プロセスの作成、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、Business Object Designer Express を使用してビジネス・オブジェクト定義を作成する方法を説明します。始める前に、「システム・インプリメンテーション・ガイド」に説明されているすべての概念について、理解していることが必要です。

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 という用語と、これを言い換えた用語は、これらの 2 つの製品の両方を指します。

対象読者

本書は、ビジネス・オブジェクトの作成や変更を担当する IBM のお客様、コンサルタント、および販売代理店を対象としています。始める前に、「システム・インプリメンテーション・ガイド」に説明されているすべての概念について、理解していることが必要です。

関連文書

本書の対象製品の一連の関連文書には、WebSphere Business Integration Server Express のどのインストールにも共通する機能とコンポーネントの解説のほか、特定のコンポーネントに関する参考資料が含まれています。

関連文書は、http://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。

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、記述されたとおりの値を示します。
イタリック、*イタリック*	初出語、変数名、または相互参照を示します。
太字	GUI 要素を示します。
青のアウトライン	オンラインで表示したときにのみ見られる青のアウトラインは、相互参照用のハイパーリンクをあらわします。アウトラインの内側をクリックすると、参照先オブジェクトにジャンプします。
{ }	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。
< >	命名規則では、個々の要素を互いに識別する目的から、<server_name><*connector_name*>tmp.log のように名前の各要素を不等号括弧で囲みます。
ProductDir	製品のインストール先ディレクトリーを表します。IBM WebSphere InterChange Server Express 環境のデフォルトの製品ディレクトリーは、IBM WebSphere Business Integration Adapters 環境の ¥WebSphereServer です。
%text% および *$text*	% 記号で囲まれたテキストは、Windows の text システム変数またはユーザー変数の値を示しています。

本リリースの新機能

本書の最初のリリースです。

Business Object Designer Express の新機能

本書は、Business Object Designer Express の以下の新機能について説明します。

- プロジェクトからのビジネス・オブジェクト定義のオープンおよびプロジェクトからのビジネス・オブジェクト定義の削除のステップの改良
- ステップ 3 からステップ 6 までのビジネス・オブジェクト・ウィザード画面の機能拡張
- Process Designer Express は、Business Object Designer Express の**「ツール」**メニューから開始できます。

第 1 部 ビジネス・オブジェクトの設計と開発

第 1 章 ビジネス・オブジェクト

ビジネス・インテグレーション・システムは、ビジネス・オブジェクトを使用して、統合ブローカーとコネクター、またはアクセス・クライアントの間でデータや処理命令を送信します。InterChange Server Express は、WebSphere Business Integration Server Express の統合ブローカーです。ビジネス・オブジェクトは、InterChange Server Express からの要求、アプリケーションまたは Web サーバーで発生したイベント、または外部サイトからの呼び出しを表します。このマニュアルには、ビジネス・オブジェクトの開発と設計に関する情報と、独自の Object Discovery Agent の開発に関する情報が記載されています。この章の主な内容は次のとおりです。

- 『WebSphere Business Integration システム内のビジネス・オブジェクト』
- 12 ページの『ビジネス・オブジェクトの構造』
- 14 ページの『開発プロセスの概要』

この章は、読者がご使用の環境の InterChange Server Express についての基本知識を持っていることが前提となっています。表 1 は、InterChange Server Express の以下の資料を参照します。

表 1. 前提資料

統合ブローカー	前提資料
InterChange Server Express	• 「*WebSphere InterChange Server Express Installation Guide*」 • 「システム・インプリメンテーション・ガイド」

WebSphere Business Integration システム内のビジネス・オブジェクト

WebSphere Business Integration システムは、以下のコンポーネントから構成されています。

- 一連のアダプター

  アダプター は、アプリケーション・ロジックの実行やデータの交換などのタスクを行う InterChange Server Express やアプリケーション (またはテクノロジー) と通信する一連のソフトウェア・モジュールです。

- 統合ブローカー

  InterChange Server Express は、WebSphere Business Integration システムの統合ブローカーです。 InterChange Server Express のタスクは、異機種混合のアプリケーション間でデータを統合することです。

WebSphere Business Integration システムでは、コンポーネント間で送受信される情報が、ビジネス・オブジェクトの形式でパッケージ化されます。

- アダプターと InterChange Server Express の間で転送されるデータに関しては、適切なアプリケーション・エンティティーをモデル化するアプリケーション*固有の*ビジネス・オブジェクト を設計してください。
- InterChange Server Express コラボレーション・オブジェクトのビジネス・ロジックの内部で処理されるデータに関しては、やり取りの必要なアプリケーション・エンティティーに関する情報のスーパーセットを含めた*汎用ビジネス・オブジェクト* を設計してください。汎用ビジネス・オブジェクトとアプリケーション固有のビジネス・オブジェクトの間のデータ変換にはマップが使用されます。これにより、コラボレーション・オブジェクトがアプリケーションに依存せずにビジネス・ロジックを適用できると同時に、アダプターがアプリケーション固有のエンティティーを使用してそのアプリケーションとやり取りできます。

アプリケーション固有のビジネス・オブジェクトと汎用オブジェクトは両方とも、設計時に*ビジネス・オブジェクト定義* としてモデル化されます。このビジネス・オブジェクト定義は、ビジネス・インテグレーション・システムに保管されます。実行時には、データは、適切な定義に基づいて *ビジネス・オブジェクト・インスタンス* (「ビジネス・オブジェクト」とも呼ばれる) に取り込まれます。ビジネス・オブジェクトは、ルーティング・ルールやビジネス・ロジック・ルールの指定に従って、ビジネス・インテグレーション・システムの中を移動します。

ビジネス・オブジェクト定義

ビジネス・オブジェクト定義 は、集合的単位として扱えるデータ用のテンプレートとなっています。これには、ビジネス・オブジェクト定義の名前およびバージョンを指定するビジネス・オブジェクト・ヘッダーが含まれています。ビジネス・オブジェクト定義には、そのほか以下の情報も含まれています。

- 5 ページの『ビジネス・オブジェクトの属性および属性プロパティー』
- 8 ページの『ビジネス・オブジェクトの動詞』
- 8 ページの『ビジネス・オブジェクトのアプリケーション固有の情報』

ビジネス・オブジェクト定義の一部を、図 1 に示します。

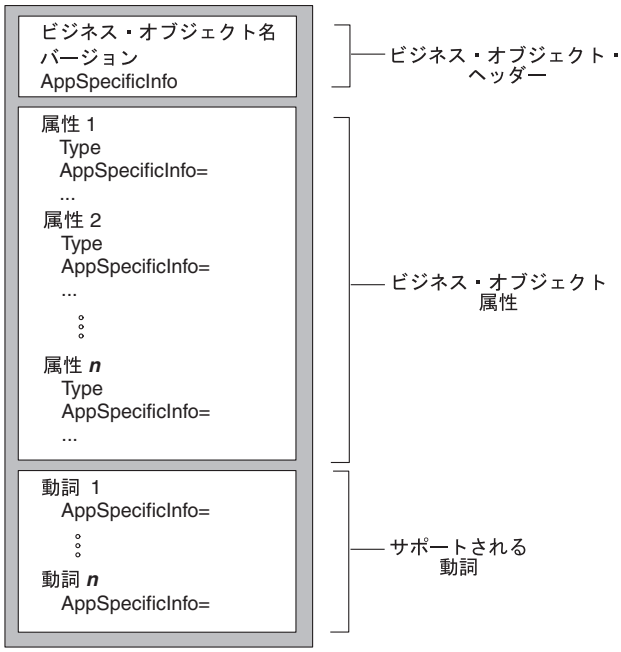


図 1. ビジネス・オブジェクト定義 (一部)

ビジネス・オブジェクトの属性および属性プロパティー

ビジネス・オブジェクトには属性が含まれています。*属性* はそれぞれ、1 つのデータ・エンティティーを表します。ビジネス・オブジェクト定義には、各属性の名前およびその他の*属性プロパティー* を定義してください。各属性の値は、ビジネス・オブジェクトのインスタンスによって保持されます (つまり、属性には値が含まれないということです)。

ビジネス・オブジェクト定義には、属性に適用される各種のプロパティーが組み込まれます。これらのプロパティーから、コネクター、データ・ハンドラー、およびその他のコンポーネントは、属性のタイプ、サイズ、デフォルト値についての情報を知ることができます。次に、属性プロパティーについて詳しく説明します。

Name プロパティー: ビジネス・オブジェクト定義の内部にある各ビジネス・オブジェクト属性は、一意の名前を持つ必要があります。この名前はその属性が格納するデータを示すものにしてください。属性の名前は、80 文字までです。英数字と下線を含めることはできますが、スペース、句読点、または特殊文字を含めることはできません。

注:

1. アプリケーション固有のビジネス・オブジェクトを設計する際には、そのアダプターのユーザーズ・ガイドまたは「*データ・ハンドラー・ガイド*」で、特定の命名要件および推奨事項を調べてください。
2. この属性名には、英語 (U.S.) のロケール (`en_US`) に関連したコード・セットで定義されている文字のみ を使用する必要があります。

Type プロパティー: Type プロパティーは属性のデータ型を定義します。

- 単純属性の場合、サポートされている型は、Boolean、Integer、Float、Double、String、Date、および LongText です。
- 複合属性の場合、型は以下のビジネス・オブジェクト定義です。
  - 属性が子ビジネス・オブジェクトを表している場合は、そのタイプを子ビジネス・オブジェクト定義の名前として指定し、カーディナリティーには 1 (単一カーディナリティー) を指定してください。
  - 属性が子ビジネス・オブジェクトの配列を表している場合、その子ビジネス・オブジェクト定義の名前を型として指定し、カーディナリティーには n (複数カーディナリティー) を指定してください。

注: 子ビジネス・オブジェクトを表すすべての属性には ContainedObjectVersion プロパティー (子オブジェクトのビジネス・オブジェクト定義のバージョン番号を指定)、および Relationship プロパティー (値として Containment を指定) も含まれています。

Cardinality プロパティー: 単純属性はそれぞれ、単一カーディナリティー (カーディナリティー 1) を持ちます。複合属性は、子ビジネス・オブジェクトを表す場合は単一カーディナリティーを、子ビジネス・オブジェクトの配列を表す場合は複数カーディナリティー (カーディナリティー n) をそれぞれ持ちます。カーディナリティーについて詳しくは、13 ページの『階層型ビジネス・オブジェクト』を参照してください。

注: 必須属性に指定されている場合、単一カーディナリティーは、子ビジネス・オブジェクトが必ず 存在することを示すのに対し、複数カーディナリティーは、子ビジネス・オブジェクトのインスタンスが存在しないか、1 個以上存在することを示します。

Key プロパティー: 各ビジネス・オブジェクトの少なくとも 1 つの属性をそのオブジェクトのキーとして指定する必要があります。キー属性には、ビジネス・オブジェクトを一意に識別する値が含まれています。属性をキーとして定義するには、Key プロパティーを true に設定します。

注: ビジネス・オブジェクト内のキー値は、一般的に基本キー と呼ばれます。

複合属性をキーとして指定した場合は、以下のようになります。

- 属性が子ビジネス・オブジェクトを表す場合、そのキーは、その子ビジネス・オブジェクト内のキーを連結する役割を果たします。
- 属性が子ビジネス・オブジェクトの配列を表す場合、そのキーは、子ビジネス・オブジェクト内のキーを連結するものとして、配列の位置 0 に置かれます。

Foreign key プロパティー: Foreign Key プロパティーは通常、ある属性の値が別のビジネス・オブジェクトの基本キーを保持するよう指定するためにアプリケーション固有のビジネス・オブジェクトで使用します。これにより、2 つのビジネス・オブジェクトがリンクされます。別のビジネス・オブジェクトの基本キーを保持する属性のことを、*foreign key* と呼びます。Foreign Key プロパティーは、外部キーを表す各属性に対して true に設定してください。

Foreign Key プロパティーは、他の処理命令にも使用することができます。例えば、このプロパティーは、コネクターが実行する外部キー検索の種類を指定する目的で

も使用できます。この場合、外部キーを true に設定して、データベース内に該当のエンティティーが存在するかをチェックし、そのエンティティーのレコードが存在している場合に限り関係を作成するようコネクターに指示します。

Required プロパティー: Required プロパティーは、属性に対する値の指定が必要であるかどうかを指定します。ビジネス・オブジェクトのデータの処理を可能にするため、そのビジネス・オブジェクト内の特定の属性に値を含めることを必須とする場合、その属性の Required プロパティーを true に設定してください。

AppSpecificInfo: AppSpecificInfo プロパティーには、1000 文字以内の String を含めることができます。これは、主にアプリケーション固有のビジネス・オブジェクトに対して指定されます。このプロパティーについては、8 ページの『ビジネス・オブジェクトのアプリケーション固有の情報』を参照してください。

注: アプリケーション固有の情報はマッピング処理では使用できません。

Max Length プロパティー: Max Length プロパティーには、String 型の属性に格納できるバイト数を設定します。この値の使用は WebSphere Business Integration システムでは規定されていませんが、この値を使用するコネクターやデータ・ハンドラーもあります。ビジネス・オブジェクトを処理する特定のアダプターのガイド、またはデータ・ハンドラーのガイドを参照して、使用可能な最小長と最大長を確認してください。

重要: Max Length プロパティーは、固定幅のデータ・ハンドラーを使用する場合にきわめて重要です。

注: 属性長はマッピング処理では使用できません。

Default value プロパティー: Default Value プロパティーは、属性に対してデフォルト値を指定するために使用します。

アプリケーション固有のビジネス・オブジェクトに対してこのプロパティーが指定されている場合、UseDefaults コネクター構成プロパティーが true に設定されていれば、コネクターは、実行時に値が指定されていない属性に値を設定するために、ビジネス・オブジェクト定義に指定されたデフォルト値を使用できます。

注:

1. 属性のデフォルト値には、現在のロケールに関連したコード・セットで定義された文字を使用することができます。
2. 属性の型が String である場合は、ブランク文字をデフォルト値として指定可能です。

Comments プロパティー: Comments プロパティーを使用すると、属性に対するコメントの指定が可能になります。AppSpecificInfo プロパティーがビジネス・オブジェクトの処理に使用されるのに対し、Comments プロパティーは、他の開発者が設計上の決定事項を理解するのに役立つ文書情報のみを備えています。

注: 属性のコメントには、現在のロケールに関連したコード・セットで定義された文字を使用することができます。ただし、改行文字は無効です。

ObjectEventId 属性: ObjectEventId 属性は必須属性であるだけでなく、各ビジネス・オブジェクト内で最後の属性にする必要があります。WebSphere Business Integration システムは、システム内でのイベントの流れの識別と追跡にこの属性を使用します。

ObjectEventId 属性は、WebSphere Business Integration システム内の各イベントを識別する一意の値を格納します。コネクター・フレームワークは、この属性の値を、親ビジネス・オブジェクト内および各子オブジェクト内に生成します。

重要: ObjectEventId 属性は、マッピングしたり、コネクターやデータ・ハンドラーから値を設定したりできません。この属性の値は、ビジネス・インテグレーション・システムによって処理されます。

ビジネス・オブジェクトの動詞

ビジネス・オブジェクト定義には、ビジネス・オブジェクトがサポートできる動詞のリストが含まれています。これらの動詞は、ビジネス・オブジェクト内のデータに対して有効な操作に対応しています。実行時には、ビジネス・オブジェクトにアクティブな動詞が 1 つ含まれますが、それは、特定のビジネス・オブジェクト内のデータに対して実行する操作について説明しています。

表 2 に、ビジネス・オブジェクト定義がサポートできる基本的な動詞を示します。

表 2. 基本的な動詞

動詞	機能
Create	新規のエンティティーをアプリケーション内に作成します。
Retrieve	キー値を使用して、完全なビジネス・オブジェクトを戻します。
Update	アプリケーション・エンティティー内の 1 つ以上のフィールドの値を変更します。
Delete	アプリケーションからエンティティーを除去します。この操作では、事実上必ず物理的な削除が行われます。

表 2 の基本的な動詞に加えて、以下の動詞のうちの 1 つ以上が、ビジネス・オブジェクト定義によってサポートされていなければなりません。

- RetrieveByContent — 非キー値を使用して、完全なビジネス・オブジェクトを戻します。
- Exist — 指定されたエンティティーに対して存在チェックを実行しますが、検索は実行しません。
- Custom — アプリケーション固有の操作を実行します。

ビジネス・オブジェクトのアプリケーション固有の情報

ビジネス・オブジェクト定義には、*アプリケーション固有の情報* を指定できます。この情報の内容によって、ビジネス・オブジェクトを処理するコンポーネントに対応したメタデータが提供されます。アプリケーション固有の情報は、一般的に、ビジネス・オブジェクトの処理の仕方に対するアプリケーション依存の指示を、コネクターまたはデータ・ハンドラーに提供する目的に使用されます。アプリケーション固有の情報は、ビジネス・オブジェクト設計時に入力されるストリングで、実行時にコネクターまたはデータ・ハンドラーが読み取ります。

注: アプリケーション固有のビジネス・オブジェクトの定義に含まれるアプリケーション固有の情報を使用するように設計されたコネクターのことをメタデータ主導型コネクター と呼びます。処理情報がハードコーディングされずに構成可能となっているため、メタデータ主導型コネクターは、メタデータ主導型でないものに比べると、はるかに柔軟であり保守も容易です。

ビジネス・オブジェクト定義に含まれるアプリケーション固有の情報は、以下の 3 つのレベルのいずれかで指定できます。

- ビジネス・オブジェクト定義
- ビジネス・オブジェクト定義の内部にある属性
- ビジネス・オブジェクトの動詞

アプリケーション固有の情報は、AppSpecificInfo プロパティーと呼ばれる、ビジネス・オブジェクト定義内のフィールドに保存されます。 AppSpecificInfo プロパティーの値は、該当のビジネス・オブジェクトまたはアプリケーションについて任意の情報を記述できるテキスト・ストリングです。図 2 に、ビジネス・オブジェクト定義の主な要素と、各要素に対するアプリケーション固有のプロパティーを示します。

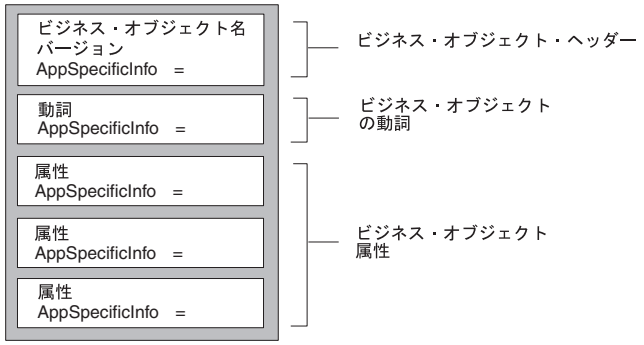


図 2. 各要素に対するアプリケーション固有のプロパティーを示すビジネス・オブジェクト定義

このセクションの内容は次のとおりです。

- ビジネス・オブジェクトのアプリケーション固有の情報
- 属性のアプリケーション固有の情報
- 動詞のアプリケーション固有の情報

ビジネス・オブジェクトのアプリケーション固有の情報: ビジネス・オブジェクト・レベルのアプリケーション固有の情報は、コネクターまたはデータ・ハンドラーがデータの処理に使う情報の提供に使用します。ビジネス・オブジェクト・レベルのアプリケーション固有の情報は、処理命令がビジネス・オブジェクト階層全体にかかわる場合は常に使用されます。例えば、オブジェクト・レベルのアプリケーション固有の情報は、次のような目的で使用されます。

- ビジネス・オブジェクト・トランザクション処理の範囲を定義すること。
- アプリケーション拡張部分でのオブジェクト処理を必要とするアプリケーションの場合、ビジネス・オブジェクトを処理するために呼び出す関数の名前を格納すること。

- レコードが属するテーブルまたはフォームの名前を指定すること。
- ビジネス・オブジェクト内にある属性のうち、論理的削除または「ソフト (回復可能)」な削除を表す属性の名前を指定すること。

図 3 に、アプリケーション内でフォーム名またはテーブル名を識別するためのアプリケーション固有の情報を示します。コネクターは、AppSpecificInfo プロパティーからテーブル名またはフォーム名を取得し、API 呼び出しにおいて、取得した名前を使ってアプリケーションからデータを検索することができます。

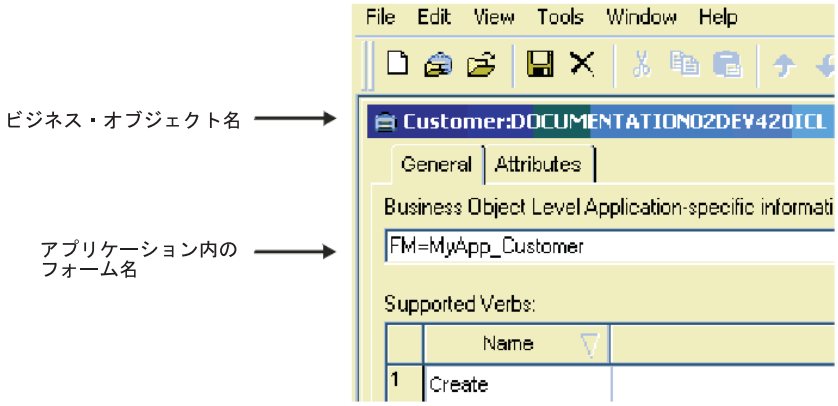


図 3. ビジネス・オブジェクトのアプリケーション固有の情報

属性のアプリケーション固有の情報: ビジネス・オブジェクト定義の各属性は、対応するアプリケーション固有の情報を持つことができます。属性レベルのアプリケーション固有の情報は、処理命令が単一の属性にかかわる場合は常に使用されます。この情報には、例えば、フォーム上のフィールド、テーブル内の列、またはコネクターが属性を見つけたり処理するために必要な任意の情報を指定できます。ビジネス・オブジェクトの特定の属性がアプリケーション内の特定のサブフォームに配置する場合、この情報をエンコードする場所としては AppSpecificInfo プロパティーが適しています。

図 4 に、AppSpecificInfo 属性のプロパティーを示します。この例では、アプリケーション固有の情報は、サブフォームおよびフィールドの名前を指定します。

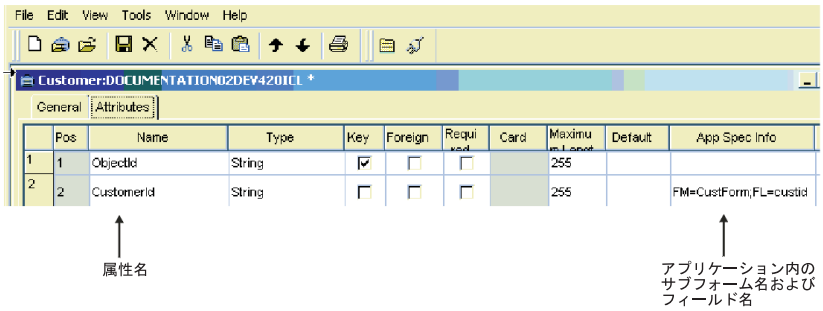


図 4. 属性のアプリケーション固有の情報

図 5 に、オブジェクト・レベルおよび属性レベルのアプリケーション固有の情報に指定されるフォーム、サブフォーム、およびフィールド名の関係を示します。この例では、請求のためのアプリケーションを示しますが、このアプリケーションはフォームをベースとしていて、このアプリケーション内の請求書とは、メイン・フォーム CustAccount のサブフォームである Invoice フォームを介して対話するものとします。Invoice サブフォーム上にあるフィールドは、CustName、CustAddr、InvNum、DollarAmount、および Terms です。

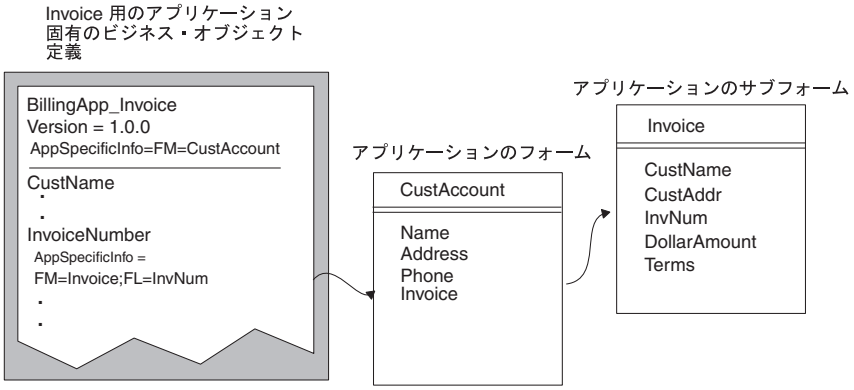


図 5. ビジネス・オブジェクト定義のアプリケーション固有の情報の使用

図 5 では、属性レベルの AppSpecificInfo プロパティーを使用して、Invoice サブフォームの名前と属性内の対応フィールドの名前を格納しています。この例では、情報の指定には名前と値のペアを使用しています。

動詞のアプリケーション固有の情報: 各動詞定義には、その動詞がアクティブなときにビジネス・オブジェクトをどのように処理するかに関する命令をコネクターまたはデータ・ハンドラーに知らせるアプリケーション固有の情報を組み込むことができます。

注: ビジネス・オブジェクト・ハンドラーは、InterChange Server Express からコネクターへ送られた要求を処理するコネクターの一部となっているため、それらのアプリケーション固有のビジネス・オブジェクト定義の動詞に含まれるアプリケーション固有の情報を使用する設計にできます。そのようなビジネス・オブジェクト・ハンドラーのことを*メタデータ主導型ビジネス・オブジェクト・ハンドラー* と呼びます。処理情報は、ハードコーディングされずに構成可能となっているため、メタデータ主導型ビジネス・オブジェクト・ハンドラーは、メタデータ主導型でないものに比べると、はるかに柔軟であり保守も容易です。

例えば、アプリケーション・データベースの更新を処理するのに API を使用しているコネクターは、アプリケーション固有の情報から API を実行するための情報を得ることができます。

動詞のアプリケーション固有の情報も、ビジネス・オブジェクトの処理のために、アプリケーション内で呼び出す関数の名前の指定に使用できます。

ビジネス・オブジェクトのインスタンス

ビジネス・オブジェクト定義は、データのコレクション用のテンプレートを意味するのに対し、ビジネス・オブジェクトのインスタンス (一般的には単に「ビジネス・オブジェクト」と呼ばれる) は、実際のデータを含んだランタイム・エンティティーです。ビジネス・オブジェクトは、ビジネス・インテグレーション・システムのコンポーネント間で渡されるものです。

ビジネス・オブジェクトに格納される情報は次のとおりです。

- 属性。各属性には関連するビジネス・オブジェクトのデータが含まれています。通常、属性のうちの 1 つはキー属性となっています。キー属性には、同じ定義を持つすべてのビジネス・オブジェクトの中から、該当のビジネス・オブジェクトのみを一意に識別する値が含まれています。
- アクティブな動詞。この動詞はビジネス・オブジェクト定義用にサポートされている動詞のうちの 1 つに該当していなければなりません。

図 6 に、`Customer` ビジネス・オブジェクト定義、およびこの定義に対応するビジネス・オブジェクトのインスタンスを示します。

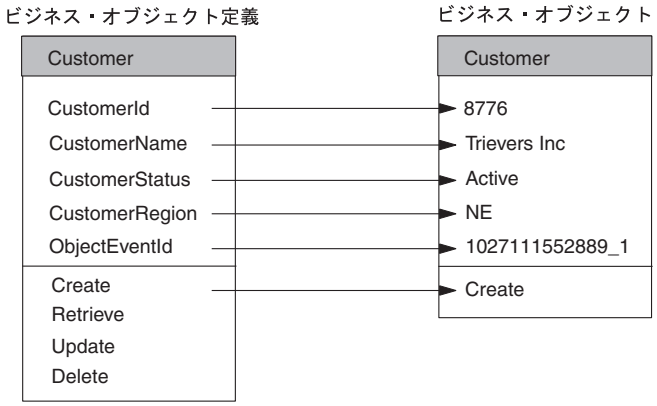


図 6. ビジネス・オブジェクト定義およびビジネス・オブジェクトの例

ビジネス・オブジェクトの構造

ビジネス・オブジェクトの構造は、以下のどちらかです。

- フラット・ビジネス・オブジェクト
- 階層型ビジネス・オブジェクト

以降のセクションでは、フラットおよび階層型のビジネス・オブジェクト構造の例を示し、ビジネス・オブジェクト構造がコネクターのロジックにどのように影響するかについて説明します。

フラット・ビジネス・オブジェクト

フラット・ビジネス・オブジェクトのビジネス・オブジェクト定義には、1 つ以上の単純属性とサポートされている動詞のリストが含まれています。*単純属性* とは、`String`、`Integer`、`Date` などのような単一値をいいます。単純属性はすべて、単一

カーディナリティーを持ちます。 図 6 の Customer ビジネス・オブジェクトは、フラット・ビジネス・オブジェクトの例です。詳しくは、 5 ページの『ビジネス・オブジェクトの属性および属性プロパティー』を参照してください。

階層型ビジネス・オブジェクト

階層型ビジネス・オブジェクト定義 では、各個別のエンティティーだけでなくエンティティー間の関係の詳細もカプセル化することで、複数の関連エンティティーの構造を定義します。階層型ビジネス・オブジェクトには、少なくとも 1 つの単純属性を含まれているほか 1 つ以上の *複合* 属性があります。つまり、属性自体に、*子ビジネス・オブジェクト* と呼ばれるビジネス・オブジェクトが 1 つ以上含まれています。複合属性が格納されているビジネス・オブジェクトのことを *親ビジネス・オブジェクト* といいます。

親ビジネス・オブジェクトと子ビジネス・オブジェクト間の関係は 2 種類あります。

- *単一カーディナリティー* — 親ビジネス・オブジェクト内の属性が *単一* の子ビジネス・オブジェクトを表す場合。属性のタイプは、子ビジネス・オブジェクトの名前に設定され、カーディナリティーは 1 に設定されます。
- *複数カーディナリティー* — 親ビジネス・オブジェクト内の属性が子ビジネス・オブジェクトの *配列* を表す場合。属性のタイプは、子ビジネス・オブジェクトの名前に設定され、カーディナリティーは `n` に設定されます。

そして、子ビジネス・オブジェクトもそれぞれ、子ビジネス・オブジェクトまたはビジネス・オブジェクトの配列を含んだ属性を包含できます。この関係は階層の下に向かって続きます。階層構造のトップにあるビジネス・オブジェクト (階層構造自体には、親がない場合) のことを *トップレベルのビジネス・オブジェクト* といいます。単一のビジネス・オブジェクトはいずれも、自身に包含される (または自身を包含する) その子ビジネス・オブジェクトから独立していることから、*独立ビジネス・オブジェクト* と呼ばれます。

一般的なビジネス・オブジェクト階層の場合、トップレベルのビジネス・オブジェクト定義には、1 つ以上の単純属性、子ビジネス・オブジェクトまたは子ビジネス・オブジェクトの配列を表す 1 つ以上の属性、およびサポートされている動詞のリストがあります。典型的な階層型ビジネス・オブジェクトを、図 7 に示します。トップレベルのビジネス・オブジェクト (Customer) の属性には、単一カーディナリティーと、子ビジネス・オブジェクトを持つ複数カーディナリティーの両方があります。

- その Address 属性は、複数カーディナリティーを持つ複合属性です。Customer は、それぞれの Address 子ビジネス・オブジェクトの親ビジネス・オブジェクトとなっています。
- その CustProfile 属性は、単一カーディナリティーを持つ複合属性です。Customer は、単一の CustProfile 子ビジネス・オブジェクトの親ビジネス・オブジェクトとなっています。

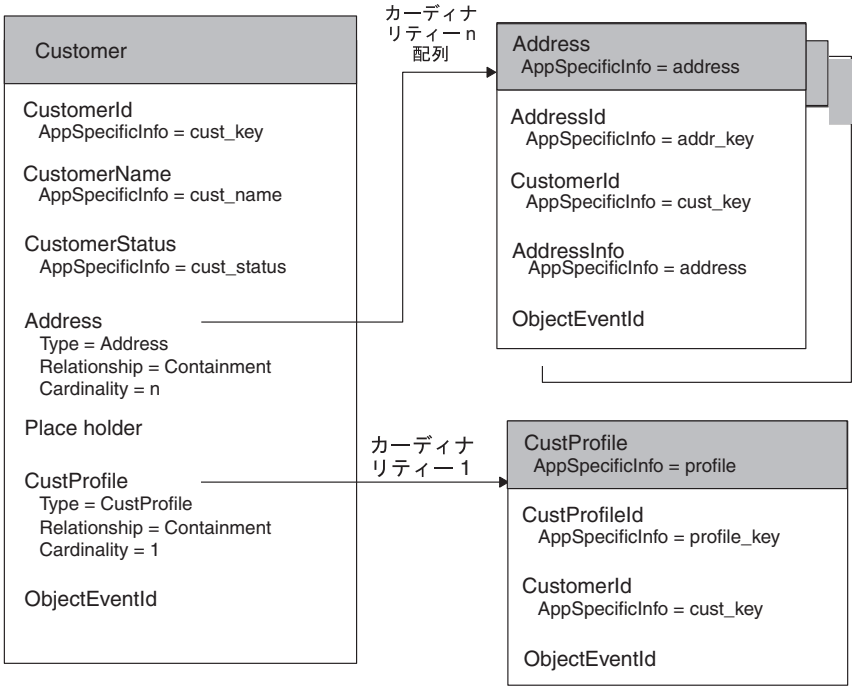


図 7. 階層型ビジネス・オブジェクト定義の例

図 7 では、Customer ビジネス・オブジェクト、CustProfile ビジネス・オブジェクト、およびそれぞれの Address ビジネス・オブジェクトは、独立ビジネス・オブジェクトとなっています。

注: トップレベルのビジネス・オブジェクトは、その子ビジネス・オブジェクトの処理に使用される情報を包含している場合、ラッパー・ビジネス・オブジェクトと呼ばれます。例えば、XML コネクターの場合、ラッパー・ビジネス・オブジェクトは、子データ・ビジネス・オブジェクトのフォーマットを決定し、その子への経路を指定する情報を格納している必要があります。

階層型アプリケーション固有のビジネス・オブジェクトの構造を設計する際、次のことを決定する必要があります。

- ビジネス・オブジェクト内でのエンティティー・データの表現方法
- 1 次アプリケーション・エンティティーと子エンティティーの関係
- アプリケーション・エンティティーに異なるエンティティーからのデータが含まれている場合は、次のことを決定する必要があります。
  - アプリケーション固有のビジネス・オブジェクトに関連データを組み込む必要があるかどうか
  - 関連データ間の関係を定義する方法

詳しくは、 28 ページの『複数エンティティーを設計する上での考慮事項』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、ビジネス・オブジェクト開発プロセスの概要を説明します。

開発環境の設定

開発プロセスを開始する前に、次の事項を確認してください。

- WebSphere Business Integration システムが、ユーザーがアクセスできるコンピューター上にインストールされていること。

  WebSphere Business Integration システムのインストールと始動の方法については、WebSphere Business Integration Server Express の「インストール・ガイド」を参照してください。

- InterChange Server Express およびそのリポジトリーのデータベース・サーバーも稼働しています。このステップは、リポジトリーに定義を保管する準備またはリポジトリーから定義を削除する準備が完了しているときにのみ必要です。開発のみの場合には、InterChange Server Express に接続せずに、Business Object Designer Express をローカルで実行できます。
- Object Discovery Agent (ODA) を使用してビジネス・オブジェクト定義を生成する場合は、Object Activation Daemon (OAD) を起動しなければ、その ODA を使用してビジネス・オブジェクト定義を生成することはできません。詳しくは、75 ページの『ODA を使用するための準備』を参照してください。

ビジネス・オブジェクト開発ステージ

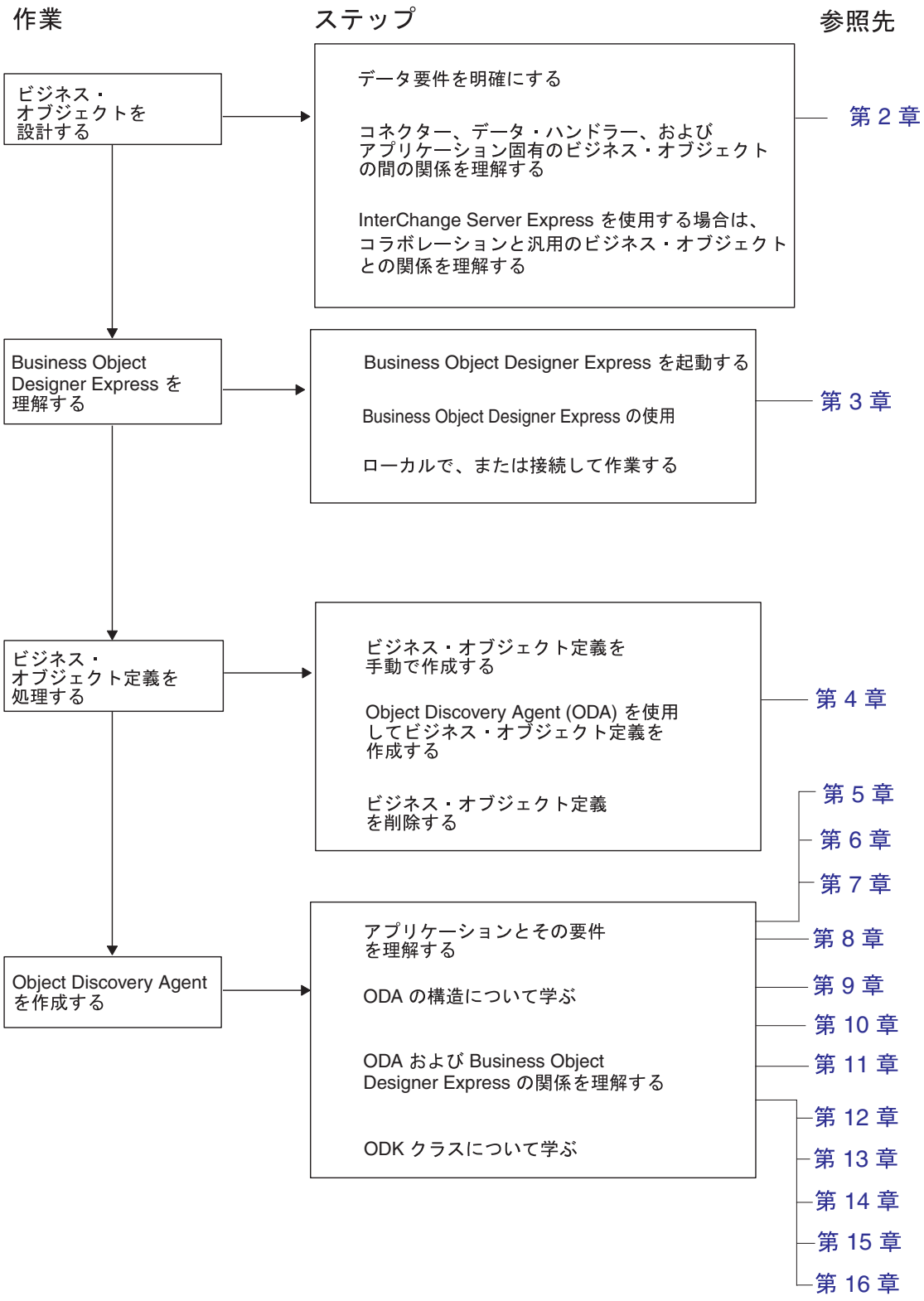
ビジネス・オブジェクトの開発ステージは次のとおりです。

1. ビジネス・プロセスの統合にとって重要なデータ要件を把握します。
   - アプリケーション固有のビジネス・オブジェクトを作成する場合には、コネクター、データ・ハンドラー、およびサポートされているアプリケーション固有のビジネス・オブジェクトの間の関係を把握します。
   - InterChange Server と併用する汎用ビジネス・オブジェクトを作成する場合は、コラボレーション・オブジェクトとビジネス・オブジェクトの間の関係を把握します。
2. ビジネス・オブジェクト定義を以下のどちらかの方法で作成します。
   a. データ・ソースからの生成 — WebSphere Business Integration システムは、いくつかのコネクターに関するビジネス・オブジェクト定義の生成を容易にするためのツールを提供しています。そのようなツールとしては、Object Discovery Agent やコマンド・ライン・ツールがあります。これらのツールは、アプリケーションへの接続、ビジネス・エンティティー固有のビジネス・オブジェクト要件の明確化、およびそれらの要件からの定義の生成を目的に設計されたものです。Business Object Designer Express は、Object Discovery Agent に対するグラフィカル・ユーザー・インターフェースを提供し、検出および定義生成のプロセスの管理を容易にします。有用なツールまたはユーティリティーがあるかどうかを調べるには、使用するアダプターおよびデータ・ハンドラーのガイドを確認してください。また、資料のメイン・ページのコネクターのカテゴリーの下にある『コネクター機能チェックリスト』にも有用な情報があります。カスタム・アダプターがアプリケーションと通信するように開発されている場合は、Object Discovery Agent Kit を使用して、アダプター用のカスタム Object Discovery Agent を作成できます。
   b. 手動 — Business Object Designer Express は、ビジネス・オブジェクト定義を手動で簡単に作成できるグラフィカル・ユーザー・インターフェースで

す。オブジェクト・ディスカバリーが実行可能なアプリケーションは存在しないため、InterChange Server Express で使える汎用ビジネス・オブジェクトを作成する場合に、このインターフェースが最も役に立ちます。

3. データ・ソースからビジネス・オブジェクト定義を自動的に生成するツールを使用した場合は、生成された構造およびアプリケーション固有の情報が要件に適合していることを検証してください。手動での作成を要する特別な構成を判別するには、ビジネス・オブジェクト定義を使用するコネクターのアダプター・ガイドを参照してください。
4. ビジネス・オブジェクトをシステム上で実行することにより、テストとデバッグを行い、必要に応じて編集します。

次の図に、ビジネス・オブジェクト開発プロセスの概要を示します。また、特定のトピックについての情報がどの章に記載されているかについても示します。


作業
ステップ
参照先
ビジネス・
オブジェクトを
設計する
データ要件を明確にする
コネクター、データ・ハンドラー、および
アプリケーション固有のビジネス・オブジェクト
の間の関係を理解する
InterChange Server Express を使用する場合は、
コラボレーションと汎用のビジネス・オブジェクト
との関係を理解する
第 2 章
Business Object
Designer Express を
理解する
Business Object Designer Express を起動する
Business Object Designer Express の使用
ローカルで、または接続して作業する
第 3 章
ビジネス・
オブジェクト定義を
処理する
ビジネス・オブジェクト定義を
手動で作成する
Object Discovery Agent (ODA) を使用
してビジネス・オブジェクト定義を
作成する
ビジネス・オブジェクト定義
を削除する
第 4 章
第 5 章
第 6 章
第 7 章
Object Discovery Agent
を作成する
アプリケーションとその要件
を理解する
ODA の構造について学ぶ
ODA および Business Object
Designer Express の関係を理解する
ODK クラスについて学ぶ
第 8 章
第 9 章
第 10 章
第 11 章
第 12 章
第 13 章
第 14 章
第 15 章
第 16 章

第 2 章 ビジネス・オブジェクト設計

ビジネス・オブジェクト設計上のかぎは、ビジネス・インテグレーション・システムのコンポーネント間での転送の必要なデータが可能な限り詳細に (かつ効率的に) モデル化されるようにビジネス・オブジェクト定義を作成することにあります。

- コネクターと InterChange Server Express の間で転送されるデータに関しては、適切なアプリケーション・エンティティーをモデル化するアプリケーション固有のビジネス・オブジェクト を設計してください。これらのエンティティーは、特定のアプリケーションによって使用されるデータ構造またはテクノロジー規格、あるいは Web サーバーによって使用される特定のテクノロジー規格に対応すると考えられます。
- InterChange Server Express コラボレーション・オブジェクトのビジネス・ロジックの内部で処理されるデータに関しては、やり取りの必要なアプリケーション・エンティティーに関する情報のスーパーセットを含めた汎用ビジネス・オブジェクト を設計してください。コラボレーション・オブジェクトがアプリケーションと情報をやり取りする際には、汎用ビジネス・オブジェクトとアプリケーション固有のビジネス・オブジェクトの間でデータがマップ変換されます。

この章では、WebSphere Business Integration システムのビジネス・オブジェクトの構造について説明したうえで、アプリケーション固有のビジネス・オブジェクトと汎用ビジネス・オブジェクトの両方の設計に関する推奨事項を紹介します。この資料は、読者が「システム・インプリメンテーション・ガイド」に記載されている基本的なオブジェクトの概念を理解していることを前提にしています。

この章の主な内容は次のとおりです。

ビジネス・オブジェクトの構造の判別

ビジネス・オブジェクトは、ビジネス・インテグレーション・システムのコンポーネントと、そのシステムによって統合されるアプリケーションとの間でのデータ移送を目的としたものです。その理由から、移送しなければならないデータは、ビジネス・オブジェクトによってモデル化する必要があります。このデータは通常、ビジネス・インテグレーション・システムによって統合されるアプリケーション、またはテクノロジーのエンティティーに関連付けられます。ビジネス・オブジェクトの構造は、以下のどちらかです。

このセクションでは、さらに 28 ページの『複数エンティティーを設計する上での考慮事項』についても取り上げます。

単一エンティティーの表現

最も単純なビジネス・オブジェクトの設計は、1 つのエンティティーを表すフラット・ビジネス・オブジェクトです。フラット・ビジネス・オブジェクトの属性はいずれも単純であり、各属性が 1 つの値 (例えば `String`、`Integer`、`Date` など) を表します。詳しくは、12 ページの『フラット・ビジネス・オブジェクト』を参照してください。

アプリケーション固有のビジネス・オブジェクトの場合、フラット・ビジネス・オブジェクトは、アプリケーションまたはテクノロジーの規格に含まれる 1 つのエンティティーを表します。ここで例として、レコードを記述するデータベース表を持つアプリケーションを考えます。また、この表には ObjectID (オブジェクト ID)、UserName (ユーザー名)、TimeStamp (タイム・スタンプ)、Detail (詳細)、および Status (状況) という 5 つの列 (図 8 を参照) があるとします。ObjectID は各行の基本キーとなっていて、その値はアプリケーションによって生成されます。この表は、他の表との関係はありません。

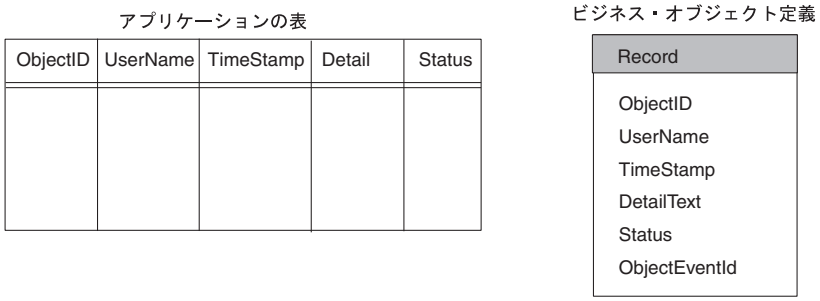


図 8. 単一のエンティティーを表すフラット・ビジネス・オブジェクト

図 8 に示すように、表を表す目的に Record ビジネス・オブジェクトを設計する場合、各列に 1 つずつ合計で 5 つの属性を与え、そのキー属性を ObjectID 列に対応させます。

フラット・ビジネス・オブジェクトを使用することにより、対応するコネクターの設計を以下のように簡略化できます。

- Create 処理時には、コネクターはその属性を 1 つずつ調べて、ビジネス・オブジェクトのインスタンスから非キー属性の値を抽出し、ビジネス・オブジェクト定義から処理命令を抽出します。コネクターは、ビジネス・オブジェクトの処理に必要な情報を組み立てると、アプリケーション関数呼び出しまたは SQL ステートメントを開始して、該当するレコードに対応する新しい行を表内に作成します。コネクターは続いて、そのキーに対応する値をビジネス・インテグレーション・システムに戻します。
- Retrieve 処理時には、コネクターは、ビジネス・オブジェクト要求から基本キーを抽出し、そのキー値を使用して該当する行に対応する現在のデータのセットを取り出し、値の完全なセットとともにビジネス・オブジェクトを戻します。

このタイプのビジネス・オブジェクトは、設計も、処理に必要なコネクター・ロジックも単純です。しかし、一般にアプリケーション・エンティティーはより複雑で、他のオブジェクトに格納されている情報が組み込まれます。

複数エンティティーの表現

ビジネス・オブジェクトは、表 3 に示す方法の 1 つを使用して、他のエンティティーからのデータを組み込むアプリケーション・エンティティーを表現できます。

表 3. 複数エンティティーの表現

ビジネス・オブジェクトの構造	データ編成のタイプ	親子関係のタイプ
親ビジネス・オブジェクトは、他のエンティティーを表す子ビジネス・オブジェクトを 1 つ以上持つことができます。	1 対 1 1 対多	構造的
親ビジネス・オブジェクトは、他のエンティティーを表す他のトップレベルのビジネス・オブジェクトを参照する外部キー属性を 1 つ以上持つことができます。	1 対 1 1 対多 多対多 多対 1	意味的
アプリケーションおよびそのインターフェースによって許可される場合、フラット・ビジネス・オブジェクトは、他のエンティティーを直接参照する属性を持つことができます。	1 対 1	なし

複数のエンティティーを表すビジネス・オブジェクトを構造化する方法を決定する場合は、次の指針に基づいて検討してください。

- エンティティー間の関係が 1 に対する多の関係の場合、従属エンティティー内のデータは子ビジネス・オブジェクトとして表します。以下に例を示します。
  - データベース表での作業では、あるエンティティーの行が別のエンティティー内の 1 つ以上の行に関連していて、そのエンティティーが従属エンティティーに関連する唯一のエンティティーの場合、関連する行ごとに別個の子ビジネス・オブジェクトを作成します。
  - DTD での作業では、ある XML 要素が * のカーディナリティーの属性を持つ場合、関連する要素の属性ごとに別個の子ビジネス・オブジェクトを作成します。
- エンティティー間の関係が多に対する多の関係の場合、関連エンティティー内のデータは、親によって格納されるのではなく、親によって参照されるトップレベルのビジネス・オブジェクトとして表します。
- あるエンティティーのビジネス・オブジェクト定義に、別のエンティティーからの属性が多数含まれていて、この第 2 のエンティティーからの属性が論理的なグループを形成している場合、両エンティティーのすべての属性を同じビジネス・オブジェクト定義に入れるよりも、第 2 のエンティティーに対応した子ビジネス・オブジェクト定義を作成する方が便利です。
- ある既存のビジネス・オブジェクトが既に他の子ビジネス・オブジェクトを含んでいる場合、新しいエンティティーを表す 1 つ以上の子ビジネス・オブジェクトを作成することにより、ビジネス・オブジェクトの構造の整合性を取ることができます。

これらの各表現については、以降のセクションで詳しく説明します。

構造的関係

構造的関係では、親ビジネス・オブジェクトはその子ビジネス・オブジェクトを物理的に含めることができます。そのようなビジネス・オブジェクトは、複合属性が少なくとも 1 つはある (属性に子ビジネス・オブジェクトまたはその配列のどちらかが含まれる) という点では、階層型ビジネス・オブジェクトであるといえます。この属性の Relationship 属性プロパティーは、包含関係を示す `containment` です。この属性のタイプは、その属性によって表される子ビジネス・オブジェクト (またはオブジェクト) のタイプとなっています。詳しくは、13 ページの『階層型ビジネス・オブジェクト』を参照してください。

以下の階層型ビジネス・オブジェクトは、構造的関係を表します。

- 注文は明細で構成されているため、Order ビジネス・オブジェクトには LineItem ビジネス・オブジェクトの配列が格納されます。各注文には明細を複数含めることができるため、その包含関係は複数カーディナリティーを持っているといえます。この構造は 1 対多の関係を表します。
- 従業員は 1 つの住所に関連付けられているため、Employee ビジネス・オブジェクトには Address ビジネス・オブジェクトが 1 つ含まれています。各従業員を関連付けることのできる住所は 1 つのみであるため、その包含関係は単一カーディナリティーを持っているといえます。この構造は 1 対 1 の関係を表します。

いずれの場合も、親ビジネス・オブジェクトは子または子の配列を持つため、その関係は構造的に定義されます。

構造的な関係の場合、親ビジネス・オブジェクトは子オブジェクト内のデータを所有することが前提となります。したがって、社員を新規に作成した場合、その従業員の住所が保持されるように、住所表に新規の行が挿入されます。同様に、従業員を削除すると、その従業員の住所も住所表から削除されます。

意味的関係

意味的な関係では、親ビジネス・オブジェクトがその子を参照するか、子が親を参照します。1 つのビジネス・オブジェクトが別のビジネス・オブジェクトを参照する場合、参照する側のオブジェクトは参照先のオブジェクトを一意に識別する値を格納しますが、オブジェクト自身は格納しません。この場合、ビジネス・オブジェクトを処理するコンポーネントは、この関係を意味的に導き出します。

一般に、意味的な関係は、*外部キー* として機能する単純属性によって定義されます。外部キー属性は、一方のビジネス・オブジェクト内に位置し、他のビジネス・オブジェクトの固有 ID (*基本キー* と呼ばれる) を保持しています。つまり、両方のビジネス・オブジェクトとも固有な ID を保持する基本キー属性を持っています。また、ビジネス・オブジェクトの 1 つは、他のビジネス・オブジェクトの基本キーの値を保持するための、外部キー属性を備えています。この外部キーにより、親と子の間に意味的にリンクが確立されます。

意味的な関係は、エンティティー間に多に対する多または多対 1 の関係がある場合、言い換えれば複数の親が同じ子との関係を持っている場合に重要になります。エンティティーを構造的にではなく意味的に関連付けると、子のデータが分離されるため、この意味的な関連付けはデータの整合性維持には重要であるといえます。

意味的に定義された関係では親に子が包含されないため、この親子に対する要求を処理するコネクターは、親に対する要求と子に対する要求をそれぞれ別個の操作で受け取ることになります。つまり、コネクターは、要求がそれぞれ別個に送られてくるため、親と子をそれぞれ別個の操作で処理します。詳しくは、28 ページの『関係におけるデータの所有権』 および 30 ページの『意味的関係と構造的関係からの選択』を参照してください。

意味的関係を指定するために、表 4 の設計オプションを考えてみましょう。

表 4. 意味的関係のための設計オプション

設計オプション	関係のタイプ
『親オブジェクトへの外部キーの格納』	1 対 1 多対 1
24 ページの『子オブジェクトへの外部キーの格納』	1 対多
25 ページの『子オブジェクト配列への外部キーの格納』	1 対多 多対多
26 ページの『ビジネス・オブジェクト・ツリーへの外部キーの格納』	1 対 1

親オブジェクトへの外部キーの格納: 外部キーの使い方がごく単純な場合は、関係を確立するための外部キーが親に格納されます。この場合、親が格納できるのは、指定されたタイプの 1 つの子への参照のみです。親と子の間の関係は、親の内部で明確に定義されます。そのため、この構造は 1 対 1 の関係を表します。ただし、複数の親ビジネス・オブジェクトから同じ子ビジネス・オブジェクトを参照すると、多対 1 の関係を実装できます。

注: 関係を確立するための外部キーを親に格納する場合、親はそれぞれ 1 つの子への参照を持つ複数の属性を格納できますが、これらの属性は一般にそれぞれ異なるタイプの子を参照します。

図 9 では、Customer ビジネス・オブジェクトに 2 つの属性 (`AddressId` および `CustInfo`) があり、各属性には子ビジネス・オブジェクトへの参照が含まれています。Customer ビジネス・オブジェクト内の外部キー属性から、親と 2 つの子との関係が即座にわかります。

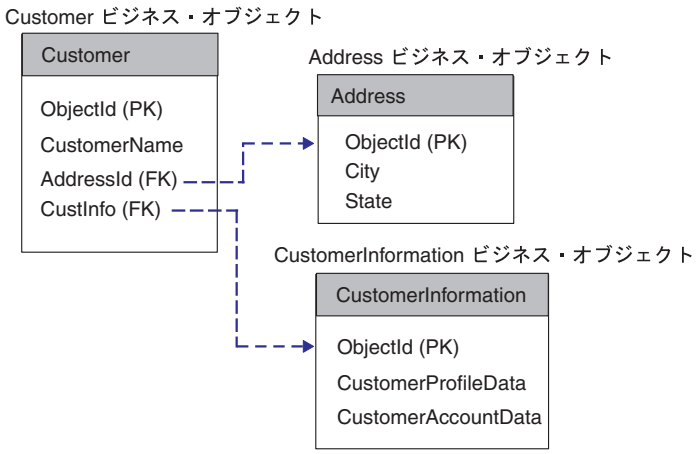


図 9. 1 対 1: 親ビジネス・オブジェクトに格納された複数の外部キー属性

注: 図 9 では、頭字語「PK」を使用して基本キーを表し、「FK」を使用して外部キーを表しています。また、これらのビジネス・オブジェクトは、基本キー属性に `ObjectId` という名前を付けることにより、汎用ビジネス・オブジェクト用の命名規則に従っています。アプリケーション固有のビジネス・オブジェクトでは、一般的には、アプリケーション内の対応するフィールドまたは列に名前を付けてから、属性に名前を付けるのが望ましいといえます。

InterChange Server Express を使用して、別のオブジェクトへの外部キー参照を格納する親オブジェクトの例として、付属の汎用 Order ビジネス・オブジェクトを調べることができます。このビジネス・オブジェクトには、トップレベルの汎用 Customer ビジネス・オブジェクトを参照先とする CustomerId 属性が含まれています。Order ビジネス・オブジェクトの図については、図 11 を参照してください。

子オブジェクトへの外部キーの格納: もう 1 つの選択肢として、関係を確立するための外部キーを子に格納する方法があります。この場合、1 対多の関係を表します。つまり、複数の子が同じ親を参照できるということです。ただし、親と子の間の関係は子の内部で定義されているため、親のみを調べた場合には、この関係が存在することがわかりません。その理由から、親ビジネス・オブジェクトによって統合のフローが起動された場合に、それらの子を検索できないため、システム全体を移動する親ビジネス・オブジェクトには、それらの子への参照は含まれません。

図 10 では、外部キー属性は、親ビジネス・オブジェクトではなく、それぞれの子ビジネス・オブジェクトに格納されています。この構造の場合、複数の子が同じ親と意味的に関係を持つことが可能です。ただし、この場合、親ビジネス・オブジェクトが子ビジネス・オブジェクトへの参照を格納した属性を持たないため、親とその子との関係を識別したり、指定された親に対して関連するすべての子を検出することはできません。

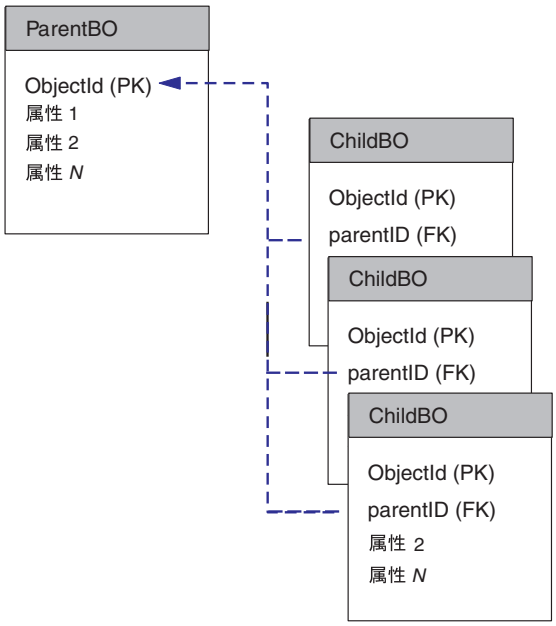


図 10. 多対 1: 外部キーが複数の子ビジネスに格納されている場合

注: 図 10 では、頭字語「PK」を使用して基本キーを表し、「FK」を使用して外部キーを表しています。

子オブジェクト配列への外部キーの格納: 1 対多の関係を表す場合、その関係を実際に確立するための外部キーを子ビジネス・オブジェクト内の単純属性に格納します。これらの子の配列は、親ビジネス・オブジェクト内に構造的に格納されます。言い換えると、親には子ビジネス・オブジェクトの配列が格納され、これらの子ビジネス・オブジェクトの配列にはそれぞれ、別のトップレベルのビジネス・オブジェクトへの外部キー参照が格納されるということです。さらに、子ビジネス・オブジェクト配列内にある同じ子ビジネス・オブジェクトが、複数の親ビジネス・オブジェクトから参照できることから、多対多の関係が実装されているといえます。

注: InterChange Server Express では、このタイプの親子関係の例として考察できるビジネス・オブジェクトがいくつか存在します。このオプションの例としては、汎用的な Order および ContactRef ビジネス・オブジェクトがあります。Order オブジェクトの OrderContactRef 属性には、汎用的な ContactRef ビジネス・オブジェクトの配列が格納されています。各 ContactRef ビジネス・オブジェクトは、トップレベルの汎用 Contact ビジネス・オブジェクトへの参照を保持した ContactId 属性を格納しています。

図 11 では、Order ビジネス・オブジェクトは、1 つの Customer ビジネス・オブジェクトへの参照を格納しており、同時に、構造的に ContactRef ビジネス・オブジェクトの配列を格納しています。各 ContactRef ビジネス・オブジェクトは、1 つの Contact ビジネス・オブジェクトへの参照を格納しています。

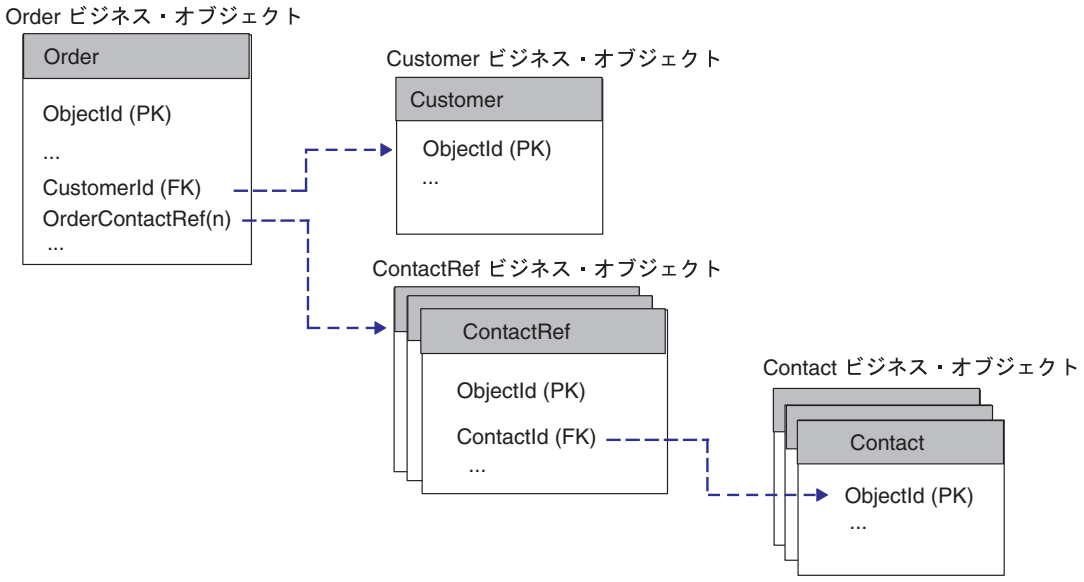


図 11. ビジネス・オブジェクトが外部キーが格納された子ビジネス・オブジェクトを格納している場合

注: 図 11 では、頭字語「PK」を使用して基本キーを表し、「FK」を使用して外部キーを表しています。

ビジネス・オブジェクト・ツリーへの外部キーの格納: この設計では、関係を確立するための外部キーが、同じタイプの別のビジネス・オブジェクトを親とする「子」ビジネス・オブジェクトに格納されます。InterChange Server Express では、この設計の例として汎用 InstalledProduct ビジネス・オブジェクトを考察できます。このビジネス・オブジェクトの ParentId 属性には、現在のビジネス・オブジェクトの直接の親である別の InstalledProduct ビジネス・オブジェクトへの参照を格納できます。

図 12 では、1 つの InstalledProduct ビジネス・オブジェクトの ParentId 属性に、その直接の親 InstalledProduct ビジネス・オブジェクトの基本キー (ObjectId) 属性への参照が格納されています。階層のヘッドにあたるのは、値が格納されていない ParentId 属性を持つビジネス・オブジェクトです。

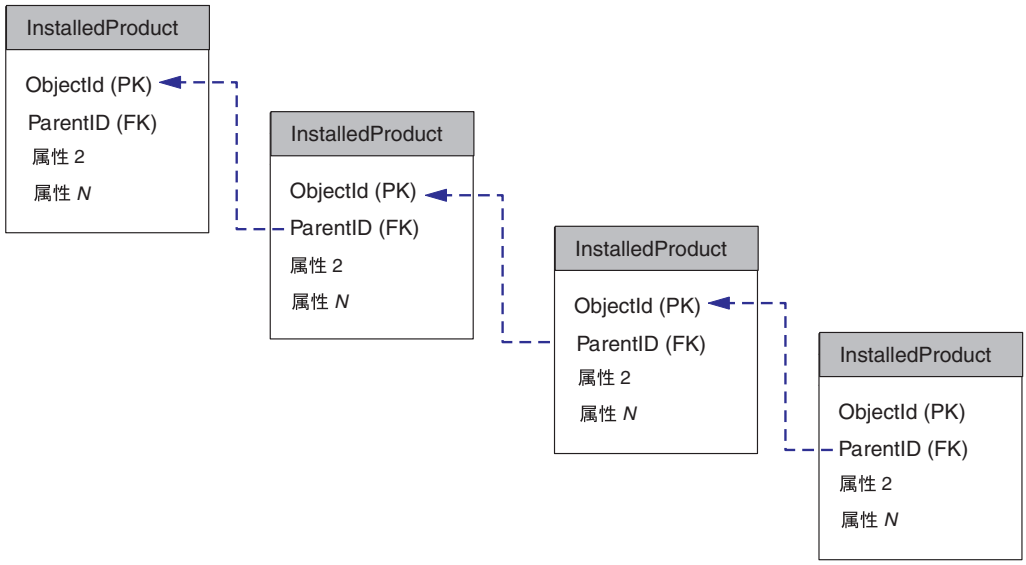


図 12. ビジネス・オブジェクトが同じタイプの親ビジネス・オブジェクトに外部キーを格納している場合

注: 図 12 では、頭字語 「PK」を使用して基本キーを表し、「FK」を使用して外部キーを表しています。

InstalledProduct ビジネス・オブジェクトにはその親ビジネス・オブジェクトへの参照を格納できるため、ビジネス・インテグレーション・システムでは、巨大な階層の一部となっているインストール済み製品を同期化できます。ビジネス・インテグレーション・システムでは、インストール済み製品の複雑な階層のコンポーネントを、個々の InstalledProduct ビジネス・オブジェクトとして管理できます。InterChange Server Express では、詳しくは「*InstalledProductSync*」のコラボレーション・テンプレートの資料を参照できます。

関連エンティティーを表すフラット・ビジネス・オブジェクト

複数のアプリケーション・エンティティーを結合して 1 つのビジネス・オブジェクトにする機能をアプリケーション・インターフェースが持っていると、基本エンティティーと関連エンティティーを参照する属性を格納したフラット・ビジネス・オブジェクトを定義できる場合があります。エンティティー間の関係が 1 対 1 の関係の場合、つまり基本エンティティーの 1 つのインスタンスが各関連エンティティーの 1 つのインスタンスに対応している場合、複数のエンティティーの属性を 1 つのビジネス・オブジェクトに組み込むことができます。

このタイプのアプリケーション固有のビジネス・オブジェクトを設計する場合、状況によっては、コネクターがデータを正しく検出して処理できるよう、アプリケーション固有の情報を使用して、アプリケーション内での属性データの位置を指定する必要があります。

図 13 に、2 つのエンティティー内のデータを表すフラット WebSphere Business Integration システムのビジネス・オブジェクトの例を示します。2 つのエンティティーのうち、一方は住所データを格納した表で、もう一方は州または地域および国の省略語の検索データを格納した表です。

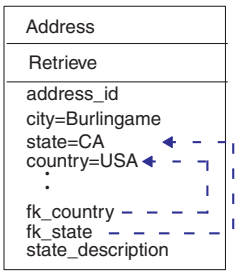


図 13. 2 つのエンティティーを表すフラット・ビジネス・オブジェクト

この例では、アプリケーション固有の情報を使用して、エンティティー間の外部キー関係を確立します。この場合、コネクターは 1 つの表を表す属性内の値から検索を実行し、別の表を表す属性の値を提供します。このデータを検出するため、コネクターは表の読み取りを 2 回行います。

フラット・ビジネス・オブジェクトは、複数のアプリケーション・エンティティーからの情報や、複数のアプリケーション・エンティティーに格納されている情報をカプセル化することができます。しかし、アプリケーション相互の統合の問題のために、フラット・ビジネス・オブジェクトでは表せない、より複雑な統合ロジックおよびデータ構造が必要とされることもあります。より複雑なアプリケーション・エンティティーおよび統合要件を扱えるように、WebSphere では階層型ビジネス・オブジェクトが提供されています。

複数エンティティーを設計する上での考慮事項

このセクションでは、複数エンティティーに対応したビジネス・オブジェクトを設計する際の考慮事項として、以下を取り上げます。

- 『関係におけるデータの所有権』
- 30 ページの『意味的関係と構造的関係からの選択』

関係におけるデータの所有権

ビジネス・オブジェクトを複数のエンティティーが表現するように設計にすると、データの所有権に以下のような影響が及びます。

- 構造的な関係の場合、親ビジネス・オブジェクトが子データを所有することが前提となります。
- 意味的な関係の場合、親ビジネス・オブジェクトが子オブジェクト内のデータを所有することを*前提にはしていません*。

この違いは、複数のビジネス・オブジェクトによって共有されるエンティティーのデータ整合性を考慮する場合に重要です。

ここで例として、ある顧客とある連絡先が同じ住所を共有している場合を考えます。Customer ビジネス・オブジェクトと Contact ビジネス・オブジェクトが、該当する Address ビジネス・オブジェクトそのものを含める (構造的な関係) のではなく、該当する Address ビジネス・オブジェクトへの参照を格納している (意味的な関係) 場合、この Address の変更は、Customer や Contact の変更と無関係に実行できます。

一方、Customer ビジネス・オブジェクトおよび Contact ビジネス・オブジェクトがそれぞれ Address ビジネス・オブジェクト自体を含めている場合は、Customer が Address を変更すると、Contact による変更が上書きされる可能性があります。この場合、2 つの異なるコラボレーション・オブジェクト (CustomerSync および ContactSync) が同じ住所データを同時に更新する可能性があるため、データの不整合が発生するおそれがあります。

Customer および Contact が Address ビジネス・オブジェクトと構造的な関係ではなく意味的な関係を持っている場合は、第 3 のインターフェースのみが Address データを変更できるように制限できます。例えば、Contact および Customer ビジネス・オブジェクトごとにそれぞれ 1 つずつインターフェースを持つことができます。その場合、それらのインターフェースの両方が、Address ビジネス・オブジェクトの管理を第 3 のインターフェースに委任できます。InterChange Server Express で、この委任を行うには、CustomerSync および ContactSync コラボレーション・オブジェクトに直接変更を行わせるのではなく、ラッパー・コラボレーション・オブジェクトを介して AddressSync を呼び出させます。InterChange Server Express の統合シナリオでのデータ整合性維持を目的としたビジネス・オブジェクトの設計について詳しくは、「*コラボレーション開発ガイド*」の『並列実行の設計』を参照してください。

図 14 に、子ビジネス・オブジェクトとの関係を意味的に定義する場合と構造的に定義する場合の違いを示します。

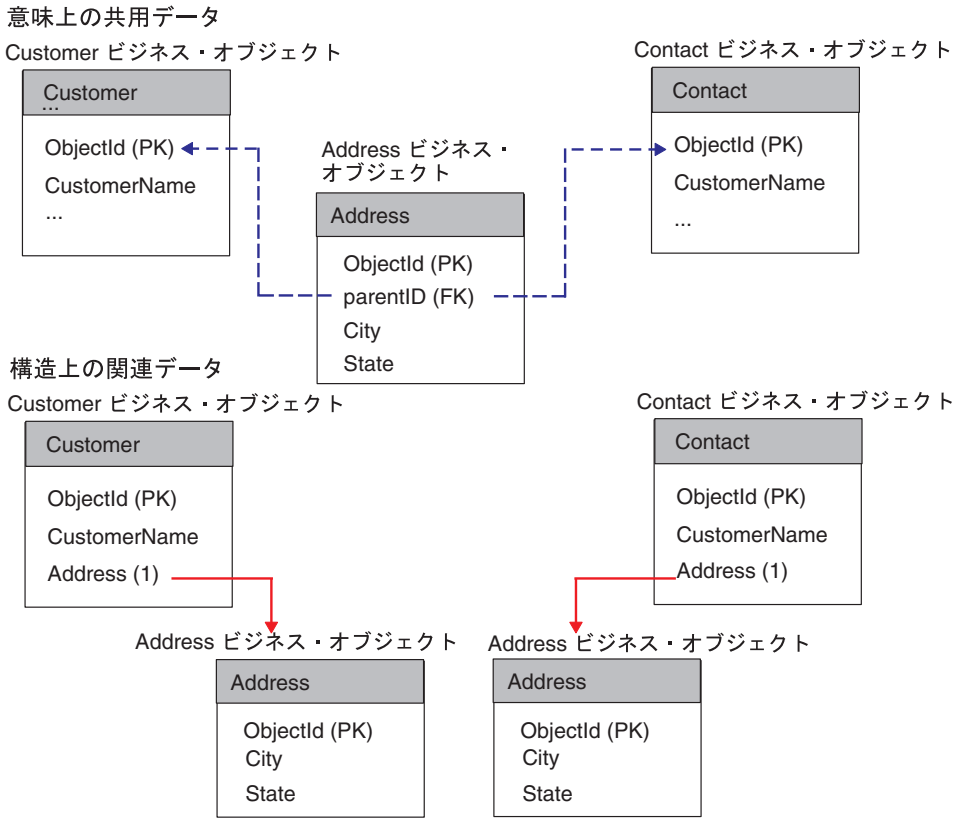


図 14. 意味的な関係と構造的な関係の比較

上の図では、子のデータとの関係が 2 種類存在しています。

- 意味的 — Customer および Contact の両オブジェクトに意味的にリンクされている子 Address ビジネス・オブジェクトは、親の基本キーの値を単純な外部キー属性に格納しています。この場合、両方の親が持つ基本キー属性の名前が同じであるため、子から親へのリンクは単純になります。
- 構造的 — Address に構造的にリンクしている 2 つのビジネス・オブジェクトは、子のインスタンスを表す属性を持ちます。この場合、子が持つデータは、その子を格納していて、かつ他の親と共有していない親のみに関連します。

意味的関係と構造的関係からの選択

21 ページの表 3 に示すように、1 対 1 の関係および 1 対多の関係は、構造的関係と意味的関係のどちらでも表現できます。これらの構造的表現および意味的表現を表 5 にまとめます。

表 5. 1 対 1 の関係および 1 対多の関係の表現

関係のタイプ	構造的表現	意味的表現
1 対 1 (単一カーディナリティー)	親ビジネス・オブジェクトの属性は、1 つの子ビジネス・オブジェクトを表す。	親ビジネス・オブジェクト内の属性は単純であり、この属性には 1 つの子ビジネス・オブジェクトを参照する外部キーが含まれている。
1 対多 (複数カーディナリティー)	親ビジネス・オブジェクト内の属性が、1 つの子ビジネス・オブジェクトの配列を表す。	複数の子ビジネス・オブジェクトにそれぞれ、親の基本キーを格納する外部キー属性が含まれている。

図 9 および図 10 は、単一カーディナリティーおよび複数カーディナリティーの関係が意味的に定義されているビジネス・オブジェクトを示したものです。これらのビジネス・オブジェクトは、データベース内に格納されたデータを表すと考えることもできます。このようなデータを表すビジネス・オブジェクト間の関係は、意味的かつ構造的に定義することができます。この種のデータの場合、親子関係は、同じ 2 つのビジネス・オブジェクト間で、意味的かつ構造的に定義することができます。

意味的関係の選択: 意味的関係を実装するには、その基礎となるアプリケーションによって外部キーがサポートされていなければなりません。例えば、あるビジネス・オブジェクトがデータベース・データを表す場合、このビジネス・オブジェクトは、エンティティー間の関係を意味的かつ構造的に確立することができます。このようなビジネス・オブジェクトは冗長な設計となります。言い換えると、これらのビジネス・オブジェクトを処理するコンポーネントは、親を介して子を見つけることができ、同時に個々の子を介して親を見つけることができます。

ここで例として、購入注文を表す表を持つアプリケーションを考えます。この表は、外部キーにより、1 つの購入注文に対応する明細を格納した表に関係付けられています。この明細表内の複数の行が、購入注文表内の 1 つの行を参照します。図 15 に、これらの表を示します。

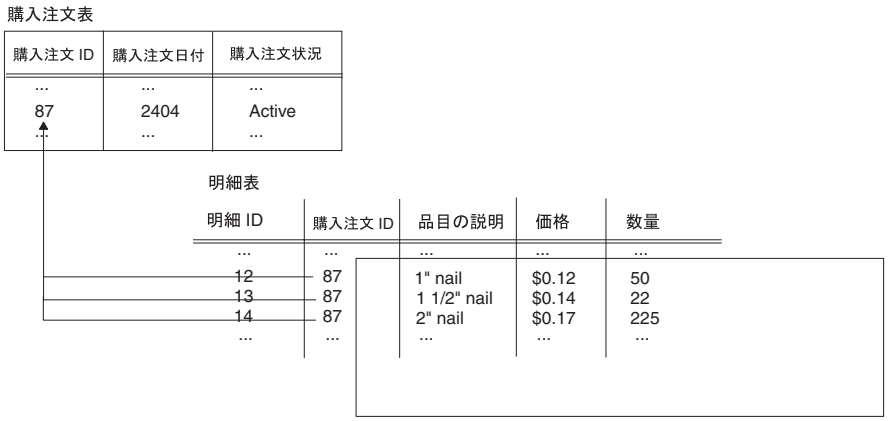


図 15. 1 対多の意味的関係を持つアプリケーション表の例

図 16 は、ビジネス・オブジェクトがこれらの表に対応している例を示したものです。この図は、トップレベルの PurchaseOrder ビジネス・オブジェクトと 3 つの子 LineItem ビジネス・オブジェクトが存在しています。

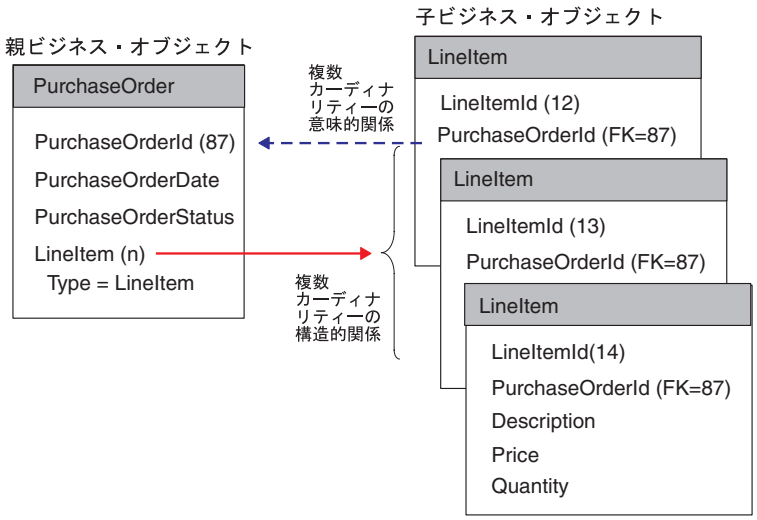


図 16. 複数カーディナリティーの意味的および構造的関係を持つビジネス・オブジェクトの例

図の PurchaseOrder ビジネス・オブジェクトは、その子ビジネス・オブジェクト LineItem に対して意味的かつ構造的な関係を持ちます。それぞれの子の PurchaseOrderId 属性は、親からその子への、外部キーによる意味的リンクを作成します。親の LineItem 属性は、カーディナリティー n で定義され、親から子への構造的リンクを作成します。

注: 外部キーを子に格納させているビジネス・オブジェクトは、IBM からは提供されていません。上記の例は、親と子のデータをリンクする方法の違いを示すために、参考として紹介しました。

構造的関係の選択: 基礎となるアプリケーションによって外部キーがサポートされていない場合は、構造的関係を実装する必要があると考えられます。例えば、1 つの XML 文書を表す DTD は外部キー情報をサポートしません。その理由から、1 対 1 の関係または 1 対多の関係をすべて構造的に定義する必要があります。次の

図に、1 つの Order アプリケーション・エンティティーに対応する要素を格納した Order DTD を示します。この DTD は単一カーディナリティーおよび複数カーディナリティーの関係を表しています。

```
<!--Order -->
<!-- Element Declarations -->
<!ELEMENT Order (Unit+)>
<!ELEMENT Unit (PartNumber?, Quantity, Price, Accessory*)>
<!ELEMENT PartNumber (#PCDATA)>
<!ELEMENT Quantity (#PCDATA)>
<!ELEMENT Price (#PCDATA)>
<!ELEMENT Accessory (Quantity, Type)>
<!ATTLIST Accessory
Name CDATA >
<!ELEMENT Type (#PCDATA)>
```

図 17 に、Order DTD を表すビジネス・オブジェクトを示します。トップレベルのビジネス・オブジェクトは単一カーディナリティーの関係を持つ Order ビジネス・オブジェクトを格納し、この Order オブジェクトは複数カーディナリティーの関係を持つ Unit 子ビジネス・オブジェクトを格納しています。さらに、この子 Unit オブジェクトは、複数カーディナリティーの関係を持つ Accessory ビジネス・オブジェクトを格納しています。

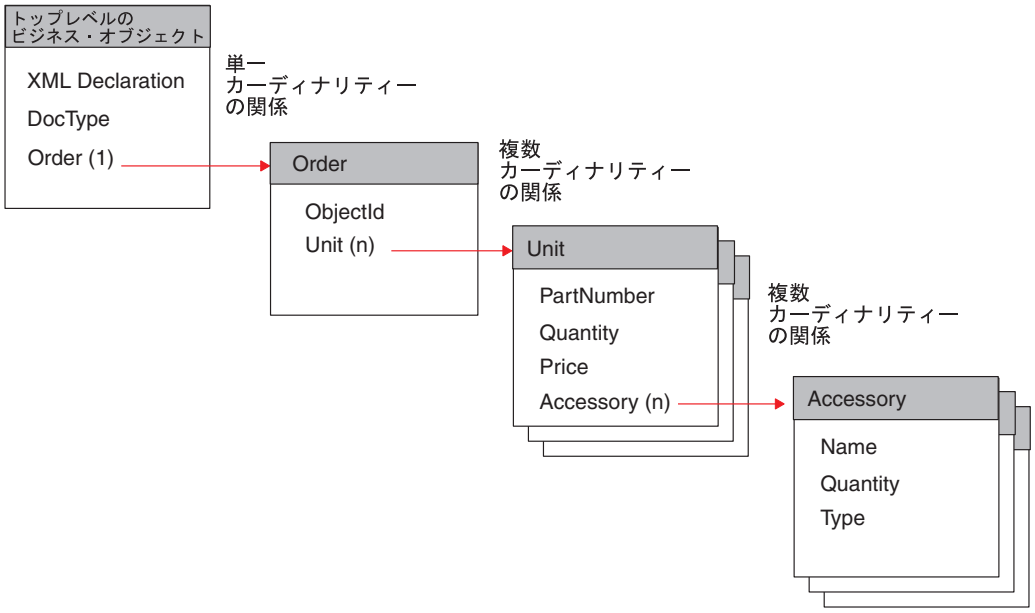


図 17. 単一カーディナリティーおよび複数カーディナリティーの構造的な関係

図 17 に示したビジネス・オブジェクトの関係は構造的に定義されています。つまり、それぞれの親ビジネス・オブジェクトには、子と同じタイプで関係が containment として指定されている属性が格納されています。

重要: XML データ・ハンドラーの場合、DTD を表すトップレベルのビジネス・オブジェクトに関して固有の要件があります。要件については、「データ・ハンドラー・ガイド」を参照してください。

アプリケーション固有のビジネス・オブジェクトの設計

各アプリケーション固有のビジネス・オブジェクトには、データ、そのデータに対して実行されるアクション (動詞)、そのデータに関する情報 (アプリケーション固有の情報) が格納されます。コネクター・メソッドの多くは、アプリケーション固有のビジネス・オブジェクトを引き数として渡します。以下に例を示します。

- アプリケーション・イベントが発生すると、一部のコネクターはデータ・ハンドラーを呼び出してデータのフォーマットをビジネス・オブジェクトに変換し、そのオブジェクトを InterChange Server Express に送ります。
- 要求が InterChange Server Express からコネクターに送られると、コネクター・フレームワークは、そのビジネス・オブジェクトをコネクターのビジネス・オブジェクト・ハンドラーへの引き数として送ります。この場合、一部のコネクターは、データ・ハンドラーを呼び出し、ビジネス・オブジェクト内の情報をアプリケーションで使用されるフォーマットに変換します。これにより、コネクターがそのアプリケーション内で操作を実行することが可能になります。

コネクター、データ・ハンドラー、およびコネクター/データ・ハンドラーによってサポートされるアプリケーション固有のビジネス・オブジェクトの間の関係の設計は、コネクターおよびデータ・ハンドラーの開発の一環として実行される作業です。アプリケーション固有のビジネス・オブジェクトの設計では、コネクター開発処理に統合する必要のある、コネクターおよびデータ・ハンドラーのプログラミング・ロジックに関する要件が生成されることがあります。したがって、コネクター、データ・ハンドラー、およびアプリケーション固有のビジネス・オブジェクトの開発者は、これらのコンポーネントに関する仕様を共同で作成する必要があります。アプリケーション固有のビジネス・オブジェクトのレイアウトと設計は、そのオブジェクトを処理するコネクターまたはデータ・ハンドラーにより決定する必要があります。

このセクションの内容は次のとおりです。

- 『アプリケーション固有のビジネス・オブジェクト定義の内容』
- 41 ページの『既存のコネクターまたはデータ・ハンドラーの設計』

アプリケーション固有のビジネス・オブジェクト定義の内容

ビジネス・オブジェクト定義には、以下の情報が含まれています。

表 6. ビジネス・オブジェクト定義の内容

ビジネス・オブジェクト定義の内容	説明	詳細情報の参照先
ビジネス・オブジェクトの構造	アプリケーション固有のビジネス・オブジェクトの構造は、コネクターまたはデータ・ハンドラーがそのアプリケーションと対話するレベル (表レベル、API レベル、または API 内の各種のレベルなど) で、アプリケーション・エンティティー (データ構造) に確実に対応するように設計するのが一般的です。	34 ページの『アプリケーション固有のビジネス・オブジェクトの構造』

表 6. ビジネス・オブジェクト定義の内容 (続き)

ビジネス・オブジェクト定義の内容	説明	詳細情報の参照先
属性プロパティー	属性には、アプリケーション・エンティティーの内部にある個々のデータが含まれています。属性には、そのほか、データのタイプ、カーディナリティー、デフォルト値などの情報を提供するプロパティーも含まれています。属性プロパティーは、同様にその属性が必須属性であるかどうか、キー属性であるかどうかについても指定します。	35 ページの『アプリケーション固有のビジネス・オブジェクトの属性』
アプリケーション固有の情報	アプリケーション固有のビジネス・オブジェクトの定義には、ビジネス・オブジェクトのアプリケーション内での表現方法や、処理方法をコネクターに通知するためのテキスト・ストリングが組み込まれることもあります。	36 ページの『ビジネス・オブジェクトのアプリケーション固有の情報』

アプリケーション固有のビジネス・オブジェクトの構造

コネクターまたはデータ・ハンドラーがビジネス・オブジェクトを処理する方法は、そのコネクターまたはデータ・ハンドラーがサポートしているビジネス・オブジェクトの構造によっても左右されます。アプリケーション固有のビジネス・オブジェクトの構造を設計するときには、どの構造により特定のアプリケーション・エンティティーを最も適切に表すことができるか、その構造がコネクターおよびデータ・ハンドラーのロジックの設計にどのような影響を及ぼすか、あるいはその構造が既存のコネクターまたはデータ・ハンドラーによってどのように処理されるかを判断する必要があります。

コネクターおよびデータ・ハンドラーの設計の目標の 1 つは、コネクターまたはデータ・ハンドラーが変更なしで新しいビジネス・オブジェクトや変更されたビジネス・オブジェクトを処理できるように、コネクターまたはデータ・ハンドラーをコーディングすることです。しかし、どんなビジネス・オブジェクトでも処理可能なコネクターやデータ・ハンドラーを作成することは困難です。

一般に、コネクターまたはデータ・ハンドラーは、自身のビジネス・オブジェクトの構造、親ビジネス・オブジェクトと子ビジネス・オブジェクトの間の関係、およびビジネス・オブジェクトのアプリケーション表現に関して、なんらかの前提事項を基に設計されています。既存のコネクターまたはデータ・ハンドラー向けのビジネス・オブジェクトを設計する場合は、このような仮定を把握し、それに基づいて設計を行う必要があります。

アプリケーション固有のビジネス・オブジェクトの構造については、まず次のことを考慮する必要があります。

- ビジネス・オブジェクトにカプセル化されるアプリケーション・エンティティーの編成またはデータベース・スキーマとは何か。 そのアプリケーション・エンティティーが階層型データあるいは 1 に対する多の関係を表しているかどうか。
- ビジネス・オブジェクトが 1 つのアプリケーション・エンティティーを表すか、それとも複数のアプリケーション・エンティティーを表すか。 つまり、1 つの個別ビジネス・オブジェクト内の属性値を複数のアプリケーション・エンティティーに格納することができるか。
- コネクターまたはデータ・ハンドラーで処理するビジネス・オブジェクト間の関係はどのような種類か。それらの関係は、ビジネス・オブジェクトでどのようにモデル化できるか、あるいはコネクターまたはデータ・ハンドラーによってどのように処理されるか。

単一または複数のアプリケーション・エンティティーを表現できるビジネス・オブジェクトの構造について詳しくは、19 ページの『ビジネス・オブジェクトの構造の判別』を参照してください。

注: トップレベルのビジネス・オブジェクトに特定の情報が含まれることを必要とするコネクターもあります。例えば、XML コネクターの場合、そのトップレベルのビジネス・オブジェクトに、単純属性 (URL、MIME タイプ、ビジネス・オブジェクト・プレフィックス用) のほか、複合属性 (応答ビジネス・オブジェクト、および要求ビジネス・オブジェクト用) を包含するよう要求します。既存のコネクターに対応したビジネス・オブジェクトを設計している場合は、そのコネクターのアダプター・ユーザーズ・ガイドを参照して、そのコネクター固有の構造上の要件を確認してください。詳しくは、41 ページの『既存のコネクターまたはデータ・ハンドラーの設計』を参照してください。

アプリケーション固有のビジネス・オブジェクトの属性

属性には、アプリケーション・エンティティーの個々のデータが保持されます。アプリケーション固有のビジネス・オブジェクトの属性を定義するときには、次の点に留意してください。

- アプリケーション・エンティティー内のどのようなデータが各属性によって表されているか。データベース・エンティティーを表すビジネス・オブジェクトの場合、各属性が表の列などのフィールドを表すかどうか。XML 文書を表すビジネス・オブジェクトの場合、各属性が要素を表すかどうか。
- アプリケーション・エンティティー内のフィールドごとにそれぞれ属性を作成する必要があるかどうか。1 つのアプリケーション・エンティティー内にある一部のデータが、そのインテグレーション内にある他のアプリケーションにとって重要ではない場合があります。それらのデータをビジネス・オブジェクト定義から除外することによって、設計の複雑度を軽減し、不要データの転送によるパフォーマンスの低下を防ぐことができます。
- アプリケーション固有のビジネス・オブジェクトが持つことになる単純属性の数が、アプリケーション・エンティティーよりも少ないかどうか。例えば、ユーザーがデータベース表の列ごとに属性を必要とするかどうかといったことです。
- 個別のビジネス・オブジェクトが持つ単純属性の数が、対応するデータベース表の列の数あるいは対応する DTD のタグの数よりも多い場合、コネクターはどのように動作するか。ビジネス・オブジェクト内の属性には、データベースや DTD で表されないものもあります。ほとんどの場合、これらの属性は、特定のアクセ

ス機構についての情報を伝達する役目を持っているか、あるいは子ビジネス・オブジェクトを表す属性を分離するために使用されます。コネクターやマップは、特定のアプリケーション固有のビジネス・オブジェクトを処理するために、コネクター固有の属性を必要とする特殊なロジックを採用している場合があります。

原則として、ビジネス・オブジェクトの構造を、対応するアプリケーション・エンティティー (データベース表や DTD など) の構造と同じになるように維持する必要があります。ビジネス・オブジェクトが大きい (多数の属性が含まれている) 場合は、ビジネス・オブジェクトを設計するビジネス・プロセスで使用される属性のみを定義してください。ただし、ビジネス・オブジェクトが小さい場合は、将来使用できるよう、すべての属性を定義してください。定義する属性の数は、ビジネス・オブジェクトのサイズとビジネス・オブジェクト間の関係の複雑さによって異なります。

ビジネス・オブジェクト内に属性として存在すべきアプリケーション・エンティティーを識別するだけでなく、ビジネス・プロセスを調べて追加の属性が必要かどうか判断することも必要とされます。ビジネス・プロセスの分析の一環として、ビジネス・オブジェクトの要件を特定します。ビジネス・プロセスを段階を追って分析していくことにより、ビジネス・オブジェクトがどのように処理され、必須属性がどのように使用されるかがわかります。ビジネス・プロセスおよび例外処理のバリエーションにより、ビジネス・オブジェクトの処理に必要な追加属性が特定されることがありますが、これらの追加属性は、アプリケーション内で検索または更新されるデータに必ずしも対応するとは限りません。

例えば、次のような属性が必要になる場合が考えられます。

- 優先順位標識として機能する属性。その値は、処理中に属性の値に基づいて決まります。
- 1 つまたは複数の属性値に基づいた経路情報を格納した検索の機能を持つ属性。

ビジネス・オブジェクトのアプリケーション固有の情報

アプリケーション固有のビジネス・オブジェクト定義の構造、およびビジネス・オブジェクト定義に含まれる属性のセットを定義し終えると、InterChange Server Express から受け取った要求の処理を可能にするにはビジネス・オブジェクトをどのように処理すべきかについての追加の情報が、コネクターまたはデータ・ハンドラーに必要とされるかどうか判断できるようになります。ビジネス・オブジェクト定義では、この追加情報をアプリケーション固有の情報に組み込むことができます。

コネクターおよびデータ・ハンドラーは、アプリケーション固有の情報に、ビジネス・オブジェクトの処理方法に関するアプリケーション依存の命令を提供します。ビジネス・オブジェクトとコネクターの関係を設計するときには、コネクターのアプリケーションやデータ・ソースとの対話を容易にするための情報が、ビジネス・オブジェクト定義に格納されているように設計してください。 このような情報は、**メタデータ**と呼ばれ、各ビジネス・オブジェクトのアプリケーション固有の情報、ビジネス・オブジェクト属性、およびビジネス・オブジェクト動詞に指定できます。

アプリケーション固有の情報は、ビジネス・オブジェクト設計時に入力されるストリングで、実行時にコネクターまたはデータ・ハンドラーが読み取ります。コネクターまたはデータ・ハンドラーは、ビジネス・オブジェクト定義内のメタデータを

使用してビジネス・オブジェクトのインスタンスを処理します。コネクターまたはデータ・ハンドラーは、サポートされているビジネス・オブジェクト定義に実行時にアクセスできるため、特定のビジネス・オブジェクトの処理方法を動的に決定できます。

ビジネス・オブジェクト設計時にアプリケーション固有の情報を使う場合、次のような利点と制限があります。

- アプリケーション固有の情報により、ビジネス・オブジェクトはその処理に必要な情報をすべて備えたオブジェクトとして機能することができます。

  ビジネス・オブジェクト定義内のアプリケーション固有の情報には、表および列の名前、処理命令、コネクターが呼び出す関数の名前、またはアプリケーション内でのデータの処理方法に関するその他の情報を組み込むことができます。

  各アプリケーション固有のビジネス・オブジェクトにはその処理に必要なすべての情報が格納されているため、コネクターのソース・コードを変更する必要なしに、コネクターは新しいビジネス・オブジェクトまたは変更したビジネス・オブジェクトを処理することができます。特定のビジネス・オブジェクトを処理するロジックがハードコーディングされていない 1 つのビジネス・オブジェクト・ハンドラーを使用して、汎用的な方法でコネクターを作成することができます。
- メタデータ主導型コネクターは、ビジネス・オブジェクトのインスタンス内の値およびビジネス・オブジェクト定義内のアプリケーション固有の情報から、アプリケーション関数呼び出しまたは SQL ステートメントを作成することができます。

  関数呼び出しまたは SQL ステートメントは、コネクターが処理するビジネス・オブジェクトおよび動詞のために、アプリケーション・データベース内での必要な変更を実行します。
- ビジネス・オブジェクト定義に格納できるアプリケーション固有情報の量は、そのビジネス・オブジェクトが表すアプリケーションによって決まります。

  アプリケーションとそのプログラミング・インターフェースによっては、ビジネス・オブジェクト内のアプリケーション固有の情報によってコネクターがほぼ全面的に駆動されるように、そのコネクターおよびビジネス・オブジェクトを設計できる場合があります。その場合、コネクターがビジネス・オブジェクトをアプリケーション処理要求に変換するために必要とするビジネス・オブジェクト・ハンドラーは 1 つのみになります。

  ただし、ビジネス・オブジェクトごとに完全に異なる処理ロジックを使わなければならない、アプリケーション・インターフェースの制約があるアプリケーションもあります。この場合、複数のビジネス・オブジェクト・ハンドラーを実装する必要があります。このようなアプリケーションの場合に可能なインプリメンテーション形態は、部分的にメタデータ主導型のインプリメンテーションか、完全にデータ主導型でないインプリメンテーションに限られます。

  ビジネス・オブジェクトが格納するアプリケーション固有情報の量は、アプリケーションによって異なります。ただし、ほとんどのアプリケーション固有のビジネス・オブジェクトは、コネクターまたはデータ・ハンドラーによるビジネス・オブジェクト処理を容易にするなんらかの情報を格納するように設計できます。

アプリケーション固有の情報に推奨されるフォーマット: アプリケーション固有の情報を定義するときには、名前と値のペアの構文を使用することをお勧めします。この構文では、プロパティーの名前と対応する値を等号 (=) で区切って指定します。

name1=value1;*name2=value2*

例えば、「表名」プロパティーは、名前と値のペアで定義されます。

TN=*TableName*

名前と値のペアを使うと、値をランダムな順序で指定できます。コネクターは、値を解釈する前に各パラメーターの名前を評価します。名前と値のペアはペアごとに区切り文字で区切ることをお勧めします。区切り文字は次のようになります。

- デフォルトとしてセミコロン (;) が使用されます。
- 区切り文字は設定可能です。

注: 既存のコネクター用にビジネス・オブジェクトを作成している場合は、そのコネクターのアダプター・ユーザーズ・ガイドを参照して、必要な構文を調べてください。コネクターによっては、デフォルトでも区切り文字としてセミコロンが使用されないため、その観点から区切り文字の構成が可能となっているものもあります。

表 7 に、属性のアプリケーション固有の情報に含めることの可能なパラメーターの例を示します。これらのパラメーターは、データベース表内のデータを表すビジネス・オブジェクトにのみ適用されます。

表 7. 属性のアプリケーション固有情報の名前と値のパラメーターの例

パラメーター	説明
TN=*TableName*	データベース表の名前です。
CN=*col_name*	この属性に対応するデータベース列の名前です。
FK=[..]*fk_attributeName*]	Foreign Key プロパティーの値によって親子関係が定義されます。
UID=AUTO	このパラメーターは、ビジネス・オブジェクトに対応する一意の ID を生成してその値をこの属性にロードするようコネクターに指示します。
CA=*set_attr_name*	Copy Attribute プロパティーは、1 つの属性の値を別の属性にコピーすることをコネクターに指示するために使用します。set_attr_name を現在の個別のビジネス・オブジェクト内にある別の属性の名前に設定すると、コネクターは、指定された属性の値を使ってこの属性の値を設定してから、Create 処理時にデータベースにビジネス・オブジェクトを追加します。
OB=[ASC|DESC]	Order By パラメーターに値が指定されていて、その属性が子ビジネス・オブジェクト内にある場合、コネクターは、検索照会の ORDER BY 文節にある属性の値に基づいて、子ビジネス・オブジェクトを昇順または降順のどちらで検索するかを決定します。
UNVL=*value*	コネクターが、値が null の属性を持つビジネス・オブジェクトを検索するときに、null を表すために使用する値を指定します。

1 つの属性のアプリケーション固有の情報を、上に例として挙げたパラメーターと組み合わせて使用することもできます。この例では、パラメーターを分離する区切り文字としてセミコロン (;) が使用されています。

```
TN=LineItems;CN=POid;FK=..PO_ID
```

この例のアプリケーション固有の情報は、表の名前と列の名前を指定し、現在の属性が子ビジネス・オブジェクトを親にリンクする外部キーであることを指定します。

アプリケーション固有の情報の内容: アプリケーション固有の情報の内容は、複雑さによって大幅に変わります。以下にいくつかの例を紹介します。

- ビジネス・オブジェクト定義内のアプリケーション固有の情報には、そのビジネス・オブジェクトに対応する表の名前をエンコードできるほか、属性ごとに、その属性に対応する列の名前をエンコードできます。これは、アプリケーション固有の情報をインプリメンテーションする形態としては比較的単純ですが、コネクターが必要とする条件はすべて満たしています。
- これより複雑なインプリメンテーション形態になると、アプリケーション固有の情報には、コネクターが各種のビジネス・オブジェクトの処理を扱う方法を指定するパラメーターのセットが格納されます。
- 最も複雑なインプリメンテーション形態では、アプリケーション固有の情報には、条件が組み込まれたり、コネクター・トランザクション処理の指示、データ検索方法の指定、プリプロセス機能の提供などがあります。

ビジネス・オブジェクト定義にアプリケーション固有の情報が組み込まれていて、コネクターがそれを使用する設計になっている場合、コネクターは、ビジネス・オブジェクト定義からアプリケーション固有の情報の内容を抽出し、それを処理に使用することができます。

例: コネクターによるアプリケーション固有の情報の処理方法: コネクターがアプリケーション固有の情報を処理する方法の例として、作成中のアプリケーションが表をベースとしていて、顧客情報を格納するためのアプリケーション表 CURRENTCUST を扱う場合を考えます。この表には、CSTName および CSTCity という 2 つの列があります。

ビジネス・オブジェクトのヘッダーの AppSpecificInfo プロパティーには、表名を格納できます。各属性の AppSpecificInfo プロパティーには列名を格納できます。さらに、このアプリケーション向けのコネクターでは、データベースとの対話に SQL ステートメントを使用するため、SQL 動詞とキーワードが保持されるように動詞のアプリケーション固有の情報を設計できます。図 18 に、この Customer ビジネス・オブジェクト定義の概念図を示します。

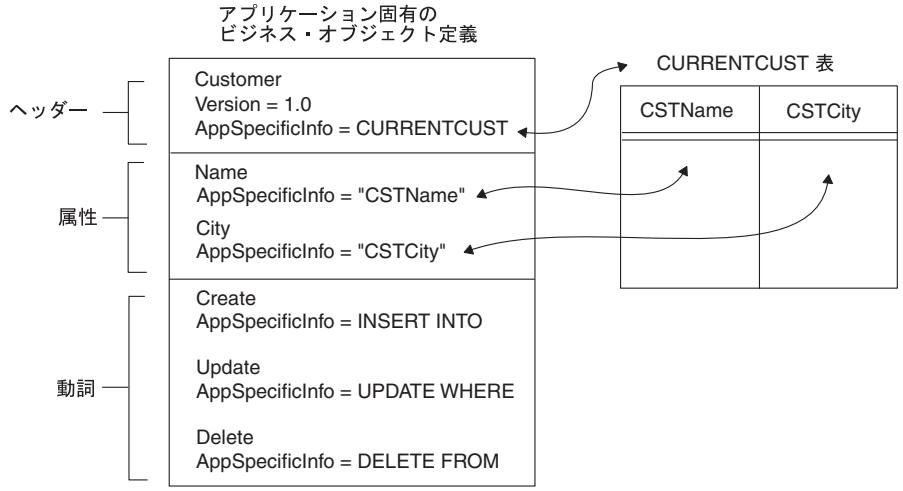


図 18. ビジネス・オブジェクト定義におけるアプリケーション固有の情報

メタデータ主導型コネクターは、InterChange Server Express からこのビジネス・オブジェクトのインスタンスを受け取ると、ビジネス・オブジェクト定義にあるアプリケーション固有の情報のプロパティーから表名および列名を抽出し、続いてビジネス・オブジェクトのインスタンスから属性および動詞の値を抽出します。動詞のアプリケーション固有の情報内の表名、列名、属性値、および SQL キーワードを使用して、コネクターは SQL ステートメントを作成することができます。

このタイプの処理の例を 図 19 に示します。コネクターは、ビジネス・オブジェクト定義から動詞処理命令、表名、および列名を抽出します。続いて、ビジネス・オブジェクト・インスタンスから属性値を取得します。この情報を使用して、コネクターは、CURRENTCUST 表を新しい情報で更新するための SQL INSERT ステートメントを作成します。

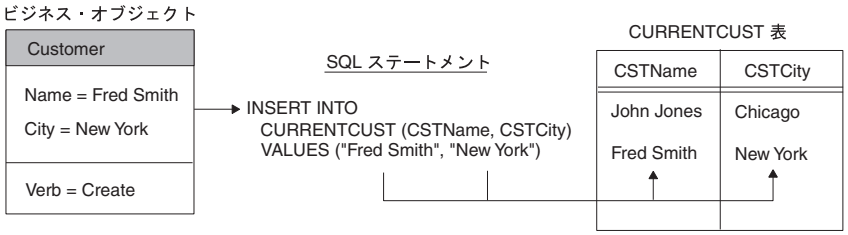


図 19. アプリケーション固有の情報を使用して Create 処理のための SQL ステートメントを作成する方法

既に述べたように、ビジネス・オブジェクト定義には、ビジネス・オブジェクト全体とその属性および動詞についての AppSpecificInfo テキストを組み込むことができます。ここからは、ビジネス・オブジェクトのこれらコンポーネントにおけるアプリケーション固有の情報の使用について詳しく説明します。

重要

アプリケーション固有の情報の長さは、1000 文字までという制約があります。

図 5 では、属性レベルの AppSpecificInfo プロパティーを使用して Invoice サブフォームの名前と属性内の対応フィールドの名前を格納しています。この例では、情報の指定には名前と値のペアを使用しています。

アプリケーション固有の情報の設計に関するヒント: コネクターのメタデータ主導型動作が最大になるようにビジネス・オブジェクトを設計する場合は、ビジネス・オブジェクト定義へのアプリケーション固有情報の格納に関して、次の一般的な推奨事項に従ってください。

- 表名やフォーム名などのエンティティー名は、トップレベルのビジネス・オブジェクトが持つビジネス・オブジェクト・レベルの AppSpecificInfo プロパティーに格納してください。サブフォーム名や表名は、子ビジネス・オブジェクトが持つビジネス・オブジェクト・レベルの AppSpecificInfo プロパティーに格納してください。
- フィールド名、列名、およびビジネス・オブジェクト属性に関するその他の情報は、各属性の AppSpecificInfo プロパティーに格納してください。
- 動詞処理命令は、各動詞の AppSpecificInfo プロパティーに格納してください。

AppSpecificInfo プロパティーを慎重に使用すれば、コネクターはさまざまなビジネス・オブジェクトを同じ方法で処理することができます。アプリケーション全体でデータ処理の取扱方法について整合性が取れていて、コネクターが実行するすべてのデータ処理についてタスクが一致している場合、ビジネス・オブジェクトは、完全にメタデータ主導型コネクターを使えるように設計できます。

既存のコネクターまたはデータ・ハンドラーの設計

既存のコネクターまたはデータ・ハンドラー用にアプリケーション固有のビジネス・オブジェクトを設計する際には、まずそのアダプター・ユーザーズ・ガイドを参照して、アプリケーション固有の情報の指定とビジネス・オブジェクト・ハンドラーの使用に関する要件を確認してください。既存のコネクターまたはデータ・ハンドラー用にビジネス・オブジェクトを設計する際には、以下の点を念頭に置いてください。

- 使用可能な Object Discovery Agent の有無を確認する場合、ビジネス・オブジェクトを処理するコネクターのアダプター・ガイド、およびデータ・ハンドラーの資料を調べてください。Object Discovery Agent を使用すると、巨大なエンティティーを必要とする場合でも、ビジネス・オブジェクトの設計の労力をかなり軽減できます。
- アプリケーション・エンティティーをモデル化するための既存のビジネス・オブジェクト (例えばサンプルなど) が使用可能かどうか確認します。既存のビジネス・オブジェクトをカスタマイズしたほうが、新しいオブジェクトをゼロから作成するよりも少ない労力で済むかどうかを判別し、その場合はサンプルのビジネス・オブジェクトを使用することを検討してください。

重要

サンプルのビジネス・オブジェクトは、IBM ではサポートされていませんが、ビジネス・オブジェクトの設計の開始点としては大変役立ちます。

- モデル化が必要なエンティティーに対応する既存のビジネス・オブジェクトが存在しない場合や、アプリケーションの Object Discovery Agent が現時点では存在していない場合は、このアプリケーションに対して新規の Object Discovery Agent を作成するのも 1 つの方法です。ただし、このアプリケーションに必要なビジネス・オブジェクトの数がごくわずかであるか、またはエンティティーが非常に小規模である場合は、この方法は効率的でないことがあります。詳しくは、101 ページの『第 5 章 Object Discovery Agent の開発』を参照してください。
- Object Discovery Agent または既存ビジネス・オブジェクトのどちらを使用するかには関係なく、オブジェクト・キー、外部キー、子ビジネス・オブジェクト、デフォルト値、データ型、サイズ制限などすべてのデータ定義要件を検査および確認することが大切です。この要件は、以下の要因によってもたらされるものです。
  - Object Discovery Agent によって設計の労力は軽減できますが、アプリケーション・エンティティーを取り巻く要件をすべて洗い出せるわけではありません。
  - 既存のビジネス・オブジェクトを使用した場合は、顧客固有の要望に応じて、アプリケーションのインストールおよび構成の仕方に違いが生じる恐れがあります。ビジネス・オブジェクトが、あるアプリケーションのインストールではエンティティーを正確にモデル化できても、そのアプリケーションの別のインストールでは同じエンティティーを正確にモデル化できない場合があります。

アプリケーション固有のビジネス・オブジェクトを設計する際には、データ・ソース内のエンティティーのモデル化こそがその基本的な役割であることを念頭に置いてください。また、対応するコネクターまたはデータ・ハンドラーでそのオブジェクトの処理がどのように扱われるのか、そのオブジェクトが参加するビジネス・プロセスの要件がどのようなものであるのかを明確にすることも重要です。

汎用ビジネス・オブジェクトの設計 (InterChange Server Express のみ)

汎用ビジネス・オブジェクト には、複数の多様なアプリケーションまたはプログラム・エンティティーによって使用されるエンティティーを表す情報のスーパーセットが反映されます。 InterChange Server Express コラボレーション・オブジェクトでは、汎用ビジネス・オブジェクトを使用して、各種の多様なアプリケーションに合った情報の提供を可能にしています。そのため、汎用ビジネス・オブジェクトの設計は、コラボレーション・オブジェクト開発の一環として行われます。汎用ビジネス・オブジェクトを設計するときには、次の点を考慮してください。

- ビジネス・プロセスの統合にとって重要なデータ要件を把握します。
- ビジネス・オブジェクトが参加する対象のビジネス・ロジックを検討し、そのロジックに基づいてすべての要件を検討します。

  次の 2 つの点を検討することにより、ビジネス・ロジック分析の複雑度を把握できます。
  - 前提条件データの処理

    コラボレーション・オブジェクトを起動するアプリケーションは、トリガー・ビジネス・オブジェクトを処理するのに必要なすべてのデータを提供しない場合もあります。必要な追加データは、宛先アプリケーションを組み込む他のアプリケーション内に常駐することもできます。

例えば、営業支援システム (SFA) アプリケーション (Siebel など) で生成される見積もりを、受注・出荷管理システム (SAP など) に Order として記録しなければならない場合があります。しかし、Quote を Order にするには、SFA アプリケーションでは入手できない追加の情報が必要になることがあります。例えば、Order には、顧客信用状況 (財務システムから)、連絡先情報 (顧客支援システムから)、または Availability To Promise 情報 (倉庫管理システムから) などの追加情報が必要となる場合が考えられます。

汎用 Order ビジネス・オブジェクトを設計するときには、属性を組み込んで、ソース・アプリケーションに存在しなくても、そのインターフェースに含まれる他のアプリケーションには存在するデータをサポートする構造を設計する必要があります。

– 個別アプリケーション・エンティティー間の相互参照

個別アプリケーション・エンティティーが互いにどのように対応しているか、あるいはビジネス・プロセス内でどのような包括的相互参照を行っているかを把握します。

例えば、Oracle では顧客は Customer、その住所は Address と表現されます。SAP では顧客は「SoldTo」エンティティー、その住所は「ship-to」エンティティーと表現されます。Clarify では顧客は「Business Organization」、その住所は「Site」と表現されます。

機能のほか、アプリケーションのエンティティー間の関係を検討して、アプリケーション間のデータの統合に関係するビジネス・オブジェクトおよびプロセスを決定します。

- 必須データのうち、そのビジネス・プロセスに参加するすべてのアプリケーションで共通している、または共有されているデータがどれであるか、そのプロセスの統合に重要なものはどれであるかを把握します。必須属性 (属性の最小限の共通特性) と、ビジネス・インテグレーション・システムがそれらのアプリケーション固有のビジネス・オブジェクト間で実行すべき変換については、属性およびその関係により最小限判別しておく必要があります。

  統合形態には次のものがあります。

  – ビジネス・プロセスによりエンタープライズ・リソース・マネージメント (ERP) システムから カスタマー・リレーションシップ・マネージメント (CRM) システムにデータを統合します。この場合、ERP システムに格納されたデータの大部分はほとんどの CRM システムに存在しないため、ビジネス・オブジェクトをあまり複雑にする必要はありません。
  – ビジネス・プロセスにより 2 つの ERP システム間でデータを統合します。この場合、ビジネス・オブジェクトはかなり複雑になるのが一般的です。
  – ビジネス・プロセスによりデータを CRM システムから ERP システムに統合します。この場合、どの程度の量のデータが実際に CRM システムから発生した (したがって、汎用ビジネス・オブジェクト内の属性で表す必要がある) か、どの程度の量のデータが宛先アプリケーション自体のデフォルトとして扱える (したがってアプリケーション固有のビジネス・オブジェクト内にデフォルト値として設定できる) かを反映した設計が必要になります。

- 汎用ビジネス・オブジェクトのマップ先アプリケーション固有のビジネス・オブジェクトが存在する場合は、それらのマップ先オブジェクトを調べます。すべてのビジネス・オブジェクトの構造と属性を分析することにより、すべてのアプリケーションに適した汎用ビジネス・オブジェクトを作成します。
- 設計しているビジネス・オブジェクトのタイプに関する規格がないか調べます。例えば、モデルによっては Electronic Data Interchange (EDI)、Open Applications Group (OAG)、または Object Management Group (OMG) イニシアチブによって提供されるエンティティーに対応している場合もあります。

汎用ビジネス・オブジェクトの設計標準

汎用ビジネス・オブジェクトとの整合性を取るため、汎用ビジネス・オブジェクトを設計するときは、次の指針に従ってください。

- 各オブジェクトの最初の属性はそのオブジェクトのキーとし、`ObjectId` という名前にします。
- 属性が外部キーを表す場合は、例えば `CustomerId`、`ItemId`、`OrderId` のように、その外部ビジネス・オブジェクトの名前に `Id` を付けた名前をその属性の名前にします。
- 名前の整合性を取ります。ある属性名で省略語を使用する場合は、親ビジネス・オブジェクトと子ビジネス・オブジェクトでも同じ省略語を使うようにします。可能であれば、関連するすべての属性の名前に同じ省略語を使用します。例えば、`Number` を `Num` という省略語にした場合は、その省略語を一貫して使うようにします。
- 「*Naming IBM WebSphere InterChange Server Express Components*」に記載されている命名ガイドラインに従ってください。

イベント分離のための設計

汎用ビジネス・オブジェクトを設計する際には、「*コラボレーション開発ガイド*」の『並列実行の設計』という題名のセクションに解説されているイベント分離の必要性を考慮するようお勧めします。

複数のコラボレーション・オブジェクトが同時に同じデータを更新するのを防ぐため、各ビジネス・オブジェクトを変更できるのは、1 つのタイプのコラボレーション・オブジェクトに限定するようにしてください。例えば、Customer ビジネス・オブジェクトは、CustomerSync コラボレーション・オブジェクトによってのみ変更できるようにします。

あるコラボレーション・オブジェクトが子ビジネス・オブジェクトを含むビジネス・オブジェクトを変更し、その子ビジネス・オブジェクトが別のトップレベルのビジネス・オブジェクトにも包含され、このトップレベルのビジネス・オブジェクトに専用の変更コラボレーション・オブジェクトが存在する場合には、これらトップレベルのビジネス・オブジェクトを、その子を構造的にではなく意味的に包含するように設計します。共有された子を変更するためには、第 3 のコラボレーション・オブジェクトを作成します。2 つのトップレベルのビジネス・オブジェクトを所有するコラボレーション・オブジェクトの場合、共有された子の処理を 3 つ目のコラボレーション・オブジェクトに委任する必要があります。

例えば、Customer ビジネス・オブジェクトと Contact ビジネス・オブジェクトの両方に同じ住所データが格納されている場合は、Customer と Contact によって参照されるトップレベルのビジネス・オブジェクトとして Address ビジネス・オブジェクトを設計します。ただし、Address ビジネス・オブジェクトは、Customer と Contact の 2 つのオブジェクトに包含されないようにします。住所データの変更のためには、別個の Address コラボレーション・オブジェクトを作成します。

一方、別の例として、Order ビジネス・オブジェクトが OrderLineItem データを変更する唯一のビジネス・オブジェクトである場合は、OrderLineItem 子ビジネス・オブジェクトを単に参照するのではなく包含するように、Order を設計することができます。

言い換えると、Customer および Contact ビジネス・オブジェクトは、Address ビジネス・オブジェクトを参照する外部キー属性を格納するように、つまり Address に対応したキー値だけを格納するように設計する必要があります。Address ビジネス・オブジェクトは、すべての値が含まれる Address ビジネス・オブジェクトを表す属性が格納されないように設計してください。一方、Order ビジネス・オブジェクトは、すべての値が含まれる OrderLineItem ビジネス・オブジェクト表す属性を格納するように設計します。

注: 共有ビジネス・オブジェクトを包含されるものではなく参照されるものとして設計すると、ビジネス・オブジェクトの分散が容易になります。複数のビジネス・オブジェクト定義に同じ子ビジネス・オブジェクトが定義されている場合、`repos_copy` ユーティリティーは、インストール時にこのビジネス・オブジェクトを 2 回ロードしようとするため、ロールバックが発生します。このデフォルト動作を変更する `repos_copy` フラグについては、「*システム管理ガイド*」を参照してください。

汎用ビジネス・オブジェクトの属性

汎用ビジネス・オブジェクトの属性を定義するときは、その汎用ビジネス・オブジェクトのマップ先のアプリケーションに固有のビジネス・オブジェクトの属性を調べる必要があります。その場合の指針を次に示します。

- アプリケーション固有のビジネス・オブジェクトの属性にあるエンティティー間の類似性に注意してください。汎用ビジネス・オブジェクトに対しては、アプリケーション固有のビジネス・オブジェクト内の属性に最も簡明に一致する属性を定義してください。
- アプリケーション固有のビジネス・オブジェクトの属性にあるエンティティー間の違いに注意してください。データを複数のフィールドに分割しているアプリケーション固有のビジネス・オブジェクトもあり、同じデータを 1 つのフィールドに結合しているオブジェクトもある場合、2 つのアプリケーション・エンティティーのマッピングを最も単純化できる設計を判別してください。詳しくは、41 ページの『既存のコネクターまたはデータ・ハンドラーの設計』を参照してください。
- コラボレーション・オブジェクトが実行する処理によって生成される要件を検討してください。例えば、42 ページの『汎用ビジネス・オブジェクトの設計 (InterChange Server Express のみ)』 に記載のようにコラボレーション・オブジェ

クトで前提条件を処理する場合は、前提条件データを格納するのに必要なすべての属性が汎用ビジネス・オブジェクトに格納されていることを確認してください。

- 汎用ビジネス・オブジェクトとインターフェースを、そのインターフェースに関与する多数のアプリケーションに対応するように開発します。例えば、インターフェースに関与するアプリケーションが 4 つあって、そのうちの 3 つは子オブジェクト内のデータをカプセル化している一方、4 つ目は親レベル・オブジェクトでデータを包含していている場合、子オブジェクト内のデータもカプセル化されるように、汎用ビジネス・オブジェクトを設計してください。これにより、マッピング、その他の関連タスクが著しく容易になります。
- 今後の開発の労作を考慮に入れたうえで、以降の一時点で必要となるデータ構造に対応するように汎用ビジネス・オブジェクトを設計して、その時点での労作および変更の影響を最小限に抑えるのがよいでしょう。ただし、実現する見込みのない将来のプロジェクトに関しては、開発スコープをあまり広げないようにしてください。

一般に、汎用ビジネス・オブジェクト定義には、その汎用ビジネス・オブジェクトがマップされるすべてのアプリケーション固有のビジネス・オブジェクトの間で変換されるすべてのデータ要素がキャプチャーされるように、属性を組み込む必要があります。

属性にはできるだけわかりやすい名前を付けてください。例えば、ある 1 つのエンティティーを顧客 (Customer) として参照するアプリケーションが複数存在し、企業 (Business Organization) として参照するアプリケーションが 1 つのみの場合、汎用属性の命名には、より汎用性の高い用語を使用します。

注: 属性の名前に使用できる文字は、英数字と下線 (_) に限定されています。

既存の汎用ビジネス・オブジェクトの評価

汎用ビジネス・オブジェクトの開発を容易にするには、既存のものをコピーしカスタマイズするのも 1 つの方法です。

汎用ビジネス・オブジェクトを評価するには、そのインターフェースに組み込まれているデータを調べます。方針としては、提供された汎用ビジネス・オブジェクトの 1 つにデータの 80% 以上が存在する場合には、その既存のオブジェクトをカスタマイズします。

この分析を行うときは、ビジネス・オブジェクトの属性よりも構造に注目することが重要です。属性は比較的簡単に追加したり削除したりできますが、構造や階層の変更には手間がかかります。

既存の汎用ビジネス・オブジェクトをカスタマイズする場合は、ビジネス・オブジェクト定義を調べて、必要な属性が欠落していないかを確認してください。属性の欠落は、マッピング設計時にさらに発見しやすくなります。汎用ビジネス・オブジェクトが 1 つ以上の追加属性を必要としている場合は、その追加属性を格納する子ビジネス・オブジェクトを作成します。カスタムの属性を子ビジネス・オブジェクトに隔離しておくと、IBM 提供のビジネス・オブジェクトに対する将来のアップグレードが容易になります。

IBM 提供のビジネス・オブジェクトにカスタム属性を埋め込むと、ビジネス・オブジェクトを新バージョンにアップグレードする際、新しいビジネス・オブジェクトにこれらの属性を埋め込み直さなければなりません。カスタム属性を独自のビジネス・オブジェクトに隔離させると、新規 IBM ビジネス・オブジェクトに、親ビジネス・オブジェクトとカスタム子ビジネス・オブジェクトの関係を作成する属性を 1 つ追加できます。親と子の両方に追加の属性を必要とする階層型ビジネス・オブジェクトをカスタマイズする場合は、それぞれに別個の子ビジネス・オブジェクトを作成してください。

カスタム属性およびビジネス・オブジェクトに、その性質がわかる名前を指定することをお勧めします。簡単な命名規則の 1 つは、各カスタム名に _x をサフィックスとして付けるという方法です。例えば、汎用 Order ビジネス・オブジェクトに属性を追加するカスタム子ビジネス・オブジェクトを作成する場合は、その子オブジェクトは Order_x という名前にします。これにより、一覧表示のときに関連のある名前がアルファベット順にまとまって表示されます。カスタム・ビジネス・オブジェクトまたは属性を識別することの方が、カスタム・オブジェクトとその汎用オブジェクトをアルファベット順に並べることより重要な場合は、各カスタム名にプレフィックスとして x_ を追加します。詳しくは、「*Naming IBM WebSphere InterChange Server Express Components*」を参照してください。

ビジネス・オブジェクトのマッピング要件の判別 (InterChange Server Express のみ)

アプリケーション固有のビジネス・オブジェクトは、アプリケーション・エンティティーと一致するように設計されていると、対応する汎用ビジネス・オブジェクトに一致しない可能性があります。したがって、アプリケーション・データを WebSphere Business Integration システム全体でやり取りできるよう、アプリケーション固有のビジネス・オブジェクトと汎用ビジネス・オブジェクトを対応付けるマップを作成する必要があります。

アプリケーション固有のビジネス・オブジェクトは、アプリケーション・エンティティー内のフィールド、列、または要素をすべて組み込まなくてもよい場合もあります。アプリケーション固有のビジネス・オブジェクトに組み込む必要のある属性を決定するときには、そのオブジェクトを使用するアプリケーションおよびビジネス・プロセスの機能要件を参考にしてください。

汎用ビジネス・オブジェクトとアプリケーション・エンティティーの間の対応を調べることもできます。汎用ビジネス・オブジェクト内のフィールドに対応するフィールドをアプリケーション固有のビジネス・オブジェクトに組み込むかどうかを選択できます。組み込んだ場合、これらデータ要素は該当のビジネス・プロセスに参加できるようになります。

ビジネス・オブジェクトを設計するときには、アプリケーション・エンティティーと汎用ビジネス・オブジェクトの間の違いに注意してください。この違いに基づいて、必要なデータ変換の内容が確定します。次のような目的でマッピングの設計が必要になることが考えられます。

- アプリケーション・エンティティー内の複数のフィールドを組み合わせて、汎用ビジネス・オブジェクト内の 1 つの属性に対応させること。

- アプリケーション・エンティティー内の 1 つのフィールドを分割して、汎用ビジネス・オブジェクト内の複数の属性に対応させること。
- 汎用ビジネス・オブジェクト内には存在しているが、アプリケーション・エンティティーには関連していないフィールドは無視すること。
- アプリケーション固有のビジネス・オブジェクトと汎用ビジネス・オブジェクトの間の意味的あるいは構造的な関係の違いに対処すること。
- アプリケーション・エンティティー間の外部キー関係などの関係に対処すること。
- データ間の関連を確立すること。以下に例を示します。
  - アプリケーション間でコード値 (既婚/独身の区別や通貨コードなど) の変換を行う関連など、非キー属性内にあるデータ間に参照する関連を確立すること。
  - アプリケーション間でキー属性 (固有 ID や製品コードなど) の変換を行う関連など、ビジネス・オブジェクト内のデータ間に一致する関連を確立すること。

マッピングと設計の概念を理解しやすいよう、表内のフィールド、アプリケーション固有のビジネス・オブジェクト内の属性、汎用ビジネス・オブジェクト内の属性の関係をごく簡単に 図 20 に示します。アプリケーション固有のビジネス・オブジェクトと汎用ビジネス・オブジェクトの違いは、マッピングにより処理することです。ビジネス・オブジェクトの属性がデータベースに対応する表現を持たない場合、コネクターはその属性のデフォルト値を設定できます。

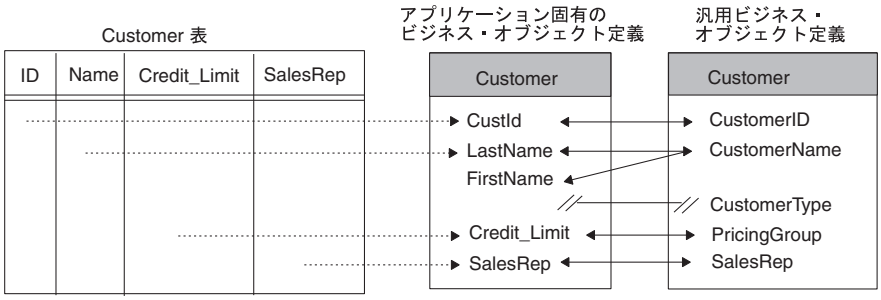


図 20. フィールド/属性の関係の概要

マップの作成については、「*マップ開発ガイド*」を参照してください。

第 3 章 Business Object Designer Express の使用

Business Object Designer Express ツールは、ビジネス・オブジェクト定義の作成、編集、および削除に使用されます。この章では、Business Object Designer Express の開始方法および使用方法について概要を示します。この章の主な内容は次のとおりです。

- 『プロジェクトの処理』
- 53 ページの『Business Object Designer Express の開始』
- 54 ページの『Business Object Designer Express からのビジネス・オブジェクト定義のオープン』
- 57 ページの『ビジネス・オブジェクト定義の処理』
- 59 ページの『Business Object Designer Express の機能』

プロジェクトの処理

Business Object Designer Express は、*プロジェクト* の概念を使用してビジネス・オブジェクト定義の作成、変更、または削除を行う仮想作業域を定義します。環境に応じて、Business Object Designer Express がダイアログ・ボックスで指す「プロジェクト」は以下のいずれかにすることができます。

表 8. Business Object Designer Express のプロジェクト

Business Object Designer Express 環境	**プロジェクト**
System Manager から Business Object Designer Express を実行していない場合	現在の Business Object Designer Express セッションで作業するためにローカル・ディレクトリーからビジネス・オブジェクト定義をインポートした仮想作業域。*ローカル・プロジェクト* とも呼びます。
System Manager から Business Object Designer Express を実行している場合	Business Object Designer Express および System Manager が実行されている Windows マシンの統合コンポーネント・ライブラリー (ICL)。*ICL ベースのプロジェクト* とも呼びます。

各タイプのプロジェクトの使用に関する詳細については、以下のセクションで説明します。

System Manager を使用せずに Business Object Designer Express を実行している場合

System Manager から Business Object Designer Express を実行していない場合は、Business Object Designer Express は「プロジェクト」としてローカル・プロジェクトを使用します。ローカル・プロジェクトは、処理するビジネス・オブジェクト定義をインポートできる仮想作業域です。

Business Object Designer Express によるローカル・プロジェクトの処理

ローカル・プロジェクトにおける Business Object Designer Express 機能の動作に関する全体像を以下に示します。これらのタスクの実行に関する詳細については、53 ページの『Business Object Designer Express の開始』以降のトピックで説明しています。

- **既存のビジネス・オブジェクト定義の編集:** 既存のビジネス・オブジェクト定義を編集するには、**「ファイル」->「ファイルから開く」** をクリックします。このメニュー項目は、ビジネス・オブジェクト定義をローカル・ディレクトリーからプロジェクトにインポートし、オプションで編集用に開きます。

  プロジェクトにインポートされているが現在は閉じられている既存のビジネス・オブジェクト定義を編集するには、**「ファイル」->「開く」**をクリックします。
- **新規ビジネス・オブジェクト定義の作成:** 新規ビジネス・オブジェクト定義を作成するには、**「ファイル」->「新規」または「ファイル」->「ODA を使用して新規作成」** をクリックします。
- **ビジネス・オブジェクト定義の保管:** 新規または変更したビジネス・オブジェクト定義を保管するには、メニュー・バーで**「保管」**をクリックします。ローカル・ディレクトリーへの保管を確認するプロンプトが表示されます。ビジネス・オブジェクト定義を別の名前で保管したり別のディレクトリーに保管したりするには、**「別名保管」**をクリックします。
- **ビジネス・オブジェクト定義の削除:** 格納先 Windows ディレクトリーからビジネス・オブジェクト定義を削除するには、Windows によって提供されているツールを使用します。Business Object Designer Express の**削除**機能を使用して行うことはできません。ローカル・プロジェクトからビジネス・オブジェクト定義を削除するには、**「ファイル」->「削除」**を選択します。プロジェクトから削除するビジネス・オブジェクト定義を選択するプロンプトが表示されます。

System Manager から Business Object Designer Express を実行している場合

System Manager から Business Object Designer Express を実行する場合は、ビジネス・オブジェクト定義の開発および管理を行うためのさらに多くの洗練された機能にアクセスできます。System Manager では、ビジネス・オブジェクト定義は、コラボレーションやマップなどの他のビジネス・インテグレーション・コンポーネントとともに*統合コンポーネント・ライブラリー* (ICL) に格納されます。ICL はビジネス・インテグレーション・コンポーネントのリポジトリーであり、ビジネス・インテグレーション・ソリューションを構築するための構成ブロックとして使用できます。各 ICL には、統合コンポーネントの種類ごとに 1 つのフォルダーの集合があります。これを 51 ページの図 21 に示します。

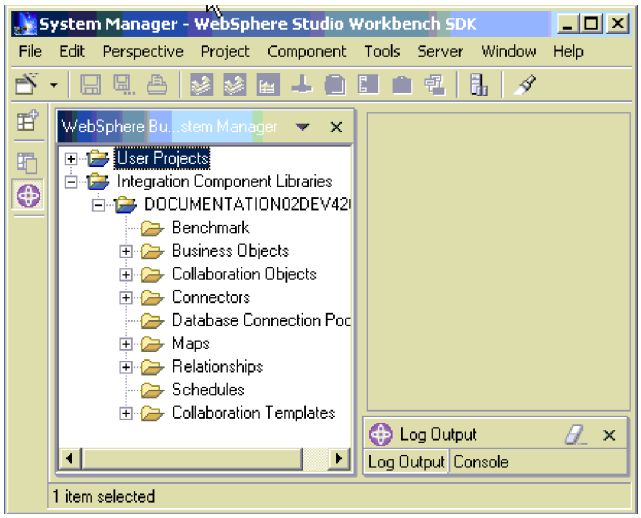


図 21. System Manager の統合コンポーネント・ライブラリー

ビジネス・オブジェクト定義の開発および展開を行う方法は以下のとおりです。Business Object Designer Express でビジネス・オブジェクト定義を開発し、ICL のビジネス・オブジェクト・フォルダーに保管します。ビジネス・オブジェクト定義をビジネス・インテグレーション・ソリューションで使用する場合は、その定義を 1 つ以上のユーザー・プロジェクト に関連付けます。各ユーザー・プロジェクトには、個々のビジネス・インテグレーション・ソリューションの実装に必要なすべてのビジネス・インテグレーション・コンポーネントが含まれます。例えば、PeopleSoft アダプターの実装に必要なコンポーネントを含むユーザー・プロジェクトでは、ビジネス・オブジェクト・フォルダーにそのアダプターが必要とするすべてのビジネス・オブジェクト定義があります。

ICL と同様に、各ユーザー・プロジェクトにはビジネス・インテグレーション・コンポーネント・フォルダーの集合が含まれます。しかし、ユーザー・プロジェクトには ICL コンポーネントの仮想的なコピーのみが含まれます。ビジネス・オブジェクト定義を変更する場合は、ICL のインスタンスを変更します。行った変更内容は、ビジネス・オブジェクト定義を含むすべてのユーザー・プロジェクトに自動的に伝搬されます。すなわち、あるビジネス・オブジェクト定義が 2 つのユーザー・プロジェクトに含まれる場合に統合コンポーネント・ライブラリーの定義を変更すると、その変更はユーザー・プロジェクトに存在する仮想コピーに自動的に反映されます。このように ICL のビジネス・オブジェクト定義と対応するユーザー・プロジェクトの仮想コピーがリンクしているため、複数のビジネス・インテグレーション・ソリューションで展開するときに、ビジネス・オブジェクト定義の変更および保守を 1 箇所に集中させることができます。

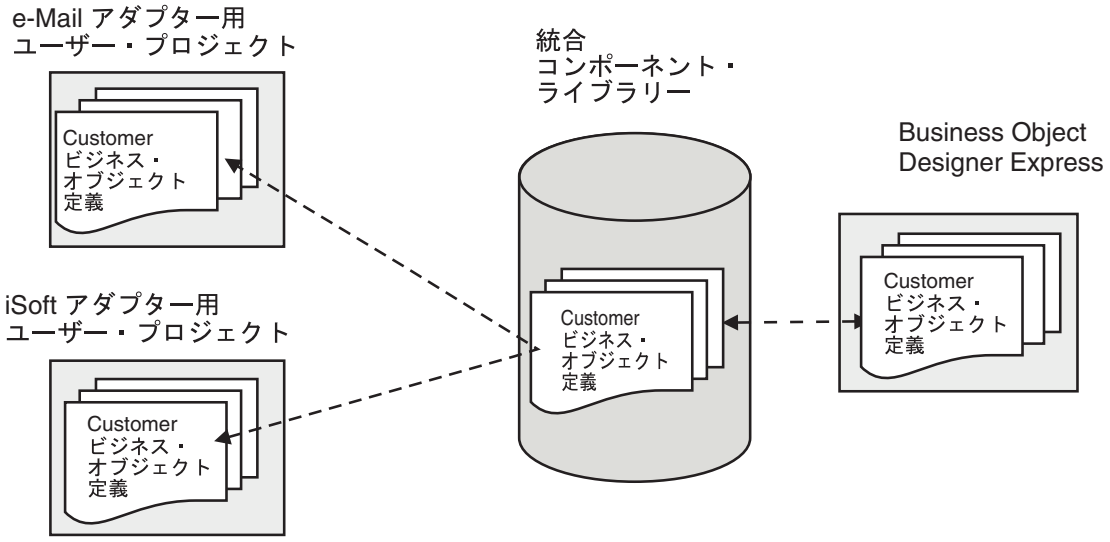


図 22. ICL のビジネス・オブジェクト定義に対する変更はユーザー・プロジェクトの仮想コピーに自動的に伝搬される

統合コンポーネント・ライブラリーを使用したビジネス・インテグレーション・コンポーネントの開発については、ご使用のシステムのインプリメンテーション・ガイドを参照してください。

Business Object Designer Express による ICL ベースのプロジェクトの処理

System Manager から Business Object Designer Express を実行する場合は、選択した統合コンポーネント・ライブラリーが「プロジェクト」として使用されます。ICL ベースのプロジェクトにおける Business Object Designer Express 機能の動作に関する全体像を以下に示します。これらのタスクの実行に関する詳細については、53 ページの『Business Object Designer Express の開始』以降のトピックで説明しています。

- **既存のビジネス・オブジェクト定義の編集:** プロジェクトに格納されているビジネス・オブジェクト定義を編集するには、**「ファイル」->「開く」**をクリックします。
- **新規ビジネス・オブジェクト定義の作成:** 新規ビジネス・オブジェクト定義を作成するには、**「ファイル」->「新規」**または**「ファイル」->「ODA を使用して新規作成」** をクリックします。
- **ビジネス・オブジェクト定義の保管:** 新規または変更したビジネス・オブジェクト定義を保管するには、**「ファイル」->「保管」**をクリックします。ビジネス・オブジェクトは、プロジェクトのビジネス・オブジェクト・フォルダーに保管されます。新規または変更したビジネス・オブジェクト定義を別の名前で保管するには、メニュー・バーから**「ファイル」->「別名保管」**をクリックします。
- **ビジネス・オブジェクト定義の削除:** ビジネス・オブジェクト定義を削除するには、メニュー・バーから**「削除」**を選択します。プロジェクトから削除するビジネス・オブジェクト定義を選択するプロンプトが表示されます。

System Manager を使用せずに Business Object Designer Express を実行している場合は、統合コンポーネント・ライブラリーにはアクセスできません。この環境では、Business Object Designer Express はローカル・プロジェクトを使用します。詳

しくは、 49 ページの『System Manager を使用せずに Business Object Designer Express を実行している場合』を参照してください。

Business Object Designer Express の開始

Business Object Designer Express は、表 9 に示すいずれの方法でも開くことができます。Business Object Designer Express が開いた後で、ビジネス・オブジェクト定義を手動で作成するか、Object Discovery Agent を使用してアプリケーション固有のビジネス・オブジェクトの定義を生成することができます。詳しくは、 65 ページの『第 4 章 ビジネス・オブジェクト定義の開発』を参照してください。

表 9. Business Object Designer Express を開く方法

System Manager から	• 統合コンポーネント・ライブラリーのビジネス・オブジェクト・フォルダーを選択し、以下のいずれかの手順を実行します。 – **「ツール」**メニューから**「Business Object Designer Express」**をクリックします。 – **「Business Object Designer Express」**ツールバー・アイコンをクリックします。 • 統合コンポーネント・ライブラリーのビジネス・オブジェクト・フォルダーを右クリックします。 • ビジネス・オブジェクト定義をダブルクリックします。
Windows ショートカットの使用 (InterChange Server Express)	**「プログラム」->「IBM WebSphere InterChange Server Express」->「IBM WebSphere Business Integration Toolset Express」->「開発」->「Business Object Designer Express」**の順にクリックします。
他の開発ツールから (InterChange Server Express のみ)	以下のいずれかの手順を実行します。 • **「ツール」**メニューで**「Business Object Designer Express」**をクリックします。 • ツールバーで Business Object Designer Express アイコンをダブルクリックします。

System Manager から直接 Business Object Designer Express を開くと、最初にビジネス・オブジェクト定義を選択しなくても「新規ビジネス・オブジェクト」ダイアログ・ボックスが自動的に開きます。System Manager が実行されていない場合、Business Object Designer Express は開きますが、「新規ビジネス・オブジェクト」ダイアログ・ボックスは開きません。

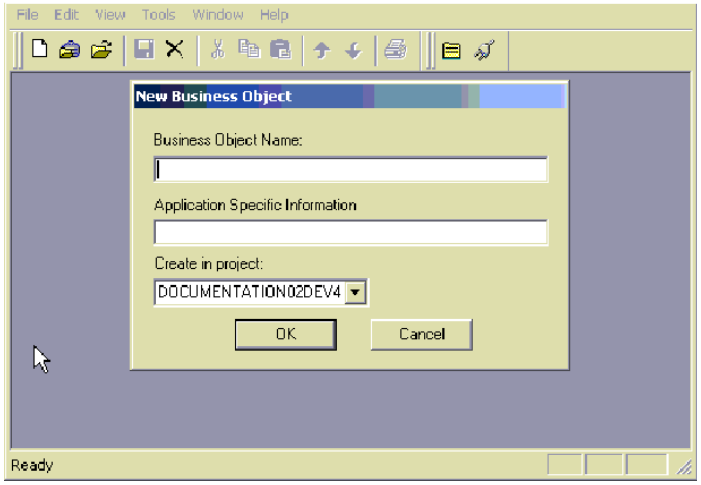


図 23. 「新規ビジネス・オブジェクト」ダイアログ・ボックス

ビジネス・オブジェクト定義をダブルクリックして Business Object Designer Express を開くと、選択した定義が Business Object Designer Express の作業域に表示されます。

Business Object Designer Express からのビジネス・オブジェクト定義のオープン

Business Object Designer Express を開くと、ファイルに格納されているオブジェクト定義を開くことができます。System Manager から Business Object Designer Express を実行している場合は、統合コンポーネント・ライブラリーに格納されているビジネス・オブジェクト定義を開くこともできます。

このセクションで説明する内容は次のとおりです。

- 『プロジェクトからのビジネス・オブジェクト定義のオープン』
- 56 ページの『名前の重複の防止』
- 55 ページの『ファイルからの定義のオープン』

プロジェクトからのビジネス・オブジェクト定義のオープン

Business Object Designer Express が既に開いている場合、プロジェクトからビジネス・オブジェクト定義を開くには以下のいずれかの操作を実行します。

注: System Manager から Business Object Designer Express を実行している場合は、プロジェクトは ICL ベースのプロジェクトです。それ以外の場合は、プロジェクトはローカル・プロジェクトであり、インポートされているビジネス・オブジェクト定義のみが含まれます。Business Object Designer Express のプロジェクトについては、49 ページの『プロジェクトの処理』を参照してください。

1. プロジェクト内のビジネス・オブジェクト定義のリストから、開きたい定義の名前を強調表示します。

2. 複数のビジネス・オブジェクト定義を選択するには、以下の手順のいずれかを実行します。
   - 連続する名前を選択するときには、最初の名前を選択し、Shift キーを押しながら最後の名前をクリックします。
   - 連続しない名前を選択するときには、Ctrl キーを押しながらそれぞれの名前をクリックして選択します。
3. 開きたい定義を選択した後、右マウス・ボタンでクリックしてから**「開く」**をクリックします。

   Business Object Designer Express は、選択された定義ごとに 1 つのウィンドウを表示します。

ファイルからの定義のオープン

ローカル・ディレクトリーに格納されているビジネス・オブジェクト定義を開くには、以下の手順を実行します。

1. **「ファイル」->「ファイルから開く」**をクリックします。

   「インポート」ダイアログ・ボックスが開きます。このダイアログ・ボックスのデフォルト設定では、タイプ XML スキーマ定義 (拡張子 .xsd を持ちます) のファイルをフィルターに掛けます。**「ファイルのタイプ」**リストから別のファイル・タイプを選択することも、全ファイル・タイプを選択することもできます。
2. 「インポート」ダイアログ・ボックスで、ファイルを検索して選択し、**「開く」**をクリックします。図 24 に、このダイアログ・ボックスを示します。

   注: System Manager から Business Object Designer Express を実行していない場合は、ダイアログ・ボックスの**「プロジェクトに」**リストは省略されます (このリストが表示された場合は、ICL ベースのプロジェクトを指定し、インポートされているビジネス・オブジェクト定義を受信できます)。代わりに、ビジネス・オブジェクト定義はローカル・プロジェクトにインポートされます。

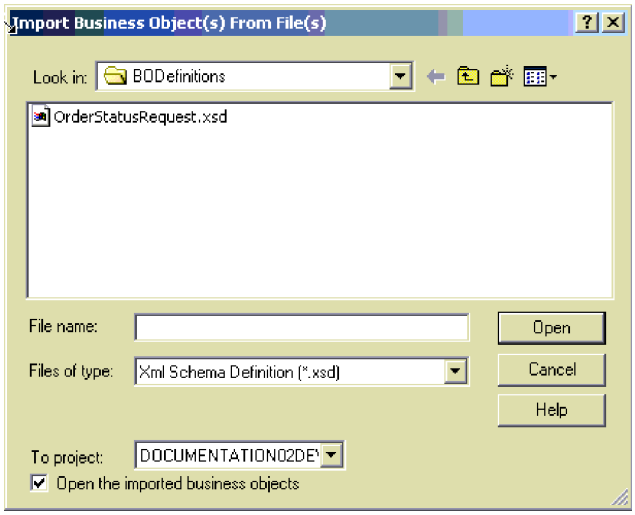


図 24. 「ファイルからビジネス・オブジェクトをインポート」ダイアログ・ボックス

「インポートしたビジネス・オブジェクトを開く」チェック・ボックスを選択すると、Business Object Designer Express はビジネス・オブジェクト定義も編集用に開きます。それ以外の場合、ビジネス・オブジェクト定義はプロジェクトにインポートされますが、編集用に開かれることはありません。詳しくは、57 ページの『ビジネス・オブジェクト定義の処理』を参照してください。

名前の重複の防止

Business Object Designer Express では、同じ名前を持つ 2 つのビジネス・オブジェクトを同じプロジェクトに置くことはできません。このようにすると、以下のいずれかの状態が発生します。

- プロジェクトに既に存在するものと同一のファイルからの定義が開かれます。
- 既にプロジェクトに存在するものと同一の新しい定義が作成されます。

  この場合、Business Object Designer Express はエラー・メッセージ「この名前のビジネス・オブジェクトは既に存在しています」を表示します。

ファイルから定義を開こうとしたときに、同じ名前の定義がローカル・プロジェクトまたは ICL ベースのプロジェクトに既に存在する場合は、図 25 に示すように、Business Object Designer Express で「インポート結果」ダイアログ・ボックスが表示されます。

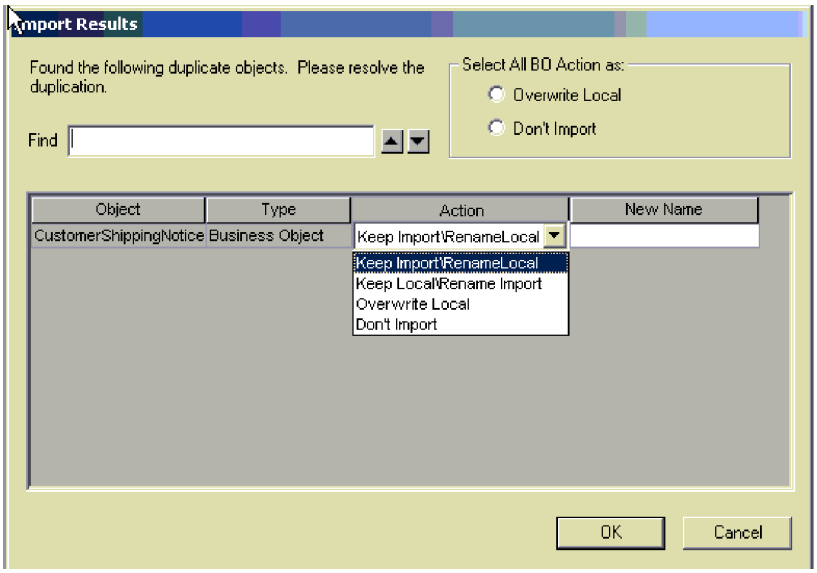


図 25. 名前の重複の防止: ローカルまたはインポートを保持

「インポート結果」ダイアログ・ボックスで、名前が重複していると報告された各ビジネス・オブジェクト定義について以下の手順を実行します。

1. **「アクション」**をクリックして、リストからアクションを選択します。各アクションの説明は以下のとおりです。
2. **「インポートを保持/ローカルの名前を変更」**または**「ローカルを保持/インポートの名前を変更」**を選択した場合は、図 25 に示すように、**「名前」**列にビジネス・オブジェクト定義の新しい名前を入力します。

また、**「すべての BO アクションを次のものとして選択」**を使用し、名前が重複していると報告されたすべてのビジネス・オブジェクト定義について 2 つのアクションのいずれかを指定できます。プロジェクトのすべてのビジネス・オブジェクト定義をインポートしている定義で上書きするには、**「ローカルを上書き」**を選択します。名前が重複しているビジネス・オブジェクト定義をインポートしないようにするには、**「インポートしない」**を選択します。

「インポート結果」ダイアログ・ボックスの**「処置」**リストには以下のオプションがあります。

- **「インポートを保持/ローカルの名前を変更」:** プロジェクトの定義の名前を変更し、ファイル内の定義の名前を変更せずに保持します。

  この変更を実行するには、図 25 に示すように**「新規の名前」**列に新しい名前を入力します。
- **「ローカルを保持/インポートの名前を変更」:** ファイル内の定義の名前を変更し、プロジェクトの定義の名前を変更せずに保持します。

  この変更を実行するには、図 25 に示すように**「新規の名前」**列に新しい名前を入力します。
- **「ローカルを上書き」:** プロジェクトに現在格納されている定義を、ファイルに格納されている定義で上書きします。
- **「インポートしない」:** ファイルに格納された定義をインポートするアクションを取り消します。

ビジネス・オブジェクト定義の処理

Business Object Designer Expressは、定義の作成と編集のために 2 つのタブ付きダイアログ・ボックスを提供します。

- **「一般」**タブ: ビジネス・オブジェクト・レベルのアプリケーション固有情報および動詞を指定または変更します。
- **「属性」**タブ: 属性プロパティーを指定または変更します。

定義を初めて作成するか、または開くときに、**「属性」**タブが開きます。

図 26 に、属性の定義および編集のための環境を示します。

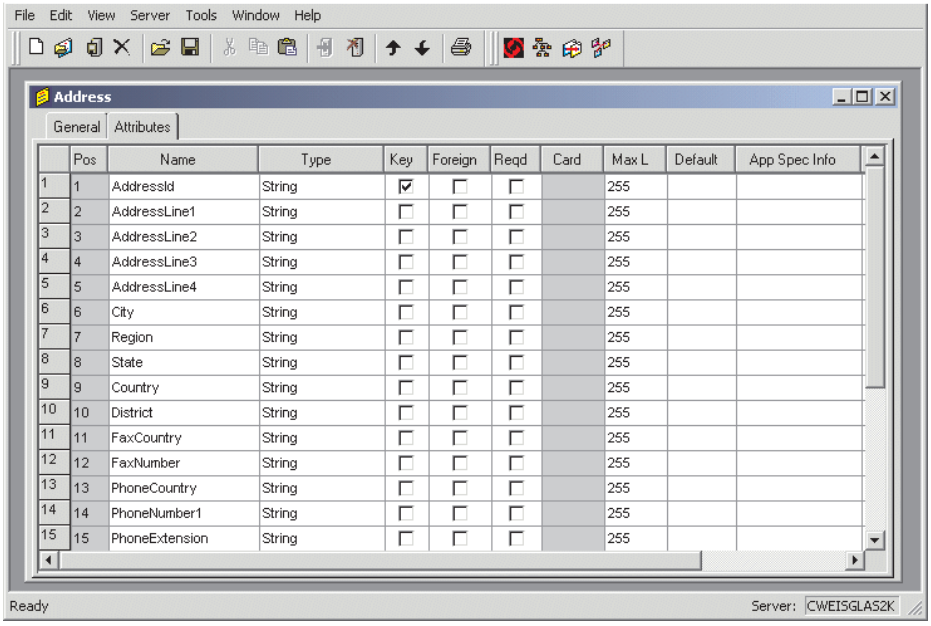


図 26. 属性の定義と編集

「一般」タブと**「属性」**タブの使用方法については、65 ページの『ビジネス・オブジェクト定義の作成』を参照してください。

ビジネス・オブジェクト定義と包含されている子のオープン

Business Object Designer Express では、親ビジネス・オブジェクト定義とその親に包含されている子の定義を編集するための、独立したウィンドウを開くことができます。

図 27 に、親ビジネス・オブジェクトと子ビジネス・オブジェクトを別個に編集するためのウィンドウを示します。

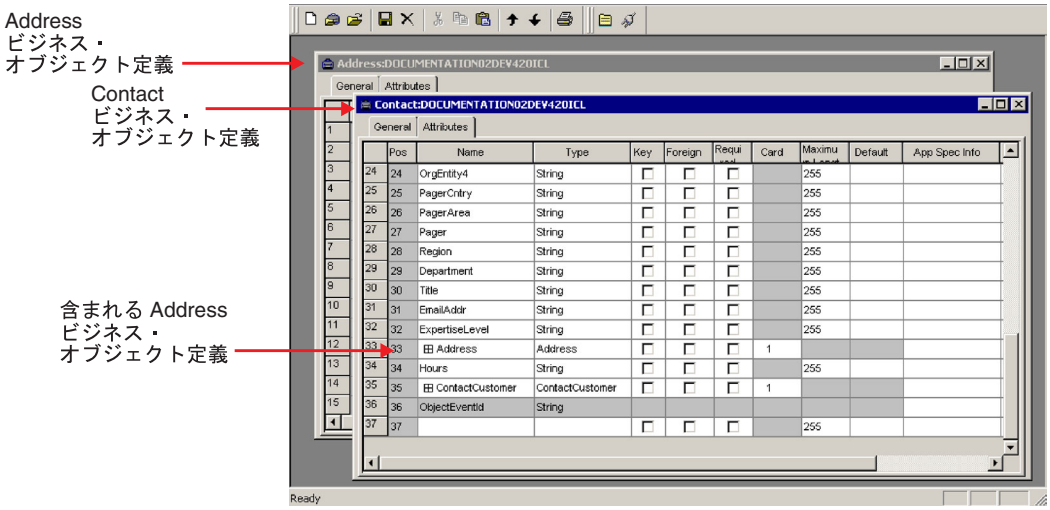


図 27. 親ビジネス・オブジェクトと子ビジネス・オブジェクトのための別個のウィンドウ

Contact ビジネス・オブジェクトの中の「住所」属性は、図 27 では展開されていないことに注意してください。属性を展開することにより、Address ビジネス・オブジェクトのすべての属性が「連絡先」ウィンドウに表示されます。これにより、親から直接、子を編集することができます。ただし、2 つの場所で同じ定義が変更されることを防ぐため、このツールでは、親ビジネス・オブジェクトの中で子ビジネス・オブジェクトを展開すると、必ず子ビジネス・オブジェクトのウィンドウが自動的にクローズされます。

図 28 は、「連絡先」の「住所」属性が展開され、「住所」ウィンドウが閉じられた後のツールの状態を示しています。

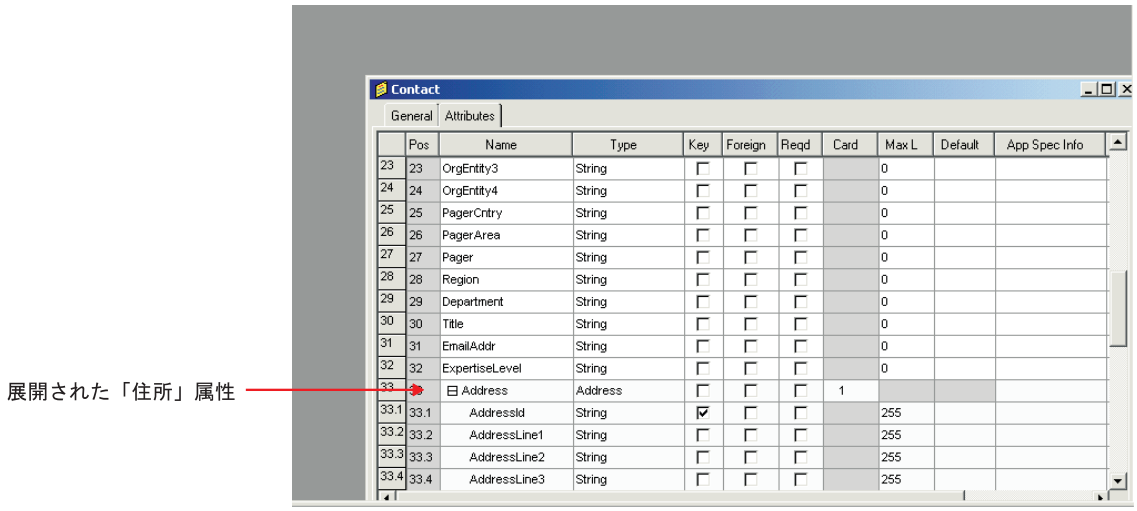


図 28. 子ビジネス・オブジェクトを表す親ビジネス・オブジェクト属性の展開

Business Object Designer Express の機能

Business Object 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。

- メニュー・バーから
- ツールバー・アイコンの使用

以下のセクションでは、Business Object Designer Express のメニューおよびメニュー・オプションの概要について説明します。

「ファイル」メニュー

「ファイル」メニューには以下の項目があります。

- **「新規ビジネス・オブジェクト」:** ビジネス・オブジェクト定義を手動で作成します。詳しくは、65 ページの『ビジネス・オブジェクト定義の作成』を参照してください。
- **「ODA を使用して新規作成」:** ビジネス・オブジェクト・ウィザードを表示します。このウィザードでは、Object Discovery Agent からビジネス・オブジェクト

定義を作成できます。詳しくは、75 ページの『Object Discovery Agent を使用してビジネス・オブジェクト定義を作成する方法』を参照してください。

- **「開く」:** プロジェクトに置かれているビジネス・オブジェクト定義を開きます。System Manager から Business Object Designer Express を実行している場合は、プロジェクトは ICL ベースのプロジェクトです。それ以外の場合は、プロジェクトはローカル・プロジェクトです。プロジェクトについては、 49 ページの『プロジェクトの処理』を参照してください。
- **「ファイルから開く」:** ローカル・ディレクトリーからビジネス・オブジェクト定義をインポートし、オプションで開きます。
- **「保管」:** 以下のようにプロジェクトにビジネス・オブジェクト定義を保管します。

  既存のビジネス・オブジェクト定義を変更した場合:

  – プロジェクトが ICL ベースの場合は、ビジネス・オブジェクト定義は元のプロジェクトに保管されます。

  – プロジェクトがローカルの場合は、ビジネス・オブジェクト定義はその既存のファイルに保管されます。

  新規ビジネス・オブジェクト定義を作成した場合:

  – Business Object Designer Express を System Manager から実行している場合は、ビジネス・オブジェクト定義を保管する ICL を選択するプロンプトが表示されます。

  – それ以外の場合は、ビジネス・オブジェクト定義のローカル宛先ディレクトリーおよびファイル名を指定するプロンプトが表示されます。

  ビジネス・オブジェクト定義は、以下のタイプのファイルとして保管できます。

  .xsd XML スキーマ定義。これはデフォルトのファイル・タイプです。

  .in または .text
  InterChange Server Express

  .sly スプレッドシート

- **「別名保管」:** ビジネス・オブジェクト定義を新しい名前で保管します。ビジネス・オブジェクト定義を含むファイルの名前は、宛先プロジェクト内で固有でなければなりません。

  既存のビジネス・オブジェクト定義を変更した場合:

  – Business Object Designer Express を System Manager から実行している場合は、ビジネス・オブジェクト定義は ICL ベースのプロジェクトに保管されます。

  – ビジネス・オブジェクト定義をファイルから開いた場合は、変更したビジネス・オブジェクト定義はそのファイルに保管されます。

  新規ビジネス・オブジェクト定義を作成した場合:

  – Business Object Designer Express を System Manager から実行している場合は、ビジネス・オブジェクト定義を保管する ICL を選択するプロンプトが表示されます。

– それ以外の場合は、ビジネス・オブジェクト定義のローカル宛先ディレクトリーおよびファイル名を指定するプロンプトが表示されます。

ビジネス・オブジェクト定義は、以下のタイプのファイルとして保管できます。

.xsd XML スキーマ定義。これはデフォルトのファイル・タイプです。

.in または .txt
InterChange Server Express

.xls スプレッドシート

- **「すべて保管」**: すべての開かれているビジネス・オブジェクト定義を保管します。詳しくは、60 ページの**「保管」**メニュー項目の説明を参照してください。
- **「コピーをファイルに保管」:** ビジネス・オブジェクト定義のコピーを別個のファイルにエクスポートします。
- **「すべてを 1 つのファイルに保管」**: プロジェクトにあるすべてのビジネス・オブジェクト定義を repos-copy 形式の 1 つのファイルにエクスポートします。
- **「閉じる」:** 選択された定義を閉じます。開いている定義が存在していなければ、このメニュー項目は利用できません。
- **「すべて閉じる」:** 開いている定義をすべて閉じます。開いている定義が存在していなければ、このメニュー項目は利用できません。
- **「削除」:** プロジェクトからビジネス・オブジェクト定義を削除できます。

  注: Business Object Designer Express でのみ、プロジェクトからビジネス・オブジェクト定義を削除できます。プロジェクトが ICL ベースの場合は、削除するビジネス・オブジェクト定義が指定の ICL から除去されます。プロジェクトがローカルの場合は、削除するビジネス・オブジェクト定義がローカル・プロジェクトから除去されますが、ローカル・ディレクトリーにあるビジネス・オブジェクト定義を含むファイルは影響を受けません。ローカル・ファイルを削除するには、Windows に付属しているツールを使用します。
- **「印刷設定」:** プリンターと印刷プロパティーを指定できます。
- **「印刷プレビュー」:** 印刷対象の定義のプレビューを表示します。開いている定義が存在していなければ、このメニュー項目は利用できません。
- **「印刷」:** 選択された定義を印刷することができます。開いている定義が存在していなければ、このメニュー項目は利用できません。
- **「終了」:** Business Object Designer Express を終了させることができます。

「編集」メニュー

開いている定義がなければ、**「編集」**メニューのどのオプションも利用できません。**「編集」**メニューには以下の項目があります。

- **「切り取り」:** 定義から属性を削除するか、または列からテキストを削除します。このメニュー項目は、列からどのテキストも選択されていなかったり、(左端の列をクリックすることにより) 属性が選択されていない場合には利用できません。属性編集用ウィンドウの例については、58 ページの図 26 を参照してください。
- **「コピー」:** 定義の属性または列のテキストをコピーします。このメニュー項目は、列からどのテキストも選択されていなかったり、(左端の列をクリックするこ

とにより) 属性が選択されていない場合には利用できません。属性編集用ウィンドウの例については、58 ページの図 26 を参照してください。

- **「貼り付け」:** 切り取られたか、またはコピーされた属性を定義に貼り付けます。あるいは切り取られたか、またはコピーされたテキストを選択された列に貼り付けます。デフォルトでは、ツールはバッファー内の属性を定義の下端に貼り付けます。ただし、特定の位置に空の行を挿入すると、バッファリング内の属性をその空の行の中に貼り付けることができます。
- **「行を削除」:** 定義から属性を削除します。このメニュー項目は、(左端の列をクリックすることにより) 属性が選択されていない場合には利用できません。属性編集用画面の例については、58 ページの図 26 を参照してください。
- **「すべて選択」:** 定義の属性をすべて選択します。
- **「上に挿入」:** 選択された属性の上に、空行を挿入します。
- **「下に挿入」:** 選択された属性の下に、空行を挿入します。
- **「上に移動」:** 選択された属性を 1 行上に移動します。このメニュー項目は、属性が選択されていない場合には利用できません。
- **「下に移動」:** 選択された属性を 1 行下に移動します。このメニュー項目は、属性が選択されていない場合には利用できません。

注: メニュー項目**「上に挿入」**、**「下に挿入」**、**「切り取り」**、**「コピー」**、**「貼り付け」**、および**「削除」** は、属性の左端列を右クリックしてアクセスできます。

「表示」メニュー

「表示」メニューの操作は、Business Object Designer Express を最初に開いたとき、および作業領域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。

「表示」メニューに表示されるオプションは次のとおりです。

- **「すべて展開」:** すべての子ビジネス・オブジェクトのすべての属性を表示します。開いている定義が存在していなければ、このメニュー項目は利用できません。
- **「すべて縮小」:** すべての子ビジネス・オブジェクトのすべての属性の表示を閉じます。開いている定義が存在していなければ、このメニュー項目は利用できません。
- **「設定」:** 「ビジネス・オブジェクト設定」ダイアログ・ボックスを開きます。このダイアログでは、ビジネス・オブジェクトの削除を確認しないように設定できます。
- **「ツールバー」:** この中のサブメニューには、Business Object Designer Express の 2 つのツールバーの表示を制御する項目があります。メニュー・オプションは次のとおりです。
  - **「標準」:** このメニュー項目をクリックすると、Business Object Designer Express に標準ツールバー用のボタンが表示されます。

– **「プログラム」:** このメニュー項目をクリックすると、Business Object Designer Express に、その他の WebSphere Business Integration Toolset 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」:** このメニュー項目をクリックすると、Business Object Designer Express のメイン・ウィンドウの下部に 1 行の状況メッセージが表示されます。

注: 1 つの子ビジネス・オブジェクトまたは複数の子ビジネス・オブジェクトの配列を表す属性の左端列を右クリックすると、**「展開」**、**「縮小」**、および**「ウィンドウで開く」**項目にアクセスできます。

「ツール」メニュー

「ツール」メニューには以下の項目があります。

- **「Log Viewer」:** Log Viewer を開きます。
- **「Connector Configurator Express」:** Connector Configurator Express を開きます。
- **「System Manager」:** System Manager を開きます。

「ウィンドウ」メニュー

「ウィンドウ」メニューは、標準的な Windows 環境の場合と同様に機能します。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、メニュー・オプションを使用します。

第 4 章 ビジネス・オブジェクト定義の開発

この章では、ビジネス・オブジェクト定義の作成と削除の基本的な手順について、順を追って説明します。この章を学習することによって、定義を手動により作成する方法と、Object Discovery Agent (ODA) を使用して作成する方法の両方を理解することができます。各 ODA では、特定のアプリケーション向けの定義が生成されます。

この章ではビジネス・オブジェクト定義を作成する定型的手順について説明しますが、実際に定義を作成する時には、設計概念についての理解も深めておくことが必要です。詳しくは、19 ページの『第 2 章 ビジネス・オブジェクト設計』を参照してください。Object Discovery Agent の作成方法については、101 ページの『第 5 章 Object Discovery Agent の開発』を参照してください。

この章の主な内容は次のとおりです。

ビジネス・オブジェクト定義の作成

ビジネス・オブジェクト定義を作成する方法は 2 つあります。

- 手動 — 汎用ビジネス・オブジェクトまたは単純なビジネス・オブジェクトを作成する場合や、Object Discovery Agent によって作成された定義を変更する場合に有用です。Business Object Designer Express では、グラフィック・インターフェースにより、ビジネス・オブジェクト定義を手動で作成することが可能となっています。このセクションには、以下の内容を説明するチュートリアルが用意されています。
  - 『手動によるフラット・ビジネス・オブジェクト定義の作成』
  - 72 ページの『手動による階層型ビジネス・オブジェクト定義の作成』
- Object Discovery Agent の使用 — アプリケーション固有のビジネス・オブジェクトを作成するときに有用です。*Object Discovery Agent* は、指定されたエンティティーを該当するアプリケーションの中で検査し、ビジネス・オブジェクト属性に対応する要素および各属性のプロパティーを「検出」して、ビジネス・オブジェクト定義を作成します。詳しくは、75 ページの『Object Discovery Agent を使用してビジネス・オブジェクト定義を作成する方法』を参照してください。

手動によるフラット・ビジネス・オブジェクト定義の作成

このセクションでは、`Hello` という名前のビジネス・オブジェクト定義を手動で作成する方法について説明します。InterChange Server Express で、このビジネス・オブジェクトは、`SampleHello` コラボレーションによって使用されます。この作成については、「*コラボレーション開発ガイド*」のチュートリアルの章に解説されています。

図 29 では、作成可能な Hello ビジネス・オブジェクト定義を示すとともに、このビジネス・オブジェクトの InterChange Server Express がトリガー・イベントとなるビジネス・オブジェクトから予想する値も示します。

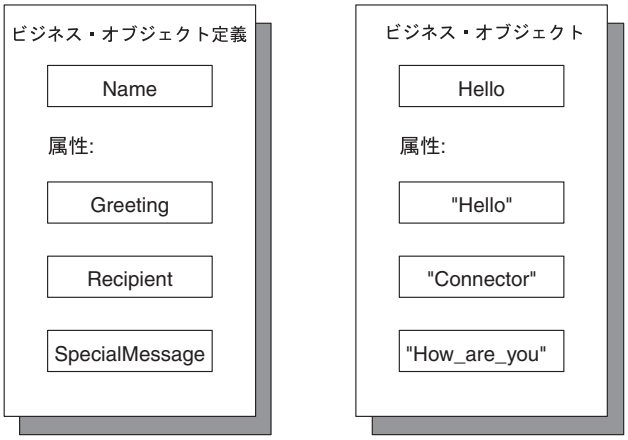


図 29. Hello ビジネス・オブジェクト

ビジネス・オブジェクト定義を手動で作成するには、以下の手順を実行します。

1. Business Object Designer Express を開始します。詳しくは、53 ページの『Business Object Designer Express の開始』を参照してください。
2. **「ファイル」->「新規」**をクリックします。

   Business Object Designer Express が「新規ビジネス・オブジェクト」ダイアログ・ボックスを表示します。図 30 に、System Manager から Business Object Designer Express を実行しているかどうかを調べる「新規ビジネス・オブジェクト」ダイアログ・ボックスのバージョンを示します。 System Manager から Business Object Designer Express を実行していない場合は、**「作成先プロジェクト」**リストはダイアログ・ボックスに表示されません。

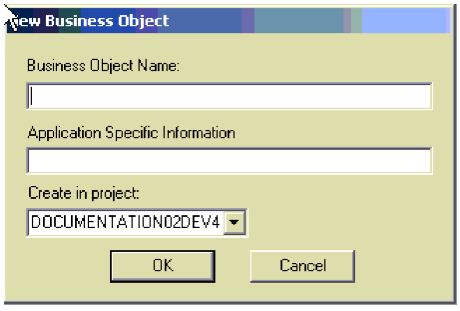


図 30. 「新規ビジネス・オブジェクト」ダイアログ・ボックス

3. 新規のビジネス・オブジェクト定義の名前として Hello を入力します。

   名前は通常、大文字と小文字が区別されるため、ここに示したとおり正確に入力してください。

注: ビジネス・オブジェクト定義の名前に使用できる文字は、英数字と下線 (_) に限定されています。この名前には、英語 (U.S.) のロケール (en_US) に関連したコード・セットで定義されている文字のみ を使用する必要があります。

4. **「アプリケーション固有の情報」**ボックスは空白のままとし、「OK」をクリックします。

図 31 に示すように、Business Object Designer Express はビジネス・オブジェクト定義ダイアログ・ボックスを表示します。

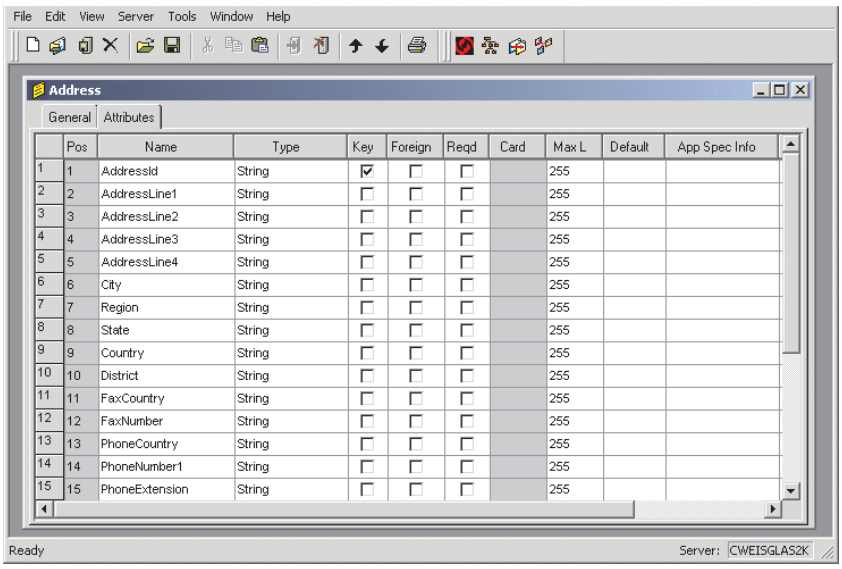


図 31. 新規ビジネス・オブジェクト定義の初期表示

注: Business Object Designer Express のインターフェースには、若干の違いがある場合があります。ただし、ツールの基本的な機能は同じです。

属性の追加

ビジネス・オブジェクトの中の情報はそれぞれ、Hello ビジネス・オブジェクト定義に含まれる*属性*によって表されます。Hello ビジネス・オブジェクトのために属性定義を指定する必要があります。図 31 に示すように、必須のオブジェクトの終端マーカーである ObjectEventId のエントリーが、Business Object Designer Express によって自動的に追加されます。

重要

ObjectEventId 属性は削除、変更、または移動しないでください。この属性は、WebSphere Business Integration システムの内部使用のために予約されています。この定義を保管すると、Business Object Designer Express は自動的にこの属性を移動します。

各属性の行により、属性のプロパティーが決まります。属性のプロパティーについては、 5 ページの『ビジネス・オブジェクトの属性および属性プロパティー』を参照してください。

66 ページの図 29 に示すように、Hello ビジネス・オブジェクト定義は、属性として Greeting、Recipient、および SpecialMessage を備えています。属性とそのプロパティーは、1 つずつ定義してください。

Greeting 属性の追加: Greeting 属性を追加するには、以下のようにします。

1. 使用できる最初の空行の**「名前」**列に属性の名前である Greeting を入力します。最初の属性の場合、この行は、2 です。

   注: この属性名には、英語 (U.S.) のロケール (en_US) に関連したコード・セットで定義されている文字のみ を使用する必要があります。

2. **「タイプ」**列をクリックし、属性タイプの**「ストリング」**を選択します。属性のタイプは、そのデータ型と同じです。

   ヒント:

   Business Object Designer Express で、他のビジネス・オブジェクトを開いた場合、その名前は **「タイプ」**リストに表示されます。**「タイプ」**から選択した既存のビジネス・オブジェクトを表示することで、階層構造を持つビジネス・オブジェクトを作成できます。そして、このようなビジネス・オブジェクトの属性タイプは、他のビジネス・オブジェクトであるような属性を持っています。

   InterChange Server Express および System Manager が実行されている場合は、処理元の統合コンポーネント・ライブラリーのすべてのビジネス・オブジェクト定義が自動的にこのリストに表示されます。

3. **「キー」**、**「外部」**、**「必要」**、および**「カード」**列は飛ばします。

   これらの列では、現在の属性がビジネス・オブジェクトの基本キーか外部キーか、属性の値が必要であるかどうか、属性が表している子ビジネス・オブジェクトが単数か複数かを指定します。これらのプロパティーについては、 19 ページの『第 2 章 ビジネス・オブジェクト設計』を参照してください。

4. **「最大長」**ボックスは、デフォルト値の 255 のままとします。

   このボックスは、この属性の値として使用できる最大バイト数を指定します。

5. **「デフォルト値」**ボックスに Hello と入力します。

   これは、実行時に属性の値の指定がなかったときに使用される値を指定します。

   これで、Greeting 属性に対して次のプロパティーを定義したことになります。

名前:	Greeting
タイプ:	String
最大長:	255
デフォルト値:	Hello

6. 他の列はすべて無視し、3 行目の**「名前」**列をクリックします。

Recipient 属性の追加: 2 つ目の属性 Recipient は、ストリングです。

InterChange Server Express では、SampleHello コラボレーション・オブジェクトによって、この属性が以下のように使用されます。

- コネクターは、メッセージをコラボレーションに送るとき、この値を Collaboration に設定します。
- コラボレーションは、メッセージをコネクターに送るとき、この値を Connector に設定します。

各ビジネス・オブジェクト定義内には 1 つ以上のキー*属性* が必要です。キー属性には、WebSphere Business Integration システムがビジネス・オブジェクトの各インスタンスを識別するための値が含まれています。Recipient 属性をキー属性にしてください。

Recipient 属性を追加するには、**「名前」**列にテキスト Recipient を入力し、以下のプロパティーを使用して Greeting 属性追加の手順に従います。

名前:	Recipient
タイプ:	String
最大長:	255
デフォルト値:	Collaboration
キー:	はい (「キー」列にチェックマークが表示されます)

他の列はブランクとし、4 行目の**「名前」**列をクリックします。

SpecialMessage 属性の追加: 3 つ目の属性 SpecialMessage は、ストリングです。

InterChange Server Express では、SampleHello コラボレーションは、コラボレーション・オブジェクトが作成された後、コラボレーション構成プロパティーへのアクセス権を持つシステム管理者などによって、この属性の値が入力されることを想定しています。コラボレーションは構成プロパティーの値を動的に取得し、メッセージの最後に追加します。

SpecialMessage 属性を追加するには、**「名前」**列にテキスト SpecialMessage を入力し、以下のプロパティーを使用して Greeting 属性追加の手順に従います。

名前:	SpecialMessage
タイプ:	String
最大長:	255

他の列はブランクとします。

「属性」タブに、Greeting、Recipient、および SpecialMessage の 3 つのユーザー定義の属性が表示されるようになります。図 32 に、Hello ビジネス・オブジェクトの属性を示します。

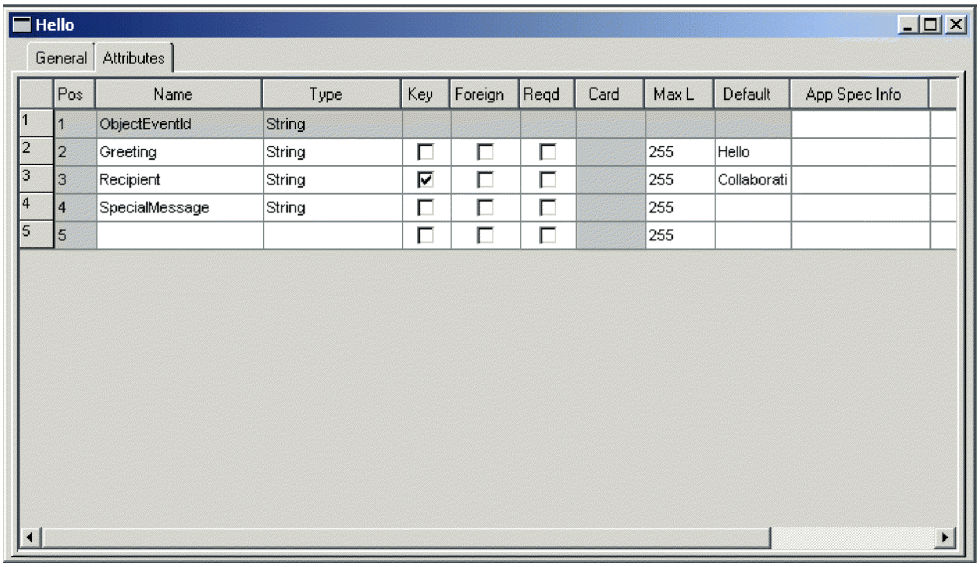


図 32. 属性が定義された新規ビジネス・オブジェクト

属性の順序の変更

ビジネス・オブジェクト定義内での属性の順序を画面上で変更することができます。例えば、キー属性である `Recipient` を `Greeting` 属性の上にするには、最初 (左端) の列をクリックし、カーソルを 1 行分上にドラッグします。

サポートされる動詞の指定

次に、この `Hello` ビジネス・オブジェクトがサポートする動詞を指定する必要があります。これらの動詞は、ビジネス・オブジェクトから InterChange Server Express に送られるトリガー・イベントを表します。 Hello ビジネス・オブジェクト定義ダイアログ・ボックスの**「一般」**タブをクリックすることにより、動詞を指定するウィンドウを表示します。図 33 に、このタブを示します。

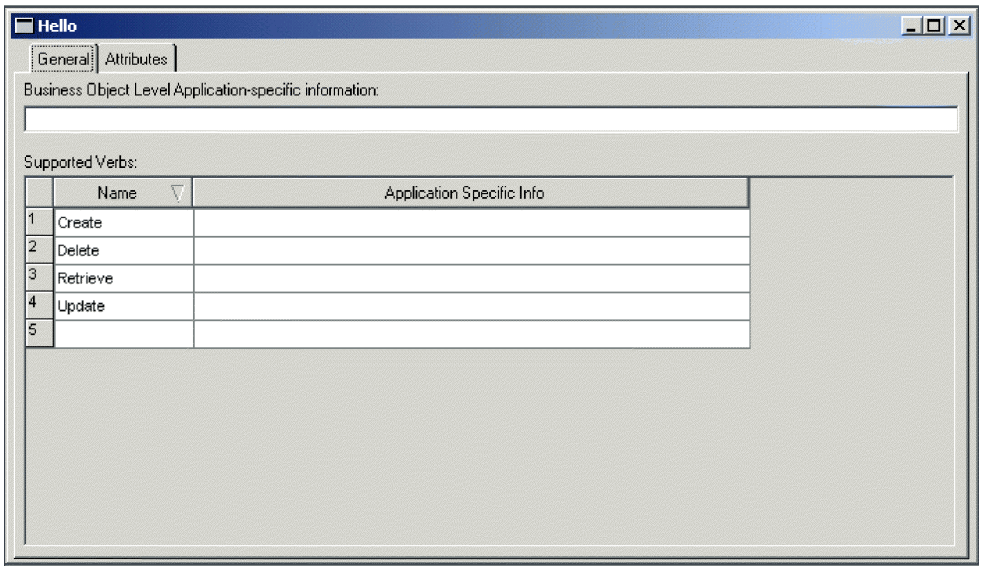


図 33. 「一般」編集タブ

ビジネス・オブジェクトは、Create、Delete、Retrieve、および Update の 4 つをデフォルトの動詞としてサポートしており、これらの動詞は**「一般」**タブにデフォルトで表示されます。このチュートリアルの目的に合わせて、トリガー・イベントとしては Create 1 つのみがサポートされています。したがって、この動詞のみをサポートするようにビジネス・オブジェクト定義を変更します。

重要: 各ビジネス・オブジェクト定義に対して 1 つ以上の動詞を指定することが必要です。

注: 動詞の名前に使用できる文字は、英数字と下線 (_) に限定されています。この名前には、英語 (U.S.) のロケール (en_US) に関連したコード・セットで定義されている文字のみ を使用する必要があります。

Hello ビジネス・オブジェクトで動詞 Create のみがサポートされることを指示するため、残りの動詞を 1 つずつまたは一括して削除することができます。

複数の動詞の削除: 動詞 Delete、Retrieve、および Update を削除するには、以下の手順を実行します。

1. Shift キーを押しながら動詞 Delete を選択し、動詞 Update をクリックします。
2. 「削除」キーを押します。

個々の動詞の削除: 各動詞を個別に削除するには、以下の手順を実行します。

1. **「サポートされている動詞」**テーブルの「削除」行の左側の数字をクリックします。

   行が選択状態になります。
2. 「削除」キーを押します。
3. **「サポートされている動詞」**テーブル内の動詞 Retrieve および Update についてステップ 1 および 2 を繰り返します。

4. Create 動詞については、**「アプリケーション固有の情報」** ボックスをブランクのままとします。

これで、Hello ビジネス・オブジェクトの定義が完了しました。この時点で、**「ファイル」->「保管」**をクリックし、変更内容を保管してください。ICL ベースのプロジェクトを使用している場合は、定義は ICL に保管されます。ローカル・プロジェクトを使用している場合は、定義を保管するファイル名およびローカル・ディレクトリーを指定するプロンプトが表示されます。

手動による階層型ビジネス・オブジェクト定義の作成

このセクションでは、1 つの子ビジネス・オブジェクトまたは複数の子ビジネス・オブジェクトの配列を表す属性を定義することにより、階層型ビジネス・オブジェクト定義を作成する方法について説明します。

直前のセクションで単純属性とサポートされる動詞を定義する方法について説明しているので、このセクションでは、子ビジネス・オブジェクトを表す属性の定義についてのみ説明します。このセクションの例では、次の 2 つの属性を持つ HierarchicalBO という名前のビジネス・オブジェクトを作成します。

- ビジネス・オブジェクトの必須キーの役目を果たす Key という名前の属性
- カーディナリティーが 1 の Address ビジネス・オブジェクトを表す Addr という名前の属性

階層型ビジネス・オブジェクト定義を手動で作成するには、以下の手順を実行します。

1. Business Object Designer Express を開きます。
2. **「ファイル」->「新規」**をクリックします。

   66 ページの図 30 に示すように、Business Object Designer Express は「新規ビジネス・オブジェクト」ダイアログ・ボックスを表示します。
3. 新規のビジネス・オブジェクト定義の名前として HierarchicalBO を入力します。
4. **「アプリケーション固有の情報」**列は空白のままとし、**「OK」**をクリックします。

   67 ページの図 31 に示すように、Business Object Designer Express はビジネス・オブジェクト定義ダイアログ・ボックスを表示します。
5. 使用できる最初の空行にキー属性を作成します。最初の属性の場合、この行は 2 です。この属性に Key という名前を付け、任意の単純なデータ・タイプを指定し、**「キー」**列をクリックします。
6. 使用できる次の空行に次の属性を作成します。この行は 3 です。この属性に Addr という名前を付けます。
7. **「タイプ」**リストをクリックし、属性タイプの**「アドレス」**を選択します。

   注: リストに子ビジネス・オブジェクトが存在しない場合には、ここで**「タイプ」**リストの**「新規ビジネス・オブジェクト」**を選択することにより、子ビジネス・オブジェクトを作成します。このステップを終了するためには、新しい子ビジネス・オブジェクトを保管することが必要です。

8. **「キー」**、**「外部」**、および**「必要」**列は飛ばします。**「カード」**リストをクリックして、**「1」**を選択します。
9. 他の列はすべて無視します。サポートされる動詞を定義し、定義を保管します。

ビジネス・オブジェクト定義の削除

InterChange Server Express では、Business Object Designer Express または System Manager を使用してビジネス・オブジェクト定義を削除できます。このセクションで説明する内容は次のとおりです。

- 『Business Object Designer Express による定義の削除』
- 74 ページの『System Manager での定義の削除』

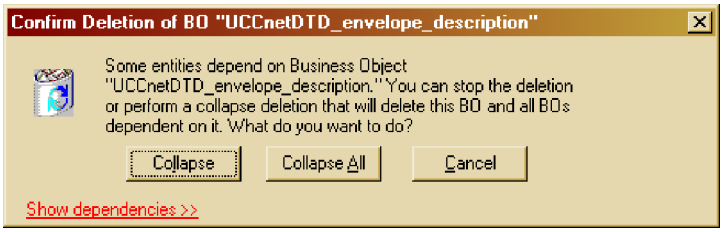
重要

ビジネス・オブジェクト定義の削除は、(InterChange Server Express を使用しており、かつ System Manager を実行している場合は) System Manager 経由で統合コンポーネント・ライブラリーから、その他の場合は Business Object Designer Express のプロジェクトから可能です。Business Object Designer Express または System Manager で削除機能を使用しても、ビジネス・オブジェクト定義を含むローカル・ファイルは削除できません。ローカル・ファイルを削除するには、Windows に付属しているツールを使用します。

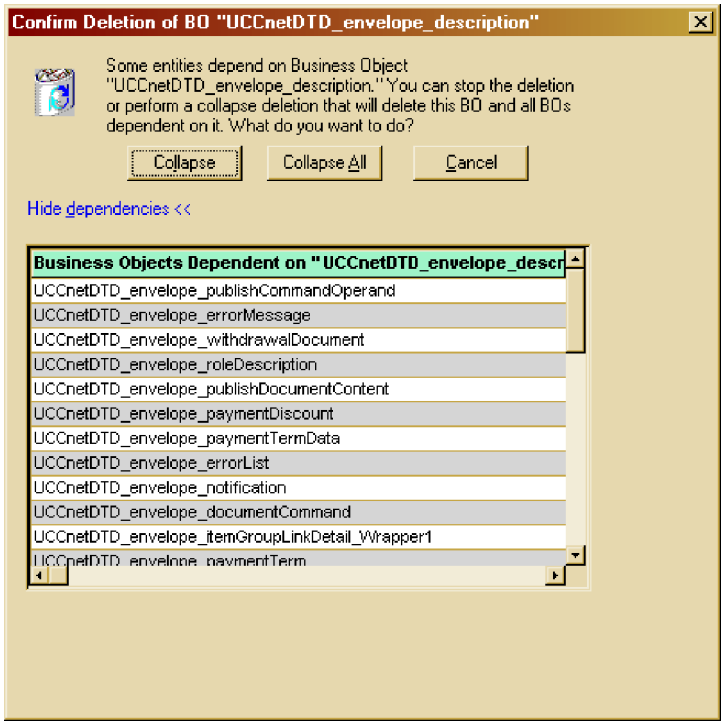
Business Object Designer Express による定義の削除

Business Object Designer Express を使用してプロジェクトからビジネス・オブジェクト定義を削除するには、以下の手順を実行します。

1. Business Object Designer Express を開きます。
2. プロジェクト内のビジネス・オブジェクト定義のリストから、削除したい定義の名前を選択します。
3. 複数の名前を選択するには、以下の手順のいずれかを実行します。
   - 連続する名前を選択するためには、最初の名前をクリックしてから、`Shift` キーを押しながら最後の名前をクリックします。
   - 連続しない名前を選択するためには、`Ctrl` キーを押しながらそれぞれの名前をクリックします。
4. 削除する定義を選択した後、右マウス・ボタンでクリックしてから**「削除」**をクリックします。
   - Business Object Designer Express が**「ビジネス・オブジェクトの削除 (Deleting business object)」**確認メッセージを表示します。**「はい」**をクリックして、ステップ 2 で選択したビジネス・オブジェクトを削除するか、またはステップ 3 で選択したビジネス・オブジェクト定義をすべて削除します。
   - ビジネス・オブジェクト定義に他のビジネス・オブジェクトとの依存関係がある場合、Business Object Designer Express は、縮小表示の削除確認メッセージを表示します。



5. 依存関係が存在する場合、**「依存関係を表示」**リンクを選択します。削除したいビジネス・オブジェクト定義について、他のビジネス・オブジェクトとの依存関係がすべてリストされます。



6. 以下のいずれかの手順を実行します。
   - **「縮小」**をクリックして、ステップ 2 で選択したビジネス・オブジェクト定義およびそれに依存するすべてのビジネス・オブジェクトを削除します。
   - ステップ 3 で複数のビジネス・オブジェクトを選択した場合は、**「すべて縮小」**をクリックして、選択した複数のビジネス・オブジェクト定義およびそれらのビジネス・オブジェクトのそれぞれに依存するすべてのビジネス・オブジェクトを削除します。
   - ビジネス・オブジェクト定義とその依存関係の削除を取り消すには、**「キャンセル」**をクリックします。

System Manager での定義の削除

System Manager を使用してビジネス・オブジェクト定義を削除するには、以下の手順を実行します。

1. System Manager を始動します。

2. **統合コンポーネント・ライブラリー**を展開し、ビジネス・オブジェクト定義を削除する統合コンポーネント・ライブラリーを展開します。
3. ビジネス・オブジェクト・フォルダーを開き、削除するビジネス・オブジェクト定義の名前を選択します。
4. 以下のいずれかを行い、ビジネス・オブジェクト定義を削除します。
   - **「削除」**ツールバー・アイコンをクリックします。
   - ビジネス・オブジェクト定義を右クリックし、**「削除」**を選択します。
5. 削除の確認を求められたときには、**「はい」**をクリックします。
6. ビジネス・オブジェクト定義が他のビジネス・オブジェクトとの依存関係を持つ場合、System Manager はエラー・メッセージを表示します。Business Object Designer Express を使用してこれらの依存関係を除去しなければ、System Manager でビジネス・オブジェクト定義を削除することはできません。

Object Discovery Agent を使用してビジネス・オブジェクト定義を作成する方法

このセクションでは、アプリケーション固有のビジネス・オブジェクトのビジネス・オブジェクト定義を、Object Discovery Agent (ODA) を使用して生成する方法について説明します。 ODA は、アダプターのオプショナル・コンポーネントです。 ODA を持つアダプターをインストールすると、ODA は自動的にインストールされます。カスタム・アダプターを開発しており、ODA を使用してビジネス・オブジェクト定義を作成する場合は、Object Discovery Agent Development Kit (ODK) を使用して開発できます。カスタム ODA の開発については、101 ページの『第 5 章 Object Discovery Agent の開発』を参照してください。

ODA の構成と実行には、Business Object Designer のビジネス・オブジェクト・ウィザードを使用します。ビジネス・オブジェクト・ウィザードは、ODA に対するグラフィカル・ユーザー・インターフェースであり、検出やコンテンツ生成のプロセスの管理を行います。このセクションの内容は次のとおりです。

ODA を使用するための準備

ODA を実行する前に、次の各項目にあてはまることを確認してください。

- システム始動ファイルが使用可能であり、かつ適切なものである。
- ODA が開始済みである。
- Business Object Designer Express が開始済みである。

システム始動ファイル

ODA を開始するには、ODA に必要なファイルがご使用のシステムにあることを確認しておく必要があります。ODA を持つ定義済みアダプターをインストールすると、必要な ODA システム始動ファイルが自動的にインストールされます。カスタム ODA を持つカスタム・アダプターを開発している場合、必要な ODA システム始動ファイルはその ODA の開発の過程で作成されます。ただし、始動スクリプトがあるかどうか、およびご使用の ODA に適したものであるかどうかを確認することをお勧めします。

各 ODA は、それぞれ*始動スクリプト* を必要とします。始動スクリプトは、ODA の実行を開始するものです。 ODA を最初に開始する前に、始動スクリプト内の変数が正しく設定されていることを確認しておく必要があります。シェル・ファイル (`start_`*`ODAname`*`.sh`) またはバッチ・ファイル (`start_`*`ODAname`*`.bat`) を編集できるように開き、表 10 の説明どおり値が正しく設定されていることを確認してください。

表 10. ODA シェル・ファイルおよびバッチ・ファイルの構成変数

変数	説明	例
`set AGENTNAME`	ODA の名前	`set AGENTNAME=`*`ODAname`*
`set AGENT`	ODA の JAR ファイルの名前	**WINDOWS:** `set AGENT =` *`%ProductDir`*`%¥ODA¥`*`srcDataName`*`¥`*`ODAname`*`.jar`
`set AGENTCLASS`	ODA の Java クラスの名前	`set AGENTCLASS=com.ibm.oda.`*`srcDataName`*`.`*`ODAname`

ODA の名前 (*ODAname*) とソース・データ名 (*srcDataName*) については、 185 ページの『ODA の名前付け』を参照してください。

ODA の開始

Windows オペレーティング・システムの場合、次の始動スクリプトを使用して ODA を開始できます。

Windows

```
start_srcDataNameODA.bat
```

Business Object Designer Express のビジネス・オブジェクト・ウィザードを使用して、ODA を構成および実行します。ビジネス・オブジェクト・ウィザードは、各スクリプト・ファイルまたはバッチ・ファイルの `AGENTNAME` 変数に指定されている名前を基に各 ODA を探し出します。

注: ODA のインスタンスを複数開始する方法については、 97 ページの『複数の ODA の同時使用』を参照してください。

Business Object Designer Express の開始

ODA を開始したら、Business Object Designer Express を開き、構成して実行する必要があります。Business Object Designer Express を開く方法については、 53 ページの『Business Object Designer Express の開始』を参照してください。Business Object Designer Express にはビジネス・オブジェクト・ウィザードがあり、このウィザードに従うと順を追って ODA を実行できます。

ビジネス・オブジェクト・ウィザードを開始するには、以下の手順を実行します。

1. 53 ページの表 9 に示す方法のいずれかを使用して Business Object Designer Express を開きます。
2. **「ファイル」->「ODA を使用して新規作成」**をクリックします。

ビジネス・オブジェクト・ウィザードが開始され、ウィザードの最初のダイアログ・ボックス (「エージェントの選択」) が表示されます。表 11 に、ビジネス・オブジェクト・ウィザードのステップについて要約します。

表 11. ビジネス・オブジェクト・ウィザードのステップ

作業	ビジネス・オブジェクト・ウィザードのステップ
1. 必要な ODA を選択する。	ステップ 1: エージェントの選択
2. 構成プロパティーを取得します (開くデータ・ソースを記述する構成プロパティーも含めて)。	ステップ 2: エージェントの構成
3. ODA によるコンテンツ生成の対象となるソース・データを取得する。	ステップ 3: ソースの選択
4. 選択されたソース・ノードがコンテンツ生成に使用したいノードであることを確認する。	ステップ 4: ソース・ノードの確認
5. コンテンツ生成プロセスを開始する。	ステップ 5: ビジネス・オブジェクトの生成中 ビジネス・オブジェクト・プロパティー
6. ビジネス・オブジェクト定義をユーザー指定のフォーマットで保管する。	ステップ 6: ビジネス・オブジェクトの保管

ビジネス・オブジェクト・ウィザードによる ODA 実行の例については、『サンプル ODA の使用』を参照してください。

サンプル ODA の使用

ローマ軍兵士のデータ (XML 形式) をビジネス・オブジェクト定義に変換するサンプル Object Discovery Agent が、IBM から提供されています。ODA の使用方法を十分に理解できるように、このサンプル ODA を使用して、ビジネス・オブジェクト定義の生成方法を以下に段階的に説明します。

注: このサンプル ODA の場所とファイルについては、112 ページの『ODA の開発サポート』を参照してください。

このセクションでは、次の作業について説明します。

- 『サンプル ODA の開始』
- 78 ページの『ODA を使用したビジネス・オブジェクト定義の作成』

サンプル ODA の開始

Adapter Development Kit (ADK) がインストールされている場合、サンプル ODA とそれを実行するためのファイルは、製品ディレクトリーの下の `DevelopmentKits¥Odk¥Samples` ディレクトリーに格納されています。サンプル ODA を実行するためのファイルは、ご使用のオペレーティング・システム環境によって異なります (表 12 を参照)。

表 12. サンプル Roman Army ODA の始動スクリプト

オペレーティング・システム	始動スクリプト
Windows	start_Agent4.bat

注: サンプル Roman Army ODA には 5 つのバージョンがあります。これは、ODA のさまざまな機能の実例を紹介するためです。このセクションでは、このサンプル ODA の 4 番目のバージョンについて説明します。このバージョンは、start_Agent4 始動スクリプトと ArmyAgent4 クラス・ファイルを使用します。

サンプル Roman Army ODA には 5 つのバージョンがあるため、すべての始動スクリプトが start_AgentX という名前の 1 つの共通始動スクリプトを呼び出し、スクリプトに ODA クラス名を渡します (この名前は start_AgentX 内の AGENTCLASS 構成変数に割り当てられます)。つまり、start_Agent4 始動スクリプトは start_AgentX に対する呼び出しを含み、このスクリプトに次のパスを ODA クラス名として渡します。

```
com.ibm.btools.ODK2.RomanArmy.ArmyAgent4
```

このサンプル ODA の構成変数を検証するには、start_AgentX バッチ・ファイルまたはスクリプト・ファイルを調べて、構成変数が表 13 の構成変数と一致していることを確認します。サンプル Roman Army ODA のバージョン 4 が使用するファイルを移動するときは、対応する構成変数も必ず変更してください。

表 13. サンプル Roman Army ODA の構成変数

変数	サンプル Roman Army ODA 用の値
AGENTNAME	set AGENTNAME=Roman
AGENT	**WINDOWS:** set AGENT = *%ProductDir%*¥DevelopmentKits¥Odk¥Samples¥RomanArmy¥ArmyODA.jar
FILE_LOCATION	**WINDOWS:** set FILE_LOCATION = *%ProductDir%*¥DevelopmentKits¥Samples¥Odk¥RomanArmy¥RomanArmy.xml

重要

接続を試みる前に、ビジネス・オブジェクト・ウィザードを使用してサンプル ODA を起動しておく必要があります。ビジネス・オブジェクト・ウィザードは、起動済みの ODA のみを探し出します。

ODA を使用したビジネス・オブジェクト定義の作成

ビジネス・オブジェクト・ウィザードを開始するには、以下の手順を実行します。

1. 53 ページの表 9 に示す方法を使用して Business Object Designer Express を開きます。
2. **「ファイル」->「ODA を使用して新規作成」**をクリックします。

ビジネス・オブジェクト・ウィザードは、ウィザードの最初のダイアログ・ボックス (「エージェントの選択」) を表示します。このダイアログ・ボックスを図 34 に示します。

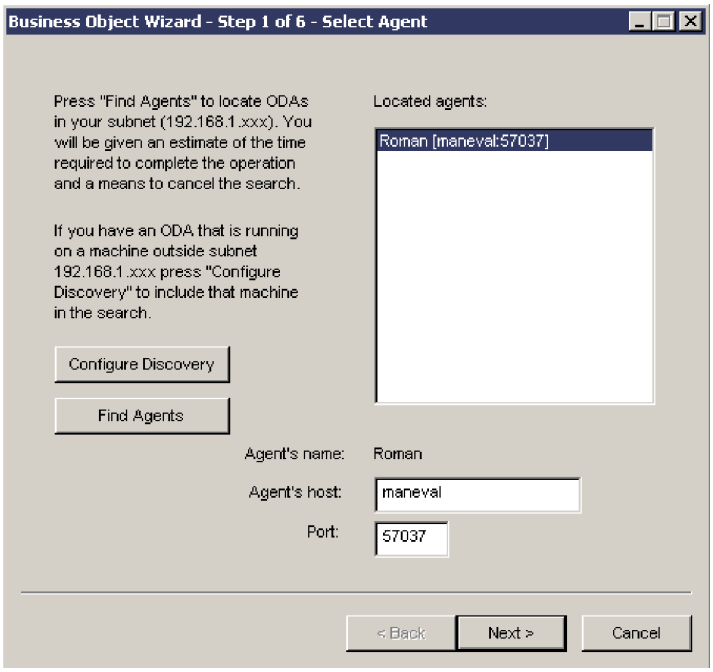


図 34. 「エージェントの選択」ダイアログ・ボックス

3. ビジネス・オブジェクト・ウィザードの接続先 ODA を、次のいずれかの操作で選択します。
   a. **「エージェントの検索」**をクリックし、**「検索されたエージェント」**リストに現在実行中の ODA (それぞれの始動スクリプトを使用して開始されたエージェント) を表示します。

      注: 必要な ODA がビジネス・オブジェクト・ウィザードで検出されない場合は、その ODA が開始済みかどうかを確認してください。

      ビジネス・オブジェクト・ウィザードは、ODA の始動スクリプトまたはバッチ・ファイルの `AGENTNAME` 変数に指定されている名前を基に、実行されている ODA のそれぞれを識別します。ここでのサンプル ODA の名前は `Roman` です。
   b. **「検索されたエージェント」**リストから必要な ODA を選択します。ビジネス・オブジェクト・ウィザードが選択内容を**「エージェント名」**として表示します。また、ODA のホスト名とポート番号を指定して、ODA を検索することもできます。
4. **「次へ」** をクリックします。ビジネス・オブジェクト・ウィザードは、指定された ODA への接続を試みます。 ODA が開始済みの場合、ビジネス・オブジェクト・ウィザードは、ODA への接続時に、図 35 に示す状況ウィンドウを表示します。

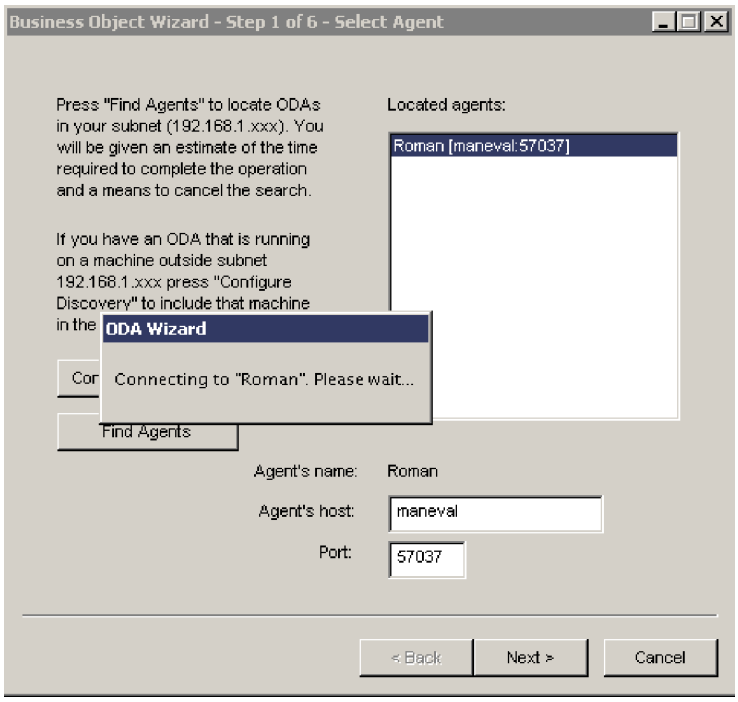


図 35. ODA への接続

5. ビジネス・オブジェクト・ウィザードが ODA に接続すると、ウィザードの 2 番目のダイアログ・ボックス (「エージェントの構成」) が表示されます。このダイアログを図 36 に示します。このダイアログ・ボックスには、データ・ソースへのアクセスおよび ODA の初期化に必要な ODA 構成プロパティーが表示されます。

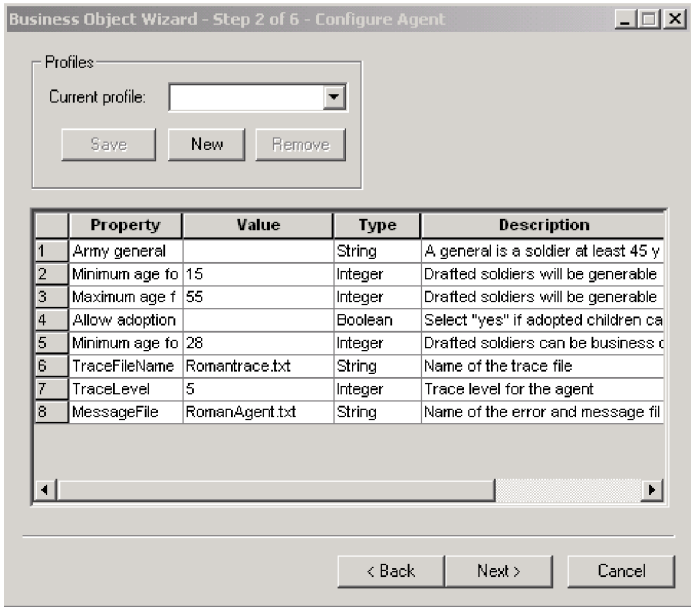


図 36. 「エージェントの構成」ダイアログ・ボックス

6. ODA の構成値を指定するか、プロファイルを選択して以前に保管した値を表示します。 ODA の必須構成領域の 1 つに、ロギングとトレースをセットアップするための領域があります。詳しくは、 88 ページの『ロギングとトレースの設定』 を参照してください。

   各 ODA を最初に使用するときには、その ODA の各構成プロパティーの値を指定する必要があります。この後、**「保管」**をクリックすることにより、指定した名前のプロファイルにプロパティー値を保管することができます。次に同じ ODA を使用するときには、保管済みのプロファイルを**「プロファイルを選択」**ボックスから選択できます。詳しくは、88 ページの『値の入力とプロファイルの保管』を参照してください。

7. **「次へ」**をクリックします。ビジネス・オブジェクト・ウィザードに、ウィザードの 3 番目のダイアログ・ボックス (「ソースの選択」) が表示されます。このダイアログ・ボックスを図 37 に示します。「ソースの選択」ダイアログ・ボックスには、ソース・*ノード階層* が表示されます。これは、トップレベルのオブジェクトが最上位に配置され、その下に子オブジェクトが配置される、ツリー構造です。通常、「ソースの選択」ダイアログ・ボックスの初期表示には、トップレベルのソース・ノードのみが表示されます。

重要

「次へ」をクリックしても ODA が動作しない場合は、`MessageFile` 構成プロパティーに指定されている ODA メッセージ・ファイルが *ProgramDir*¥ODA¥messages ディレクトリーにあるかどうかを確認してください。ここでのサンプル ODA の場合、このメッセージ・ファイルのデフォルトの名前は `RomanAgent.txt` です。詳しくは、 90 ページの『ODA メッセージ・ファイルの指定』 を参照してください。

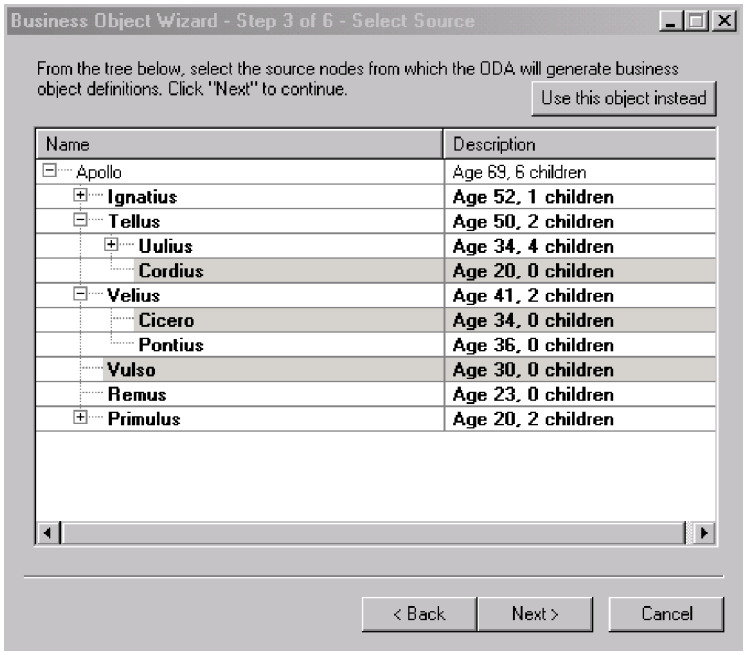


図 37. 「ソースの選択」ダイアログ・ボックスの初期表示

ソース・ノード階層のノードになるのは、テーブル名、ビジネス・オブジェクト名、スキーマ、または関数です (ODA のデータ・ソースによって異なります)。ここでのサンプル ODA は、`RomanArmy.xml` という名前の XML ファイルに含まれるオブジェクトから、ノードを生成します。図 37 に、`Army general` 構成プロパティー (80 ページの図 36 を参照) に指定されているローマ軍将官に対応する、単一のトップレベルのソース・ノードを示します。

8. ODA によって生成するビジネス・オブジェクト定義の対象になるオブジェクトを、ソース・ノード階層から選択します。ソース・ノードを 1 つだけ選択するには、そのノードの名前をクリックします。ノードを追加で選択するには、`Ctrl` キーを使用します。図 38 では、いくつかのソース・ノードが展開され、XML オブジェクトに対応する 3 つのソース・ノードが選択されています。

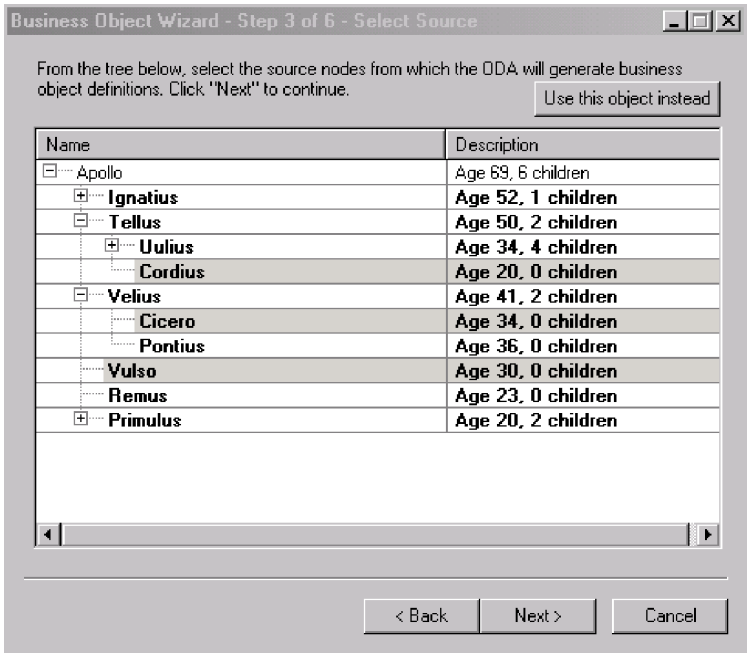


図 38. ソース・ノードが展開および選択された「ソースの選択」ダイアログ・ボックス

ソース・ノードを展開してその子ノードを表示するには、次のいずれかの操作を行います。

- ノード名の左側にある + 記号をクリックする。
- ノード名を右クリックします。ビジネス・オブジェクト・ウィザードは、図 39 に示すポップアップ・メニューを表示します。選択されているノードを展開するには、**「すべての項目を検索」**をクリックします。ビジネス・オブジェクト・ウィザードに、ソース・ノードの次のレベル、つまり展開した親ノードの子ノードが表示されます。同様の手順でさらに下のレベルを表示できます。

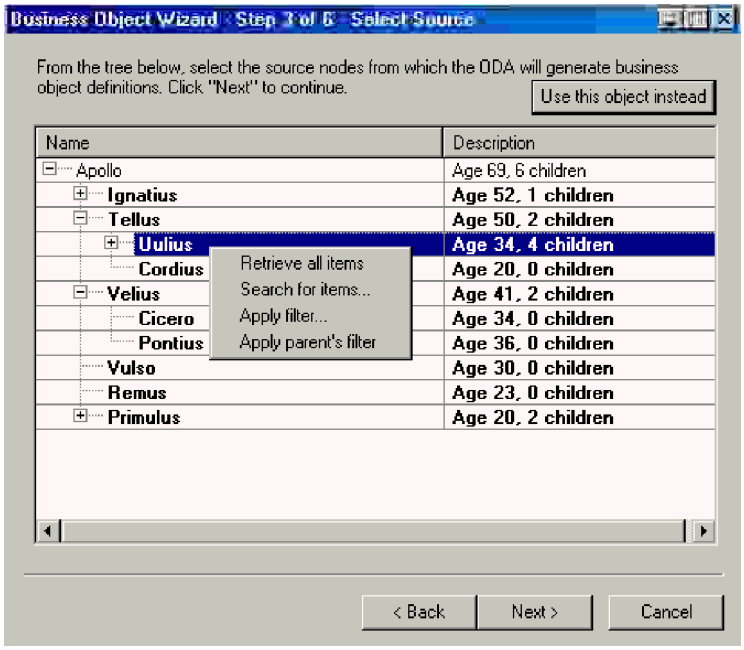


図 39. ノードの右クリック

注: ビジネス・オブジェクト・ウィザードには、ソース・ノード階層のノード間を移動できる仕組みが、ほかにもいくつか用意されています。詳しくは、92 ページの『ソース・ノード階層内での移動』を参照してください。

9. 生成するビジネス・オブジェクト定義の対象になるソース・ノードを選択したら、**「次へ」**をクリックします。ビジネス・オブジェクト・ウィザードに、ウィザードの 4 番目のダイアログ・ボックス (「ソース・ノードの確認」) が表示されます。このダイアログ・ボックスを図 40 に示します。このダイアログ・ボックスでは、どのソース・ノードを選択したかを確認できます。選択したソース・ノードは、太字で表示されます。図 40 では、**Cordius**、**Cicero**、および **Vulso** に対応するソース・ノードが選択されています。

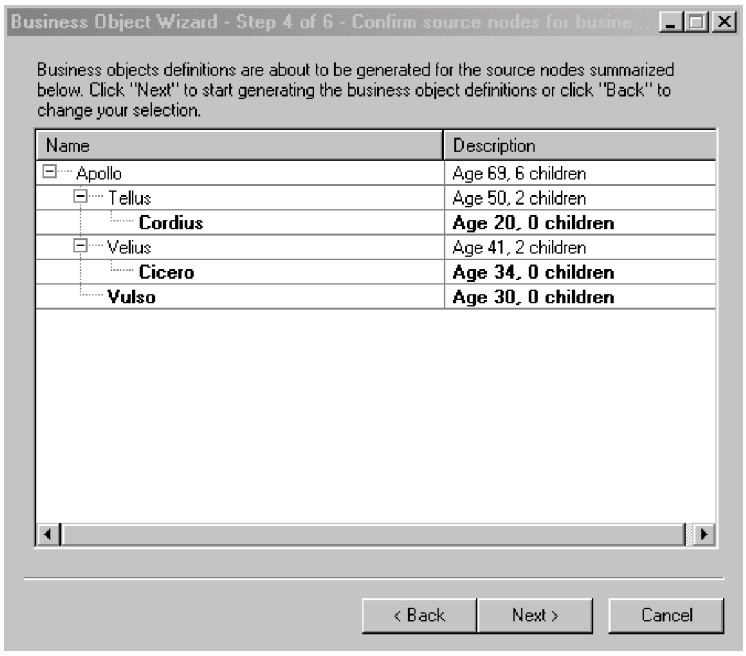


図 40. 生成するビジネス・オブジェクト定義の対象になるオブジェクトの確認

選択に誤りがあった場合には、**「戻る」**をクリックして直前のダイアログ・ボックスに戻り、必要な変更を加えます。

10. 選択が正しい場合には、**「次へ」**をクリックします。ビジネス・オブジェクト・ウィザードに、ウィザードの 5 番目の画面 (「ビジネス・オブジェクトの生成中」) が表示されます。この画面を図 41 に示します。この画面は、ODA がビジネス・オブジェクト定義を生成中であることをユーザーに通知します。

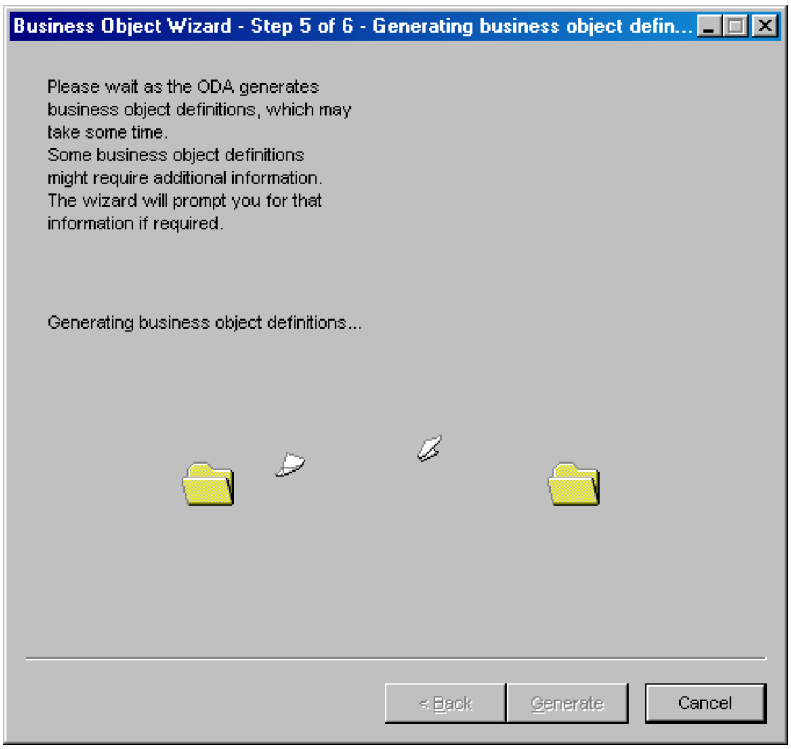


図 41. 定義の生成

ODA に追加情報を指定する必要がある場合、ビジネス・オブジェクト・ウィザードは「BO プロパティー」ダイアログ・ボックスを表示し、ユーザーにその情報を入力するように促します。ただし、ここでのサンプル ODA には、追加情報は必要ありません。「BO プロパティー」ダイアログ・ボックスについて詳しくは、96 ページの『追加情報の指定』を参照してください。

11. ODA によるビジネス・オブジェクト定義の生成が完了すると、ビジネス・オブジェクト・ウィザードにはウィザードの最後のダイアログ・ボックス (「ビジネス・オブジェクトの保管」) が表示されます。このダイアログ・ボックスを図 42 に示します。このダイアログ・ボックスには、ODA によって生成されたビジネス・オブジェクト定義を保管できる、以下のオプションが用意されています。
    - ビジネス・オブジェクト定義を ICL ベースのプロジェクトに保管します (Business Object Designer Express を System Manager から実行している場合)。
    - ビジネス・オブジェクト定義をファイルに保管します (InterChange Server Express の場合)。
    - Business Object Designer Express でビジネス・オブジェクト定義を編集用に開きます。
    - ODA をシャットダウンします。

重要

ODA がビジネス・オブジェクト定義を生成するときに、キー要素が特定されていない データ・ソース・オブジェクトが生成元のオブジェクトになっていると、生成されたビジネス・オブジェクト定義はキー属性を持ちません。どのビジネス・オブジェクトも、少なくとも *1* つのキー を持っていなければなりません。キーを持たないビジネス・オブジェクト定義が ODA によって生成されたおそれがある場合には、ビジネス・オブジェクト定義を保管する代わりに、「別のウィンドウで新規ビジネス・オブジェクトを開く」オプションを選択するようにします。Business Object Designer Express では、各ビジネス・オブジェクト定義にキー属性があるかどうかを検証して、ない場合にはキー属性を追加することができます。Business Object Designer Express では、キーを持たないビジネス・オブジェクト定義は保管できません。

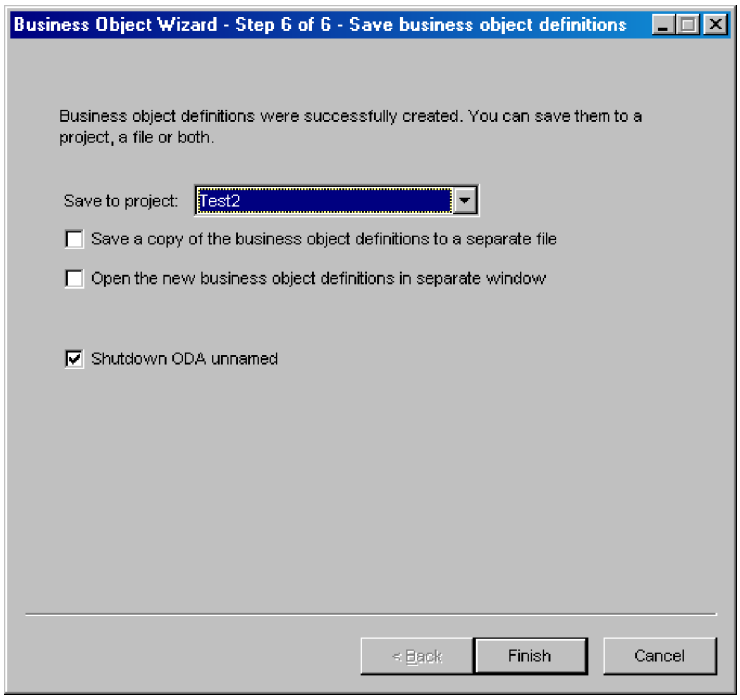


図 42. ビジネス・オブジェクト定義の保管

「完了」 をクリックしてビジネス・オブジェクト定義を保管します。または、**「キャンセル」** をクリックし、定義を保管しないで終了します。どちらの場合も、ビジネス・オブジェクト・ウィザードは ODA から切断されます。このダイアログ・ボックスには、ビジネス・オブジェクト・ウィザードに、ODA からの切断後 ODA をシャットダウンさせるオプションもあります。ODA を引き続き使用する必要がない場合は、このオプションを選択します。

ビジネス・オブジェクト定義をファイルに保管するオプションを選択した場合、**「完了」**をクリックすると、参照ウィンドウが開き、そのファイルの名

前、保管場所、および使用するフォーマット (テキスト・ファイルまたは InterChange Server Express 固有のフォーマット) を指定できます。

以上で、Object Discovery Agent を使用することによりビジネス・オブジェクト定義が正常に作成されました。

値の入力とプロファイルの保管

ODA の構成値のセットをプロファイルに保管して、その ODA を将来使用するときに利用できるようにすることができます。プロファイルを保管するには、以下の手順を実行します。

1. ステップ 2 の、ビジネス・オブジェクト・ウィザードの「エージェントの構成」ダイアログ・ボックスで、**「プロファイル」**の下の**「新規」**ボタンをクリックします。

   注: 既存のプロファイルを基にして新しいプロファイルを作成する場合には、プロファイルのリスト (ドロップダウン・リスト) から、目的のプロファイルを探し出します。「新規」ボタンはクリックしないでください。

2. **「現在」**リストにプロファイルの名前を入力します (80 ページの図 36 を参照)。

   注: 既存のプロファイルを基にして新しいプロファイルを作成する場合には、プロファイルのリスト (ドロップダウン・リスト) にある既存のプロファイルの名前を上書きします。

3. **「エージェントの構成」** ダイアログのテーブルに、適切な構成値を入力します。
4. **「保管」**ボタンをクリックします。

   ビジネス・オブジェクト・ウィザードによって、プロファイルが次のディレクトリーに保管されます。

   ```
   C:¥Documents and Settings¥All Users¥Application Data¥CrossWorlds¥
      BusObjDesigner¥profiles.bod
   ```

ロギングとトレースの設定

ODA を構成する際には、ロギングとトレースをセットアップしなければなりません。ODA のロギングとトレースに関する情報は、ビジネス・オブジェクト・ウィザードの「エージェントの構成」ダイアログ・ボックスで指定します。ビジネス・オブジェクト・ウィザードには、表 14 に示す ODA の標準構成プロパティーが必ず表示されます。

表 14. ODA の標準構成プロパティー

プロパティー名	プロパティーの型	説明
TraceFileName	String	ODA がトレース情報を書き込む先のファイルを指定。詳しくは、89 ページの『トレース・ファイルとトレース・レベルの指定』を参照してください。
TraceLevel	Integer	ODA に適用されるトレース・レベル。詳しくは、89 ページの『トレース・ファイルとトレース・レベルの指定』を参照してください。

表 14. ODA の標準構成プロパティー (続き)

プロパティー名	プロパティーの型	説明
MessageFile	String	ODA のエラー/メッセージ・ファイルの名前。このプロパティーを使用して、既存のファイルの確認や指定をします。詳しくは、90 ページの『ODA メッセージ・ファイルの指定』を参照してください。

このセクションの内容は次のとおりです。

- 『トレース・ファイルとトレース・レベルの指定』
- 90 ページの『ODA メッセージ・ファイルの指定』

トレース・ファイルとトレース・レベルの指定

図 43 は、ビジネス・オブジェクト・ウィザードの「エージェントの構成」ダイアログ・ボックスを示しています。トレース・ファイルの名前とトレース・レベルの指定は、このダイアログ・ボックスで行います。

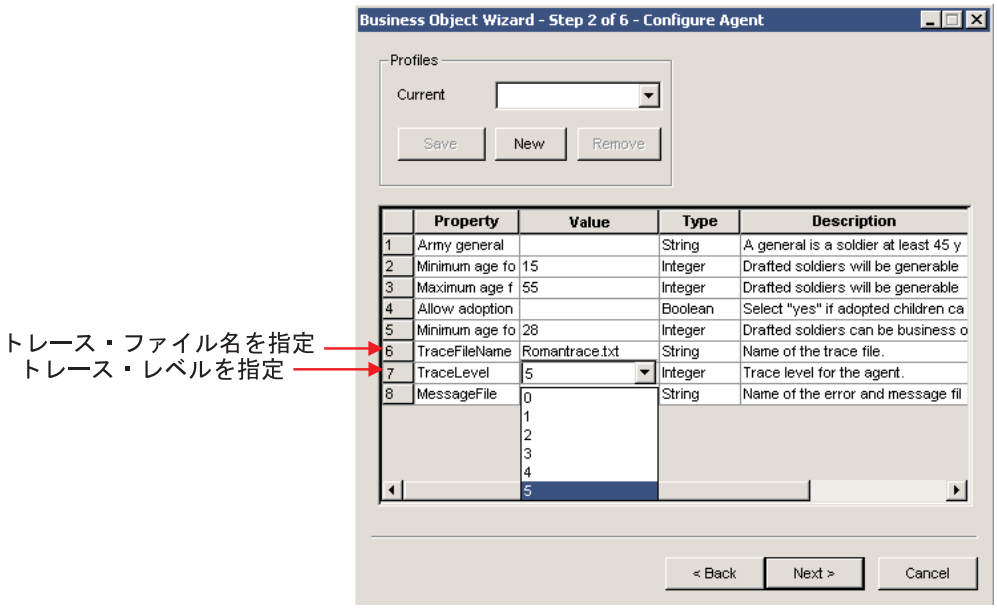


図 43. トレース情報の指定

トレース・ファイルの指定: TraceFileName 構成プロパティーは、ODA のトレース・ファイル の名前を指定します。このファイルは、ODA が記録するすべての トレース・メッセージおよびエラー・メッセージの記録先になります。デフォルトでは、トレース・ファイルには次の命名規則に基づく名前が ODA ランタイムによって付けられます。

*ODAname*trace.txt

ここで、*ODAname* は、ODA を一意的に識別できる名前です。詳しくは、 185 ページの『ODA の名前付け』 を参照してください。例えば、ODA の名前が HTMLODA である場合、HTMLODAtrace.txt という名前のトレース・ファイルが生成されます。

注: ODK API では、1 つのメソッドでトレース・メッセージとエラー・メッセージの*両方* を記録します。そのため、ODA はこれらの 2 種類のメッセージを 1 つのファイルに記録します。したがって、このファイルは、トレース・ファイルという名前でありながら、ODA が生成したエラー・メッセージも含むことになります。

指定されたトレース・ファイルが存在しない 場合、ODA は、そのファイルを ODA のランタイム・ディレクトリー (製品ディレクトリーの ODA¥*srcDataName* サブディレクトリー) 内に作成します。指定されたトレース・ファイルが既に存在する場合、ODA はそのファイルに追記します。ODA を構成するときに TraceFileName プロパティーを再設定することで、別のトレース・ファイル名を指定することができます。

トレース・レベルの設定: TraceLevel 構成プロパティーは、ODA のシステム・トレース・レベル を指定します。 ODA のトレース・メソッドは、メッセージのトレース・レベルがこのシステム・トレース・レベルよりも低いか等しい場合に、指定されたメッセージをトレース・ファイルに送信します。したがって、記録されたトレース・メッセージに含まれる情報の詳しさは、システム・トレース・レベルによって決まります。トレース・レベルとそれに対応する動作を表 15 に示します。

表 15. トレース・レベル

レベル	動作
0	指定されたトレース・ファイルにエラー・メッセージを書き込みます。
1	メソッドが開始された場合は必ずトレースします。各ビジネス・オブジェクト定義に対応する状況メッセージおよびキー情報の収集に有用です。
2	エージェント・プロパティーおよび受け取った値のトレースをとります。
3	• ビジネス・オブジェクトの名前のトレースをとります。 • ビジネス・オブジェクト・プロパティーおよび受け取った値のトレースをとります。
4	• すべてのスレッドの作成のトレースをとります。 • メソッドの開始と終了に関するメッセージのトレースをとります。
5	• Object Discovery Agent の初期状態を表し、すべての Object Discovery Agent プロパティーを対象として検索した値をログに記録します。 • Object Discovery Agent により作成された各スレッドの詳細状況のトレースをとります。 • ビジネス・オブジェクト定義ダンプのトレースをとります。

ODA でのトレース・メッセージの生成方法については、175 ページの『トレース・メッセージとエラー・メッセージの処理』を参照してください。

ODA メッセージ・ファイルの指定

MessageFile 構成プロパティーは、ODA のメッセージ・ファイル の名前を指定します。 ODA のエラー・メッセージとトレース・メッセージは、この ODA メッセージ・ファイルに格納しておくことができます。ODA 自体にメッセージ・テキストを作成する代わりに、ファイルに格納したメッセージをメッセージ番号で検索でき

ます。メッセージをメッセージ・ファイル内に分離することで、ODA が使用される可能性があるさまざまなロケールの言語に、ODA メッセージを容易に翻訳できるようになります。

デフォルトでは、メッセージ・ファイルには次の命名規則に基づく名前が ODA ランタイムによって付けられます。

*ODAname*Agent.txt

ここで、*ODAname* は、ODA を一意的に識別できる名前です。詳しくは、185 ページの『ODA の名前付け』 を参照してください。例えば、ODA の名前が HTMLODA である場合、MessageFile プロパティーの値は、デフォルトでは HTMLODAAgent.txt になります。メッセージ・ファイルは、次のメッセージ・ファイル・ディレクトリー内になければなりません。

ProductDir¥ODA¥messages

重要

ODA は、指定されたメッセージ・ファイルが存在しない 場合や、メッセージ・ファイル・ディレクトリー内にない場合には、実行時例外を生成します。ODA の実行に進む前に、MessageFile に指定されているメッセージ・ファイルが存在することを確認してください。

別のファイルを ODA のメッセージ・ファイルとして使用する場合は、MessageFile プロパティーを設定して、別のトレース・ファイル名を指定してください。

米国英語以外のロケールを使用している場合は、ファイル名にロケールの名前を含む ODA メッセージ・ファイルをビジネス・オブジェクト・ウィザードが自動的に参照します。

*ODAname*Agent_*locale*.txt

locale の書式は「*ll_TT*」です。*ll* は 2 文字の言語名 (小文字)、*TT* は 2 文字の国名または地域名 (大文字) です。例えば、HTMLODA という名前の ODA のメッセージ・ファイルが日本語ロケール向けにローカライズされると、そのメッセージ・ファイルの名前は次のようになります。

HTMLODAAgent_ja_JP.txt

注: 米国英語以外のロケールにログインしているときに、MessageFile プロパティーに米国英語以外の名前を指定する必要はありません。例えば HTML ODA を使用している場合、MessageFile を米国英語のファイル名 (HTMLODAAgent.txt) に設定します。日本語ロケールにログインしている場合、ビジネス・オブジェクト・ウィザードは、日本語ロケールの正しいメッセージ・ファイルである HTMLODAAgent_ja_JP.txt を探し出します。

ODA スクリプト・ファイルまたはバッチ・ファイルのインスタンスを複数作成し、各インスタンスに対応する ODA に固有の名前を指定した場合には、*それぞれの* ODA インスタンスごとにメッセージ・ファイルを持つことができます。詳しくは、97 ページの『複数の ODA の同時使用』 を参照してください。

ソース・ノード階層内での移動

ビジネス・オブジェクト・ウィザードの「ソースの選択」ダイアログ・ボックスには、ソース・ノード階層のノード間での移動のための以下の仕組みが用意されています。

- 『子ノードの表示の制限』
- 94 ページの『オブジェクトのパスの指定』
- 95 ページの『オペレーティング・システム・ファイルの関連付け』

子ノードの表示の制限

ステップ 8 (82 ページ) に示したソース・ノードの展開方法は、展開可能なノードのすべての 子ノードを表示する方法を説明したものです。表示するオブジェクトを制限するには、ノード名を右クリックすると表示される以下のいずれかのメニュー・アイテムを使用します (84 ページの図 39 を参照)。

- **フィルターを適用**
- **項目を検索**

フィルターの使用: **「フィルターを適用」**メニュー項目を使用すると、現在選択されているソース・ノードの中で開かれるものを制限できるフィルター を指定できます。このメニュー項目をクリックすると、ビジネス・オブジェクト・ウィザードは、図 44 に示す「ノードへのフィルターの適用」ダイアログ・ボックスを表示します。

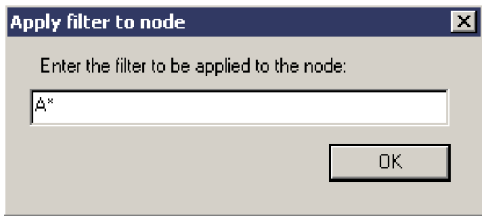


図 44. 表示内容を制限するためのフィルターの指定

フィルター条件のテキストには、任意の数 (0 個を含む) の文字を表すワイルドカードとして、アスタリスク文字 (*) を使用できます。このワイルドカード文字は、必要に応じて、どの位置にも、何箇所でも指定できます。例えば、SAP*、*SAP、*SAP*、または *S*AP* といった指定が可能です。

「OK」をクリックすると、ビジネス・オブジェクト・ウィザードは、親ノードに属する子ノードのうち現在取得済みのものの中から、名前がフィルター条件のテキストに一致するものを検索します。その親ノードを展開すると、名前がこのテキストに一致する子ノードだけが表示されます。

重要: ビジネス・オブジェクト・ウィザードは、フィルターを受け付けると、現在取得済みのソース・ノード内で、親ノードに属する子ノードのうち条件に一致するものを検索します。つまり、条件に一致する子ノードをデータ・ソース内で検索するわけではありません。ビジネス・オブジェクト・ウィザードにデータ・ソース内を検索させるには、検索パターンを指定します。詳しくは、 93 ページの『検索パターンの指定』 を参照してください。

例えば、サンプル Roman ODA の場合、Uulius ノードには、Ares、Cronus、Atlas、Metis の 4 つの子ノードがあります。図 44 に示したフィルター (「A*」) を Uulius ノードに適用した場合、ビジネス・オブジェクト・ウィザードでこのノードを展開すると、図 45 のように表示されます。

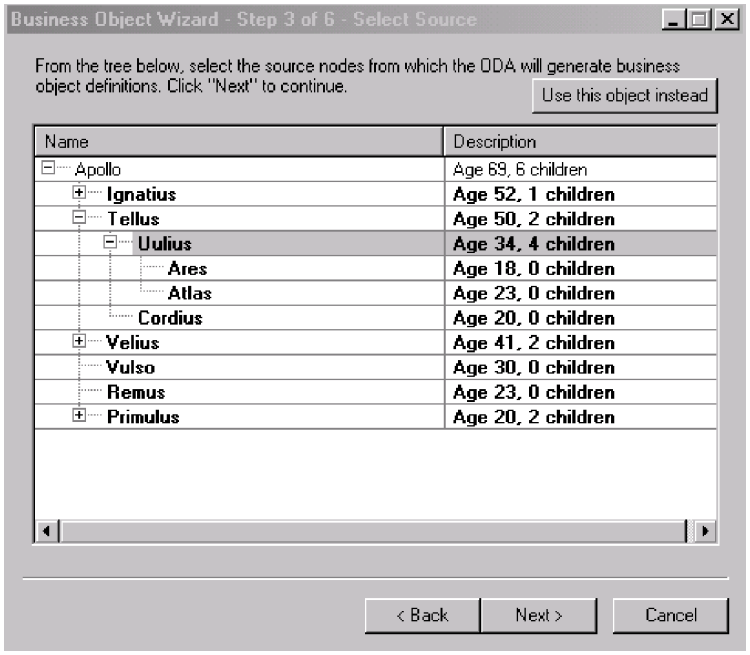


図 45. 展開されたフィルター適用済みノード

ノードのトップでフィルターを指定してからそのノードを展開した場合、そのノードを右クリックして**「親のフィルターを適用」**をクリックすると、子オブジェクトに同じフィルターを適用することができます。**「すべての項目を検索」**メニュー項目をクリックした場合には、親ノードのフィルターがすべての要素に適用されます。

検索パターンの指定: **「項目を検索」** メニュー項目を使用すると、ビジネス・オブジェクト・ウィザードがデータ・ソースから選択するソース・ノードを絞り込む、*検索パターン* を指定することができます。**「項目を検索」**をクリックすると、ビジネス・オブジェクト・ウィザードによって「検索パターンの入力」ダイアログ・ボックスが表示されます。 94 ページの図 46 に、このダイアログ・ボックスを示します。

注: ODA が検索パターン機能をサポートしていなければ、**「項目を検索」** メニュー項目は使用可能になりません。このメニュー項目が選択不可になっている場合、ODA は検索パターンをサポートしていません。

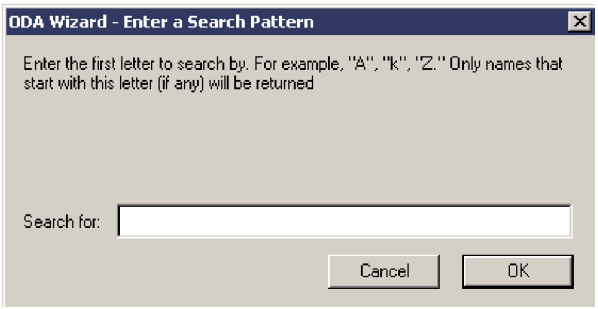


図 46. 検索結果を絞り込むための検索パターンの指定

「検索パターンの入力」ダイアログ・ボックスには、検索パターンによる検索の条件についての説明が表示されます。図 46 のダイアログ・ボックスのテキストは、検索パターンに含めることができるのは 1 文字であることを示しています。ODA が示す検索基準の説明はカスタマイズ可能です。入力する検索パターンは、説明されている検索基準に適合したものにしてください。適合していない場合、ODA は例外をスローします。

「OK」をクリックすると、ビジネス・オブジェクト・ウィザードは、親ノードに属する子ノードのうち名前が検索パターンに一致するものを、データ・ソース内で検索します。その親ノードを展開すると、名前がこのパターンに一致する子ノードだけが表示されます。

重要: ビジネス・オブジェクト・ウィザードは、検索パターンを受け付けると、データ・ソース内で、親ノードに属する子ノードのうち条件に一致するものを検索します。つまり、データ・ソースからツリー・ノードを新しく取得します。単に現在取得済みのツリー・ノード内で条件に一致する子ノードを検索するのではありません 。ビジネス・オブジェクト・ウィザードに現在取得済みのツリー・ノード内を検索させるには、フィルターを指定します。詳しくは、 92 ページの『フィルターの使用』 を参照してください。

オブジェクトのパスの指定

ソース・ノード階層内を移動する代わりに、目的のオブジェクトの正確なパスを指定することもできます。このためには、「ソースの選択」ダイアログ・ボックスの右上にある**「代わりにこのオブジェクトを使用」** をクリックします。ビジネス・オブジェクト・ウィザードが「オブジェクトのパス」ダイアログ・ボックス (図 47) を表示します。このダイアログ・ボックスではパスを指定します。

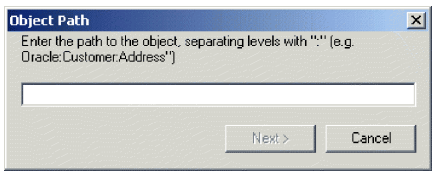


図 47. 「オブジェクトのパス」の指定

オブジェクトのパスとしては、ソース・ノードの完全修飾パス (トップレベルの親ノードから目的のノードまでを含むパス) を指定します。このパス内のノード名は、コロン (:) で区切られています。

オペレーティング・システム・ファイルの関連付け

オペレーティング・システム・ファイルをソース・ノード階層の現在のノードに関連付けるには、ノードを右クリックし、**「関連付けられたファイル」**をクリックします (図 48 を参照)。ファイルをソース・ノードに関連付けると、ODA は、ODA のデータ・ソースを使用しないで、そのファイルをそのソース・ノードのデータのソースとして使用します。

注: ODA がファイルを関連付ける機能をサポートしていなければ、**「関連付けられたファイル」**メニュー項目は使用可能になりません。このメニュー項目が選択不可になっている場合、ODA は、ファイルを現在のソース・ノードと関連付ける機能をサポートしていません。

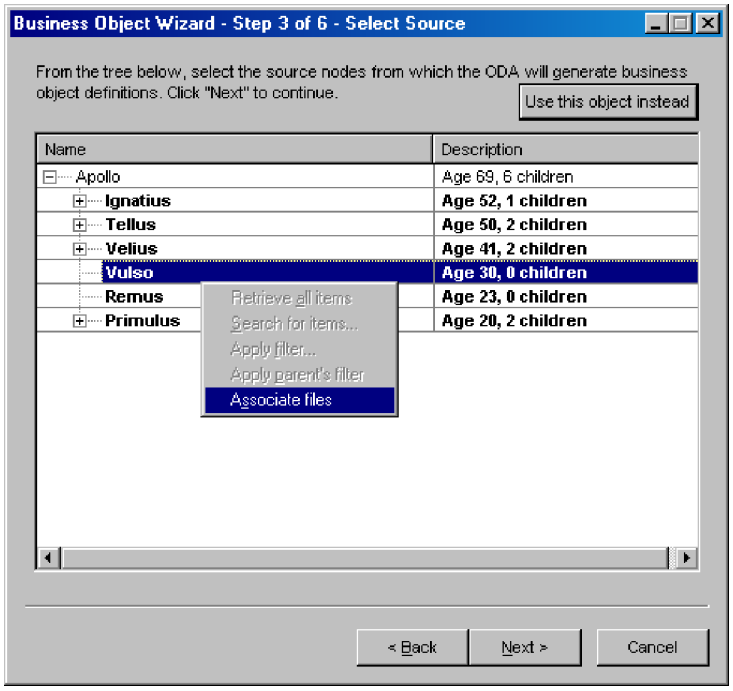


図 48. ファイルとソース・ノードの関連付け

「関連付けられたファイル」メニュー項目をクリックすると、ビジネス・オブジェクト・ウィザードは、図 49 に示す「開く」ウィンドウを表示します。このウィンドウからファイル構造を参照し、現在のノードに関連付けるファイルを選択することができます。

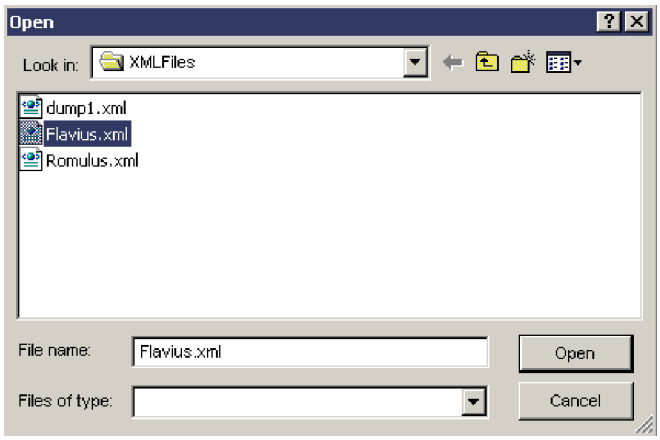


図 49. 関連付けるファイルを選択するための「開く」ウィンドウ

ソース・ノードに関連付けるファイルを選択してから、**「開く」**をクリックします。ビジネス・オブジェクト・ウィザードが「ソースの選択」ダイアログ・ボックスに制御を戻すと、図 50 に示すように、選択したファイルが、そのファイルを関連付けたソース・ノードの下に表示されます。

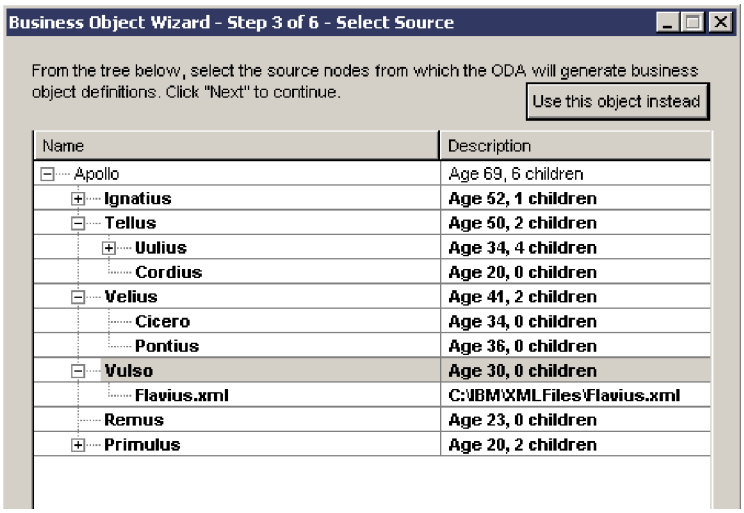


図 50. ソース・ノードに関連付けられたファイル

追加情報の指定

ODA に追加情報を指定する必要がある場合、図 51 に示すように、ステップ 5 (「ビジネス・オブジェクトの生成中」) で「BO プロパティー」ダイアログ・ボックスが開きます。

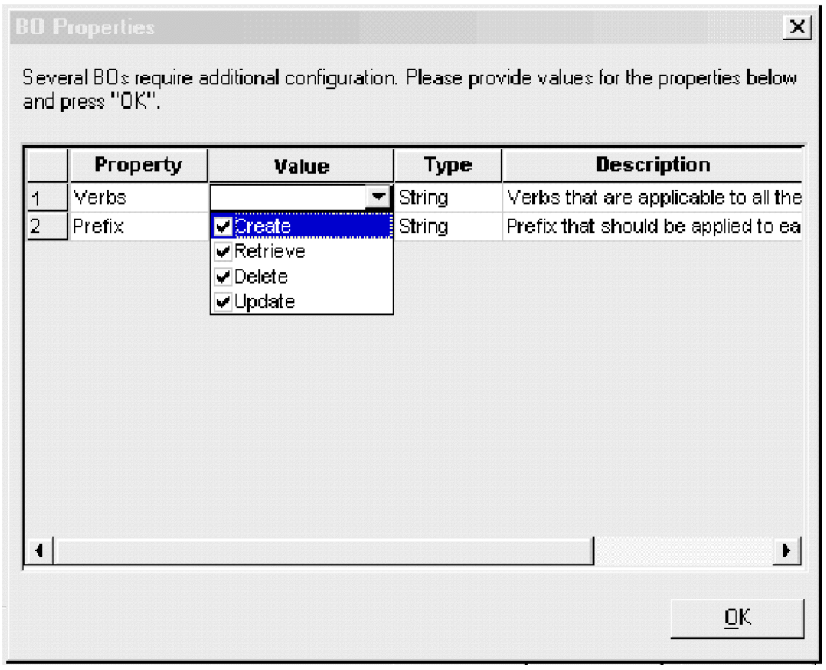


図 51. 追加情報の指定

注: 「BO プロパティー」ダイアログ・ボックスのセルに複数の値が設定されている場合、このダイアログ・ボックスを初めて開いたときは、このセルには何も表示されません。値のリストを表示するには、セルをクリックします。

「BO プロパティー」ダイアログ・ボックスで必要な情報をすべて指定したら、**「OK」**をクリックします。ODA によるビジネス・オブジェクト定義の生成の続きが行われます。

複数の ODA の同時使用

ローカル・ホスト・マシンまたはリモート・ホスト・マシンのいずれかで ODA の複数のインスタンスを実行できます。各インスタンスは固有のポートで実行されます。このポート番号は、ビジネス・オブジェクト・ウィザード内から各 ODA を開始するときに指定できます。

Business Object Designer で複数の Object Discovery Agent を同時に実行するには、以下の手順を実行します。

1. 各 Object Discovery Agent を、それぞれの `start_ODAname.bat` または `start_ODAname.sh` ファイルを実行することにより、開始します。
2. Business Object Designer を開きます。
3. **「ファイル」->「ODA を使用して新規作成」**をクリックします。

   ビジネス・オブジェクト・ウィザードの最初のダイアログ・ボックス (「エージェントの選択」) が開きます (79 ページの図 34 を参照)。
4. **「エージェントの検索」**ボタンをクリックし、**「検索されたエージェント」**リストに現在実行されている ODA を表示させます。ホスト名およびポート番号を使用して ODA を検索することもできます。

5. 表示されたリストから最初の ODA を選択します。選択内容が**「エージェント名」**としてリストされます。
6. 再び**「ファイル」->「ODA を使用して新規作成」**をクリックします。
7. **「エージェントの検索」**ボタンをクリックし、**「検索されたエージェント」**リストに現在実行されている ODA を表示させるか、ホスト名およびポート番号を使用して ODA を検索します。
8. 表示されたリストから 2 番目の ODA を選択します。
9. 78 ページの『ODA を使用したビジネス・オブジェクト定義の作成』 のステップ 4 の説明に従って、各 ODA の構成を続行します。

ODA スクリプト・ファイルまたはバッチ・ファイルのインスタンスを複数作成し、各インスタンスに対応する ODA に固有の名前を指定した場合には、*それぞれ*の ODA インスタンスごとにメッセージ・ファイルを持つことができます。異なる名前の付いた ODA インスタンスが複数存在しても、メッセージ・ファイルは共通にすることも可能です。有効なメッセージ・ファイルを指定する方法は 2 つあります。

- ODA の名前を変更した場合、それに対応するメッセージ・ファイルを新しく作成*しない* ときは、ビジネス・オブジェクト・ウィザードで ODA を構成する際にメッセージ・ファイルの名前を変更する必要があります。ビジネス・オブジェクト・ウィザードはメッセージ・ファイルの名前を指定しますが、実際にファイルを作成するわけではありません。ODA 構成の一部として表示されたファイルが存在しない場合には、既存のファイルを指すように値を変更してください。
- 特定の ODA に対応する既存のメッセージ・ファイルをコピーし、必要に応じて変更することもできます。ビジネス・オブジェクト・ウィザードは、各ファイルが命名規則に従って命名されることを前提としています。例えば、ODA 始動スクリプトに含まれる `AGENTNAME` 変数に `HTMLODA` と指定されている場合、このツールでは、関連するメッセージ・ファイルは `HTMLODAAgent.txt` であると見なされます。したがって、ビジネス・オブジェクト・ウィザードが、確認のために ODA 構成の一部としてファイル名を表示した場合、このファイル名は ODA 名が基本になっています。デフォルトのメッセージ・ファイルが正しく命名されていることを確認し、必要ならば訂正してください。

第 2 部 Object Discovery Agent の開発

第 5 章 Object Discovery Agent の開発

この章では、Object Discovery Agent Development Kit (ODK) API に定義されているクラスを使用して Object Discovery Agent (ODA) を開発する方法について説明します。 ODA は、Business Object Designer Express のビジネス・オブジェクト・ウィザードと連携して、特定のアプリケーション、データベース、またはファイル・システムに対して作用する特定のコネクターまたはデータ・ハンドラー向けのビジネス・オブジェクト定義を作成します。

この章の主な内容は次のとおりです。

ODA の実行

実行時に、ODA の実行には次のコンポーネントが関与します。

- Business Object Designer Express。ODA と対話するためのグラフィカル・インターフェースをウィザード形式で提供します。ビジネス・オブジェクト・ウィザード。コンテンツ生成のために ODA が必要とする情報を取得するための一連のダイアログ・ボックスを表示します。
- *ODA ランタイム*。ビジネス・オブジェクト・ウィザードと ODA の間に位置する中間コンポーネントです。 ODK API のクラスと ODK インフラストラクチャーを使用して ODA と通信を行います。ODA 始動スクリプトで開始するのは、ODA ランタイムです。
- *ODA*。データ・ソース内のソース・ノードを「検出」して、コンテンツを生成するコンポーネントです。 ODA は、ビジネス・オブジェクト・ウィザードのダイアログ・ボックスで情報を ODA ランタイムから受け取ります。次に ODA は、情報 (生成されたコンテンツなど) を ODA ランタイムを経由してビジネス・オブジェクト・ウィザードに送ります。

図 52 は、ODA ランタイム・アーキテクチャーのコンポーネントを示しています。

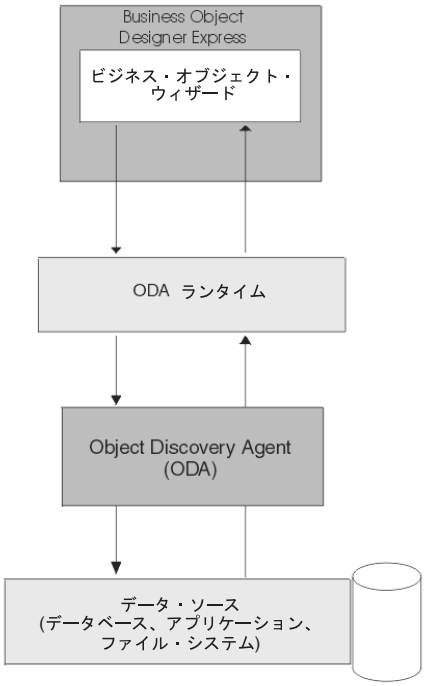


図 52. Object Discovery Agent のアーキテクチャー

ビジネス・オブジェクト定義を生成するために、ODA は次のステップを実行する必要があります。

1. データ・ソース (アプリケーション、データベース、またはファイル・システムなど) に接続するために ODA が必要とする ODA 構成プロパティーの値 (ユーザー名やデータベース・タイプなど) を取得します。
2. これらの構成プロパティーを使用してデータ・ソースに接続します。
3. ビジネス・オブジェクト定義を生成するソース・ノードのリストを取得します。
4. (アプリケーション、データベースの表、ファイル・システム、または DTD によって定義された) ソース・ノードの基礎となるデータ・ソース・エンティティーの要件を検出します。
5. WebSphere Business Integration システムの要件、およびビジネス・オブジェクトを処理するコンポーネントの要件を満たすビジネス・オブジェクト定義を作成し、それらの定義を戻します。

注: ビジネス・オブジェクト定義の他に、ODA はファイルをコンテンツとして生成することもできます。詳しくは、106 ページの『コンテンツの生成』を参照してください。

表 16 は、ODA 実行のステップと、それを開始するビジネス・オブジェクト・ウィザードのステップを示しています。

表 16. Object Discovery Agent の実行

作業	ビジネス・オブジェクト・ウィザードのステップ	詳細情報の参照先
1. 開始する ODA を選択します。	ステップ 1: エージェントの選択	103 ページの『ODA の選択』

表 16. Object Discovery Agent の実行 (続き)

作業		ビジネス・オブジェクト・ウィザードのステップ	詳細情報の参照先
2.	ODA 構成プロパティーを取得します (開くデータ・ソースを記述する構成プロパティーも含めて)。	ステップ 2: エージェントの構成	『ODA 構成プロパティーの取得』
3.	ODA コンテンツを生成するソース・データを取得します。	ステップ 3: ソースの選択	105 ページの『ソース・データの選択および確認』
4.	選択したソース・データを確認します。	ステップ 4: ソース・ノードの確認	105 ページの『ソース・データの選択および確認』
5.	ビジネス・オブジェクト定義を生成します。	ステップ 5: ビジネス・オブジェクトの生成中	106 ページの『コンテンツの生成』
		ビジネス・オブジェクト・プロパティー	108 ページの『ビジネス・オブジェクト・プロパティーの取得』
6.	ビジネス・オブジェクト定義を保管します。	ステップ 6: ビジネス・オブジェクトの保管	110 ページの『コンテンツの保管』

ODA の選択

「ファイル」->「ODA を使用して新規作成」を選択すると、Business Object Designer Express は、ODA を実行するためにビジネス・オブジェクト・ウィザードを起動します。ビジネス・オブジェクト・ウィザードのステップ 1 では、「エージェントの選択」ダイアログ・ボックスが表示されます。このダイアログ・ボックスでは、使用可能なすべての Object Discovery Agent にグラフィックによりアクセスすることができます。このダイアログ・ボックスから、ユーザーが、実行する ODA を選択します。

ビジネス・オブジェクト・ウィザードはその ODA に接続し、次のステップを実行します。

- ODA オブジェクトをインスタンス化します。ODA オブジェクトは、*ODA クラス* のオブジェクトです。 ODA クラスは、ODA 基底クラス (`ODKAgentBase2`) を拡張したものです。ODA の振る舞いが定義されています。
- ODA オブジェクトへのハンドルを取得します。このハンドルを使用すると、開始時にそのオブジェクトにアクセスできます。

注: ODA を事前に開始しておかないと、ビジネス・オブジェクト・ウィザードは、その ODA を選択可能な ODA としてリストしません。詳しくは、 75 ページの『ODA を使用するための準備』を参照してください。

ODA クラスの作成方法については、 116 ページの『ODA 基底クラスの拡張』を参照してください。

ODA 構成プロパティーの取得

ビジネス・オブジェクト・ウィザードのステップ 2 では、「エージェントの構成」ダイアログ・ボックスが表示されます。このダイアログ・ボックスでは、ODA の構成プロパティーが表示されます。*構成プロパティー* は、ODA が実行を開始するために必要なプロパティーです。ODK API は、構成プロパティーをエージェント・プロパティー (`AgentProperty`) オブジェクトとして表します。このステップでは、ウィザードに構成プロパティーが表示されます。ユーザーがそれらの構成プロパティ

ーを更新すると、ユーザーにより初期化されたプロパティーが ODA ランタイム・メモリーに書き込まれます。

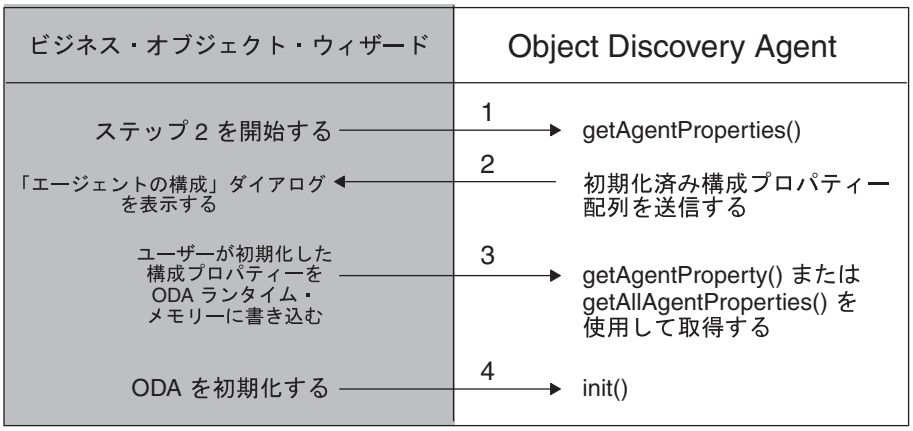


図 53. ビジネス・オブジェクト・ウィザードのエージェントの構成 (ステップ 2)

図 53 に示すように、ビジネス・オブジェクト・ウィザードは次のステップを実行します。

1. 選択された ODA から構成プロパティーを取得し、それを「エージェントの構成」ダイアログ・ボックスに表示します。

   ODA から構成プロパティーを取得する際、ウィザードは、ODA 基底クラス (ODKAgentBase2) に定義されている `getAgentProperties()` メソッドを呼び出します。このメソッドは、ODA クラスの一部として開発者が実装する必要のある抽象メソッドです。これにより、ODA の構成プロパティーが `AgentProperty` オブジェクトの配列としてビジネス・オブジェクト・ウィザードに戻されます。これらの構成プロパティーには、名前、タイプ、任意の有効な値、説明、入力上の制限、および任意のデフォルト値を含めることができます。

   `getAgentProperties()` が提供する構成プロパティーの他に、ビジネス・オブジェクト・ウィザードは常に、すべての ODA に共通の以下の標準構成プロパティーを提供します。

   - `MessageFile`
   - `TraceLevel`
   - `TraceFileName`

   詳しくは、 118 ページの『構成プロパティーの取得』を参照してください。

2. 「エージェントの構成」ダイアログ・ボックスから、ユーザーが入力または変更した構成プロパティーの値を受け入れます。ウィザードは、ユーザーが初期化した構成プロパティーを ODA に送ります。

   ビジネス・オブジェクト・ウィザードは、これらのプロパティーを ODA ランタイム・メモリーに保管します。ODA 内では、`ODKUtility` クラスのインスタンスを通じて、これらのプロパティーにアクセスできます (`ODKUtility` クラスには、プロパティーへのアクセスのために `getAgentProperty()` メソッドと `getAllAgentProperties()` メソッドが用意されています)。

3. ODA のメタデータを初期化します。ODA のメタデータには、ODA とその機能についての情報が記述されています。

   getAgentProperties() を呼び出した後、ビジネス・オブジェクト・ウィザードは ODA 基底クラス (ODKAgentBase2) の getMetaData() メソッドを呼び出します。このメソッドは、ODA クラスの一部として開発者が実装する必要のある抽象メソッドです。これにより、ODA メタデータを含む初期化済みの AgentMetaData オブジェクトが戻されます。
4. ユーザーが初期化した始動プロパティーに基づいて ODA を初期化します。

   ODA を初期化する際、ウィザードは ODA 基底クラス (ODKAgentBase2) の init() メソッドを呼び出します。このメソッドは、ODA クラスの一部として開発者が実装する必要のある抽象メソッドです。これにより、リソース割り振りやデータ・ソースへの接続作成などの初期化タスクが実行されます

この章では、ODA の初期化に必要なメソッドの実装方法に関して、次のトピックを取り上げます。

初期化メソッド	詳細情報の参照先
getAgentProperties() getMetaData() init()	118 ページの『構成プロパティーの取得』 120 ページの『ODA メタデータの初期化』 122 ページの『ODA 開始の初期化』

ソース・データの選択および確認

ビジネス・オブジェクト・ウィザードのステップ 3 では、「ソースの選択」ダイアログ・ボックスが表示されます。このダイアログ・ボックスでは、データ・ソースのソース・ノード が表示されます。ソース・ノードはソース・ノード階層内に配列されています。各ソース・ノードは、ODA がデータ・ソース内で「検出」したオブジェクトの名前です。ソース・ノードを展開すれば、その他の子ノードを表示することができます。また、ソース・ノードを選択すれば、コンテンツを生成することができます。ユーザーは、このソース・ノード階層を展開し、データ・ソースからオブジェクトを選択して、コンテンツへの変換を行うことができます。詳しくは、92 ページの『ソース・ノード階層内での移動』を参照してください。

ステップ 3 でウィザードは次のステップを実行します。

1. 選択された ODA からソース・ノード階層を取得し、そのトップレベルを「ソースの選択」ダイアログ・ボックスに表示します。

   ソース・コード階層を取得する際、ウィザードは IGeneratesBoDefs インターフェースの getTreeNodes() メソッドを呼び出します。 ODA 開発者は、ODA クラスに IGeneratesBoDefs インターフェースを実装する際に、このメソッドを実装する必要があります。このメソッドは、データ・ソースを検索してソース・ノードを「検出」し、それらのソース・ノードを TreeNode オブジェクトの配列としてビジネス・オブジェクト・ウィザードに戻します。最初にノードを展開したとき、ウィザードは getTreeNodes() を呼び出して、ソース・ノード階層の特定のレベルを表示します。この階層を横断することにより、詳細のレベルを選択できます。詳しくは、92 ページの『ソース・ノード階層内での移動』を参照してください。

2. 「ソースの選択」ダイアログ・ボックスで、コンテンツ生成のために選択された階層内のソース・ノードの名前を追跡します。ウィザードは、選択されたソース・ノードの名前を含む配列を生成します。

ビジネス・オブジェクト・ウィザードのステップ 4 では、「ソース・ノードの確認」ダイアログ・ボックスが表示されます。このダイアログでは、ユーザーが選択したソース・ノードが表示されます。ユーザーは、選択内容を確認したり、「ソースの選択」ダイアログ・ボックスに戻ってソース・ノードを選択し直したりすることができます。**「次へ」**ボタンをクリックすると、ウィザードはコンテンツの生成を開始します。

`getTreeNodes()` メソッドの実装方法については、129 ページの『ソース・ノードの生成』を参照してください。

コンテンツの生成

ODA を作成すると、表 17 に示すコンテンツ・タイプ のどちらか一方または両方を生成できます。 ODA が生成するデータの構造体は、コンテンツ・タイプによって決まります。 ODA で特定のコンテンツをサポートするには、ODA に対して適切なコンテンツ生成インターフェース を実装する必要があります。表 17 は、ODA がサポートするコンテンツ・タイプと、それに対して ODA が実装する必要のあるコンテンツ生成インターフェースを示しています。

表 17. ODA のコンテンツ・タイプ

コンテンツ・タイプ	説明	コンテンツ生成インターフェース
ビジネス・オブジェクト定義	ODA は、データ・ソース内のオブジェクトを表すビジネス・オブジェクト定義を生成します。	`IGeneratesBoDefs`
バイナリー・ファイル	ODA は、生成されたコンテンツに関する情報を保持するファイル・オブジェクトを生成します。	`IGeneratesBinFiles`

注: 今回のリリースでは、ODA は、コンテンツとしてのビジネス・オブジェクト定義の生成をサポートする必要があります。したがって、ODA は `IGeneratesBoDefs` インターフェースを実装する必要があります。また、ODA は `IGeneratesBinFiles` インターフェースを実装することにより、コンテンツとしてのファイルの生成をサポートすることができます。

ソース・ノードを選択し、確認すると、ビジネス・オブジェクト・ウィザードはステップ 5 のコンテンツ生成を開始します。このステップでは、「ビジネス・オブジェクトの生成中」画面が表示されます。また、ビジネス・オブジェクト定義のコンテンツ生成メソッド `generateBoDefs()` が呼び出され、(ステップ 4 で) ユーザーが選択したソース・ノードの配列が ODA に渡されます。このメソッドは、選択したソース・ノードに対応するビジネス・オブジェクト定義を生成します。 ODA は、要求時コンテンツ・プロトコルでビジネス・オブジェクト定義の生成をサポートする必要がある ため、ビジネス・オブジェクト・ウィザードは必ず

generateBoDefs() メソッドを呼び出します。したがって、ODA 開発者は、ODA に IGeneratesBoDefs インターフェースを実装する際に、このメソッドを実装する必要があります。

ODA でファイル・コンテンツが生成されるかどうかは、IGeneratesBinFiles インターフェースが実装されているかどうかで決まります。ODA クラスがこのインターフェースを実装している場合、生成されたコンテンツを実際に提供するメソッドは、ファイル・コンテンツ・タイプに対して ODA が使用しているコンテンツ・*プロトコル* によって異なります。

- ODA が*要求時* コンテンツ・プロトコルを使用してコンテンツを生成する場合、ビジネス・オブジェクト・ウィザードは、コンテンツ生成メソッド generatesBinFiles() を呼び出すことにより、ステップ 5 の一環としてコンテンツ生成を開始します。このメソッドには、ユーザーが選択したソース・ノードの配列が渡されます。したがって、ODA でファイル・コンテンツをサポートするには、開発者は ODA に IGeneratesBinFiles インターフェースを実装する際に、このメソッドを実装する必要があります。
- ODA が*コールバック・* コンテンツ・プロトコルを使用してコンテンツを生成する場合、ODA (または何らかの外部プロセス) は、ユーザー定義メソッドを呼び出すことにより、コンテンツ生成を開始します。 ODA 開発者は、ファイル生成機構を実装する必要があります。

したがって、ビジネス・オブジェクト・ウィザードがファイル用のコンテンツ生成メソッド generateBinFiles() を呼び出すかどうかは、次のことによって決まります。

- ODA が IGeneratesBinFiles インターフェースを実装しているかどうか
- ODA が IGeneratesBinFiles を実装している場合は、ファイルを生成する際に ODA がどのコンテンツ・プロトコルを使用するのか

注: コンテンツ・プロトコルについては、126 ページの『ODA コンテンツ・プロトコルの選択』を参照してください。

どのコンテンツ・プロトコルを使用する場合でも、コンテンツを生成するには、次のステップを実行する必要があります。

1. 必要に応じて、動詞の値などの追加情報をビジネス・オブジェクト・プロパティーとして取得します。
2. 要求されたコンテンツを生成し、それを ODA メモリー内の生成済みコンテンツ構造体に保管します。

以降のセクションでは、これらのステップについて概説します。コンテンツ生成プロセスの概要についてもう少し詳しく知りたい方は、表 18 に示す参照先をご覧になり、サポートされるコンテンツ・タイプの詳細な情報を確認してください。

表 18. コンテンツ生成プロセス

コンテンツ・タイプ	詳細情報の参照先
ビジネス・オブジェクト定義 バイナリー・ファイル	137 ページの『ビジネス・オブジェクト定義の生成』 157 ページの『ファイルの生成』

ビジネス・オブジェクト・プロパティーの取得

通常 ODA は、ビジネス・オブジェクト定義を生成する前に、追加情報を必要とします。 ODA は、*ビジネス・オブジェクト・プロパティー* を定義することにより、このような追加情報を要求できます。 ODK API は、ビジネス・オブジェクト・プロパティーをエージェント・プロパティー (AgentProperty) オブジェクトとして表します。ビジネス・オブジェクト・プロパティーを収集するために、ODA はビジネス・オブジェクト・ウィザードに「BO プロパティー」ダイアログ・ボックスを表示することができます。このダイアログ・ボックスでは、ウィザードによりビジネス・オブジェクト・プロパティーが表示されます。それらのビジネス・オブジェクト・プロパティーを更新すると、ユーザーにより初期化されたプロパティーが ODA ランタイム・メモリーに書き込まれます (97 ページの図 51 を参照)。

「BO プロパティー」ダイアログ・ボックスを表示するために、ODA のコンテンツ生成メソッドは、ODKUtility クラスで定義されている getBOSpecificProps() メソッドを呼び出します。

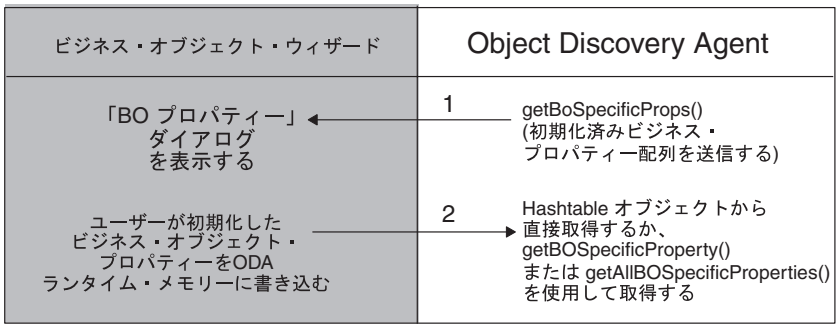


図 54. ビジネス・オブジェクト・プロパティーの取得

図 54 に示すように、getBOSpecificProps() メソッドでは次のステップが実行されます。

1. ビジネス・オブジェクト・プロパティーをビジネス・オブジェクト・ウィザードに送ります。これらのビジネス・オブジェクト・プロパティーは、「BO プロパティー」ダイアログ・ボックスに表示されます。

   ビジネス・オブジェクト・プロパティーを送信するために、getBOSpecificProps() メソッドは、エージェント・プロパティー (AgentProperty) オブジェクト (表示する各ビジネス・オブジェクト・プロパティーごとに 1 つのオブジェクト) の初期化済みの配列を引き数として送ります。
2. 「BO プロパティー」ダイアログ・ボックスから、値を追加または変更できます。**「次へ」**ボタンがクリックされると、ウィザードは、ユーザーが初期化したビジネス・オブジェクト・プロパティーを ODA 内の getBOSpecificProps() メソッドに戻します。

   ODA 内のこれらのビジネス・オブジェクト・プロパティーにアクセスするには、getBOSpecificProps() によって戻される Java Hashtable オブジェクトを使用します。または、getBOSpecificProperty() メソッドと

getAllBOSpecificProperties() メソッドを提供する ODKUtility クラスのインスタンスを使用してこれらのプロパティーにアクセスすることもできます。

ODA は、getBOSpecificProps() を繰り返し呼び出すことで、さまざまなビジネス・オブジェクトのプロパティー・セットを取得できます。getBOSpecificProps() メソッドの使用方法については、138 ページの『ビジネス・オブジェクト・プロパティーの要求』を参照してください。

生成済みコンテンツの提供

ODA は、生成したコンテンツを次の 2 つの部分に分けてビジネス・オブジェクト・ウィザードに提供します。

- コンテンツ・メタデータ

  コンテンツ・メタデータ (ContentMetaData) オブジェクトには、ODA の生成済みコンテンツについての情報が含まれます。ビジネス・オブジェクト・ウィザードはこの情報を使用して、生成済みコンテンツを取得する際に使用するコンテンツ取得メソッドを決定します。

- コンテンツ本体

  ODA は、ODA クラスのメソッドでアクセス可能な*生成済みコンテンツ構造体*内のある場所に対して、生成済みコンテンツを書き込みます。例えば、ODA クラスのメンバー変数である配列にコンテンツを書き込むことができます。

生成済みコンテンツを提供するメソッドは、特定のコンテンツ・タイプに対して ODA が使用しているコンテンツ・プロトコルによって異なります。

- ODA が要求時コンテンツ・プロトコルを使用してコンテンツを生成する場合、生成済みコンテンツ構造体にデータを追加し、ビジネス・オブジェクト・ウィザードにコンテンツ・メタデータ・オブジェクトを戻すのは、コンテンツ生成メソッドです。ビジネス・オブジェクト・ウィザードは、コンテンツ・タイプに基づいてコンテンツ生成メソッドを呼び出します。
  - ビジネス・オブジェクト定義の場合は、IGeneratesBoDefs インターフェースの generateBoDefs()
  - ファイルの場合は、IGenerateBinFiles インターフェースの generateBinFiles()
- ODA がコールバック・コンテンツ・プロトコルを使用してコンテンツを生成する場合、生成済みコンテンツ構造体にデータを追加し、ビジネス・オブジェクト・ウィザードにコンテンツ・メタデータ・オブジェクトを送るのは、ユーザー定義メソッドです。

注: コンテンツ・プロトコルについては、126 ページの『ODA コンテンツ・プロトコルの選択』を参照してください。

次の表は、生成済みコンテンツの提供方法に関する情報の参照先を示しています。

コンテンツ・タイプ	詳細情報の参照先
ビジネス・オブジェクト定義	150 ページの『生成済みのビジネス・オブジェクト定義の提供』
バイナリー・ファイル	160 ページの『生成されたファイルの提供』

生成済みコンテンツを取得する際、ビジネス・オブジェクト・ウィザードは、表 19 に示すコンテンツ取得メソッドを呼び出します。

表 19. コンテンツ取得メソッド

コンテンツ・タイプ	コンテンツ取得メソッド	詳細情報の参照先
ビジネス・オブジェクト定義	`IGeneratesBoDefs.getBoDefs()`	152 ページの『生成済みビジネス・オブジェクト定義へのアクセスの提供』
バイナリー・ファイル	`IGeneratesBinFiles.getBinFile()`	162 ページの『生成済みファイルへのアクセスの提供』

コンテンツ取得メソッドは、ODA オブジェクト内の生成済みコンテンツ構造体にアクセスし、配列内の指定されたコンテンツをビジネス・オブジェクト・ウィザードに戻します。ビジネス・オブジェクト・ウィザードは、ステップ 6 におけるコンテンツ保管要求を処理する前に、生成済みコンテンツへのアクセスが必要となります。詳しくは、『コンテンツの保管』を参照してください。

コンテンツの保管

ビジネス・オブジェクト・ウィザードのステップ 6 では、「ビジネス・オブジェクトの保管」ダイアログが表示されます。このダイアログ・ボックスでは、生成済みのビジネス・オブジェクト定義を保管する方法を選択できます。 87 ページの図 42 に示すように、ビジネス・オブジェクト・ウィザードでは、生成済みコンテンツを ICL プロジェクトまたはファイルに保管したり、Business Object Designer Express で各ビジネス・オブジェクト定義を開いたりすることができます。生成済みのビジネス・オブジェクト定義を指定した形式で保管するには、ビジネス・オブジェクト・ウィザードは生成済みコンテンツにアクセスできなければなりません。ビジネス・オブジェクト・ウィザードは、表 19 に示す ODA のコンテンツ取得メソッドを使用して、前のステップ (ステップ 5) でこのコンテンツを取得しています。

ODA 開発プロセスの概要

このセクションでは、ODA の開発プロセスに関して、次のトピックを取り上げます。

- 『ODA 開発用のツール』
- 113 ページの『ODA 開発プロセス』

ODA 開発用のツール

ODA は、IBM WebSphere Business Integration Adapter で実行可能なコンポーネントの 1 つです。アダプター には、InterChange Server Express とアプリケーション (またはテクノロジー) 間の通信をサポートするランタイム・コンポーネントが含まれています。これらのランタイム・コンポーネントの 1 つが ODA です。ODA は、コネクターが実行時に使用するビジネス・オブジェクト定義を作成します。コネクター は、アプリケーション (またはテクノロジー) と InterChange Server Express 間の通信を処理するランタイム・コンポーネントです。アダプターには、アダプター・フレームワーク も含まれます。アダプター・フレームワークは、カスタム・アダプターの構成、ランタイム、および開発用のコンポーネントを備えています。これらのコンポーネントは、既存のアプリケーションまたは特定のアプリケー

ション用のビルド済みアダプターが、WebSphere Business Integration Adapter 製品の一部として現在提供されていない場合に使用します。

ODA 開発用として、アダプター・フレームワークには、表 20 に示す開発サポートが含まれています。

表 20. ODA 開発用のアダプター・フレームワーク・サポート

アダプター・コンポーネント	Configuration Tool Express	API
ビジネス・オブジェクト定義	Business Object Designer Express	適用されない
Object Discovery Agent (ODA)	Business Object Designer Express	Object Discovery Agent Development Kit (ODK)

注: アダプター・フレームワークは、コネクター開発のサポートも提供します。

WebSphere Business Integration Adapter Framework と同様、Adapter 開発キット (ADK) も ODA およびコネクターのコード・サンプルを提供するツールキットです。詳しくは、『Adapter 開発キット』を参照してください。

Adapter 開発キット

Adapter 開発キット (ADK) は、アダプター開発を支援するファイルおよびサンプルを提供します。Adapter 開発キットは、多くの Object Discovery Agent (ODA)、コネクター、およびデータ・ハンドラーを含むアダプター・コンポーネントにサンプルを提供します。ADK が提供するサンプルは、製品ディレクトリーの `DevelopmentKits` サブディレクトリーにあります。

注: ADK は WebSphere Business Integration Adapters 製品の一部で、個別のインストールが必要です。そこで、ADK の開発サンプルにアクセスするには、WebSphere Business Integration Adapters 製品にアクセスして ADK をインストールする必要があります。ADK は *Windows* システムでしか使用できないことに注意してください。

表 21 は、ADK が ODA の開発のために提供するサンプルおよび、サンプルがある `DevelopmentKits` ディレクトリーのサブディレクトリーをリストします。

表 21. ODA 開発のための ADK サンプル

Adapter 開発キットのコンポーネント	説明	DevelopmentKits サブディレクトリー
Object Discovery Agent Development Kit (ODK)	ODA サンプルを提供します。	`Odk`
Twineball アダプター (Twineball adapter) サンプル	ODA が含まれるサンプル・アダプターを提供します。	`Twineball_sample`

表 21 に示すように、Adapter 開発キットには Object Discovery Agent (ODA) のサンプルが含まれます。これらのサンプルは、次のディレクトリーに入っています。

```
DevelopmentKits¥Odk
```

詳しくは、 112 ページの『ODA の開発サポート』を参照してください。

注: 表 21 に示すように、ADK は別のアダプター・コンポーネントであるコネクターの開発もサポートします。

ビジネス・オブジェクト定義の開発サポート

表 22 は、ビジネス・オブジェクト定義の開発を支援するために WebSphere Business Integration Adapters 製品と WebSphere InterChange Server Express 製品で提供されるツールを示しています。

表 22. ビジネス・オブジェクト定義を開発するためのツール

開発ツール	説明
Business Object Designer Express	ビジネス・オブジェクト定義を手動または ODA を通じて作成することを支援するグラフィック・ツール。

ビジネス・オブジェクト定義の概要については、4 ページの『ビジネス・オブジェクト定義』を参照してください。

ODA の開発サポート

表 23 は、ODA の開発を支援するために WebSphere Business Integration Adapters 製品と WebSphere InterChange Server Express 製品で提供されるツールを示しています。

表 23. ODA を開発するためのツール

開発ツール	説明
Business Object Designer Express Object Discovery Agent Development Kit (ODK)	ビジネス・オブジェクト定義を手動または ODA を通じて作成することを支援するグラフィック・ツール。 内容: • ODK API: カスタム ODA の作成を可能にする一連の Java クラス。これらのクラスの概要については、193 ページの『第 7 章 ODK API の概要』を参照してください。 • ODA ランタイム: ODA と Business Object Designer Express 間の通信を処理するために ODA ランタイムが使用する一連の Java クラス。 • ODA サンプル: Adapter 開発キット (ADK) の一部としてインストールされます。詳しくは、111 ページの『Adapter 開発キット』を参照してください。

表 23 に示すように、ODK は ODA 開発者向けに ODK API (ODA で使用するメソッドのライブラリー) とサンプル ODA を提供します。これらは、次の製品サブディレクトリーに入っています。

```
DevelopmentKits¥Odk¥Samples
```

ODK には、次のサンプル ODA が含まれています。

表 24. サンプル ODA

ODA サンプル	説明	DevelopmentKits¥Odk のサブディレクトリー
Roman Army ODA	ローマの将軍 (general) と兵士 (soldier) の名前を XML ファイルからビジネス・オブジェクト定義に変換し、その変換を説明する特定のバイナリー・ファイルを提供します。この ODA は、本章で説明する ODK API を使用します。	始動スクリプトの場合: `Samples` 外部ファイルおよび .jar ファイルの場合: `RomanArmy` Java ソースの場合: `com¥ibm¥btools¥ODK2¥RomanArmy`
JDBC ODA	JDBC データ (表およびスキーマ) をビジネス・オブジェクト定義に変換します。このサンプル ODA を実行するには、JDBC データベースへのアクセスが必要となります。このサンプルは、ビジネス・オブジェクト定義の生成のみ を処理し、ファイル・コンテンツの生成は処理しない、前のバージョンの ODK API に基づいています。 **注:** 新しい ODA を開発する場合は、より複雑なビジネス・オブジェクト定義の作成例としてのみ、このサンプルを利用してください。新しい ODA の構築方法の例としては、Roman Army ODA サンプルを使用してください。	始動スクリプトの場合: `Samples` Java ソースの場合: `com¥crossworlds¥JDBC`

ODA の概要については、75 ページの『Object Discovery Agent を使用してビジネス・オブジェクト定義を作成する方法』を参照してください。サンプル Roman Army ODA の実行方法については、77 ページの『サンプル ODA の使用』を参照してください。

ODA 開発プロセス

このセクションでは、ODA 開発プロセスについて概説します。ODA 開発プロセスには、次の主要なステップが含まれます。

1. WebSphere Business Integration システム・ソフトウェアのインストールと設定、および Java Development Kit (JDK) のインストール。
2. ODA の設計と実装

開発環境の設定

開発プロセスを開始する前に、次の事項を確認してください。

- アクセス可能なマシンに WebSphere Business Integration システム・ソフトウェアがインストールされていること。

ODA を実行するには、ODA ライブラリー CwODA.jar が利用できなければなりません。したがって、この ODA ライブラリーのインストールが必要となります。詳しくは、製品のインストール情報を参照してください。

InterChange Server Express

ビジネス・インテグレーション・システムで InterChange Server Express を使用する場合は、CwODA.jar ファイルがソフトウェアの一部としてインストールされます。製品のインストール情報を参照してください。この中にはシステムのインストールおよび始動方法が説明されています。

- Java Development Kit (JDK) または JDK 準拠の開発製品が開発マシン上にインストールされていること。

  JDK の必要なバージョンとそのインストール方法については、製品のインストール情報を参照してください。Java がインストールされているディレクトリーが含まれるように、PATH 環境変数を更新してください。 InterChange Server Express を統合ブローカーとして使用しているため、パスを更新してから InterChange Server Express を再始動します。
- 開発環境では、ODA ライブラリー・ファイル (CwODA.jar) が含まれる次のディレクトリーにアクセスできなければなりません。

  ProductDir¥lib

  ODA をコンパイルするために、コンパイラーはこのディレクトリー ODA へのアクセスが*可能*でなければなりません。ODA のコンパイル方法については、185 ページの『ODA のコンパイル』を参照してください。

注: ODA を作成し、生成済みコンテンツをテストする際に、InterChange Server Express またはコネクターを実行しておく必要はありません。ただし、ある時点に到達したらコネクターを実行し、ODA の生成済みコンテンツにコネクターのビジネス・オブジェクトが正しく記述されているかどうかをテストする必要があります。 IBM WebSphere Business Integration システム全体をテストするには、InterChange Server Express とコネクターが通信可能でなければなりません。

ODA 開発ステージ

ODA を開発するには、表 25 に示すステップを実行する必要があります。

表 25. ODA 開発ステップ

	ODA 開発ステップ	詳細情報の参照先
1.	ODA 基底クラス (ODKAgentBase2) を拡張し、ODA クラスを作成します。	116 ページの『ODA 基底クラスの拡張』
2.	ODA を開始する手段を提供する、ODA クラスのメソッドを実装します。	117 ページの『ODA の開始』

表 25. ODA 開発ステップ (続き)

	ODA 開発ステップ	詳細情報の参照先
3.	ODA コンテンツを設計および実装します。 • ODA がサポートするコンテンツ・タイプ: – ビジネス・オブジェクト定義: `IGeneratesBoDefs` インターフェースを実装します (必須) – バイナリー・ファイル: `IGeneratesBinFiles` インターフェースを実装します (任意) • ODA が使用するコンテンツ・プロトコル: – 要求時 (ビジネス・オブジェクト定義で必要) – コールバック	124 ページの『ODA で生成されるコンテンツの決定』
4.	すべての ODA メソッドに対して、エラー処理とメッセージ処理を実装します。トレース・メッセージは、適切なトレース・レベルで実装します。	183 ページの『例外処理』および 175 ページの『トレース・メッセージとエラー・メッセージの処理』
5.	データ・ソース対話の処理に必要なクラスを作成します。 • 接続管理 • コンテンツの分析および定義	IBM では、Object Discovery Agent をモジュール化し、その主要なプロセスをそれぞれ個別に扱うコンポーネント・クラスを生成することをお勧めしています。詳細は、データ・ソースごとに異なります。
6.	ODA をビルドします。	185 ページの『ODA のコンパイル』
7.	新しい ODA 用の始動スクリプトを作成します。	186 ページの『新規の ODA の始動』
8.	ODA のテストとデバッグを行い、必要に応じて結果を記録します。	

ODA コードの作成は、ビジネス・オブジェクト開発の作業全体の一部にすぎません。Object Discovery Agent コードの作成を開始する前に、ビジネス・オブジェクトの設計の問題、ビジネス・オブジェクトが表すエンティティーが存在するアプリケーションと、ビジネス・オブジェクトを処理するコネクターおよびデータ・ハンドラーを明確に把握してください。さらに、Object Discovery Agent を使用してビジネス・オブジェクト定義を作成するときに、Business Object Designer Express 上でユーザーがたどる手順も十分理解しておくことが必要です。

注: ビジネス・オブジェクトの設計については、 19 ページの『第 2 章 ビジネス・オブジェクト設計』を参照してください。Business Object Designer Express での Object Discovery Agent の使用については、 75 ページの『Object Discovery Agent を使用してビジネス・オブジェクト定義を作成する方法』を参照してください。

ODA 基底クラスの拡張

ODA を作成するには、ODA 基底クラス (ODKAgentBase2) を拡張して、独自の *ODA* クラス を作成します。 ODKAgentBase2 クラスには、ODA の初期化、設定、および終了のためのメソッドが含まれています。独自の ODA を実装するには、この ODA 基底クラスを拡張して、ODA クラスを作成する必要があります。

ODA クラスを派生するには、次のステップを実行します。

1. ODKAgentBase2 クラスを拡張して ODA クラスを作成します。この ODA クラスには、次の名前を付けることを推奨します。

   ODAname.java

   ここで、*ODAname* は ODA を固有に識別する名前で、ODA のソース・データに ODA の拡張子を付けた形式 (*srcDataName*ODA) となります。ソース・データ名については、185 ページの『ODA の名前付け』を参照してください。例えば、HTML オブジェクト用の ODA を作成する場合は、HTMLODA.java という名前の ODA クラス・ファイルを作成します。
2. ODA クラス・ファイル内で、ユーザーの ODA を含むパッケージ名を定義します。 ODA パッケージ名は、次のような形式で指定するきまりになっています。

   com.ibm.oda.*srcDataName*.*ODAname*

   上記の形式において、*ODAname* はステップ 1 で定義されたものと同じであり、*srcDataName* もステップ 1 で定義されたものと同じです (ただし、小文字で指定する点は異なる)。例えば、HTML オブジェクト用の ODA のパッケージ名は、次のように ODA クラス内に定義できます。

   ```
   package com.ibm.oda.html.HTMLODA;
   ```
3. ODA クラス・ファイルで com.crossworlds.ODK パッケージのクラスをインポートします。

   ```
   import com.crossworlds.ODK.*;
   ```

   ODK API のメソッドにアクセスするには、ODA クラスで ODK パッケージをインポートする必要があります (これは、製品ディレクトリー内の lib サブディレクトリーにある CwODK.jar ファイルに含まれています)。 ODA クラス・コードを保持するファイルを複数作成する場合は、ODK パッケージのクラスを各ソース・ファイルにインポートする必要があります。
4. ODA クラスを定義し、ODA が使用するコンテンツ生成インターフェースをその定義に組み込みます。ビジネス・オブジェクト定義をコンテンツとして生成するには、ODA に IGeneratesBoDefs コンテンツ生成インターフェースを実装する必要があります。ODA に IGeneratesBinFiles コンテンツ生成インターフェースを実装すれば、バイナリー・ファイルのコンテンツを生成することもできます。

   例えば、必須の IGeneratesBoDefs インターフェースのみ を HTML 用の ODA に実装するとします。その場合の定義は、次のようになります。

   ```
   public class HTMLODA extends ODKAgentBase2 implements IGeneratesBoDefs {
   ```

   コンテンツ生成インターフェースの詳細については、124 ページの『ODA で生成されるコンテンツの決定』を参照してください。

5. ODKAgentBase2 クラスの抽象メソッドを ODA クラスに実装します。 表 26 は、それらのメソッドの概要を示しています。メソッドは、ビジネス・オブジェクト・ウィザードが呼び出す順番に記載されています。これらの抽象メソッドの実装方法については、表 26 を参照してください。

表 26. ODKAgentBase2 クラスの抽象メソッドの拡張

ODKAgentBase2 の抽象メソッド	説明	詳細情報の参照先
getAgentProperties()	このメソッドは、次のタスクを実行します。 • ODA の初期化に必要な構成プロパティーの定義 (データ・ソースへの接続のために ODA が必要とする情報の定義も含む)。 • 配列内の構成プロパティーのビジネス・オブジェクト・ウィザードへの送信。	118 ページの『構成プロパティーの取得』
getMetaData()	ODA のメタデータ (そのコンテンツ生成機能を含む) を保持する AgentMetaData オブジェクトをインスタンス化します。	120 ページの『ODA メタデータの初期化』
init()	リソース割り振りやデータ・ソースへの接続も含めて ODA を初期化します。	122 ページの『ODA 開始の初期化』
terminate()	データ・ソースからの切断、および ODA が使用するリソースの解放などのクリーンアップを実行します。	174 ページの『ODA のシャットダウン』

6. 1 つまたは複数のコンテンツ生成インターフェースのメソッドを ODA クラスに実装します。表 26 は、コンテンツ生成インターフェースのメソッドと、それらのメソッドの作成方法について詳細な情報が記載されている場所を示しています。

表 27. コンテンツ生成インターフェースのメソッドの定義

コンテンツ生成インターフェース	説明	詳細情報の参照先
IGeneratesBoDefs	getContentProtocol()	124 ページの『ODA で生成されるコンテンツの決定』
	getTreeNodes() generateBoDefs() getBoDefs()	128 ページの『コンテンツとしてのビジネス・オブジェクト定義の生成』
IGeneratesBinFiles	getContentProtocol()	124 ページの『ODA で生成されるコンテンツの決定』
	generateBinFiles() getBinFile()	153 ページの『コンテンツとしてのバイナリー・ファイルの生成』

ODA の開始

ODA を始動すると、関連する ODA クラス (ODKAgentBase2 から拡張したクラス) が ODA ランタイムによってインスタンス化され、表 28 に示すクラス・メソッドが呼び出されます。

表 28. ODA の開始

初期化タスク	ODKAgentBase2 メソッド	詳細情報の参照先
1. 構成プロパティーを取得します (開くデータ・ソースを記述する構成プロパティーも含めて)。	`getAgentProperties()`	『構成プロパティーの取得』
2. ODA メタデータを初期化し、ODA に関する情報 (特にそれがサポートするコンテンツ) をビジネス・オブジェクト・ウィザードが取得できるようにします。	`getMetaData()`	120 ページの『ODA メタデータの初期化』
3. データ・ソースへの接続を開くなどの必要な開始ステップを実行するために ODA を初期化します。	`init()`	122 ページの『ODA 開始の初期化』

以降のセクションでは、表 28 に示すステップをそれぞれ説明します。

構成プロパティーの取得

ODA の初期化を開始する際に、ビジネス・オブジェクト・ウィザードは ODA クラスの `getAgentProperties()` メソッドを呼び出します。 `getAgentProperties()` メソッドは、低レベルの ODA 基底クラス (`ODKAgentBase`) に含まれています。このクラスは、ODA 基底クラスの `ODKAgentBase2` からユーザーの ODA クラスへと順に継承されます。

重要: ODA クラスを実装する際は、`getAgentProperties()` メソッドを実装する*必要があります。*

`getAgentProperties()` メソッドは、次のタスクを実行します。

- 『ODKUtility オブジェクトへのハンドルの取得』
- 119 ページの『構成プロパティー配列の初期化』

ODKUtility オブジェクトへのハンドルの取得

`getAgentProperties()` は、ビジネス・オブジェクト・ウィザードが*最初に* 呼び出す ODA メソッドであるため、`ODKUtility` オブジェクトをインスタンス化するのに適した場所となります。ODKUtility オブジェクトは、ODA コードに対して次のものへのアクセスを提供します。

- ODA ランタイム・メモリー内のオブジェクト (構成プロパティーやビジネス・オブジェクト・プロパティーなど)
- トレース、およびユーザー応答ダイアログ・ボックスの表示を行うユーティリティー・メソッド

`ODKUtility` オブジェクトへのアクセスを取得するには、`getODKUtility()` メソッドを使用します。このメソッドは `ODKUtility` クラス内に定義されており、`ODKUtility` オブジェクトへのハンドルを戻します。

```
odkUtil = ODKUtility.getODKUtility()
```

`ODKUtility` オブジェクトへのハンドルを ODA クラス全体に対してグローバルであると宣言すると、このクラス内のすべての メソッドが、ユーティリティー・メソッドにアクセスできるようになります。

注: ODKUtility オブジェクトをその getAgentProperties() メソッドでインスタンス化する代わりに、サンプル Roman Army ODA は m_utility という名前のメンバー変数を ODA クラス内に用意し、それを次のように初期化します。

```
final ODKUtility m_utility = ODKUtility.getODKUtility();
```

構成プロパティー配列の初期化

103 ページの『ODA 構成プロパティーの取得』で説明したように、ビジネス・オブジェクト・ウィザードは getAgentProperties() が戻す構成プロパティー配列を使用して、「エージェントの構成」ダイアログ・ボックス (ステップ 2) を初期化します。このダイアログ・ボックスでは ODA 構成プロパティーがすべて表示され、ユーザーは値を入力または変更することができます。構成プロパティー配列は、AgentProperty オブジェクトの配列です。AgentProperty クラスは、構成プロパティーが次の機能を持つためのサポートを提供します。

- デフォルト値
- 値を 1 つだけ保持するか、複数保持するかの指定
- ユーザーが選択する有効な値のリスト
- 入力可能な値を制限する条件

注: 詳しくは、 163 ページの『エージェント・プロパティーの使用』を参照してください。

getAgentProperties() の目的は、ODA 構成プロパティーを記述する AgentProperty オブジェクトの配列をビジネス・オブジェクト・ウィザードに送ることです。 getAgentProperties() で構成プロパティー配列を初期化するには、次のステップを実行します。

1. 構成プロパティーの AgentProperty オブジェクトをインスタンス化し、該当する適切なプロパティー情報でそれを初期化します。

   getAgentProperties() メソッドの実装では、ビジネス・オブジェクト・ウィザードがユーザーに対して表示する各構成プロパティーごとに、エージェント・プロパティー・オブジェクトをインスタンス化する必要があります。エージェント・プロパティー・オブジェクトをインスタンス化するときは、そのメンバー変数 (163 ページの表 49 を参照) の一部またはすべてを初期化します。
2. 初期化した AgentProperty オブジェクトを構成プロパティー配列に保管します。
3. 初期化した構成プロパティー配列を getAgentProperties() メソッドから戻します。

図 55 は、getAgentProperties() メソッド (サンプル Roman Army ODA の ArmyAgent2 クラスで定義) の実装を示しています。

```
public AgentProperties[] getAgentProperties()
   throws com.crossworlds.ODK.ODKException
{
   AgentProperty general = new AgentProperty("Army general",
      AgentProperty.TYPE_STRING, true, false, false,
      "A general is a soldier at least 45 years old", true,
      ODKConstant.SINGLE_CARD, m_generals.toArray(), null);

   AgentProperty recAdop = new AgentProperty("Allow adoption",
      AgentProperty.TYPE_BOOLEAN, true, false, false,
      "Select ¥"yes¥" if adopted children can be business objects", true,
      ODKConstant.SINGLE_CARD, new Object[]{"true", "false"}, null);

   AgentProperty minAge = new AgentProperty("Minimum age for drafting",
      AgentProperty.TYPE_INTEGER, true, false, false,
      "Drafted soldiers will be generable nodes", false,
      ODKConstant.SINGLE_CARD, null, new Object[] {"15"});

   AgentProperty maxAge = new AgentProperty("Maximum age for drafting",
      AgentProperty.TYPE_INTEGER, true, false, false,
      "Drafted soldiers will be generable nodes", false,
      ODKConstant.SINGLE_CARD, null, new Object[] {"55"});

   AgentProperty minAdo = new AgentProperty("Minimum age for adopting",
      AgentProperty.TYPE_INTEGER, true, false, true,
      "Drafted soldiers will be generable nodes", false,
      ODKConstant.SINGLE_CARD, null, new Object[] {"" + m_minAdoptionAge});

   AgentProperty[] props = new AgentProperty[]
      {general, minAge, maxAge, recAdop, minAdo});

return props;
}
```

図 55. 構成プロパティー配列の初期化

図 55 では、サンプル Roman Army ODA の ODA 構成プロパティーが 5 つ初期化されています。実際に定義するプロパティーは、ODA がアクセスするデータ・ソースによって異なります。

構成プロパティーの値を指定すると、ビジネス・オブジェクト・ウィザードはそれらのプロパティーを ODA ランタイム・メモリー内に保管します。 ODA は、ODKUtility クラス内の getAgentProperty() などのメソッドを通じて、これらのプロパティーにアクセスします。詳しくは、122 ページの『ODA 構成プロパティーの取得』を参照してください。

ODA メタデータの初期化

ビジネス・オブジェクト・ウィザードは、ODA の getAgentProperties() メソッドを呼び出した後に getMetaData() メソッドを呼び出し、ODA メタデータを初期化します。 getMetaData() メソッドは ODA 基底クラス (ODKAgentBase2) に定義されており、ユーザーの ODA クラスによって継承されます。このメソッドは、ODA のメタデータを含む AgentMetaData オブジェクトを、サポートされる生成済みコンテンツとともに戻します。

重要: getMetaData() メソッドは、抽象メソッドです。 ODA クラスを実際する際は、getMetaData() メソッドを実装する必要があります。

ODA のメタデータを取得する必要がある場合、AgentMetaData オブジェクトは、表 29 の情報を ODA ランタイムに提供します。

表 29. AgentMetaData オブジェクトのコンテンツ

メンバー変数	説明
agentVersion	ODA のバージョン
searchableNodes、searchPatternDesc	ODA の検索パターンを指定するための情報。この検索パターンを指定すると、表示されるデータ・ソースのツリー・ノードの数を削減できます。
supportedContent	ODA でサポート可能な生成済みコンテンツの説明

ODA メタデータを初期化するには getMetaData() メソッドを実装する必要がありますが、そのためには次のステップを実行する必要があります。

- AgentMetaData クラスのインスタンスを生成し、ODA それ自体およびオプションの ODA バージョンに参照を渡します。

  いずれかの形式の AgentMetaData() コンストラクターを使用します。どちらの形式の場合でも、this 参照を ODA オブジェクト (ODA クラスのインスタンス) に渡す必要があります。コンストラクターは、ODA が実装している 1 つまたは複数のコンテンツ生成インターフェースについての情報を取得するために ODA オブジェクトを照会します。次にこの情報を使用して、サポートされる各コンテンツ・タイプに対して ODA がサポートするコンテンツ・プロトコルで、supportedContent メンバー変数を初期化します。ODA のサポートされるコンテンツについては、124 ページの『ODA で生成されるコンテンツの決定』を参照してください。

  必要な場合は、ODA バージョンを引き数としてコンストラクターに渡し、agentVersion メンバー変数を初期化することもできます。
- ODA のその他のメンバー変数を必要に応じて初期化します。

  ODA で検索パターン機能をサポートするには、AgentMetaData オブジェクトをインスタンス化した後に、searchableNodes メンバー変数と searchPatternDesc メンバー変数を明示的に初期化する必要があります。詳しくは、132 ページの『検索パターン機能の実装』を参照してください。
- 初期化した AgentMetaData オブジェクトを getMetaData() メソッドから戻します。

図 56 は、getMetaData() メソッド (サンプル Roman Army ODA の ArmyAgent2 クラスで定義) の実装を示しています。

```
public AgentMetaData getMetaData(){
   odkUtil.trace(TRACELEVEL1, XRD_TRACE, "Entering getMetaData()...");
   AgentMetaData amdObj = new AgentMetaData(this, "Sample ODA v1.0.0");

   //Initialize search-pattern feature for tree nodes
   amd.searchableNodes = true;
   amd.searchPatternDesc = "Enter the first letter to search by. For example, " +
      "¥"A¥", ¥"k¥", "¥Z¥". Only names that start with this letter will be " +
      "returned."

   return amd;
}
```

図 56. ODA メタデータの初期化

図 56 の getMetaData() メソッドは、(IGeneratesBoDefs インターフェースを実装する) ArmyAgent3 クラスによって継承されます。そのため、このコード・フラグメントの AgentMetaData() コンストラクターの呼び出しにより、ODA のコンテンツ・タイプおよびそれに関連するコンテンツ・プロトコルが初期化されます。ArmyAgent3 内で getMetaData() が開始されると、ODA のコンテンツ・タイプが ContentType.BusinessObject に初期化され、そのコンテンツ・プロトコルが「要求時」に初期化されます。詳しくは、 124 ページの『ODA で生成されるコンテンツの決定』を参照してください。

この getMetaData() メソッドは、searchableNodes メンバー変数と searchPatternDesc メンバー変数を初期化することにより、検索パターン機能のサポートも提供します。searchPatternDesc 変数には「検索パターンの入力」ダイアログ・ボックスに表示されるテキストが含まれています (94 ページの図 46 を参照)。

ODA 開始の初期化

ビジネス・オブジェクト・ウィザードは、ODA の getMetaData() メソッドを呼び出した後に init() メソッドを呼び出し、ODA 開始の初期化を始めます。 init() メソッドは、低レベルの ODA 基底クラス (ODKAgentBase) に含まれています。このクラスは、ODA 基底クラスの ODKAgentBase2 からユーザーの ODA クラスへと順に継承されます。このメソッドは、ODA の初期化ステップを実行します。

重要: init() メソッドは、抽象メソッドです。ODA クラスを実装する際は、init() メソッドを実装する*必要があります*。

init() メソッドの主要なタスクを次に示します。

- 『ODA 構成プロパティーの取得』
- 123 ページの『接続の確立』
- 124 ページの『ODA バージョンの確認』

ODA 構成プロパティーの取得

init() メソッドは、ODA の初期化を完了するために必要な、ユーザーが初期化した各構成プロパティーを取得できます。ODA は、構成プロパティーを getAgentProperties() メソッドで初期化します。ユーザーは必要に応じて、ビジネス・オブジェクト・ウィザードの「エージェントの構成」ダイアログ・ボックスでこれらのプロパティーを更新できます。構成プロパティーが更新されると、ビジネス・オブジェクト・ウィザードはそれを ODA ランタイム・メモリーに書き込みます。

ODK API は、ODA 構成プロパティーの値を ODA ランタイム・メモリーから取得するためのメソッドを提供します (表 30 を参照)。

表 30. ODA 構成プロパティーの値を取得するためのメソッド

ODK ライブラリー・メソッド	説明
getAgentProperty()	指定した ODA 構成プロパティーの値を取得します。
getAllAgentProperties()	すべて の ODA 構成プロパティーを Java Hashtable オブジェクトとして取得します。

表 30 のメソッドはすべて、ODKUtility クラス内で定義されています。したがって、構成プロパティーにアクセスする*前に*、このクラスの singleton オブジェクトへのハンドルを取得する必要があります。詳しくは、118 ページの『ODKUtility オブジェクトへのハンドルの取得』を参照してください。

図 57 は、init() メソッド (サンプル Roman Army ODA の ArmyAgent3 クラスで定義) の実装を示しています。

```
public void init() throws com.crossworlds.ODK.ODKException
{
   Hashtable h = m_utility.getAllAgentProperties();

// Obtain values of ODA configuration properties
   AgentProperty property = (AgentProperty) h.get("Army general");
   m_general = property.allValues[0].toString();

   property = (AgentProperty) h.get("Minimum age for drafting");
   m_minAge = Integer.parseInt(property.allValues[0].toString());

   property = (AgentProperty) h.get("Maximum age for drafting");
   m_maxAge = Integer.parseInt(property.allValues[0].toString());

   property = (AgentProperty) h.get("Allow adoption");
   m_allowAdoption = new Boolean(
      property.allValues[0].toString()).booleanValue();

// Clear the generated-content structure
   m_generatedBOs.clear();
}
```

図 57. ODA の初期化

図 57 で init() メソッドは、次のものを使用して構成プロパティーの値を取得しています。

- ODKUtility クラスで定義されている getAllAgentProperties() メソッド。すべての構成プロパティーを取得して、Java Hashtable オブジェクトを生成します。
- Java Hashtable クラスで定義されている get() メソッド。ハッシュ・テーブルの要素を名前を指定して取得します。
- AgentProperty クラスで定義されている allValues メンバー変数。各構成プロパティーに指定された値を取得します。

この init() メソッドに含まれている構成プロパティーはすべて、単一カーディナリティー・プロパティーです。したがって、allValues メンバー変数に含まれる値は 1 つのみです。複数カーディナリティー・プロパティーの使用例については、139 ページの『ビジネス・プロパティー配列の作成』を参照してください。この init() メソッドは、ODA の生成済みコンテンツ構造体 (m_generatedBOs と呼ばれるベクトル) も初期化します。このベクトルは、生成済みのビジネス・オブジェクト定義を保持します。

接続の確立

通常 init() 初期化メソッドの主要なタスクは、データ・ソースへの接続を確立することです。 ODA はデータ・ソースを検索し、ビジネス・オブジェクト定義への変換が可能なオブジェクトを「検出」します。接続を確立するために、init() メソッドは次のタスクを実行することができます。

- 接続情報を提供する ODA 構成プロパティーを取得し、それを使用してデータ・ソースに接続する。必要な構成プロパティーが空の場合、init() メソッドは、デフォルト値を提供するか、ODKInvalidPropException 例外をスローします。

  getAgentProperty() メソッドを使用すると、ODA 構成プロパティーの値を取得できます。詳しくは、 122 ページの『ODA 構成プロパティーの取得』を参照してください。
- 必要な接続またはファイルを取得する。例えば、通常 init() メソッドは、データ・ソースとの接続を確立します。ODA が接続を開くことができない 場合、init() メソッドは、失敗の原因を示すために ODKException 例外 (または、そのサブクラスの 1 つ) をスローする必要があります。

init() メソッドが正常に実行されるのは、ODA が正常に接続を開き、さらに ODA がデータ・ソース内のデータを処理する準備を整えた場合です。ODA が接続を開くことができない 場合、init() メソッドは、失敗の原因を示すために ODKException 例外をスローする必要があります。

ODA バージョンの確認

getVersion() メソッドは、ODA ランタイムのバージョンを戻します。このメソッドは、低レベルの ODA 基底クラス (ODKAgentBase) に含まれています。このクラスは、ODA 基底クラスの ODKAgentBase2 からユーザーの ODA クラスへと順に継承されます。これは、次の 2 つの状況で呼び出されます。

- init() メソッドが ODA ランタイムのバージョンを確認するために getVersion() を呼び出す必要がある。
- バージョンを取得する必要があるときに ODA ランタイムが getVersion() メソッドを呼び出す。

注: getVersion() メソッドは、ODA のバージョンではなく、ODA ランタイムのバージョンを戻します (ODA のバージョンは ODA のメタデータの一部として保管されています)。

ODA で生成されるコンテンツの決定

このセクションでは、ご使用の ODA が生成可能なコンテンツを決定する際に検討すべき問題に関して、次のトピックを取り上げます。

- 『ODA コンテンツ・タイプの選択』
- 126 ページの『ODA コンテンツ・プロトコルの選択』

ODA コンテンツ・タイプの選択

ODK API は、ODA がサポートする有効なコンテンツ・タイプを ContentType クラスを使用して識別します。このクラスには、サポートされる各コンテンツ・タイプの静的メンバー変数が含まれます (表 31 を参照)。

表 31. コンテンツ・タイプの表現方法

コンテンツ・タイプ	ContentType メンバー変数
ビジネス・オブジェクト定義 バイナリー・ファイル	BusinessObject BinaryFile

ContentType クラスは、サポートされる ODA コンテンツ・タイプの列挙型リストをシミュレートします。例えば、ビジネス・オブジェクト定義を表すコンテンツ・タイプ・オブジェクトでは、次のように BusinessObject メンバー変数のみが使用されます。

```
ContentType.BusinessObject
```

特定のコンテンツ・タイプの生成をサポートするために、ODA は、106 ページの表 17 に示すコンテンツ生成インターフェースを適切に実装する必要があります。ODA はすべて、ビジネス・オブジェクト定義の生成をサポートする必要があります。ODA は、必要に応じてバイナリー・ファイルをそのコンテンツとして生成することもできます。コンテンツ生成インターフェースには、表 32 に示すタイプのメソッドが含まれます。コンテンツ生成インターフェースを実装する際は、これらのメソッドを実装する必要があります。

表 32. コンテンツ生成インターフェース内のメソッド

メソッド	メソッドの目的	IGeneratesBoDefs	IGeneratesBinFiles
ソース・ノード生成メソッド	ビジネス・オブジェクト・ウィザードは、ユーザーに対して表示するソース・ノード階層を取得するためにこのメソッドを呼び出します (ステップ 3: ソースの選択)。	`getTreeNodes()`	なし
コンテンツ生成メソッド	ビジネス・オブジェクト・ウィザードは、指定されたソース・データ・コンテンツの生成を開始するためにこのメソッドを呼び出します (ステップ 5: ビジネス・オブジェクトの生成中)。	`generateBoDefs()`	`generateBinFiles()`
コンテンツ取得メソッド	ビジネス・オブジェクト・ウィザードは、生成済みコンテンツを ODA メモリーから取得するためにこのメソッドを呼び出します (ステップ 5: ビジネス・オブジェクトの生成中)。	`getBoDefs()`	`getBinFile()`

どのコンテンツ生成インターフェースのメソッドを呼び出すのかを決定する際に、ビジネス・オブジェクト・ウィザードは ODA のメタデータを確認します。このメタデータのコンポーネントの 1 つが、supportedContent メンバー変数です。このメンバー変数は、ODA の getMetaData() メソッド内で呼び出される AgentMetaData() コンストラクターによって初期化されます。詳しくは、120 ページの『ODA メタデータの初期化』を参照してください。

表 33 は、コンテンツ生成インターフェース内のメソッドの実装方法に関して、この章で取り上げるトピックを示しています。

表 33. コンテンツ生成インターフェースの開発方法

コンテンツ生成インターフェース	詳細情報の参照先
IGeneratesBoDefs	128 ページの『コンテンツとしてのビジネス・オブジェクト定義の生成』
IGeneratesBinFiles	153 ページの『コンテンツとしてのバイナリー・ファイルの生成』

ODA コンテンツ・プロトコルの選択

ODA は、表 34 に示すコンテンツ・プロトコル のいずれかを使用して、特定のコンテンツ・タイプを生成できます。コンテンツ・プロトコルは、サポートされるコンテンツを生成する際の ODA とビジネス・オブジェクト・ウィザード との対話方法を決定します。つまり、ビジネス・オブジェクト・ウィザードが ODA からのコンテンツ生成を明示的に開始できるかどうかを決定します。

表 34. ODA のコンテンツ・プロトコル

コンテンツ・プロトコル	説明	コンテンツ・プロトコル定数
要求時	ビジネス・オブジェクト・ウィザードは、コンテンツ生成メソッドを呼び出すことにより、ODA に対してコンテンツを生成するように明示的に要求します。このメソッドが完了すると、要求時コンテンツの準備が整います。ビジネス・オブジェクト・ウィザードは、コンテンツ取得メソッドを呼び出すことにより、都合の良いときにこのコンテンツを取得できます。	CONTENT_PROTOCOL_ONREQUEST
コールバック	ODA は特定の方法でコンテンツを生成し、コンテンツの準備が整った時点でその旨をビジネス・オブジェクト・ウィザードに通知します。通知を受け取ると、ビジネス・オブジェクト・ウィザードはコンテンツ取得メソッドを呼び出して、このコンテンツを取得します。	CONTENT_PROTOCOL_CALLBACK

注: ODA は、要求時コンテンツ・プロトコルを使用したビジネス・オブジェクト定義コンテンツの生成をサポートする必要があります。さらに、ODA はこれらのコンテンツ・プロトコルでのファイル・コンテンツの生成をサポートできます。

コンテンツ・プロトコルをサポートするには、ODA で次のステップを実行する必要があります。

- 『実装されるコンテンツ・プロトコルの指定』
- 127 ページの『コンテンツ生成メソッドの実装』

実装されるコンテンツ・プロトコルの指定

IGeneratesBoDefs インターフェースと IGeneratesBinFiles インターフェースは、どちらも IGeneratesContent インターフェースから拡張されています。したがって、これらのインターフェースは、IGeneratesContent で定義される単一のメソッド getContentProtocol() を継承します。 ODA のコンテンツ生成インターフェースを実装する際は、getContentProtocol() メソッドを実装して、サポートされるコンテンツ・タイプに使用するコンテンツ・プロトコルを指定する必要があります。

注: ODA は、指定されたコンテンツ・タイプに対して *1* つ のコンテンツ・プロトコルをサポートします。

`getContentProtocol()` メソッドは、ODA でサポートされるコンテンツ・タイプを示す `ContentType` オブジェクトを引き数として受け入れます。
`getContentProtocol()` メソッドは、この指定されたコンテンツ・タイプに対してサポートされるコンテンツ・プロトコルを戻します。このメソッドは、サポートされるコンテンツ・プロトコルをコンテンツ・プロトコル定数 (表 34 を参照) の 1 つとして戻します。これらの定数は、`ODKConstant` インターフェース内で定義されています。

注: 今回のリリースでは、ODA は要求時にビジネス・オブジェクト定義を生成する必要があります。したがって、`ContentType.BusinessObject` というコンテンツ・タイプ用の `CONTENT_PROTOCOL_ONREQUEST` 定数を戻すために、`getContentProtocol()` メソッドを実装する必要があります。さらに、ODA はこれらのプロトコルでのファイル生成をサポートし、`ContentType.BinaryFile` というコンテンツ・タイプ用の適切なコンテンツ・プロトコル定数を戻すことができます。

図 58 は、`getContentProtocol()` の実装を示しています。ファイルの生成にはコールバック・プロトコル、ビジネス・オブジェクト定義の生成には要求時プロトコルがサポートされていることがわかります。

```
public long getContentProtocol(ContentType contentType)
{
    if (contentType == ContentType.BinaryFile)
      return ODKConstant.CONTENT_PROTOCOL_CALLBACK;
    else
      return ODKConstant.CONTENT_PROTOCOL_ONREQUEST;
}
```

図 58. サポートされるコンテンツ・プロトコルの指定

コンテンツ生成メソッドの実装

表 35 に示すように、コンテンツ生成メソッドの実装は、コンテンツ・タイプがサポートするコンテンツ・プロトコルによって異なります。

表 35. コンテンツ・プロトコルおよびコンテンツ生成メソッド

コンテンツ・プロトコル	コンテンツ生成メソッドの呼び出し方法	コンテンツ生成メソッドの実装
要求時	ビジネス・オブジェクト・ウィザードは、コンテンツ生成 (ビジネス・オブジェクト定義またはファイル) を開始する際に、コンテンツ生成メソッドを明示的に呼び出します。	メソッドは、引き数として渡されたソース・ノードのコンテンツを生成し、適切なコンテンツ・メタデータをビジネス・オブジェクト・ウィザードに戻す必要があります。

表 35. コンテンツ・プロトコルおよびコンテンツ生成メソッド (続き)

コンテンツ・プロトコル	コンテンツ生成メソッドの呼び出し方法	コンテンツ生成メソッドの実装
コールバック	ビジネス・オブジェクト・ウィザードは、コンテンツ生成メソッドを明示的に呼び出すことはありません。なぜなら、コンテンツ生成 (ファイルのみ) は、このコンテンツ・プロトコル用の ODA によって開始されるからです。	メソッドを直接呼び出してはならない ため、例外をスローする必要があります。コンテンツは実際には、コンテンツ生成メソッドの外部にある別のメソッド、クラス、またはプロセス内で生成されます。

表 36 は、コンテンツ生成メソッドの実装方法に関して、この章で取り上げるトピックを示しています。

表 36. コンテンツ生成メソッドの開発方法

コンテンツ生成メソッド	詳細情報の参照先
`IGeneratesBoDefs.generateBODefs()`	138 ページの『generateBoDefs() メソッドの定義』
`IGeneratesBinFiles.generateBinFiles()`	158 ページの『generateBinFiles() メソッドの定義』

コンテンツとしてのビジネス・オブジェクト定義の生成

4 ページの『ビジネス・オブジェクト定義』で説明したように、*ビジネス・オブジェクト定義* は、集合的単位として扱えるデータ用のテンプレートとなっています。ODA の目的は、データ・ソース内のオブジェクトに対するビジネス・オブジェクト定義を生成することです。 ODA でビジネス・オブジェクト定義コンテンツを生成するには、その ODA クラスに `IGeneratesBoDefs` インターフェースを実装する必要があります。

注: ODA ではビジネス・オブジェクト定義の生成をサポートする必要がある ため、その ODA クラスには `IGeneratesBoDefs` インターフェースを実装する必要があります。

表 37 は、`IGeneratesBoDefs` インターフェースを実装するために ODA クラスに定義する必要のあるメソッドを示しています。

表 37. IGeneratesBoDefs インターフェース内のメソッド

メソッド	IGeneratesBoDefs メソッド	説明
ソース・ノード生成メソッド	getTreeNodes()	次の処理を繰り返し実行します。 • データ・ソース内でオブジェクトのソース・ノードを検出します。 • ソース・ノード階層を表すツリー・ノードの配列を作成します。 • ツリー・ノードの配列をビジネス・オブジェクト・ウィザードに戻します。戻されたツリー・ノードは、「ソースの選択」ダイアログ・ボックスに表示されます。
コンテンツ生成メソッド	generateBoDefs()	ユーザーが選択したソース・データのビジネス・オブジェクト定義を生成し、それを ODA メモリーに書き込みます。
コンテンツ取得メソッド	getBoDefs()	指定したビジネス・オブジェクト定義、またはすべてのビジネス・オブジェクト定義を ODA メモリーから取得します。

注: 表 37 に示すメソッドの他に、IGeneratesBoDefs には getContentProtocol() メソッドも含まれます。このメソッドは、ビジネス・オブジェクト定義を生成するために ODA がサポートするコンテンツ・プロトコルを指定します。詳しくは、 126 ページの『ODA コンテンツ・プロトコルの選択』を参照してください。

IGeneratesBoDefs インターフェースを実装している場合、ビジネス・オブジェクト・ウィザードは、表 38 に示すメソッドを呼び出して、ソース・ノードを取得したり、コンテンツを生成および取得したりします。

表 38. ビジネス・オブジェクト・ウィザードと IGeneratesBoDefs メソッド

ビジネス・オブジェクト・ウィザードのステップ	IGeneratesBoDefs メソッド	詳細情報の参照先
ステップ 3: ソースの選択	getTreeNodes()	『ソース・ノードの生成』
ステップ 5: ビジネス・オブジェクトの生成中	generateBoDefs()	137 ページの『ビジネス・オブジェクト定義の生成』
ステップ 5: ビジネス・オブジェクトの生成中	getBoDefs()	152 ページの『生成済みビジネス・オブジェクト定義へのアクセスの提供』

以降のセクションでは、表 38 に示す各メソッドの実装について説明します。

ソース・ノードの生成

ビジネス・オブジェクト・ウィザードは、ODA データ・ソース内のソース・ノードを検出してソース・ノード階層を作成するために getTreeNodes() メソッドを呼び出します。ビジネス・オブジェクト・ウィザードは、作成したソース・ノード階層を「ソースの選択」ダイアログ・ボックス (ステップ 3) に表示します。

getTreeNodes() メソッドは IGeneratesBoDefs インターフェースに含まれおり、ODA クラスは、ビジネス・オブジェクト定義の生成をサポートするためにこのメソッドを実装する必要があります。

重要: IGeneratesBoDefs インターフェースを実装する際は、ODA に getTreeNodes() メソッドを実装する必要があります。

105 ページの『ソース・データの選択および確認』で説明したように、ビジネス・オブジェクト・ウィザードは、getTreeNodes() が戻すツリー・ノード配列を使用して、「ソースの選択」ダイアログ・ボックスを初期化します。このダイアログ・ボックスにはソース・ノード階層が表示され、ユーザーはデータ・ソースから取得されたソース・ノード間を移動したり、ODA がビジネス・オブジェクト定義を生成するソース・ノードを選択したりできます。ソース・ノードが展開されるたびに、ビジネス・オブジェクト・ウィザードは getTreeNodes() メソッドを呼び出します。これにより、展開されたソース・ノードのコンテンツを持つツリー・ノード配列が戻されます。

例えば、サンプル Roman Army ODA の getTreeNodes() メソッドは、サンプルのデータ・ソースである RomanArmy.xml ファイルから取得したトップレベルの army general を使用して、「ソースの選択」ダイアログ・ボックスを初期化します。特定のノードが展開されると、getTreeNodes() はそのノードの army general の子を XML ファイルから取得し、それをツリー・ノード配列に保管します。ビジネス・オブジェクト・ウィザードは、このツリー・ノード配列を使用して、展開されたソース・ノードを表示します。

したがって、getTreeNodes() の目的は、データ・ソース内のソース・ノードを検出してから、ツリー・ノードの配列を作成して戻すことです。これを行うには、getTreeNodes() で次のタスクを実行する必要があります。

- 『親ノード・パスの決定』
- 132 ページの『検索パターン機能の実装』
- 133 ページの『データ・ソースの照会』
- 134 ページの『ツリー・ノードの組み立て』

親ノード・パスの決定

ビジネス・オブジェクト・ウィザードは、getTreeNodes() メソッドを呼び出すときに、*親ノード・パス* の値をこのメソッドに渡します。このパスは、ユーザーが選択した、getTreeNodes() によって展開されるノードを識別します。このパスは、トップレベルの親からユーザーが選択したノードまでの完全修飾ノード・パスを含む String です。このパス内のノード名は、コロン (:) で区切られています。

例えば、図 59 は、サンプル Roman Army ODA のソース・ノード階層のビューを表示する「ソースの選択」ダイアログ・ボックスを示しています。

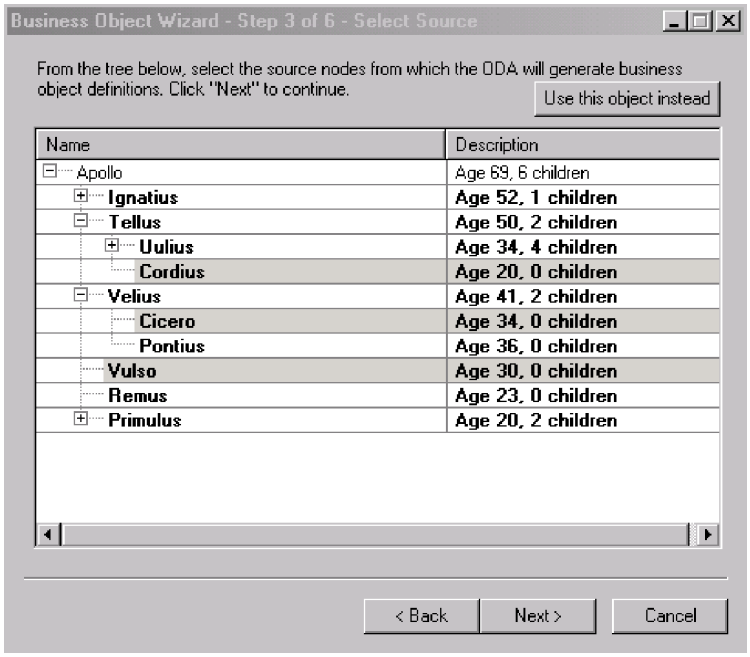


図 59. サンプルのソース・ノード階層

図 59 において、Uulius ソース・ノードの親ノード・パスは次のようになります。

```
Apollo:Tellus:Uulius
```

展開する親ノードは、次のいずれかの方法で指定できます。

- 親ノード名の左側にある + 記号をクリックする。

  ビジネス・オブジェクト・ウィザードは、選択したソース・ノードの親ノード・パスを作成し、このパスを `getTreeNodes()` に渡します。
- 「ソースの選択」ダイアログ・ボックスの上部にある**「代わりにこのオブジェクトを使用」**を選択し、「オブジェクトのパス」ダイアログ・ボックスに親ノード・パスを明示的に指定する。

  親ノード・パスは、`getTreeNodes()` で指定する親ノード・パスと同じ構文を使用して指定する必要があります。詳しくは、94 ページの『オブジェクトのパスの指定』を参照してください。

`getTreeNode()` メソッドは、ツリー・ノード配列に戻すソース・ノード階層のレベルを、親ノード・パスを使用して決定します。このツリー・ノード配列には、親ノード・パスで識別されるノードの子ノードがすべて保管されます。ソース・ノード階層のトップレベルを戻すように `getTreeNodes()` に指示する際、ビジネス・オブジェクト・ウィザードは、「空」の親ノード・パスを渡します。したがって、`getTreeNodes()` メソッドは、最初のステップとして「空」のノード・パスがないかどうかを確認する必要があります。次のコード・フラグメントを参照してください。

```
if (parentNodePath = null || parentNodePath.length() == 0)
    //return the top-level of the source-node hierarchy
```

親ノード・パスが空でない 場合、getTreeNodes() は、指定した親ノード・パスの子のツリー・ノードを構築し、TreeNode オブジェクトの該当する配列をビジネス・オブジェクト・ウィザードに戻す必要があります。

図 60 は、getTreeNodes() メソッド (サンプル Roman Army ODA の ArmyAgent3 クラスで定義) の実装を示しています。

```
public TreeNode[] getTreeNodes(String parentNodePath, String searchPattern)
   throws ODKException
{
   if (parentNodePath == null || parentNodePath.length() == 0)
      return getNodes(m_army, searchPattern);
   return getNodes(findSon(parentNodePath, searchPattern));
}
```

図 60. ツリー・ノード配列の生成

図 60 は、getTreeNodes() メソッドの実装における重要な概念を示しています。このメソッドはモジュラー化されることが多く、データ・ソースの実際の検索は個別のメソッド、または個別のクラスに組み込まれます。この getTreeNodes() メソッドは、選択したデータ・ソース・データのツリー・ノード配列を実際に生成する際に getNodes() メソッドを呼び出します。親ノード・パスが空の場合、getTreeNodes() は、(m_army 変数内の) XML ファイルのコンテンツ全体を getNodes() に送ります。親ノード・パスが空でない場合、getTreeNodes() は、データ・ソースを実際に照会する findSon() メソッドの結果を getNodes() に送ります。

検索パターン機能の実装

検索パターン を使用すると、親ノードを展開したときに、子ノードを表示するために満たす必要のある基準を指定できます。右マウス・ボタンをクリックしてから **「項目を検索」** をクリックして、検索パターン機能を開始します。検索基準を指定するための「検索パターンの入力」ダイアログ・ボックスが開きます。

注: 検索パターン機能の使用方法については、93 ページの『検索パターンの指定』を参照してください。

ビジネス・オブジェクト・ウィザードは、検索パターンを受け取ると、getTreeNode() を再び呼び出して、データ・ソースから新しいツリー・ノード配列を取得します。ビジネス・オブジェクト・ウィザードは、検索パターンを引き数として getTreeNodes() に渡します。検索パターンには、基盤となるデータ・ソースで認識されるワイルドカードやその他の記号が含まれます。例えば、データ・ソースがデータベースである場合、有効な検索基準には、パーセント (%) や疑問符 (?) などの SQL 検索記号を含めることができます。

getTreeNodes() メソッドは、検索パターンに一致する子ノードがデータ・ソースにあるかどうかを検索し、ビジネス・オブジェクト・ウィザードに戻すツリー・ノード配列に結果の子ノードを保管します。このようにして、ソース・ノードで満たす必要のある新しい条件を動的に指定することができます。

注: フィルターとは異なり、検索パターンを使用すると、ビジネス・オブジェクト・ウィザードは getTreeNodes() メソッドを再度呼び出します。フィルターを使用した場合、ビジネス・オブジェクト・ウィザードは現在表示されている

親ノードの子ノードを検索します。つまり、ビジネス・オブジェクト・ウィザードは現在のソース・ノード階層内にすでに存在する子ノードを探します。getTreeNodes() を呼び出して、一致する新しい子ノードがデータ・ソース内に存在するかどうかは検索しません。

ODA に検索パターン機能を実装するには、次のステップを実行します。

- searchableNodes メンバー変数に true を設定することにより、ODA のメタデータ (AgentMetaData) 内の検索パターン機能を使用可能にします。

  また、searchPatternDesc メンバー変数を、有効な検索基準をユーザーに対して記述するストリングで初期化する必要もあります。ビジネス・オブジェクト・ウィザードは、「検索パターンの入力」ダイアログ・ボックスのテキストとしてこのストリングを表示します。ODA のメタデータは、getMetaData() メソッドで初期化します。詳しくは、120 ページの『ODA メタデータの初期化』を参照してください。
- getTreeNodes() メソッドを実装し、データ・ソースの照会で *searchPattern* 引き数の値を使用します。

  例えば、データ・ソースがデータベースである場合は、データベースの表を照会する SQL ステートメントに検索パターンを含めることができます。

サンプル Roman Army ODA では、検索パターンに検索条件として 1 つの文字を入力できます。 getTreeNodes() メソッドは、ツリー・ノードの実際の生成を処理する getNodes() メソッドを呼び出します。次のコード・フラグメントは、この getNodes() メソッド (ArmyAgent3 内で定義) からのものです。これを見ると、メソッドがデータ・ソースを検索する際にどのように検索パターンを使用しているのかがわかります。

```
TreeNode[] getNodes(Son parent, String searchPattern)
{
   Vector nodes = new Vector();

   if (searchPattern == null || searchPattern.length() == 0)
      searchPattern = "";
   else
      searchPattern = new String(new char[] {searchPattern.charAt(0)});
```

XML ファイル (データ・ソース) 内のオブジェクトの名前と親ノード・パス内の現在の名前とを getNodes() メソッドで後で比較するときは、指定された検索パターンでオブジェクトの名前が始まっているかどうかを次のようにして検査します。

```
if (currSon.name.getValue().startsWith(searchPattern))
```

検索パターンをこのように使用する状況については、136 ページの図 61 を参照してください。

データ・ソースの照会

getTreeNodes() メソッドの主な目的は、データ・ソースを照会し、ソース・ノード、つまり、ODA がコンテンツを生成できるオブジェクトを検出することです。データ・ソースを照会する仕組みは、ODA が作業の対象とするデータ・ソースのタイプによって異なります。例えば、XML ODA (WebSphere Business Integration Adapters 製品に含まれる事前ビルド済み ODA) は、XML ファイルを照会し、そのファイル内のコンテンツ生成が可能なオブジェクトの名前を戻します。別の例とし

ては、JDBC ODA (WebSphere Business Integration Adapters に含まれる別の事前ビルド済み ODA) は、JDBC データベースを照会し、データベース内のコンテンツ生成が可能な表の名前を戻します。

114 ページの表 25 で推奨したように、データ・ソースの照会に必要なロジックがかなり複雑な場合は、この対話を処理する特別な Java クラスを開発する必要があります。そうすれば、`getTreeNodes()` メソッドは、これらのクラスを必要に応じてインスタンス化してアクセスできます。これらのクラスは、ODA ライブラリー・ファイルに含めてください。詳しくは、185 ページの『ODA のコンパイル』を参照してください。

サンプル Roman Army ODA では、`findSon()` メソッド (`ArmyAgent3` クラスで定義) はデータ・ソースの照会タスクを実行します。このメソッドは、親ノード・パスによって識別される特定の soldier を Roman-Army XML ファイル内で検索します。このメソッドは、指定された名前の情報を `Son` オブジェクトとして戻します。サンプルでは、XML ファイル内のオブジェクトを読み取る `Son` クラスが定義されています。

ツリー・ノードの組み立て

ODA がソース・ノードについてデータ・ソースを照会するときは、検出した各ソース・ノードを表すための関連するツリー・ノードを生成する必要があります。ODK API は、ツリー・ノードを `TreeNode` クラスのオブジェクトとして表します。ツリー・ノードには、表 39 に示す情報が含まれています。

表 39. ツリー・ノードの内容

メンバー変数	説明
メタデータ	
`name`	このツリー・ノードの名前。「ソースの選択」ダイアログ・ボックスの「名前」列に表示されます。
`description`	このツリー・ノードの説明。「ソースの選択」ダイアログ・ボックスの「説明」列に表示されます。
`polymorphicNature`	ツリー・ノードのタイプが「標準」(展開可能またはリーフ・ノード)、または「ファイル」(ファイルへの関連付けが可能) のどちらであるのか。
`isExpandable`	このツリー・ノードが展開可能であるかどうか。つまり、ツリー・ノードに子ノードが含まれるのか、それともツリー・ノードがリーフ (終了) ノードであるのかどうか。
`isGeneratable`	このツリー・ノードに対してコンテンツを生成できるかどうか。
データ	
`nodes`	子ノードの配列 (このツリー・ノードが展開可能の場合)。

ツリー・ノードを作成するには、いずれかの形式の `TreeNode()` コンストラクターを使用します。これらの形式については、310 ページの『TreeNode()』を参照してください。

標準タイプ・ノード: ノードが「標準」タイプである場合、ノードは次の構造体のいずれかを持つことができます。

- 展開可能ノード

ビジネス・オブジェクト・ウィザードが展開可能ノードを表示するときは、ノード名の左側に正符号 (+) または負符号 (-) が付いています。
正符号 (+) はノードの展開が可能であることを示し、負符号 (-) はノードの縮小が可能であることを示します。次の表は、TreeNode オブジェクトを展開可能ノードとして表示するために必要な初期設定を示しています。

TreeNode メンバー変数	値
isExpandable	true
nodes	子ノードの配列
isGeneratable	false (通常)

ソース・ノード階層内の移動方法については、92 ページの『ソース・ノード階層内での移動』を参照してください。

- リーフ・ノード

  ビジネス・オブジェクト・ウィザードは、リーフ (終了) ノードを単にノード名で表示します。次の表は、TreeNode オブジェクトをリーフ・ノードとして表示するために必要な初期設定を示しています。

TreeNode メンバー変数	値
isExpandable	false
nodes	null
isGeneratable	true (通常)

リーフ・ノードと展開可能ノードはどちらも標準タイプ・ノードです。したがって、両方のノードの polymorphicNature メンバー変数には、NODE_NATURE_NORMAL ノード・タイプ定数を設定する必要があります。この定数は、(TreeNode クラスが実装する) ODKConstant インターフェース内で定義されています。TreeNode() コンストラクターの最初の 2 つの形式では、polymorphicNature メンバー変数は指定しません。したがって、このメンバー変数は、NODE_NATURE_NORMAL にデフォルト設定されています。

131 ページの図 59 に示すソース・ノード階層を ODA が生成するものと想定しましょう。ユーザーが Uulius ノードを展開した場合、getTreeNodes() メソッドは、Uulius の子ノードを含むツリー・ノード配列を生成する必要があります。親ノード・パスは Apollo:Tellus:Uulius であり、空ではない ため、getTreeNodes() メソッドは、次のように getNodes() メソッドを呼び出します (132 ページの図 60 を参照)。

```
getNodes(findSon(parentNodePath), searchPattern))
```

この getNodes() の呼び出しでは、findSon() メソッドを使用してデータ・ソースに Uulius ノードを照会し、XML ファイルからの情報を含む Son オブジェクトを戻します。この Son オブジェクトのメンバー変数の 1 つは、Uulius の子に関する情報を含む XML オブジェクトのベクトル (XmlObjectVector) です。図 61 は、この XML オブジェクト・ベクトルをループ処理し、それぞれの子ごとに TreeNode オブジェクトを生成する getNodes() メソッドからのコード・フラグメントを示しています。

```
for (int i=0; i<sons.size(); i++)
   {
   Son currSon = (Son) sons.getAt(i);
   if (currSon.name.getValues().startsWith(searchPattern))
      {
      int age = currSon.age.getIntValue();
      int children = currSon.Son == null ? 0 : currSon.Son.size();
      int nature = TreeNode.NODE_NATURE_NORMAL;

      TreeNode tn = new TreeNode(currSon.name.getValue(), " ",
         canRecruit(currSon), children > 0, null, nature);

      nodes.add(tn);
      }
   }
```

図 61. ツリー・ノードの組み立て

図 61 に示すコード・フラグメントは、それぞれの子 soldier ノードの新しい TreeNode オブジェクトを soldier の名前で、このノードが生成可能であるかどうか (soldier が従軍可能な年齢であるかどうかに基づく)、展開可能であるかどうか (この soldier が所有する子の数に基づく) を設定して初期化します。この TreeNode() コンストラクターの呼び出しでは、ツリー・ノードは説明 (“”) で初期化されず、どの子ノードも戻されません。それぞれの新しい TreeNode オブジェクトがインスタンス化されると、コードはそれを Java Vector (nodes) に追加します。 getNodes() によってすべての 子ノードの TreeNode オブジェクトが生成されると、このベクトルの内容が次のコードでツリー・ノード配列にコピーされます。

```
TreeNode[] tn = new TreeNode[nodes.size()];
System.arraycopy(nodes.toArray(), 0, tn, 0, nodes.size());
```

getNodes() メソッドは、このツリー・ノード配列を getTreeNodes() 経由で呼び出し側プログラムのビジネス・オブジェクト・ウィザードに戻します。ビジネス・オブジェクト・ウィザードは、この新しいツリー・ノード配列を使用して、展開された Uulius ノードのコンテンツを表示します (図 62 を参照)。

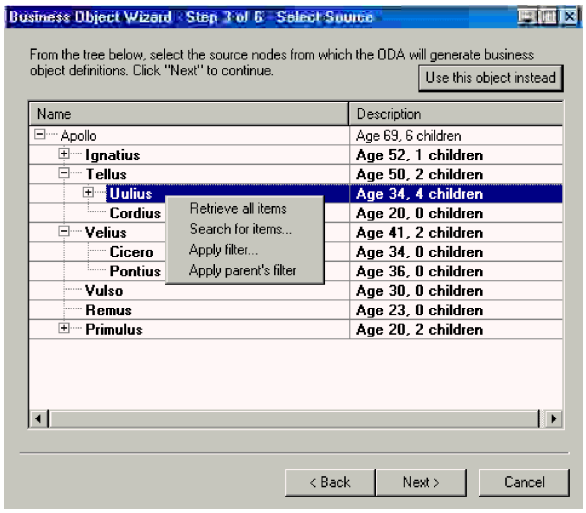


図 62. Uulius ソース・ノードの展開

ファイル・タイプ・ノード: ノードが「ファイル」タイプの場合は、オペレーティング・システム・ファイルとノードを関連付けることができます。ユーザーがノー

ド名を右マウス・ボタンでクリックすると、ビジネス・オブジェクト・ウィザードは「関連付けられたファイル」メニュー項目を活動化することにより、ノードがファイル・タイプであることを示します。このメニュー項目の使用方法については、95 ページの『オペレーティング・システム・ファイルの関連付け』を参照してください。

ファイル・タイプ・ノードの polymorphicNature メンバー変数には、NODE_NATURE_FILE ノード・タイプ定数が設定されています。この定数は、(TreeNode クラスが実装する) ODKConstant インターフェース内で定義されています。次の表は、TreeNode オブジェクトがファイル・タイプ・ノードとして機能するために必要な初期設定を示しています。

TreeNode メンバー変数	値
polymorphicNature	NODE_NATURE_FILE
isExpandable	false
nodes	null
isGeneratable	false (通常)

サンプル Roman Army ODA で ArmyAgent4 クラスは、ファイル・タイプ・ノードをサポートする getNode() メソッドを実装しています。このサンプルでは、28 歳 (デフォルトの最小年齢) 以上で自分の子を持たない soldier を表す任意のノードのソース・ノードに対して、ファイルを関連付けることができます。このバージョンの getNode() のコードは、136 ページの図 61 に示すコードとほぼ同じです。唯一の違いは、polymorphicNature メンバー変数への値の代入にあります。ノードが 28 歳以上で子を持たない soldier を表している場合、ArmyAgent4 バージョンの getNode() は、NODE_NATURE_NORMAL 定数をすべての ノードに割り当てる代わりに、次のコード行を使用してノード・タイプに NODE_NATURE_FILE を設定します。

```
int nature = m_allowAdoption && canAdopt(currSon) ?
   TreeNode.NODE_NATURE_FILE : TreeNode.NODE_NATURE_NORMAL;
```

ビジネス・オブジェクト定義の生成

ユーザーが「ノードを選択」ダイアログ・ボックス (ステップ 3) でソース・ノードを選択すると、ODA は、コンテンツ生成を開始する準備が整います。ビジネス・オブジェクト・ウィザードは、generateBoDefs() コンテンツ生成メソッドを呼び出して、ユーザーが選択したソース・ノードのビジネス・オブジェクト定義を生成します。ビジネス・オブジェクト・ウィザードは、(ステップ 3 で選択された) ソース・ノードのリストを ODA に送ります。ビジネス・オブジェクト定義の生成プロセスの目的は、選択された各ソース・ノードごとにビジネス・オブジェクト定義を作成することです。 generateBoDefs() メソッドの実行中、ビジネス・オブジェクト・ウィザードは「ビジネス・オブジェクトの生成中」画面 (ステップ 5) を表示します。

注: ODA は「要求時」にビジネス・オブジェクト定義を生成するため、ビジネス・オブジェクト・ウィザードは generateBoDefs() メソッドを明示的に呼び出して、ビジネス・オブジェクト定義の生成を開始します。したがって、generateBoDefs() は、ビジネス・オブジェクト定義 (BusObjDef オブジェクト)

を生成し、それを生成済みコンテンツ構造体に保管して、コンテンツ・メタデータをビジネス・オブジェクト・ウィザードに戻すような処理を行うように実装する必要があります。

このセクションでは、ビジネス・オブジェクト定義を生成する際に generateBoDefs() メソッドで実行する必要のある次のステップについて説明します。

1. 『generateBoDefs() メソッドの定義』
2. 『ビジネス・オブジェクト・プロパティーの要求』
3. 142 ページの『ビジネス・オブジェクト定義の作成』
4. 150 ページの『生成済みのビジネス・オブジェクト定義の提供』

generateBoDefs() メソッドの定義

ビジネス・オブジェクト定義の生成を可能にするには、IGeneratesBoDefs インターフェース内で定義されている generateBoDefs() メソッドを、(ODKAgentBase2 から派生した) ODA クラスに実装する必要があります。 generateBoDefs() メソッドは、ユーザーが選択したこれらのソース・ノードを引き数、つまりソース・ノード・パスの配列 (String オブジェクト) として受け取ります。このメソッドは、この配列内の各ソース・ノードごとにビジネス・オブジェクト定義を生成する必要があります。また、このメソッドは、そのパスを使用してデータ・ソース内のソース・ノードを特定することができます。最後のステップとして、generateBoDefs() は、生成したビジネス・オブジェクト定義を記述するコンテンツ・メタデータ (ContentMetaData) オブジェクトを戻します。

サンプル Roman Army ODA は、ビジネス・オブジェクト定義を生成するために、要求時コンテンツ・プロトコルをサポートします (127 ページの図 58 を参照)。ArmyAgent3 クラス内のこの generateBoDefs() メソッドの実装には、図 63 に示すコード・フラグメントを含めます。このコード・フラグメントは、生成済みコンテンツ構造体 (m_generatedBOs) の変数を宣言し、generateBoDefs() メソッド自体を定義します。

```
final Vector m_generatedBOs = new Vector();

public ContentMetaData generateBoDefs(String[] nodes)
    throws ODKException
{
```

図 63. generateBoDefs() メソッドの定義

ビジネス・オブジェクト・プロパティーの要求

コンテンツ生成プロセスの間に ODA で追加情報が必要となった場合は、「BO プロパティー」ダイアログ・ボックスを表示して、ビジネス・オブジェクト・プロパティー の値を入力するようユーザーに求めることができます。ビジネス・オブジェクト・プロパティーの概要については、108 ページの『ビジネス・オブジェクト・プロパティーの取得』を参照してください。

図 64 は、2 つのビジネス・オブジェクト・プロパティーを表示しているサンプルの「BO プロパティー」ダイアログ・ボックスを示しています。

- Verbs ビジネス・オブジェクト・プロパティーでは、ビジネス・オブジェクト定義でサポートされる動詞を指定できます。このプロパティーには有効な動詞のドロップダウン・リストが用意されており、そこから 1 つまたは複数の値を選択できます。
- Prefix ビジネス・オブジェクト・プロパティーでは、生成されるすべてのビジネス・オブジェクト定義の名前に追加するプレフィックス (JDBC、SAP、LegacyApp など) を入力できます。このプロパティーには空のフィールドしか用意されておらず、そのフィールドにプレフィックスのストリングを指定します。

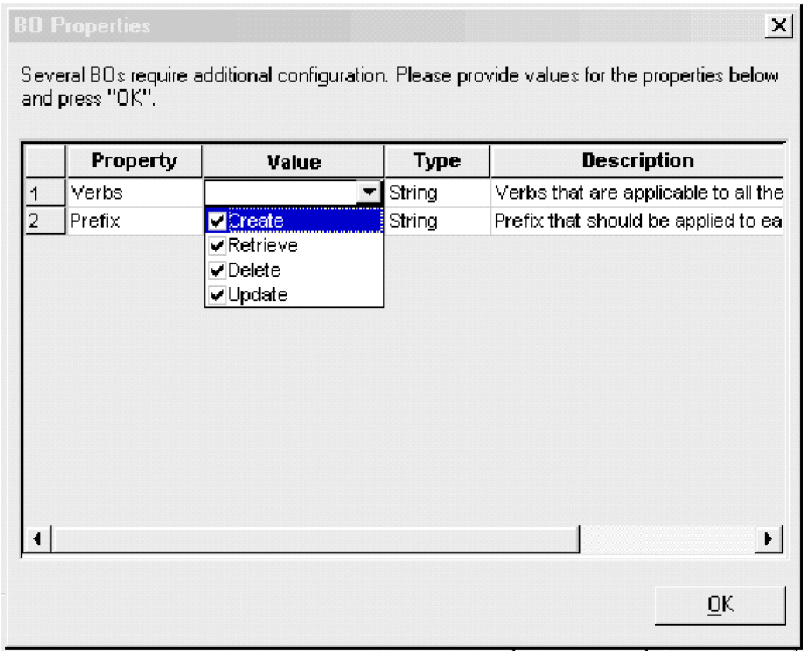


図 64. 必要な追加プロパティー情報

図 64 に示すプロパティーをユーザーに提供するために、generateBoDefs() メソッドは次のステップを実行します。

1. Verbs プロパティーおよび Prefix プロパティー用のビジネス・プロパティー配列を作成します。
2. getBOSpecificProps() メソッドを呼び出し、ビジネス・オブジェクト・プロパティーを表示します。
3. ユーザーが初期化したビジネス・オブジェクト・プロパティー値を取得します。

ビジネス・プロパティー配列の作成: getBOSpecificProps() メソッドは引き数として、エージェント・プロパティー・オブジェクトの配列を必要とします。この引き数は、*ビジネス・オブジェクト・プロパティー配列* であり、「BO プロパティー」ダイアログ・ボックスに表示されるそれぞれのビジネス・オブジェクト・プロパティーごとに、1 つのエージェント・プロパティー・オブジェクトが保持されています。 generateBoDefs() は、getBOSpecificProps() を呼び出す前のステップとして、ビジネス・オブジェクト・プロパティーを定義する配列を作成し、ビジネス・オブジェクト・プロパティーを初期化して、それらのプロパティーを配列に保管する必要があります。

最初のステップでは、Verbs プロパティーと Prefix プロパティーを保持するビジネス・オブジェクト・プロパティー配列を定義します。次のステップでは、AgentProperty() コンストラクターを使用して、ビジネス・オブジェクト・プロパティーを初期化します。このコンストラクターでは、AgentProperty クラスがサポートする各種のメタデータの値を指定します。AgentProperty クラスは、ビジネス・オブジェクト・プロパティーが次の機能を持つためのサポートを提供します。

- デフォルト値
- 値を単一値に制限したり、複数値を許可する機能
- ユーザーが選択する有効な値のリスト
- 入力可能な値を制限する条件

注: 詳しくは、 163 ページの『エージェント・プロパティーの使用』を参照してください。

Verbs プロパティーと Prefix プロパティーを初期化するには、AgentProperty() コンストラクターに次の情報を指定します。

- Verbs プロパティーは、選択対象の値を複数提供する複数カーディナリティー・プロパティーである。デフォルト値もあります。

  したがって、このプロパティーは、AgentProperty() コンストラクター内で次のメタデータを必要とします。

メタデータ	AgentProperty メンバー変数	値
複数カーディナリティー ユーザーは複数の値の中から選択できる	cardinality isMultiple	ODKConstant.MULTIPLE_CARD true
	allValidValues	validValues 配列 (表示する有効な値を含む)
デフォルト値がある	allDefaultValues	defaultValues 配列 (表示するデフォルト値を含む)
ユーザーは値を入力する必要がない	isRequired	false

  AgentProperty() コンストラクターを呼び出す前に、 141 ページの図 65 に示すコードによって最初に validValues 配列と defaultValues 配列が作成および初期化され、それらがコンストラクターで使用できるようになります。

- Prefix プロパティーは、選択対象の値を複数表示しない 単一カーディナリティー・プロパティーである。デフォルト値はありません。

  したがって、このプロパティーは、AgentProperty() コンストラクター内で次のメタデータを必要とします。

メタデータ	AgentProperty メンバー変数	値
単一カーディナリティー ユーザーは複数の値の中から選択できない	cardinality isMultiple	ODKConstant.SINGLE_CARD false

メタデータ	AgentProperty メンバー変数	値
デフォルト値はない ユーザーは値を入力する必要がない	allValidValues allDefaultValues isRequired	null null false

図 65 に示すコード・フラグメントは、ビジネス・オブジェクト・プロパティー配列を作成して初期化します。

```
// Create the business-object-property array
   AgentProperty AgtProps[] = new AgentProperty[2];

  // Provide list of valid values for Verbs property
   Object[] validValues = new Object[4];
   validValues[0] = new String("Create");
   validValues[1] = new String("Retrieve");
   validValues[2] = new String("Delete");
   validValues[3] = new String("Update");

 // Provide list of default values for Verbs property
   Object[] defaultValues = new Object[4];
   defaultValues[0] = new String("Create");
   defaultValues[1] = new String("Retrieve");
   defaultValues[2] = new String("Delete");
   defaultValues[3] = new String("Update");

// Instantiate the Verbs property
   AgtProps[0] = new AgentProperty("Verbs", AgentProperty.TYPE_STRING,
         "Verbs that are applicable to all the selected objects",
         false, true, ODKConstant.MULTIPLE_CARD, validValues,
         defaultValues);

// Instantiate the Prefix property
   AgtProps[1] = new AgentProperty("Prefix", AgentProperty.TYPE_STRING,
         "Prefix that should be applied to each business object name",
         false, false, ODKConstant.SINGLE_CARD, null, null);
```

図 65. ビジネス・オブジェクト配列の作成

ビジネス・オブジェクト・プロパティーのメタデータについては、163 ページの『エージェント・プロパティーの使用』を参照してください。

「BO プロパティー」ダイアログ・ボックスの表示: ビジネス・オブジェクト・プロパティー配列を初期化すると、ODA は getBOSpecificProps() メソッドを呼び出してこの配列をビジネス・オブジェクト・ウィザードに渡し、「BO プロパティー」ダイアログ・ボックスに表示されるようにします。このメソッドは ODKUtility クラス内で定義されているため、ODKUtility オブジェクトへのアクセスが必要となります。通常、このオブジェクトは、ODA 初期設定の一環としてインスタンス化します。詳しくは、118 ページの『ODKUtility オブジェクトへのハンドルの取得』を参照してください。

注: 各プロパティー値が無効である場合、getBOSpecificProps() は ODKInvalidPropException 例外をスローします。

図 66 に示す getBOSpecificProps() の呼び出しでは、(図 65 で初期化された) AgtProps 配列 がビジネス・オブジェクト・ウィザードに送られ、「BO プロパティー」ダイアログ・ボックスに表示されます。

```
// Display BO Properties dialog box, initializing it with AgtProps
   Util.getBOSpecificProps(AgtProps, "For all the Tables selected");
```

図 66. 「BO プロパティー」ダイアログ・ボックスの表示

ユーザー指定値の取得: ユーザーが「BO プロパティー」ダイアログ・ボックスでビジネス・オブジェクト・プロパティーの値を指定して**「次へ」**をクリックすると、ビジネス・オブジェクト・ウィザードはユーザー指定値を ODA に戻します。ODA は、これらの値を次のいずれかの方法で取得できます。

- ビジネス・オブジェクト・ウィザードはユーザー指定値を Java Hashtable オブジェクトに保管し、getBOSpecificProps() の戻り値としてこのオブジェクトを送ります。各プロパティーは、この Hashtable オブジェクト内で自身の名前をキーとして持ちます。 ODA は、Hashtable メソッドを使用してこれらのプロパティーにアクセスできます。プロパティーのユーザー指定値は、そのエージェント・プロパティー (AgentProperty) オブジェクトの allValues メンバー変数に入っています。
- ビジネス・オブジェクト・ウィザードは、ユーザー指定値を ODA ランタイム・メモリーに書き込みます。ODA は、ODKUtility クラス内の getBOSpecificProperty() メソッドまたは getAllBOSpecificProperties() メソッドを使用して、これらの値にアクセスできます。

図 66 に示す getBOSpecificProps() の呼び出しでは、ビジネス・オブジェクト・ウィザードが作成する Hashtable オブジェクトが保管されませんでした。したがって、このコード・フラグメントは getBOSpecificProperty() メソッドを使用して、動詞および各ビジネス・オブジェクト・プレフィックスに指定されたプロパティーの値を取得します。

```
// Get the value of the Verbs and the Prefix properties
AgentProperty propVerb =
      Util.getBOSpecificProperty("Verbs");
AgentProperty propPrefix =
      Util.getBOSpecificProperty("Prefix");
```

ビジネス・オブジェクト定義の作成

generateBoDefs() メソッドは、ビジネス・オブジェクト・ウィザードから引き数として受け取った配列内の各ソース・ノードごとに、ビジネス・オブジェクト定義を生成する必要があります。ビジネス・オブジェクト定義を生成するために、generateBoDefs() は次のステップを実行する必要があります。

1. generateBoDefs() がビジネス・オブジェクト・ウィザードから受け取った配列のソース・ノード・パスを使用して、データ・ソース内の関連するオブジェクトを特定します。
2. ビジネス・オブジェクト定義にデータを設定するために必要な情報を、データ・ソース内の関連オブジェクトから取得します。
3. ソース・ノードを表すビジネス・オブジェクト定義オブジェクトを生成します。
4. データ・ソース内の関連オブジェクトから取得した情報を使用して、このビジネス・オブジェクト定義オブジェクトにデータを設定します (ステップ 2)。

ODK API は、ビジネス・オブジェクト定義をビジネス・オブジェクト定義 (BusObjDef) オブジェクトとして表します。 BusObjDef() コンストラクターを使用すると、新しいビジネス・オブジェクト定義をインスタンス化し、それに名前を提供することができます。そうすれば、表 40 に示す情報をビジネス・オブジェクト定義に提供できます。

表 40. ビジネス・オブジェクト定義の内容

ビジネス・オブジェクト定義の情報	説明	accessor メソッド
メタデータ		
名前	ビジネス・オブジェクト定義の名前	getName()
アプリケーション固有の情報	ビジネス・オブジェクト・レベルのアプリケーション固有の情報。ビジネス・オブジェクト定義全体に適用される情報が含まれます。	getAppInfo()、setAppInfo()
データ		
属性リスト	ビジネス・オブジェクト定義内の属性のリスト。各属性は BusObjAttr オブジェクトです。	getAttributeList()、setAttributeList()、insertAttribute()、removeAttribute()
動詞リスト	ビジネス・オブジェクト定義内のサポートされる動詞のリスト。各動詞は BusObjVerb オブジェクトです。	getVerbList()、setVerbList()、insertVerb()、removeVerb()

表 40 に示すように、ビジネス・オブジェクト定義にはメタデータとデータの両方が含まれています。以降のセクションでは、ビジネス・オブジェクト定義のこれらの部分にアクセスする方法について説明します。

- 『ビジネス・オブジェクト定義のメタデータの定義』
- 145 ページの『属性の生成』
- 149 ページの『サポートされる動詞の提供』

ビジネス・オブジェクト定義のメタデータの定義: 表 40 に示すように、ビジネス・オブジェクト定義のメタデータは、次の情報から構成されます。

- ビジネス・オブジェクト定義の名前
- アプリケーション固有の情報 (ビジネス・オブジェクト定義レベルでの)

ビジネス・オブジェクト定義の命名: generateBoDefs() メソッドは、ユーザーが選択したソース・ノードのリストを引き数として受け取ります。このリストは、ユーザーが選択したソース・ノードのノード・パスを含む String オブジェクトの配列です (ノード・パスについては、130 ページの『親ノード・パスの決定』を参照してください)。この配列を使用して、ODA は各ソース・ノードに関連付けられているビジネス・オブジェクト定義の名前を適切に作成する必要があります。通常 ODA は、ビジネス・オブジェクト定義の名前はソース・ノードが表すデータ・ソース・オブジェクトの名前に一致する (またはそれに基づく) と想定しています。ODA は、ソース・ノード・パスの構文を解析してソース・ノードの名前を取得し、

このソース・ノード名を使用して関連するソース・データ・オブジェクトを特定し、このソース・データ・オブジェクトから名前を取得する必要があります。

例えば、Roman Army サンプルでは、データ・ソース・オブジェクトの名前とビジネス・オブジェクト定義の名前は同じです。したがって、サンプル・コードは、findSon() メソッド (ArmyAgent3 クラスと ArmyAgent4 クラスで定義) を呼び出し、ソース・ノード (nodes) の入力配列内に格納されているソース・ノードのノード・パスを使用して、ソース・ノードが表すデータ・ソース・オブジェクトを取得します (次の例を参照)。

```
for (int i=0; i<nodes.length; i++)
   {
   Son sonNode = findSon(nodes[i]);
   BusObjDef sonBo = new BusObjDef(sonNode.name.getValue());
   ...
```

注: BusObjDef() コンストラクターは、どの形式のものでもビジネス・オブジェクト定義の名前を指定します。

findSon() メソッドは、ソース・ノード・パスの構文を解析し、パス内の最後のノード名を取得します。

別の例として、データ・ソースがデータベースであり、そのソース・ノードが表を表しているものと仮定しましょう。ソース・ノード・パスにスキーマ名 (*schema*:*table*) が含まれる場合、ODA はソース・ノード・パスの構文を解析し、対応するビジネス・オブジェクト定義に表名のみを割り当てる必要があります。 (構成変数を持つ) ビジネス・オブジェクト定義にユーザーが指定したプレフィックスが ODA でサポートされる場合、ODA は、このプレフィックスを前に付加してから、BusObjDef() コンストラクターを呼び出し、ビジネス・オブジェクト定義オブジェクトを生成する必要があります (次のコード・フラグメントを参照)。

```
AgentProperty propPrefix = getBOSpecificProperty("Prefix");
for (int i=0; i<names.length; i++)
   {
   strToken = new StringTokenizer(names[i], ":");
   schemaName = strToken.nextToken();
   tableName = strToken.nextToken()

   if (propPrefix.allValues != null && propPrefix.allValues[0] != null)
      boDef = new BusObjDef(propPrefix.allValues[0] + tableName);
   else
      boDef = new BusObjDef(tableName);
   ...
```

ビジネス・オブジェクト定義に割り当てたい正確な名前がデータ・ソース・オブジェクトにない 場合、ODA はその名前を必要に応じて構文解析するか、何らかの方法でフォーマットする必要があります。

ビジネス・オブジェクトのアプリケーション固有の情報の生成: 8 ページの『ビジネス・オブジェクトのアプリケーション固有の情報』で説明したように、アプリケーション固有の情報は、アプリケーション固有の処理情報をビジネス・オブジェクト定義に保管するための有効な手段となります。処理プログラム (コネクターなど) からこの情報を移動すれば、処理プログラムをメタデータ主導型にすることができます。つまり、処理プログラムは、より一般的な手法で作成することが可能となり、アプリケーション固有の処理命令をビジネス・オブジェクト定義から取得できるようになります。したがって、ビジネス・オブジェクト定義をメタデータ主導型

の処理プログラムとともに使用する場合は、ビジネス・オブジェクト、属性、および動詞の各レベルのアプリケーション固有の情報を、適切にフォーマットしてビジネス・オブジェクト定義に含める必要があります。

注: 属性のアプリケーション固有の情報については、『属性の生成』を参照してください。動詞のアプリケーション固有の情報については、149 ページの『サポートされる動詞の提供』を参照してください。

Roman Army サンプルが生成するビジネス・オブジェクト定義は、アプリケーション固有の情報を提供しません。ただし、データ・ソースがそのソース・ノードとして表を持つデータベースであると仮定します。ODA は、ユーザーが選択した表ごとにビジネス・オブジェクト定義を生成します。表の名前は、ビジネス・オブジェクト・レベルのアプリケーション固有の情報として、このビジネス・オブジェクト定義に含めることができます。次のコード・フラグメントは、BusObjDef クラスで定義されている setAppInfo() メソッドを使用して、このビジネス・オブジェクト・レベルのアプリケーション固有の情報に対応する名前と値のペアを適切に作成します。

```
boDef.setAppInfo("TN=" + tableName + ";SCN=" + schemaName +";");
```

このコードは、TN および SCH の名前と値のペアを作成して、表名とスキーマ名をそれぞれ表します。この表名とスキーマ名は、要素の命名に使用するタグを用いて連結されます。次にこのコードは、setAppInfo() メソッドを使用して、このストリング全体をビジネス・オブジェクト・レベルのアプリケーション固有の情報として割り当てます。

属性の生成: ビジネス・オブジェクト定義には、*属性* が含まれます。属性は、ビジネス・オブジェクト定義で表されるオブジェクトを記述するものです。ビジネス・オブジェクト定義は、その*属性リスト* 内に属性を保持しています。ODK API は、属性を属性 (BusObjAttr) オブジェクトとして表します。属性オブジェクトをインスタンス化するには、BusObjAttr() コンストラクターを使用します。

表 41 は、属性オブジェクトのプロパティーをまとめたものです。これらのプロパティーは、属性メタデータに対応しています。

表 41. 属性のプロパティー

属性プロパティー	説明	accessor メソッド
名前	属性の名前	getName()、setName()
アプリケーション固有の情報	属性レベルのアプリケーション固有の情報。属性に適用される情報が含まれます。	getAppText()、setAppText()
Type	属性値のデータ型	getAttrType()、getAttrTypeName()、setAttrType()
Cardinality	属性のカーディナリティー。属性が保持する値の数を識別します。	getCardinality()、setCardinality()
Default value	ユーザーが値を入力する前に属性に割り当てられる値。	getDefault()、setDefault()
Maxlength	属性値の最大長	getMaxLength()、setMaxLength()

表 41. 属性のプロパティー (続き)

属性プロパティー	説明	accessor メソッド
Comments	属性の目的を記述するコメント (オプション)	getComments()、setComments()
Relationship type	属性が参加する関係のタイプを識別するストリング	getRelationType()、setRelationType()
Primary key	属性が基本キーの一部であるかどうか	isKey()、setIsKey()
Foreign key	属性が外部キーの一部であるかどうか	isForeignKey()、setIsForeignKey()
Required key	属性が必須であるかどうか	isRequiredKey()、setIsRequiredKey()

重要

ビジネス・オブジェクト定義の生成プロセスでは、ObjectEventId 属性が自動的に作成されます。ビジネス・オブジェクト・ウィザードがビジネス・オブジェクト定義をファイルに保管する場合は、リポジトリーのバージョンがこのファイルの先頭に自動的に追加されます。リポジトリーのバージョンは、統合ブローカーが InterChange Server である場合に必要となります。

サンプル Roman Army ODA では、各ビジネス・オブジェクト定義はローマの兵士 (soldier) を表します。 generatesBoDefs() メソッドは、ビジネス・オブジェクト定義に対して次の属性を作成します。

- Age 属性。ローマの兵士 (soldier) の年齢を保持します。
- ChildNo 属性。soldier が所有する子 (養子を含む) の数を保持します。

図 67 は、ビジネス・オブジェクト定義に対してこれらの属性オブジェクトを生成するコード・フラグメントを示しています。

```
// 1. Create an attribute object for Age attribute
   BusObjAttr attr = new BusObjAttr("Age", BusObjAttrType.INTEGER,
      BusObjAttrType.AttrTypes[BusObjAttrType.INTEGER]);
//    Set the Age attribute as the business object definition's key
   attr.setIsKey(true);
//    Add the attribute to the business object definition's attribute list
   sonBo.insertAttribute(attr);
// 2. Create an attribute object for ChildNo attribute
   attr = new BusObjAttr("ChildNo", BusObjAttrType.INTEGER,
      BusObjAttrType.AttrTypes[BusObjAttrType.INTEGER]);
//    Set the default value to number of children
   attr.setDefault(sonNode.Son == null ? "0" : "" + sonNode.Son.size());
//    Add the attribute to the business object definition's attribute list
   boDef.insertAttribute(attr);
```

図 67. 属性の生成

Age 属性を作成する際、図 67 に示すコード・フラグメントは次のステップを実行します。

1. BusObjAttr() コンストラクターを使用して Age 属性オブジェクト (attr) を作成します。ここでは、このコンストラクターの形式を使用して属性オブジェクトをその名前、タイプ、およびタイプ名で初期化します。

   コードは、タイプを初期化する際に Integer (BusObjAttrType.INTEGER) の属性タイプ定数を指定します。タイプ名を初期化する際は、BusObjAttrType インターフェース内の AttrTypes メンバー変数を使用します。この静的メンバー変数は、サポートされるすべての属性タイプについてタイプ名を提供し、属性タイプ定数によってインデックスを付けることができます。このようにして、タイプ名ストリングをハードコーディングすることなく、タイプ名を割り当てることができます。

2. setIsKey() メソッドを使用して、Primary key プロパティーに true を明示的に設定します。

   この形式の BusObjAttr() コンストラクターで指定する属性プロパティーは 3 つだけであるため、その他の属性プロパティーはすべて「未定義」としてデフォルト設定されます。したがって、BusObjAttr() を呼び出した後、Primary key 属性プロパティーは false になります。 Age 属性がキー属性である ことを示すために、コード・サンプルは setIsKey() を呼び出します。

3. BusObjDef クラスで定義されている insertAttribute() を使用して、Age 属性をビジネス・オブジェクト定義の属性リストに追加します。

146 ページの図 67 に示すコード・フラグメントは、これらの基本ステップを繰り返して ChildNo 属性を生成します。主な違いは、ChildNo はキー属性ではない ために setIsKey() の呼び出しが不要であることです。ただし、コード・フラグメントは setDefault() メソッドを呼び出して、この属性のデフォルト値を提供します。

Roman Army サンプルが生成するビジネス・オブジェクト定義は非常に単純です。ビジネス・オブジェクト定義内に存在する属性は 2 つだけであり、それらの名前はコンパイル時に既知です。また、設定する必要のある属性プロパティーは数個しかありません。もう少し複雑な例として、データ・ソースが、そのソース・ノードとしての表を持ち、またそのビジネス・オブジェクト定義の名前としての表を持つデータベースであると仮定しましょう。この場合、データベースの列は、ビジネス・オブジェクト定義の属性に対応します。これらの属性に対して、さらに多くの属性プロパティーを設定する必要があります。

次のコード・フラグメントは、データベースの表の列の属性を作成します。

```
Vector Attributes;

// 1. Retrieve columns from database table into 'rst' result set
try{
   ResultSet rst = null;

   // Retrieve columns from database
   rst = db.dbmd.getColumns(null, schemaName, tableName, "%");

   String colName = null;
   String colType = null;
   int cType = 0;
   int colSize = 0;
```

```
   // Obtain next column from result set
   rst.next();
   do{
      // Get column name & type
      colName = rst.getString(4);
      colType = rst.getString("DATA_TYPE");

      // Convert database types to supported types.
      // Load converted types into the cType variable
      // (steps not shown)

// 2. Create an attribute object for each column in the result set.
      Attributes = new Vector(1, 10);

      try
        {
         // Create attribute object for column
         BusObjAttr attrib = new BusObjAttr(colName, cType);

         // Set the cardinality and maxLength attribute properties
         attrib.setCardinality(BusObjAttr.CARD_SINGLE);
         colSize = rst.getInt("COLUMN_SIZE");
         attrib.setMaxLength(colSize);

         // Determine whether it is a primary key in the table: compare
         // column name against earlier retrieve of table's primary keys
         // (stored in pKeys -- code not included here)
         if (pKeys.contains(colName))== true {
            attrib.setIsKey(true);
         }else
            attrib.setIsKey(false);

         // Determine whether it is a foreign key in the table: compare
         // column name against earlier retrieve of table's primary keys
         // (stored in fKeys -- code not included here)
         if (fKeys.contains(colName))== true {
            attrib.setIsForeignKey(true);
         }else
            attrib.setIsForeignKey(false);

         // Set the isRequired property
         if ((rst.getString("IS_NULLABLE").equals("NO")) &&
               (attrib.isKey() != true)){
            attrib.setIsRequiredKey(true);
         }

        // Create attribute application-specific information:
        // CN tag provides column name
         String asi = "CN="+colName;
         attrib.setAppText(asi);
         attrib.setDefault("");

        // Add attribute object to Attributes vector
         Attributes.add(attrib);
         ...

// 3. Save the attribute vector as the business object definition's attribute list
        boDef.setAttributeList(Attributes);
```

このプロセスにおけるステップは、次のとおりです。

1. BusObjAttr() コンストラクターを使用して、列情報から単純なビジネス・オブジェクト属性を作成します。この形式のコンストラクターは、属性の名前とタイプのみを指定します。
2. データベースの列からのこれらの値に基づいて、Cardinality および maxLength 属性プロパティーを設定します。

注: 子ビジネス・オブジェクトまたは子ビジネス・オブジェクトの配列を表す属性を作成するには、その子ビジネス・オブジェクトの名前を属性のタイプとして指定し、カーディナリティーを 1 または n のうち該当する方に設定します。 例えば、OrderLineItems ビジネス・オブジェクトの配列を表す LineItems という名前の属性を作成するには、次のコードを使用します。

```
BusObjAttr attrib = new BusObjAttr(LineItems, OrderLineItems);
attrib.setCardinality(BusObjAttr.CARD_MULTIPLE);
```

3. 基本キーおよび外部キーに関する情報を取得し、基本キーまたは外部キーを表す属性を設定します。コード・フラグメントは、データベースから選択した基本キー列 (pKey) および外部キー列 (fKey) を含む既存の配列内の名前と現在の列の名前とを比較します。基本キー列と外部キー列を選択するコードはここでは示しません。
4. 属性が基本キーであるかどうかに基づいて「is required key」属性プロパティーを設定します。
5. 属性レベルのアプリケーション固有の情報を設定します。

   データベースの表に対して生成されたビジネス・オブジェクト定義の場合は、属性レベルのアプリケーション固有の情報として、列名を各属性ごとにビジネス・オブジェクト定義に含めることができます。このコード・フラグメントは、BusObjAttr クラスで定義されている setAppText() メソッドを使用して、属性レベルのアプリケーション固有の情報に対応した名前と値のペア CN を作成します。コードは、列名と CN タグを連結します。次に、setAppText() メソッドを使用して、このストリング全体を属性のアプリケーション固有の情報として割り当てます。
6. BusObjDef クラスで定義されている setAttributeList() メソッドを使用して、生成済みの属性ベクトル (Attributes) をビジネス・オブジェクト定義の属性リストとして割り当てます。

サポートされる動詞の提供: ビジネス・オブジェクト定義には、サポートされる*動詞* が含まれます。この動詞は、ビジネス・オブジェクト定義のビジネス・オブジェクト上で実行可能な操作を記述します。ビジネス・オブジェクト定義は、サポートしている動詞をその*動詞リスト* 内に保持しています。ODK API は、動詞をビジネス・オブジェクト動詞 (BusObjVerb) オブジェクトとして表します。動詞オブジェクトをインスタンス化するには、BusObjVerb() コンストラクターを使用します。

表 42 は、動詞オブジェクト内のメタデータをまとめたものです。

表 42. 動詞のメタデータ

動詞メタデータ	説明	accessor メソッド
名前	サポートされる動詞の名前 (Create、Retrieve、Update、または Delete など)	getName()、setName()
アプリケーション固有の情報	動詞レベルのアプリケーション固有の情報。動詞にのみ適用される情報が含まれます。	getAppInfo()、setAppInfo()

Roman Army サンプルでは、generatesBoDefs() メソッドが、サポートされる単一の動詞である Create を各ビジネス・オブジェクト定義に割り当てます。次のコー

ド・フラグメントは、BusObjDef クラスで定義されている insertVerb() メソッドを使用して、Create 動詞をビジネス・オブジェクト定義の動詞リストに追加します。

```
sonBo.insertVerb("Create", null);
```

Roman Army サンプルが生成するビジネス・オブジェクト定義は、アプリケーション固有の情報を提供しません。したがって、この insertVerb() 呼び出しの 2 番目の引き数 (動詞のアプリケーション固有の情報を指定する) は null となります。

ODA は、「BO プロパティー」ダイアログ・ボックスを使用して、生成済みのビジネス・オブジェクト定義の動詞サポートを実現できます。 Verbs と呼ばれるビジネス・オブジェクト・プロパティーを定義し、サポートされる動詞をユーザーが選択できるようにすることで、ODA はよりカスタマイズされた動詞サポートを取得できます。「BO プロパティー」ダイアログ・ボックスの使用方法については、138 ページの『ビジネス・オブジェクト・プロパティーの要求』を参照してください。

次のコード・フラグメントでは、ODA が Verbs というビジネス・オブジェクト・プロパティーのユーザー指定値を取得済みで、かつ ODA がこのプロパティーを使用して、ビジネス・オブジェクト定義の動詞リストに追加する動詞を取得するものと想定しています。

```
Vector Verbs;
AgentProperty propVerbs = getBOSpecificProperty("Verbs");

if (propVerbs.allValues[0] != null)
   {
   int len = propVerbs.allValues.length;
   BusObjVerb verb;

   for(int i=0; i<len; i++)
      {
      if(propVerbs.allValues[i] != null)
         {
         try {
            verb = new BusObjVerb(propVerbs.allValues[i].toString(), "");
            Verbs.add(verb);
         }
      }
   }
...
boDef.setVerbList(Verbs);
```

このコード・フラグメントは、BusObjVerb() コンストラクターを使用して、タイプ BusObjVerb の verb 変数に動詞をコピーします。次に、その動詞オブジェクトの String バージョンを Verbs ベクトルにロードします。このコードでは、動詞のアプリケーション固有の情報は指定されません。最後にコード・フラグメントは、BusObjDef クラスで定義されている setVerbList() メソッドを使用して、生成済みの動詞ベクトル (Verbs) をビジネス・オブジェクト定義の動詞リストとして割り当てます。

生成済みのビジネス・オブジェクト定義の提供

109 ページの『生成済みコンテンツの提供』で説明したように、ODA は生成済みコンテンツを 2 つの部分に分けてビジネス・オブジェクト・ウィザードに戻す必要があります。したがって、ODA がビジネス・オブジェクト定義をコンテンツとして生成する場合は、次のものを戻す必要があります。

- 生成済みコンテンツ構造体。生成済みのビジネス・オブジェクト定義が含まれます。
- コンテンツ・メタデータ (ContentMetaData) オブジェクト。生成済みコンテンツ構造体内のビジネス・オブジェクト定義を記述します。

ODA は「要求時」にビジネス・オブジェクト定義を生成する必要があるため、generateBoDefs() メソッドは次のようにコンテンツ情報を提供します。

- 生成済みコンテンツ構造体にデータを設定します。この構造体は、generateBoDefs() と getBoDefs() がともにアクセスできるように、何らかの方法で ODA クラスに対してグローバルにする必要があります。
- 生成済みのビジネス・オブジェクトを記述するコンテンツ・メタデータ (ContentMetaData) オブジェクトを、その呼び出し側であるビジネス・オブジェクト・ウィザードに戻します。

  ビジネス・オブジェクト・ウィザードは、このコンテンツ・メタデータ・オブジェクトを受け取ると、getBoDefs() メソッドを必要に応じて使用することにより、(生成済みコンテンツ構造体内の) 生成済みビジネス・オブジェクト定義にアクセスできます。

注: getBoDefs() については、152 ページの『生成済みビジネス・オブジェクト定義へのアクセスの提供』を参照してください。

図 68 のコード・サンプルは、サンプル Roman Army ODA の generateBoDefs() メソッドの最後の部分を示しています。

```
            m_generatedBOs.add(sonBo);
        } // this for loop terminates when all bus obj defs are generated
        return new ContentMetaData(ContentType.BusinessObject, -1,
           m_generatedBOs.size());
} // end of generateBoDefs()
```

図 68. 生成済みのビジネス・オブジェクト定義の提供

図 68 のコード・フラグメントは、生成済みコンテンツを次のように処理します。

- generatedBoDefs() メソッドにより、生成したビジネス・オブジェクト定義を生成済みコンテンツ構造体 (m_generatedBOs) に保管します。

  138 ページの図 63 に示すように、Roman Army サンプルは m_generatedBOs と呼ばれる Java ベクトルを生成済みコンテンツ構造体として使用します。この構造体は、Roman Army ODA クラスのメソッドに対してグローバルです。生成されたビジネス・オブジェクト定義を保管する際、generateBoDefs() は、それを m_generatedBOs ベクトルに保管します。このステップはループに含まれていますが、このループは、ユーザーが選択したすべての ソース・ノードに対してビジネス・オブジェクト定義が生成された時点で終了します。ビジネス・オブジェクト・ウィザードは、生成済みコンテンツ構造体にアクセスする必要があるときは、コンテンツ取得メソッド (getBoDefs()) を呼び出します。
- 最後のステップとして、generateBoDefs() は、生成済みコンテンツを記述するコンテンツ・メタデータ・オブジェクトを戻します。

generateBoDefs() メソッドは、ContentMetaData オブジェクトをインスタンス化し、表 43 に示す情報をこのコンストラクターに渡します。

表 43. ビジネス・オブジェクト定義を生成するためのコンテンツ・メタデータの初期化

ContentMetaData 情報	コード	説明
コンテンツ・タイプ	ContentType.BusinessObject	コンテンツ・タイプがビジネス・オブジェクト定義であることを示します。
生成済みコンテンツのサイズ	-1	合計サイズが必須でない ことを示します。現在の ContentMetaData オブジェクトの実装では、長さの値は必要ありません。
生成済みコンテンツの数	m_generatedBOs.size()	size() メソッドは、現在ベクトル内にある要素の数を戻します。

生成済みビジネス・オブジェクト定義へのアクセスの提供

generateBoDefs() メソッドは、実際に生成されたビジネス・オブジェクト定義を戻しません。ビジネス・オブジェクト・ウィザードで生成済みコンテンツにアクセスできるようにするには、ビジネス・オブジェクト定義用のコンテンツ取得メソッドを ODA クラスに実装する必要があります。ビジネス・オブジェクト・ウィザードは、generateBoDefs() によって戻される コンテンツ・メタデータ・オブジェクト内の情報を使用して、適切なコンテンツ取得メソッドを呼び出すかどうかを決定します。generateBoDefs() が正常にビジネス・オブジェクトを生成すると、ビジネス・オブジェクト・ウィザードは getBoDefs() メソッドを呼び出して、生成済みのビジネス・オブジェクト定義を取得します。

注: 今回のリリースでは、ビジネス・オブジェクト・ウィザードは、ビジネス・オブジェクト定義の生成を開始する際に必ず generateBoDefs() メソッドを呼び出します。なぜなら、ODA は要求時コンテンツ・プロトコルをサポートする必要があるからです。 ODA は、ビジネス・オブジェクト定義の生成において、コールバック・コンテンツ・プロトコルをサポートしてはなりません。コンテンツ・プロトコルについては、 126 ページの『ODA コンテンツ・プロトコルの選択』を参照してください。

生成済みのビジネス・オブジェクト定義へのアクセスを可能にするには、ODA クラスに getBoDefs() メソッドを実装する必要があります。このメソッドは、IGeneratesBoDefs インターフェース内で定義されています。このメソッドは、戻されるビジネス・オブジェクト定義の数を識別するインデックスを引き数として受け入れます。また、このメソッドは、生成済みコンテンツ構造体内のビジネス・オブジェクト定義にアクセスし、取得したビジネス・オブジェクト定義 (BusObjDef) オブジェクトの配列を戻します。この配列内のビジネス・オブジェクト定義の数は、表 44 に示すように、インデックス引き数の値によって異なります。

表 44. ビジネス・オブジェクト定義の取得

インデックスの値	説明	getBoDefs() が戻す配列内の要素数
0 から *count* - 1 の範囲。ここで、*count* は生成済みコンテンツ構造体内のビジネス・オブジェクト定義の総数です。	取得するビジネス・オブジェクト定義の生成済みコンテンツ構造体にインデックスの位置を指定します。	1 つのビジネス・オブジェクト定義
ODKConstant.GET_ALL_OBJECTS	生成済みコンテンツ構造体内のビジネス・オブジェクト定義をすべて 戻すことを指定する特別な定数。	生成済みコンテンツ構造体内のすべてのビジネス・オブジェクト定義 (*count*)

サンプル Roman Army ODA の場合、generateBoDefs() メソッド (ArmyAgent3 クラスで定義) は、m_generatedBOs ベクトルに生成済みのビジネス・オブジェクト定義を設定します。したがって、getBoDefs() メソッド (これも ArmyAgent3 クラスで定義) は、指定した数のビジネス・オブジェクト定義をこのベクトルから取得し、それを戻り配列内にコピーします。次のコードは、サンプル Roman Army ODA の getBoDefs() メソッドを示しています。

```
public BusObjDef[] getBoDefs(long index) throws ODKException
{
   BusObjDef[] bos = null;

   if (index == ODKConstant.GET_ALL_OBJECTS)
      {
      bos = new BusObjDef[m_generatedBOs.size()]
      System.arraycopy(m_generatedBOs.toArray(), 0, bos, 0,
         m_generatedBOs.size());
      }

   else
      bos = new BusObjDef[] {(BusObjDef)m_generatedBOs.get((int)index)};

return bos;
}
```

コンテンツとしてのバイナリー・ファイルの生成

バイナリー・ファイル は、Java File オブジェクトとして表されるオペレーティング・システム・ファイルです。 ODA でバイナリー・ファイル・コンテンツを生成するには、その ODA クラスに IGeneratesBinFiles インターフェースを実装する必要があります。 表 45 は、IGeneratesBinFiles インターフェースを実装する際に ODA クラスに定義する必要のあるメソッドを示しています。

表 45. IGeneratesBinFiles インターフェース内のメソッド

メソッド	IGeneratesBinFiles メソッド	説明
ソース・ノード生成メソッド	なし	ソース・ノードは、IGeneratesBoDefs インターフェースの getTreeNodes() メソッドで生成する必要があります。詳しくは、154 ページの『ファイルの使用』を参照してください。

表 45. *IGeneratesBinFiles* インターフェース内のメソッド (続き)

メソッド	IGeneratesBinFiles メソッド	説明
コンテンツ生成メソッド	generateBinFiles()	バイナリー・ファイルを生成し、それを ODA メモリーに書き込みます。
コンテンツ取得メソッド	getBinFile()	指定したバイナリー・ファイル、またはすべてのバイナリー・ファイルを ODA メモリーから取得します。

注: 表 45 に示すメソッドの他に、IGeneratesBinFiles には getContentProtocol() メソッドも含まれています。このメソッドは、ODA がファイル生成用にサポートするコンテンツ・プロトコルを指定します。詳しくは、126 ページの『ODA コンテンツ・プロトコルの選択』を参照してください。

ビジネス・オブジェクト・ウィザードは、「ビジネス・オブジェクトの生成中」ダイアログ・ボックス (ステップ 5) を表示して、コンテンツを生成および取得します。IGeneratesBinFiles インターフェースを実装している場合、ビジネス・オブジェクト・ウィザードは、表 46 に示すメソッドを呼び出して、コンテンツを生成および取得します。

表 46. ビジネス・オブジェクト・ウィザードと *IGeneratesBinFiles* メソッド

メソッドの用途	IGeneratesBinFiles メソッド	詳細情報の参照先
ファイルをコンテンツとして生成します。	generateBinFiles()	157 ページの『ファイルの生成』
生成されたファイルを取得します。	getBinFile()	162 ページの『生成済みファイルへのアクセスの提供』

以降のセクションでは、表 46 に示す各メソッドの実装について説明します。

ファイルの使用

IGeneratesBinFiles インターフェースを実装すると、ODA は、次の状況でオペレーティング・システム・ファイルの使用をサポートできるようになります。

- ODA は、新しいファイルを*作成する* ことにより、ファイル・コンテンツの生成をサポートします。
- ODA は、既存のファイルを*読み取る* ことにより、ファイルとソース・ノードとの関連付けをサポートします。

ファイル・コンテンツ用ファイルの作成

ODA に IGeneratesBinFiles インターフェースを実装すると、ファイルをコンテンツとして作成することが可能となります。ODA が作成するファイルには、ビジネス・オブジェクト定義の生成プロセスやその他のプロセスから ODA が収集した情報が保持されます。ユーザーが選択したソース・ノードの配列 (ビジネス・オブジェクト・ウィザードがステップ 3「ソースの選択」の結果として作成するもの) がファイル生成プロセスで必要になった場合、ODA は、ビジネス・オブジェクト・ウ

ィザードからこの配列を受け取ることができます。ファイルを生成するメソッドの実装方法については、 157 ページの『ファイルの生成』を参照してください。

ただし、ソース・ノードを検出し、ビジネス・オブジェクト・ウィザードにより「ソースの選択」ダイアログ・ボックスで表示されるツリー・ノードの配列を生成するソース・ノード生成メソッドは、IGeneratesBinFiles インターフェースには定義されていません。 ODA がファイル・コンテンツの生成をサポートし、ユーザーが選択したソース・ノードの配列がこのファイル生成で必要な場合、ODA は、IGeneratesBoDefs インターフェース内のソース・ノード生成メソッドである getTreeNodes() を使用する必要があります。このメソッドは、指定した親ノードの子ノードについてデータ・ソースを照会し、関連するツリー・ノードを作成します (129 ページの『ソース・ノードの生成』を参照)。

注: 今回のリリースでは、ODA はすべて、ビジネス・オブジェクト定義の生成をサポートする必要があります。したがって、ODA には IGeneratesBoDefs インターフェースとそのすべてのメソッド (getTreeNodes() メソッドを含む) を実装する必要があります。

ODA で新しいファイルの作成 (ファイル生成) のみ をサポートする場合は、IGeneratesBoDefs に定義されている getTreeNodes() メソッドを使用することができます。このメソッドは、指定した親ノードの子ノードについてデータ・ソースを照会し、関連するツリー・ノードを作成します (129 ページの『ソース・ノードの生成』を参照)。

ソース・データ用ファイルの読み取り

IGeneratesBinFiles インターフェースを実装すると、ODA は、ソース・ノードに関連付けられたオペレーティング・システム・ファイルの読み取りをサポートできるようになります (ファイルをノードに関連付ける方法については、 95 ページの『オペレーティング・システム・ファイルの関連付け』を参照してください)。
ODA が読み取るファイルには、ソース・データが含まれています。ODA は、ソース・ノードとして表されるオブジェクトを、このソース・データから検索する必要があります。ファイルとノードの関連付けをサポートするために、ODA は次のステップを実行する必要があります。

- ファイルの関連付けが可能なツリー・ノードのノード・タイプに「ファイル」を設定します (その polymorphicNature メンバー変数は ODKConstant.NODE_NATURE_FILE に設定します)。詳しくは、 136 ページの『ファイル・タイプ・ノード』を参照してください。
- ソース・ノードが表すオブジェクトへのアクセスが必要なメソッドを実装し、ODA のデータ・ソースだけではなく、ソース・ノードに関連付けられているファイルも照会します。ソース・ノード・パスによって指定されたオペレーティング・システム・ファイルのコンテンツを取得するには、ODKUtility クラスで定義されている getClientFile() メソッドを使用します。

重要

指定されたオペレーティング・システム・ファイルを getClientFile() メソッドで正常に取得するには、ODA に IGeneratesBinFiles インターフェースを実装する必要があります。 ODA に IGeneratesBoDefs インターフェースしか実装していない場合、getClientFile() は UnsupportedContentException 例外をスローします。

getClientFile() メソッドの引き数には、取得するファイルのソース・ノード・パスを指定します。このソース・ノード・パスは、次のような形式で指定します。

fileNodePath:*filePath*

ここで、*fileNodePath* は関連するファイルを持つノードのノード・パス (ノード名はコロン (:) コロンで区切られている)、*filePath* は関連するファイルのオペレーティング・システム・パスです。関連するファイルであるノードを展開または選択すると、ビジネス・オブジェクト・ウィザードはそのノードに対してこのパスを作成します。

例えば、サンプル Roman Army ODA の ArmyAgent5 クラスは、IGeneratesBinFiles インターフェース、およびファイルとノードの関連付けの両方をサポートします。 96 ページの図 49 に示すように、ユーザーが (C:¥IBM¥XMLFiles ディレクトリー内の) Flavius.xml ファイルを Vulso ソース・ノードに関連付けるものと仮定しましょう。ソース・ノード階層から Flavius.xml ノード (96 ページの図 50 を参照) を選択すると、ビジネス・オブジェクト・ウィザードはソース・ノードの配列に次のノード・パスを組み込みます。

```
Apollo:Vulso:Flavius.xml:C:¥IBM¥XMLFiles¥Flavius.xml
```

この ODA は、ソース・ノード・パスの構文を解析する findSon() メソッドを提供し、ソース・ノードが表す関連オブジェクトを特定します。 ArmyAgent3 クラスの findSon() メソッドは、指定したソース・ノードに関連付けられているオブジェクトについて、ODA のデータ・ソース (RomanArmy.xml と呼ばれる XML ファイル) のみを照会します。ArmyAgent4 クラスの修正版のメソッドは、remoteSon() メソッドを提供することにより、関連ファイルを照会する機能を追加しています。remoteSon() メソッドは、指定されたファイルのコンテンツを getClientFile() を使用して取得し、そのコンテンツを Son オブジェクトとして戻します。

注: remoteSon() メソッドを実装している ArmyAgent4 クラスは、IGeneratesBinFiles インターフェースをサポートしていません。したがって、remoteSon() メソッドは getClientFile() がスローする UnsupportedContentException 例外をキャッチし、「ダミー」の Son オブジェクトを生成します (184 ページの図 76 を参照)。ArmyAgent4 を拡張した ArmyAgent5 クラスは、IGeneratesBinFiles を実装しています。したがって、このバージョンの ODA は、getClientFile() を使用することにより、ソース・ノードに関連付けられているファイルへのアクセスを完全にサポートします。

ソース・ノードとファイルとの関連付けが可能な場合は、ファイルのソース・ノード・パスを解釈し、そのファイルのコンテンツを読み取る機能が、コンテンツ生成時に必要となります。コンテンツを生成するメソッドは、ファイル内に存在するノードに格納された情報にアクセスできなければなりません。ノードに関連付けられているオペレーティング・システム・ファイルを getClientFile() を使用して取得するように、コンテンツを生成するメソッドを実装します。このサポートを提供するメソッドは、次のとおりです。

- generateBoDefs() メソッドは、ビジネス・オブジェクト定義を生成します。getClientFile() メソッドは、ビジネス・オブジェクト定義を作成するために必要な情報を generateBoDefs() で取得できるように、指定されたファイルのコンテンツを提供します。 ODA のデータ・ソースからのソース・ノード情報を取得するように generateBoDefs() メソッドがすでに実装されている場合は、関連ファイルからの情報も取得できるようにそれを拡張する必要があります。

  注: ODA が IGeneratesBinFiles インターフェースを実装していない場合、getClientFile() メソッドは、generateBoDefs() 内から呼び出されたときに、指定したファイルのコンテンツを取得できません。

- ファイルを生成するメソッドは、ODA がサポートするコンテンツ・プロトコルによって異なります。要求時生成の場合は、generateBinFiles() メソッドがファイルを生成します。コールバック生成の場合は、ユーザー定義メソッドがファイルを生成します。いずれの場合も、指定したファイルのコンテンツは getClientFile() メソッドが提供し、これによりメソッドは、ファイル生成に必要な情報を取得できるようになります。

ファイル・コンテンツの生成方法については、『ファイルの生成』を参照してください。

ファイルの生成

ユーザーが「ノードを選択」ダイアログ・ボックスでソース・ノードを選択すると、ODA は、コンテンツ生成を開始する準備が整います。ファイル生成プロセスの目標は、ODA またはその他のプロセスが必要とするファイルを 1 つ以上作成することです。ファイル生成を開始するステップは、ファイル・コンテンツ・タイプ (ContentType.BinaryFile) に関連付けられているコンテンツ・プロトコルによって異なります。

- ファイルが*要求時* に生成される場合、ビジネス・オブジェクト・ウィザードは、コンテンツ生成メソッド generateBinFiles() を呼び出すことにより、コンテンツ生成を開始します。このメソッドは、IGeneratesBinFiles インターフェースに含まれています。
- ファイルがコールバック により生成される場合、ODA は、ユーザーが定義した方法でコンテンツ生成を開始します。ビジネス・オブジェクト・ウィザードは、generateBinFiles() を呼び出すのではなく、ODA からの「コンテンツ生成が完了した」という旨のメッセージを待ち、それから生成されたコンテンツにアクセスします。

このセクションでは、ファイルを生成する際に generateBinFiles() メソッドで実行する必要のある次のステップについて説明します。

1. 158 ページの『generateBinFiles() メソッドの定義』

2. 159 ページの『ファイル情報のプロパティーの要求』
3. 159 ページの『ファイルの作成』
4. 160 ページの『生成されたファイルの提供』

generateBinFiles() メソッドの定義

generateBinFiles() メソッドは、IGeneratesBinFiles インターフェース内で定義されています。したがって、IGeneratesBinFiles インターフェースを実装するときは、(ODKAgentBase2 から派生した) ODA クラスにこのメソッドを実装する必要があります。 generateBinFiles() メソッドの目的は、ファイル (ContentType.BinaryFile) コンテンツの生成用に ODA が使用するコンテンツ・プロトコルによって以下のように異なります。

- ODA がファイルを「要求時」に生成する場合、ビジネス・オブジェクト・ウィザードは generateBinFiles() メソッドを明示的に呼び出してファイルを生成します。
- ODA がファイルをコールバックにより生成する場合、ビジネス・オブジェクト・ウィザードは generateBinFiles() メソッドを明示的に呼び出しません。代わりに ODA は、ビジネス・オブジェクト・ウィザードがアクセス可能なファイルをその他の方法で生成します。

要求時のファイル生成: ODA がファイルを「要求時」に生成する場合、ビジネス・オブジェクト・ウィザードは generateBinFiles() メソッドを明示的に呼び出してファイルの生成を開始します。したがって、generateBinFiles() は、ファイル・オブジェクトの生成を処理し、それを生成済みコンテンツ構造体に保管し、コンテンツ・メタデータをビジネス・オブジェクト・ウィザードに戻すように実装する必要があります。

generateBinFiles() メソッドの実行中、ビジネス・オブジェクト・ウィザードは「ビジネス・オブジェクトの生成中」画面 (ステップ 5) を表示します。最後のステップとして、generateBinFiles() は、生成されたファイルを記述するコンテンツ・メタデータ (ContentMetaData) オブジェクトを戻します (これには実際に生成されたファイルは含まれません)。

コールバックによるファイル生成: ODA がファイルをコールバックにより生成する場合、ビジネス・オブジェクト・ウィザードは generateBinFiles() メソッドを明示的に呼び出しません。代わりに、ODA はその他の方法を使用して「自発的に」ファイルを生成します。開発者は、ファイルの生成を処理し、それを生成済みコンテンツ構造体に保管して、コンテンツ生成が完了した旨をビジネス・オブジェクト・ウィザードに通知するようなメソッドを開発する必要があります。ただし、IGeneratesBinFiles インターフェースでは、generateBinFiles() メソッドを定義する必要があります。したがって、呼び出してはならないことを呼び出し側に警告するように generateBinFiles() を実装する必要があります。

サンプル Roman Army ODA は、ファイル生成用のコールバック・コンテンツ・プロトコルをサポートします (127 ページの図 58 を参照)。これは、ArmyAgent5 クラス内に generateBinDefs() メソッドを定義しています。このメソッドの実装には、図 69 に示すコードが含まれます。このコードは、呼び出された場合に例外をスローするように generateBinFiles() メソッドを定義しています。

```
public ContentMetaData generateBinFiles(String[] nodes)
    throws ODKException
{
   throw new ODKException(
      "Files are produced as callbacks. Do not call for file generation.");
}
```

図 69. generateBinFiles() メソッドの定義

例外をスローする代わりに、generateBinFiles() メソッドは、contentUnavailable() メソッド (ContentMetaData で定義) を使用して、そのコンテンツ・メタデータをビジネス・オブジェクト・ウィザードに戻すことができます。

```
return (ContentMetaData.contentUnavailable(ContentType.BinaryFile));
```

ファイル情報のプロパティーの要求

ファイル生成プロセスの間に ODA で追加情報が必要となった場合は、「BO プロパティー」ダイアログ・ボックスが開くため、ユーザーはそこでビジネス・オブジェクト・プロパティーの値を入力することができます。これらのプロパティーはビジネス・オブジェクト・プロパティーと呼ばれますが、getBOSpecificProps() メソッドを使用すれば、ファイル生成プロセスで必要とされる情報を表示することができます。「BO プロパティー」ダイアログ・ボックスの使用方法については、138 ページの『ビジネス・オブジェクト・プロパティーの要求』を参照してください。

ファイルの作成

ODK API では、バイナリー・ファイルを表す特別なクラスは提供されません。なぜなら、Java では java.io パッケージに File クラスがすでに用意されているからです。このパッケージには、ファイルの生成やアクセスに役に立つ入出力クラスが多数含まれています。ODA は、生成する各ファイルごとに、次のステップを実行する必要があります。

- 新しい File オブジェクトを生成し、適切なファイル名を付ける。
- このファイルにコンテンツを書き込み、書き込みが完了した時点でファイルを閉じる。

ODA が実行する実際のファイル生成は、ODA の設計によって異なります。ODA の要件に最も適合するようにファイル生成を実装してください。また、ファイルを必要とするすべてのコンポーネントを実装してください。

サンプル Roman Army ODA の ArmyAgent5 クラスは、独立したクラス (FileCreator) を定義することにより、実際のファイル生成を処理します。「自発的な」ファイル生成をシミュレートする際に、このサンプルは、generateBoDefs() メソッドから FileCreator() コンストラクターを呼び出します (次のコード・フラグメントを参照)。

```
public ContentMetaData generateBoDefs(String[] nodes) throws ODKException
{
   ContentMetaData cmd = super.generateBoDefs(nodes);

   new FileCreator(this, nodes).start();

   return cmd;
}
```

FileCreator() コンストラクターは、ファイル生成用のスレッドを作成します。このコンストラクターは、現在の ODA オブジェクト (this) への参照と、選択したソース・ノードのノード・パスを持つ配列を、引き数として受け取ります。続いて、次のファイルを作成します。

- stats.zip ファイル。ODA が生成したビジネス・オブジェクト定義の数が含まれます。
- adopted.txt ファイル (ユーザーが選択したソース・ノードのどれかが養子の場合)。

生成されたファイルの提供

109 ページの『生成済みコンテンツの提供』で説明したように、ODA は生成済みコンテンツを 2 つの部分に分けてビジネス・オブジェクト・ウィザードに戻す必要があります。したがって、ODA がファイルをコンテンツとして生成する場合は、次のものを戻す必要があります。

- 生成済みコンテンツ構造体。生成済みのファイルが含まれます。
- コンテンツ・メタデータ (ContentMetaData) オブジェクト。生成済みコンテンツ構造体内のファイルを記述します。

この情報を提供するメソッドは、ファイル生成用に ODA が使用するコンテンツ・プロトコルによって異なります。

- ODA がファイルを「要求時」に生成する場合は、generateBinFiles() メソッドがこのコンテンツ情報を提供します。
- ODA がファイルをコールバックにより生成する場合は、ユーザー定義メソッドがこのコンテンツ情報を提供する必要があります。

要求時ファイルのコンテンツの提供: ODA がファイルを「要求時」に生成する場合、ビジネス・オブジェクト・ウィザードは generateBinFiles() メソッドを呼び出してファイル生成を処理します。したがって、generateBinFiles() は、生成済みコンテンツを次のように提供します。

- 生成済みコンテンツ構造体にデータを設定します。この構造体は、generateBinFiles() と getBinFile() がともにアクセスできるように、何らかの方法でそれらのメソッドから可視にする必要があります。
- 生成済みのファイルを記述するコンテンツ・メタデータ (ContentMetaData) オブジェクトを、その呼び出し側であるビジネス・オブジェクト・ウィザードに戻します。

  ビジネス・オブジェクト・ウィザードは、このコンテンツ・メタデータ・オブジェクトを受け取ると、getBinFile() メソッドを必要に応じて使用することにより、(生成済みコンテンツ構造体内の) 生成済みファイルにアクセスできるようになります。

getBinFile() については、162 ページの『生成済みファイルへのアクセスの提供』を参照してください。

コールバックにより生成されたファイルのコンテンツの提供: ODA がファイルをコールバックにより生成する場合、ビジネス・オブジェクト・ウィザードは、ファイル生成を処理する際に generateBinFiles() メソッドを呼び出しません。代わりに、ODA はユーザー定義メソッドを使用して「自発的に」ファイルを生成します。

このメソッドは、ODA クラスであっても、ODA パッケージ内のクラスであってもかまいません。ただし、このメソッドは、生成済みコンテンツを次のように提供する必要があります。

- 生成済みコンテンツ構造体にデータを設定します。この構造体は、ファイルを生成するユーザー定義メソッドおよび getBinFile() (ODA クラスで実装) に対して何らかの方法で可視となるようにし、両方のメソッドでアクセスできるようにする必要があります。
- 生成済みのファイルを記述するコンテンツ・メタデータ (ContentMetaData) オブジェクトをビジネス・オブジェクト・ウィザードに送ります。

  ファイルを生成するユーザー定義メソッドは、コンテンツ・メタデータを直接ビジネス・オブジェクト・ウィザードに戻すことはできません。なぜなら、ビジネス・オブジェクト・ウィザードはこのメソッドを呼び出していないからです。代わりに、このメソッドは、ODKUtility クラスで定義されている contentComplete() メソッドを呼び出すことにより、「コンテンツ生成が完了した」という旨のメッセージをビジネス・オブジェクト・ウィザードに送る必要があります。このメソッドは、コンテンツ・メタデータ・オブジェクトを引き数として受け入れます。このコンテンツ・メタデータ・オブジェクトに含める必要のある情報については、 162 ページの表 47 を参照してください。このコンテンツ・メタデータは、ビジネス・オブジェクト・ウィザードに送られます。ビジネス・オブジェクト・ウィザードは、コンテンツ・メタデータ・オブジェクトを受け取ると、getBinFile() メソッドを使用して、(生成済みコンテンツ構造体内の) 生成済みファイルにアクセスできるようになります。

注: getBinFiles() については、 162 ページの『生成済みファイルへのアクセスの提供』を参照してください。

サンプル Roman Army ODA の ArmyAgent5 クラス内で、生成済みコンテンツ構造体は、m_files と呼ばれる File オブジェクトの配列として定義されています。

```
File[] m_files = null;
```

図 70 のコード・フラグメントは、ArmyAgent5.java ファイルに定義されている FileCreator.run() メソッドの最後の部分を示しています。

```
            for (int i=0; i<fileV.size(); i++)
               m_agent.m_files[i] = (File) fileV.get(i);
      }
      ODKUtility.getODKUtility.contentComplete(
         new ContentMetaData(ContentType.BinaryFile, 0,
         m_agent.m_files.length);
} // end of run() in FileCreator class
```

図 70. ファイル・コンテンツの提供

図 70 のコード・フラグメントは、生成済みコンテンツを次のように処理します。

- FileCreator.run() メソッドで、生成されたファイルを生成済みコンテンツ構造体 (m_files) に保管します。

  Roman Army サンプル ODA が m_files 配列を生成済みコンテンツ構造体として使用します。生成したファイルを保管する際は、run() がそのファイルをこの

m_files 配列に保管します。このステップは、run() によってすべての ファイルが生成された後に実行されます。ビジネス・オブジェクト・ウィザードは、コンテンツ取得メソッド (getBinFile()) の呼び出しを通じて、m_files 配列にアクセスできます。

- 最後のステップとして、FileCreator.run() は、生成済みコンテンツを記述するコンテンツ・メタデータ・オブジェクトをビジネス・オブジェクト・ウィザードに送ります。

  run() メソッドは、contentComplete() メソッドを呼び出して、新しい ContentMetaData オブジェクトを渡します。 run() は、表 47 に示す情報を ContentMetaData() コンストラクターに渡します。

表 47. ファイル生成用のコンテンツ・メタデータの初期化

ContentMetaData 情報	コード	説明
コンテンツ・タイプ	ContentType.BinaryFile	コンテンツ・タイプがファイルであることを示します。
生成済みコンテンツのサイズ	0	合計サイズが必須でない ことを示します。現在の ContentMetaData オブジェクトの実装では、長さの値は必要ありません。
生成済みコンテンツの数	m_files.length	length メンバー変数には、現在配列内にある要素の数が含まれます。

生成済みファイルへのアクセスの提供

generateBinFiles() メソッドは、実際に生成されたビジネス・オブジェクト定義を戻しません。ビジネス・オブジェクト・ウィザードで生成済みコンテンツにアクセスできるようにするには、ファイル用のコンテンツ取得メソッドを ODA クラスに実装する必要があります。ビジネス・オブジェクト・ウィザードは、generateBinFiles() によって戻される コンテンツ・メタデータ・オブジェクト内の情報を使用して、どのコンテンツ取得メソッドを呼び出すのかを決定します。ファイル・コンテンツの場合、ビジネス・オブジェクト・ウィザードは getBinFile() メソッドを呼び出して、生成済みのビジネス・オブジェクト定義を取得します。

注: ODA が要求時コンテンツ・プロトコルをサポートしている場合、ビジネス・オブジェクト・ウィザードは、ファイル生成の際に generateBinFile() メソッドを呼び出します。 ODA がファイル生成用のコールバック・コンテンツ・プロトコルをサポートしている場合、ファイルを実際に生成するのはユーザー定義メソッドです。ただし、このメソッドは、実際に生成されたコンテンツは戻しません。したがって、ビジネス・オブジェクト・ウィザードは依然として、生成済みファイルにアクセスするために、getBinFile() メソッドを必要とします。

ファイル生成用に ODA でサポートされるコンテンツ・プロトコルとは無関係に、ODA クラスには getBinFile() メソッドを実装する必要があります。このメソッドは、IGeneratesBinFiles インターフェース内で定義されています。このメソッドは、戻されるファイルの数を識別するインデックスを引き数として受け入れます。

また、このメソッドは、生成済みコンテンツ構造体内のファイルにアクセスし、取得したファイル (File) オブジェクトの配列を戻します。この配列内のファイルの数は、表 48 に示すように、インデックス引き数の値によって異なります。

表 48. ファイルの取得

インデックスの値	説明	getBinFile() が戻す配列内の要素数
0 から *count* - 1 の範囲。ここで、*count* は生成済みコンテンツ構造体内のファイルの総数です。	取得するファイルの生成済みコンテンツ構造体にインデックスの位置を指定します。	1 つのファイル・オブジェクト
ODKConstant.GET_ALL_OBJECTS	生成済みコンテンツ構造体内のファイルをすべて戻すことを指定する特別な定数。	生成済みコンテンツ構造体内のすべてのファイル・オブジェクト (*count*)

サンプル Roman Army ODA の場合、FileCreator.run() メソッド (ArmyAgent5 クラスで定義) は、m_files 配列に生成済みファイルを設定します。したがって、getBinFile() メソッド (これも ArmyAgent5 クラスで定義) は、指定した数のファイルをこの配列から取得します。次のコードは、サンプル Roman Army ODA の getBinFile() メソッドを示しています。

```
public File[] getBinFile(long index) throws ODKException
{
   if (index == ODKConstant.GET_ALL_OBJECTS)
      return m_files;
   else
      return new File[] {m_files[(int)index]};
}
```

エージェント・プロパティーの使用

ODA がビジネス・オブジェクト・ウィザードにエージェント・プロパティーを提供する状況は 2 つあります。

- 初期化した ODA 構成プロパティーを (「エージェントの構成」ダイアログ・ボックスに) 提供するため
- 初期化したビジネス・オブジェクト・プロパティーを (「BO プロパティー」ダイアログ・ボックスに) 提供するため

エージェント・プロパティーを表すために、ODK API は、AgentProperty クラスのインスタンスであるエージェント・プロパティー・オブジェクトを定義します。エージェント・プロパティー・オブジェクトをインスタンス化するときは、表 49 に示すそのメンバー変数の一部またはすべてを初期化します。

表 49. エージェント・プロパティー・オブジェクトのコンテンツ

メンバー変数	説明
propName	エージェント・プロパティーの名前。
description	エージェント・プロパティーの目的を記述するテキスト・ストリング。
type	エージェント・プロパティーのデータ型。プロパティー・タイプ定数によって表されます。

表 49. エージェント・プロパティー・オブジェクトのコンテンツ (続き)

メンバー変数	説明
cardinality	エージェント・プロパティーのカーディナリティー。プロパティーが持つことのできる値が単一の値か複数の値かを指定します。
isHidden	ビジネス・オブジェクト・ウィザードがプロパティー値を標準テキストまたは暗号形式のどちらで表示するのかを決定します。
isMultiple	ビジネス・オブジェクト・ウィザードがエージェント・プロパティーの有効な値のドロップダウン・リストを表示するかどうかを決定します (このドロップダウン・リストはユーザーの選択対象となります)。
isReadOnly	エージェント・プロパティーの値を読み取り専用にするかどうかを決定します (表示値の変更が可能であるかどうか)。
isRequired	エージェント・プロパティーの値が必須であるかどうかを決定します (値の指定が必要であるかどうか)。
allDefaultValues	エージェント・プロパティーのデフォルト値の配列。
allDependencies	エージェント・プロパティーの条件の配列。
allValidValues	エージェント・プロパティーの有効な値の配列。
allValues	ユーザーが初期化したエージェント・プロパティー値の配列。

エージェント・プロパティー・オブジェクトをインスタンス化するには、次のいずれかの構文形式の AgentProperty() コンストラクターを使用します。

- 最初の構文形式は、新しいエージェント・プロパティー・オブジェクトを定義し、そのオブジェクトのプロパティー名に限り 初期化します。
- 2 番目の構文形式は、新しいエージェント・プロパティー・オブジェクトを定義し、そのオブジェクトのすべての メンバー変数を初期化します。
- 3 番目の構文形式は、新しいエージェント・プロパティー・オブジェクトを定義し、そのオブジェクトのメンバー変数を、isHidden と isReadOnly を除いて すべて初期化します。

エージェント・プロパティーの定義

表 50 は、エージェント・プロパティー・オブジェクトに含まれるエージェント・プロパティーについての基本情報を示しています。

表 50. エージェント・プロパティーの基本情報

基本プロパティー情報	AgentProperty メンバー変数	説明
名前	propName	エージェント・プロパティーを識別します。 ビジネス・オブジェクト・ウィザードは、この値を「エージェントの構成」ダイアログ (構成プロパティー) または「BO プロパティー」ダイアログ・ボックス (ビジネス・オブジェクト・プロパティー) の「プロパティー」列に表示します。エージェント・プロパティーの名前は、どの形式の AgentProperty() コンストラクターでも初期化できます。
説明 (オプション)	description	エージェント・プロパティーの目的を説明する追加情報を提供します。 ビジネス・オブジェクト・ウィザードは、この値を「エージェントの構成」ダイアログ (構成プロパティー) または「BO プロパティー」ダイアログ・ボックス (ビジネス・オブジェクト・プロパティー) の「説明」列に表示します。エージェント・プロパティーの説明を初期化する際は、2 番目か 3 番目の形式の AgentProperty() コンストラクターを使用する必要があります。
データ型	type	エージェント・プロパティーが保持している値のデータ型を定義します。 ビジネス・オブジェクト・ウィザードは、この値を「エージェントの構成」ダイアログ (構成プロパティー) または「BO プロパティー」ダイアログ・ボックス (ビジネス・オブジェクト・プロパティー) の「タイプ」列に表示します。 1 番目の形式の AgentProperty() コンストラクターを使用してエージェント・プロパティーを初期化すると (プロパティー名のみの指定)、エージェント・プロパティーのタイプは String にデフォルト設定されます。タイプを指定するには、2 番目か 3 番目の形式の AgentProperty() コンストラクターを使用して、エージェント・プロパティーを初期化します。エージェント・プロパティーのタイプを表す際は、 201 ページの表 67に示すプロパティー・タイプ定数のいずれかを使用します。

プロパティー値の定義

ビジネス・オブジェクト・ウィザードは、エージェント・プロパティーの値を「エージェントの構成」ダイアログ (構成プロパティー) または「BO プロパティー」ダイアログ・ボックスの「値」列に表示します。エージェント・プロパティーを初期化するプロセスの一環として、次の操作を行う必要があります。

- 『表示コントロールのタイプの選択』
- 168 ページの『デフォルト値の指定』
- 168 ページの『単一カーディナリティー・プロパティーの初期化』
- 169 ページの『複数カーディナリティー・プロパティーの初期化』

表示コントロールのタイプの選択

ビジネス・オブジェクト・ウィザードは、次の `AgentProperty` メタデータを使用して、プロパティー値を表示するコントロールのタイプを決定します。

- `isMultiple` パラメーター。プロパティー値のコントロールにおいてドロップダウン・リストに複数の値を表示するかどうかを決定します。ドロップダウン・リストを値で初期化するには、`allValidValues` 配列にその値を指定します。
- `cardinality` パラメーター。コントロールにおいて単一のプロパティー値の指定を可能にするか、複数のプロパティー値の指定を可能にするかを決定します。

カーディナリティー	説明	カーディナリティー定数
単一	プロパティーが保持できる値は 1 つのみです。したがって、ユーザーは、プロパティーに対して値を *1* つのみ 指定できます。	`ODKConstant.SINGLE_CARD`
複数 (n)	プロパティーが保持できる値は 1 つ以上です。したがって、ユーザーは、プロパティーに対して値を *複数* 指定できます。	`ODKConstant.MULTIPLE_CARD`

表 51 は、プロパティー値コントロールを表示する際の可能な組み合わせを示しています。

表 51. 考えられるプロパティー値コントロールのタイプ

カーディナリティー	複数の値を表示 (isMultiple)	有効な値 (すべての ValidValues) が提供されるか?	説明
1	`false`	いいえ	プロパティー値は、プレーン・テキスト編集コントロール (1 つの値を入力または編集可能なシンプルなボックス) として表示されます。
1	`true`	はい	プロパティー値は、指定された有効な値を含むドロップダウン・リストとして表示されます (167 ページの図 71 を参照)。このリストからは、値を 1 つのみ選択できます。
n	`true`	はい	プロパティー値は、指定された有効な値を含むドロップダウン・リストとして表示されます。このリスト内のそれぞれの値はチェック・ボックスとともに表示されます。チェック・ボックスを選択すると、その値がプロパティーの値セットに含められます (167 ページの図 71 を参照)。

表 51. 考えられるプロパティー値コントロールのタイプ (続き)

カーディナリティー	複数の値を表示 (isMultiple)	有効な値 (すべての ValidValues) が提供されるか?	説明
n	true	いいえ	プロパティー値は、表示値を持たないグリッド・コントロールとして表示されます。最初このグリッドには、1 行の空行を持つサブグリッドが表示されます。この行にテキストを入力すると、ビジネス・オブジェクト・ウィザードは別の空行を挿入します。このプロセスは、ユーザーが新しい行を入力している間続きます。値を削除する場合、ユーザーはその値のテキストを削除します。ビジネス・オブジェクト・ウィザードは、空でない行のみをプロパティーの値セットに含めます。

有効な値を持たない 単一カーディナリティー・プロパティーをビジネス・オブジェクト・ウィザードが表示する場合、プロパティーの「値」フィールドは空のままになります。ただし、プロパティーにデフォルト値を定義することができます。その場合、ビジネス・オブジェクト・ウィザードは、「値」フィールドにデフォルト値を表示します。詳しくは、168 ページの『デフォルト値の指定』を参照してください。

図 71 は、複数の値を表示する (isMultiple = true) ビジネス・オブジェクト・ウィザードの 2 つのコントロールを示しています。

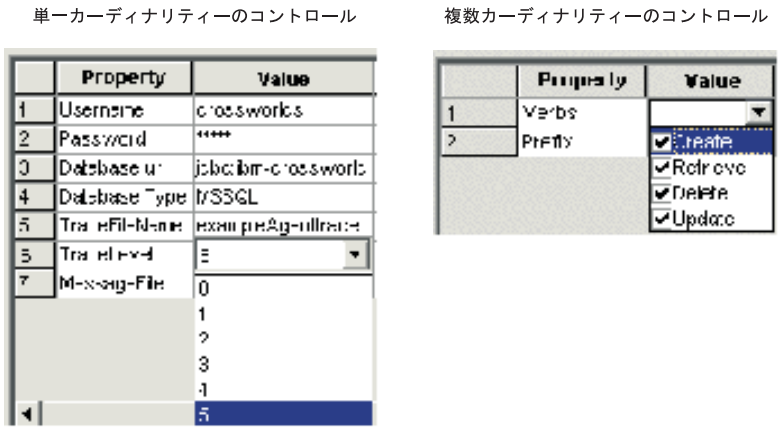


図 71. 複数の値を持つプロパティー用の単一カーディナリティー・コントロールと複数カーディナリティー・コントロール

図 71 は、単一カーディナリティー・コントロールと複数カーディナリティー・コントロールを示しています。どちらのコントロールも、ドロップダウン・リスト内に複数の値が表示されています。

- 単一カーディナリティー・コントロール (図 71 の左側) では、ドロップダウン・リスト内に複数のトレース・レベルが表示されています。ただし、このリストからユーザーが選択できる値は *1* つのみ (cardinality = ODKConstant.SINGLE_CARD) です。

- 複数カーディナリティー・コントロール (図 71 の右側) では、ドロップダウン・リスト内に複数の動詞が表示されています。ユーザーはこのリストから動詞をいくつでも (cardinality = ODKConstant.MULTIPLE_CARD) 選択できます。

デフォルト値の指定

エージェント・プロパティーのデフォルト値を指定するには、そのデフォルト値を allDefaultValues メンバー変数に提供します。このメンバー変数は、Object 値の配列です。この配列内の要素の数は、プロパティーのカーディナリティーと同じでなければなりません。

- 単一カーディナリティー・プロパティーの場合、allDefaultValues 配列に含まれる要素は 1 つのみでなければなりません。
- 複数カーディナリティー・プロパティーの場合、allDefaultValues 配列には複数の要素を含めることができます。

ビジネス・オブジェクト・ウィザードは、デフォルト値をプロパティーに割り当ててから、プロパティーを表示します。プロパティー値を指定してこのデフォルトをオーバーライドしない場合、このデフォルトの値はプロパティー値のままとなります。

注: 有効な値をプロパティーに指定しても、その値は自動的にデフォルト値にはなりません。デフォルト値は、明示的に指定する必要があります。

表 52 は、デフォルト値の振る舞いをまとめたものです。

表 52. エージェント・プロパティーのデフォルト値

カーディナリティー	allDefaultValues のコンテンツ	表示
単一	1 つの要素	有効な値を持つ場合 (isMultiple=true): デフォルト値は、有効な値のドロップダウン・リスト内で「チェックマーク付き」項目として表示されます。 有効な値を持たない場合 (isMultiple=false): デフォルト値は、プロパティーの「値」フィールドに表示されます。
複数	1 つまたは複数の要素	デフォルト値は、有効な値のドロップダウン・リスト内の「チェックマーク付き」項目として表示されます。

単一カーディナリティー・プロパティーの初期化

単一カーディナリティーのエージェント・プロパティーを初期化するには、次のステップを実行します。

- 指定可能な値の数を 1 つに制限します。プロパティーの cardinality メンバー変数に ODKConstant.SINGLE_CARD を設定します。
- プロパティーの有効な値のリストを提供し、その中からユーザーが 1 つだけ選択できるようにするかどうかを決定します。有効な値のリストを提供する場合は、次のようにします。
  - isMultiple 変数に true を設定します。

– 有効な値 (allValidValues) の配列を有効な値のリストで初期化します。

有効な値のリストを提供しない 場合は、isMultiple 変数に false を設定し、有効な値の配列を渡さないようにします。

- 必要な場合は、Object を含む allDefaultValues 配列を単一のデフォルト値で初期化します。

次のコード・フラグメントは、選択対象の値のリストを提供せず、デフォルト値が 256 である単一カーディナリティー・エージェント・プロパティーを初期化します。

```
defaultVal[0] = 256;
AgentProperty("Property1", AgentProperty.TYPE_INTEGER, "Description of property",
   false, false, ODKConstant.SINGLE_CARD, null, defaultVal);
```

複数カーディナリティー・プロパティーの初期化

複数カーディナリティー・エージェント・プロパティーを初期化するには、次のステップを実行します。

- 複数のプロパティー値の指定が可能であることを指示します。プロパティーの cardinality メンバー変数に ODKConstant.MULTIPLE_CARD を設定します。
- 複数のプロパティー値の入力をビジネス・オブジェクト・ウィザードで処理する必要があることを指示します。プロパティーの isMultiple メンバー変数に true を設定します。
- 選択元となる有効な値のリストを提供するかどうかを決定します。値のリストを提供する場合は、allValidValues 配列内の有効な値のリストを初期化します。これらの値でプロパティーのドロップダウン・リストが初期化されます。

  有効な値のリストを提供しない 場合、ビジネス・オブジェクト・ウィザードは、各プロパティー値を指定するためのサブグリッドを提供します。
- 必要な場合は、各デフォルト値の Object を含む allDefaultValues を初期化します。

141 ページの図 65 に示すコード・フラグメントは、有効な値とデフォルト値のリストを持つ、複数カーディナリティー・エージェント・プロパティーである Verbs を初期化します。

プロパティー値に対する条件の設定

AgentProperty クラスは、エージェント・プロパティーに対する条件を定義する機能を備えています。*条件* を設定すると、*従属プロパティー* と呼ばれる 1 つのエージェント・プロパティーの値を別のエージェント・プロパティーの値に基づいて制限できます。条件は、表 53 に示すように、2 つの部分 (それぞれの部分は特定のタイプの副条件) から構成されます。

表 53. エージェント・プロパティー条件の各部分

副条件	説明	ODK API クラス
入力条件	現在のエージェント・プロパティー値に対する条件を定義します。	InputCondition

表 53. エージェント・プロパティー条件の各部分 (続き)

副条件	説明	ODK API クラス
従属条件	関連する入力条件が true に評価されたときに満たす必要のある、従属プロパティーに対する条件を定義します。	DependentCondition

完了条件の定義

条件を表すために、ODK API は、CompleteCondition クラスのインスタンスである完了条件オブジェクトを定義します。表 54 は、完了条件オブジェクトに含まれるメンバー変数を示しています。

表 54. 完了条件オブジェクトのコンテンツ

メンバー変数	説明
allInputConditions	入力条件 (InputCondition) オブジェクトの配列。各オブジェクトには、エージェント・プロパティーの値の条件が 1 つだけ定義されています。
allDependentConditions	従属条件 (DependentCondition) オブジェクトの配列。各オブジェクトには、従属プロパティーの値の制限が 1 つだけ定義されています。この制限は、(allInputConditions 配列内の) 関連する入力条件が true に評価されたときに、従属プロパティーの値に適用されます。

完了条件オブジェクトには、エージェント・プロパティーに対する条件を *1* つだけ記述した情報が含まれます。エージェント・プロパティーには、多数の条件を定義することができます。それぞれの条件の完了条件オブジェクトは、エージェント・プロパティーの AgentProperty オブジェクトの allDependencies メンバー変数に保管されます。

エージェント・プロパティーに対する条件を 1 つ作成するには、次のステップを実行します。

1. 条件に関する情報を保持する CompleteCondition オブジェクトをインスタンス化します。
2. エージェント・プロパティーの入力条件を記述する適切な InputCondition オブジェクトをインスタンス化します。完了条件オブジェクトの入力条件配列 (allInputConditions メンバー変数) に各 InputCondition オブジェクトを保管します。入力条件については、 171 ページの『入力条件の定義』を参照してください。
3. エージェント・プロパティーの従属条件を記述する適切な DependentCondition オブジェクトをインスタンス化します。完了条件オブジェクトの従属条件配列 (allDependentConditions メンバー変数) に各 DependentCondition オブジェクトを保管します。詳しくは、 172 ページの『従属条件の定義』を参照してください。
4. 完了条件オブジェクトをエージェント・プロパティーの条件配列に保管します。この条件配列は、エージェント・プロパティー・オブジェクトの allDependencies メンバー変数に含まれています。

入力条件の定義

InputCondition クラスは、現在のエージェント・プロパティー値に対する条件を記述する入力条件 を表します。入力条件が true に評価されると、関連する従属条件が従属エージェント・プロパティーに適用されます。 表 55 は、入力条件を定義する際に必要な情報を示しています。

表 55. 入力条件の情報

入力条件の情報	説明	InputCondition メンバー変数
演算子	エージェント・プロパティー値に対して行う比較のタイプ。比較は関係演算子として表され、CompleteCondition クラス内の演算子定数の 1 つとして指定されます。	operatorType
特定の値	エージェント・プロパティーの値と比較する値。この値は、定数にすることも、別のエージェント・プロパティーの名前にすることもできます。	specificValue、typeOfSpecificValue
エージェント・プロパティーの値の比較を動的に行うかどうか	現在のエージェント・プロパティーの値を別のプロパティーの値と動的に比較するかどうかを指定する boolean 値。定数が関与する比較では、動的な比較は必要ありません。	isDynamic

入力条件を作成するには、いずれかの形式の InputCondition() コンストラクターを使用します。 174 ページの図 74 に示すコード・フラグメントは、エージェント・プロパティー値と 2 つの定数ストリング値「optionA」および「optionB」とを比較する入力条件を作成します。エージェント・プロパティー値をその他のプロパティー値と比較することもできます。図 72 に示すコード・フラグメントは、エージェントのプロパティーの値を現在 Property2 エージェント・プロパティーに入っている値と比較する入力条件を作成します。

```
// Instantiate a complete-condition object
condition1 = new CompleteCondition();

// Input condition to compare property value with Property2's value
condition1.allInputConditions[0] = new InputCondition(
   CompleteCondition.OP_NOT_EQUAL, true, AgentProperty.TYPE_INTEGER, "Property2");
```

図 72. プロパティー値と別のプロパティー値を比較する入力条件

図 72 で isDynamic メンバー変数には true が設定されています。これによりビジネス・オブジェクト・ウィザードは、最初に Property2 プロパティーの現行値を取得してから、ユーザー指定値とこの値を比較するようになります。また、specificValue には「Property2」、つまり比較対象のプロパティー名が設定されています。この入力条件の結果、プロパティーの従属条件は、このプロパティーの値が Property2 の値と*同じでない* 場合に*限って* 適用されます。

従属条件の定義

DependentCondition クラスは、特定の従属プロパティーの値に対する制限を記述する*従属条件* を表します。従属プロパティーは、その値が何らかのかたちで現在のプロパティーの値に依存しているプロパティーです。関連する 1 つまたは複数の入力条件が true に評価される場合、従属プロパティーの値は、従属条件で指定される制限を満たす必要があります。 表 56 は、従属条件を定義する際に必要な情報を示しています。

表 56. 従属条件の情報

従属条件の情報	説明	DependentCondition メンバー変数
名前	関連する 1 つまたは複数の入力条件が true に評価された場合に従属条件が適用される従属プロパティーの名前。	propertyName
演算子	従属プロパティー値に対して行う比較のタイプ。比較は関係演算子として表され、CompleteCondition クラス内の演算子定数の 1 つとして指定されます。	operatorType
特定の値	従属プロパティー値と比較する値。この値は、定数にすることも、別のエージェント・プロパティーの名前にすることもできます。	specificValue、typeOfSpecificValue
ユーザー指定値の比較を動的に行うかどうか	従属プロパティーの値を別のプロパティーの値と動的に比較するかどうかを指定する boolean 値。定数が関与する比較では、動的な比較は必要ありません。	isDynamic

従属条件を作成するには、いずれかの形式の DependentCondition() コンストラクターを使用します。 174 ページの図 74 に示すコード・フラグメントは、次の従属条件を作成します。

- 現在のプロパティー値が「optionA」である場合、「optionA」入力条件の 4 つの従属条件は、DepProperty1 従属プロパティーの 4 つの指定可能な値を指定します。
- 現在のプロパティー値が「optionB」である場合、「optionB」入力条件の 2 つの従属条件は、「DepProperty2」従属プロパティーの可能な値の範囲を指定します。

従属プロパティー値をその他のプロパティー値と比較することもできます。図 73 に示すコード・フラグメントは、従属プロパティーの値を現在 Property2 エージェント・プロパティーに入っている値と比較する従属条件を作成します。

```
// Dependent condition to compare property value with Property2's value
condition1.allDependentConditions[0] = new DependentCondition(
   CompleteCondition.OP_EQUAL, true, AgentProperty.TYPE_INTEGER, "Property2");
```

図 73. プロパティー値を別のプロパティー値と比較する従属条件

図 73 で isDynamic メンバー変数には true が設定されています。これによりビジネス・オブジェクト・ウィザードは、最初に Property2 プロパティーの現行値を取得してから、従属プロパティーの値とこの値とを比較するようになります。また、specificValue には「Property2」、つまり比較対象のプロパティー名が設定されています。

サンプル条件の定義

あるエージェント・プロパティー (Property1) に対して条件を定義し、この Property1 の値に基づいて 2 つの従属プロパティーに対する制限を指定したいとします。

- 1 番目の条件は、DepProperty1 従属プロパティーの値を、4 つの整数値 (0、1、256、または 512) のいずれか 1 つに制限します (Property1 の値が「optionA」の場合)。
- 2 番目の条件は、DepProperty2 従属プロパティーの値を、1 から 5 の範囲 (1 および 5 を含む) の範囲内に制限します (Property1 の値が「optionB」の場合)。

図 74 は、上記 2 つの条件を実装したコードを示しています。

```
// 1. Instantiate the complete-condition object
condition1 = new CompleteCondition();

// 2. Create the condition on the "optionA" value
//   a) Instantiate the input condition on "optionA"
   condition1.allInputConditions[0] = new InputCondition(
      CompleteCondition.OP_EQUAL, false, AgentProperty.TYPE_STRING,
      "optionA");

//   b) Instantiate the dependent conditions for DepProperty1
   condition1.allDependentConditions[0] = new DependentCondition(
      "DepProperty1", CompleteCondition.OP_EQUAL, false,
      AgentProperty.TYPE_INTEGER, "0");

   condition1.allDependentConditions[1] = new DependentCondition(
      "DepProperty1", CompleteCondition.OP_EQUAL, false,
      AgentProperty.TYPE_INTEGER, "1");

   condition1.allDependentConditions[2] = new DependentCondition(
      "DepProperty1", CompleteCondition.OP_EQUAL, false,
      AgentProperty.TYPE_INTEGER, "256");

   condition1.allDependentConditions[3] = new DependentCondition(
      "DepProperty1", CompleteCondition.OP_EQUAL, false,
      AgentProperty.TYPE_INTEGER, "512");

// 3. Instantiate the next complete-condition object
condition2 = new CompleteCondition();

// 4. Create the condition on the "optionB" value
//   a) Instantiate the input condition on "optionB"
   condition2.allInputConditions[0] = new InputCondition(
      CompleteCondition.OP_EQUAL, false, AgentProperty.TYPE_STRING,
      "optionB");

//   b) Instantiate the dependent conditions for DepProperty2
   condition2.allDependentConditions[0] = new DependentCondition(
      "DepProperty2", CompleteCondition.OP_GREATER_THAN_EQUAL, false,
      AgentProperty.TYPE_INTEGER, "1");

   condition2.allDependentConditions[1] = new DependentCondition(
      "DepProperty2", CompleteCondition.OP_LESS_THAN_EQUAL, false,
      AgentProperty.TYPE_INTEGER, "5");

// Save conditions in the agent-property object
agentProp.allDependencies[0] = condition1;
agentProp.allDependencies[1] = condition2;
```

図 74. 2 つのエージェント・プロパティー条件の定義

ODA のシャットダウン

ODA が適切なコンテンツを生成した後、ビジネス・オブジェクト・ウィザードは「ビジネス・オブジェクトの保管」ダイアログ・ボックス (ステップ 6) を表示します。このダイアログ・ボックスでは、生成したコンテンツの保管方法をユーザーが指定できます。このステップの一環として、ビジネス・オブジェクト・ウィザードは ODA を終了します。ODA ランタイムは、`terminate()` メソッドを呼び出してクリーンアップ・タスクを実行し、ODA のリソースを解放します。例えば、ODA が `init()` メソッドでデータ・ソースに接続した場合、このソースから切断するには `terminate()` メソッドを使用する必要があります。 ODK API における ODA の `terminate()` メソッドは、低レベル ODA 基底クラス (`ODKAgentBase`) に含まれています。このクラスは、ODA 基底クラスの `ODKAgentBase2` からユーザーの ODA クラスへと順に継承されます。

図 75 は、データベース接続を閉じ、データベースにアクセスしたオブジェクトをクリーンアップする ODA のサンプル terminate() メソッドを示しています。

```
public void terminate()
{
   specList = null;
   //close connection
   if(db != null)
      db.disconnect();
   if(dbAnalizer != null)
      dbAnalizer.cleanup();
 }
```

図 75. サンプルの ODA terminate() メソッド

トレース・メッセージとエラー・メッセージの処理

メッセージ は、ODA が外部の ODA ログに送信することができる情報のストリングです。システム管理者や開発者は、このログでメッセージを確認することにより、ODA の実行時の状況を知ることができます。 ODA が ODA ログに送信できるメッセージのカテゴリーは次の 2 つです。

- エラー・メッセージまたは通知メッセージ
- トレース・メッセージ

メッセージは、ODA コード内で生成することも、メッセージ・ファイルから取得することもできます。 ODK API は、トレース・メッセージとエラー・メッセージをログに記録する trace() メソッドを用意しています。このメソッドは、ODKUtility クラス内で定義されています。このセクションでは、次のトピックについて説明します。

- 『ログの宛先の指定』
- 176 ページの『トレース・ファイルへのメッセージの送信』
- 179 ページの『メッセージ・ファイル』

ログの宛先の指定

ODA はメッセージをログの宛先に送信します。*ログ* は外部の宛先であり、ODA の開始状態を確認する必要のあるユーザーが参照するために用意されています。ログの宛先は、外部ファイルの絶対パス名として、ODA の構成時に構成プロパティー TraceFileName によって定義されます。ログの宛先は、ODA のプロセスと同じマシン上に存在している必要があります。

注: ODK API では、1 つのメソッドでトレース・メッセージとエラー・メッセージの両方を記録します。そのため、ODA はこれらの 2 種類のメッセージを 1 つのファイルに記録します。したがって、このファイルは*トレース・ファイル* と呼ばれていますが、ODA が生成するエラー・メッセージも中に含まれます。

トレース・ファイル名の形式については、 89 ページの『トレース・ファイルの指定』を参照してください。

トレース・ファイルへのメッセージの送信

ODK API は、トレース・メッセージとエラー・メッセージをログに記録する trace() メソッドを用意しています。このメソッドは、ODKUtility クラス内で定義されています。 ODA トレース機構によって送信されるメッセージのタイプは、メッセージのトレース・レベルによって異なります。

表 57. トレース・レベルとメッセージ・タイプ

メッセージのトレース・レベル	説明
ゼロ (0)	ODA トレース機構を使用して、エラー・メッセージ のログがトレース・ファイルに記録されます。
1 から 5 の間のレベル	ODA トレース機構を使用して、トレース・メッセージのログがトレース・ファイルに記録されます。トレース・メッセージは、状況メッセージ、プロパティー値、およびビジネス・オブジェクト名などの通知を目的としています。

注: ODA ランタイムは、ODA トレース機構を暗黙的に操作します。トレース機構は、ビジネス・オブジェクト・ウィザードの「エージェントの構成」ダイアログ・ボックスでトレース・ファイルを設定するまでは機能しません。詳しくは、117 ページの『ODA の開始』を参照してください。

trace() を呼び出す際は、トレース・レベルを引き数として指定します。そのために、ODK API はトレース・レベル定数を用意しています。メッセージの生成方法については、 180 ページの『メッセージ・ストリングの生成』を参照してください。トレース・レベルの設定については、89 ページの『トレース・ファイルとトレース・レベルの指定』を参照してください。

ODA トレース機構は、Connector Development Kit および InterChange Server Express と同じ形式のファイルを生成します。

エラー・メッセージと通知メッセージ

トレース・レベルをゼロ (0) に設定すると、ODA は状況に関する情報をログの宛先に送信します。エラーおよび状況の記録を作成することは、一般に*ロギング* と呼ばれます。次のタイプの情報をログの対象にすることをお勧めします。

- エラーおよび致命的エラー (コードからログ・ファイルに送信)
- 警告: システム管理者の注意を喚起する必要があるもの (コードからログ・ファイルに送信)
- 次のような通知メッセージ:
  - ODA の開始および終了のメッセージ
  - アプリケーションからの重要なメッセージ

ODA は通知メッセージやエラー・メッセージを送信できますが、このロギング・プロセスはエラー・*ロギング* と呼ばれます。

重要: 例外が発生した場合は必ず、例外をスローしてビジネス・オブジェクト・ウィザードにその例外を表示し、例外を説明するエラー・メッセージをトレース・ファイルに書き込むことをお勧めします。例外をすべてトレース・ファ

イルに記録すれば、ODA または Business Object Designer Express に万一障害が発生したとしても、例外を特定できるようになります。

トレース・レベルがゼロ (0) のときは、エラー・ロギングがオンになります。デフォルトでは、トレース・レベルが 5 に設定されているため、ODA のロギングはオフになっています。トレース・レベルは、TraceLevel ODA 構成プロパティーで設定します。 TraceLevel に 0 の値を設定すると、メッセージがエラー・メッセージに指定されます。

エラー・メッセージをログに送信するには、trace() メソッドを使用します。表 58 は、エラー・メッセージを送信する trace() のトレース情報をまとめたものです。

表 58. エラー・メッセージのトレース情報

トレース情報	説明	ODKConstant 定数
トレース・レベル メッセージ・タイプ	0 エラー 警告 通知	TRACELEVEL0 XRD_FATAL、XRD_ERROR XRD_URGENTWARNING、 XRD_WARNING XRD_INFO

表 58 に示す情報の他に、trace() メソッドはエラー・メッセージの内容も要求します。メッセージの内容は、次のいずれかの方法により、メッセージ・テキストとして取得することができます。

- メッセージ・ストリング (String 値)

```
Util.trace(ODKConstant.TRACELEVEL0, ODKConstant.XRD_ERROR,
      "Invalid property value");
```

ODKException クラスの getMsg() メソッドを使用すれば、例外からメッセージ・ストリングを取得することもできます。

```
try
   {
   boDef.setAttributeList(Attributes);
   boDef.setVerbList(Verbs);
   defList[i] = boDef;
   }
catch(BusObjInvalidAttrException e)
   {
   Util.trace(ODKConstant.TRACELEVEL0,
ODKConstant.XRD_ERROR, e.getMsg());
   }
```

- メッセージ・ファイルから取得するメッセージ

```
Util.trace(ODKConstant.TRACELEVEL0, ODKConstant.XRD_WARNING, 1009);
```

メッセージ・ファイルの使用方法については、 179 ページの『メッセージ・ファイル』を参照してください。

トレース・メッセージ

トレースは、ODA でオンにすることができるトラブルシューティングおよびデバッグ用のオプションの機能です。トレースをオンにすると、システム管理者は、ODA

がタスクを実行している間にコンテンツ生成を追跡できます。トレースを使用すると、開発者や、開発者が作成した ODA コードを使用するその他のユーザーは、ODA の振る舞いをモニターできます。トレースは、特定の ODA メソッドを呼び出したときに追跡することもできます。

トレースは、トレース・レベルが 1 から 5 のときにオンになります。デフォルトではトレース・レベルが 5 に設定されているため、ODA のトレースはオンになっています。トレース・レベルは、TraceLevel ODA 構成プロパティーで設定します。 TraceLevel は、詳細レベルに応じて 1 から 5 を設定できます。トレース・レベルを 5 に設定すると、それより下のレベルのトレース・メッセージがすべてログに記録されます。ODA が各トレース・レベルで戻す情報のタイプを定義するのは開発者の責任です。 90 ページの表 15 は、推奨される ODA トレース・メッセージの内容を示しています。詳しくは、 90 ページの『トレース・レベルの設定』を参照してください。

トレース・メッセージをトレース・ファイルに送信するには、trace() メソッドを使用します。表 58 は、トレース・メッセージを送信する trace() のトレース情報をまとめたものです。

表 59. トレース・メッセージのトレース情報

トレース情報	説明	ODKConstant 定数
トレース・レベル	1	TRACELEVEL1
	2	TRACELEVEL2
	3	TRACELEVEL3
	4	TRACELEVEL4
	5	TRACELEVEL5
メッセージ・タイプ	トレース	XRD_TRACE

表 59 に示す情報の他に、trace() メソッドはトレース・メッセージの内容も要求します。メッセージの内容は、次のいずれかの方法で取得できます。

- メッセージ・テキストとして:
  - メッセージ・ストリング (String 値)

    ```
    Util.trace(ODKConstant.TRACELEVEL1, ODKConstant.XRD_TRACE,
          "Entering method getProperties");
    ```

  - メッセージ・ファイルから取得するメッセージ

    ```
    Util.trace(ODKConstant.TRACELEVEL1, ODKConstant.XRD_TRACE, 1009);
    ```

    メッセージ・ファイルの使用方法については、 179 ページの『メッセージ・ファイル』を参照してください。
- ビジネス・オブジェクト定義 (BusObjDef オブジェクト) として

  この場合、trace() は、指定されたビジネス・オブジェクト定義のコンテンツをフォーマットします。

  ```
  BusObjDef boDef = new BusObjDef();
  // code that populates business object definition
  ...
  // write out the business object definition
  ODKUtility.getODKUtility().trace(ODKConstant.TRACELEVEL5,
     ODKConstant.XRD_TRACE, boDef);
  ```

- エージェント・プロパティー (AgentProperty オブジェクト) の配列として

この場合、trace() は、指定可能なストリングを前に付けてエージェント・プロパティーのリストをフォーマットします。

```
AgentProperties[] propArray;
// code that populates agent-property array
...
// write out the agent-property array
ODKUtility.getODKUtility().trace(ODKConstant.TRACELEVEL2,
   ODKConstant.XRD_TRACE, propArray, "List of configuration properties:");
```

メッセージ・ファイル

エラー・メッセージの場合でもトレース・メッセージの場合でも、メッセージの内容は、ハードコーディングされたストリング、またはメッセージ・ファイル から取得したストリングとして提供できます。メッセージ・ファイルは、メッセージ番号とそれに関連するメッセージ・テキストが含まれたテキスト・ファイルです。メッセージ・テキストには、ODA からの実行時データを渡すための定位置パラメーターを含めることができます。メッセージ・ファイルは、ファイルを作成し、必要なメッセージを定義することにより提供できます。

このセクションでは、メッセージ・ファイルに関して次のトピックを取り上げます。

- 『メッセージ・フォーマット』
- 180 ページの『メッセージ・ファイルの名前および位置』
- 180 ページの『メッセージ・ストリングの生成』
- 182 ページの『メッセージ・ファイルの保守』

メッセージ・フォーマット

メッセージ・ファイル内のメッセージの形式は次のとおりです。

```
MessageNum
Message
[EXPL]
Explanation
```

MessageNum は、各メッセージを固有に識別する整数です。このメッセージ番号は、1 行で指定する必要があります。メッセージ・テキストは、複数の行にまたがって記述できます (各行は復帰で終了します)。*説明* テキストには、メッセージを表示する原因となった状況をより詳細に記述します。説明テキストの最後の行の後ろには空白行を入れないでください。次のメッセージの番号が説明テキストの次の行に来るようにします。メッセージ・ファイルは、メモ帳など任意のテキスト・エディターで編集してください。

例えば、メッセージ番号 1005 は次のようになります。

```
1005
ODA content generation is complete.
[EXPL]
This is a log message that indicates successful completion of the ODA.
```

メッセージには、実行時にプログラムからの値で置き換えるパラメーターを含めることができます。このパラメーターは定位置パラメーターであり、中括弧付きの番

号でメッセージ内に指定します。例えば、次のメッセージには、エージェント・プロパティー名を指定するパラメーターが 3 つ含まれています。

```
1003
The agent configuration properties are {1}, {2}, {3}.
[EXPL]
This is a trace message that provides startup properties.
```

メッセージ・パラメーターの指定方法については、181 ページの『パラメーター値の使用』を参照してください。

メッセージ・ファイルの名前および位置

ODA は、次の 2 つのメッセージ・ファイルのいずれかからメッセージを取得できます。

- ODA メッセージ・ファイル

  *ODAname*Agent.txt

  ここで、*ODAname* は ODA を固有に識別する名前です。詳しくは、185 ページの『ODA の名前付け』を参照してください。このメッセージ・ファイルには、ODA に固有のメッセージを含めます。例えば、LegacyApp という名前の ODA を作成した場合は、そのメッセージ・ファイルに LegacyAppAgent.txt という名前を付けます。

  注: ビジネス・オブジェクト・ウィザードは、自動的に構成プロパティーのリストに MessageFile を含め、*ODAname* Agent.txt という形式のメッセージ・ファイル名を指定します。 ODA を構成する際は、このメッセージ・ファイル名を変更し、既存のファイルを指し示すことができます。ODA が実行を継続するためには、指定したメッセージ・ファイルが存在しなければなりません。メッセージ・ファイルの指定方法については、90 ページの『ODA メッセージ・ファイルの指定』を参照してください。

- グローバル ODA メッセージ・ファイル useragentmessages.txt。

  すべての Object Discovery Agent に対してグローバルなメッセージを作成する場合は、それらのメッセージをグローバル・メッセージ・ファイルに追加します。

上記 2 つのメッセージ・ファイルは、製品ディレクトリーの次のサブディレクトリーに含める必要があります。

ProductDir¥ODA¥messages

メッセージ・ストリングの生成

次のメソッドは、定義済みのメッセージをメッセージ・ファイルから取得します。

表 60. メッセージ・ストリングを生成するメソッド

メッセージ・メソッド	説明
getMsg()	指定した重大度のメッセージをメッセージ・ファイルから生成します。
trace()	指定した重大度のメッセージをメッセージ・ファイルから生成し、それをトレース・ファイルに送ります。

表 60 に示すメッセージ生成メソッドは、ODKUtility クラス内に定義されています。これらのメソッドは、次の情報を必要とします。

- 『メッセージ番号の指定』
- 『メッセージ・タイプの指定』
- 『パラメーター値の使用』

メッセージ番号の指定: 表 60 に示すメッセージ生成メソッドは、引き数としてメッセージ番号を必要とします。この引き数は、メッセージ・ファイルから取得するメッセージの番号を指定します。 179 ページの『メッセージ・フォーマット』で説明したように、メッセージ・ファイル内の各メッセージには、固有のメッセージ番号 (整数) を付ける必要があります。これらのメッセージ生成メソッドは、指定されたメッセージ番号がメッセージ・ファイル内にあるかどうかを検索し、その番号が付いているメッセージ・テキストを抽出します。

これらのメソッドは、該当するメッセージ番号を探す際に、次の順序で ODA メッセージ・ファイルを検索します。

1. ODA 固有のメッセージ・ファイル (デフォルト名は *ODAname*Agent.txt)
2. グローバル ODK メッセージ・ファイル (useragentmessages.txt)

メッセージ・タイプの指定: 180 ページの表 60 に示すメッセージ生成メソッドは、引き数としてメッセージ・タイプも必要です。この引き数は、メッセージの重大度 を指定します。 表 61 は、有効なメッセージ・タイプとそれに関連するメッセージ・タイプ定数を示しています。

表 61. メッセージ・タイプ

メッセージ・タイプ定数	重大度レベル	説明
XRD_FATAL	致命的エラー	プログラムの実行が停止するエラーを示します。
XRD_ERROR	エラー	調査が必要なエラーを示します。
XRD_URGENTWARNING	緊急警告	問題である可能性が高く、無視しないほうがよい状況を示します。
XRD_WARNING	警告	問題である可能性があるものの、無視しても構わない状況を示します。
XRD_INFO	通知	単なる通知メッセージ。アクションは不要です。
XRD_TRACE	----	トレース・メッセージで使用します。

メッセージに関連付けるメッセージ・タイプを指定するには、表 61 に示すメッセージ・タイプ定数のいずれかを使用します。

- エラー・メッセージの場合は、メッセージの重大度を示すメッセージ・タイプ定数を使用します (メッセージ・タイプ定数は、重大度が高い順に XRD_FATAL、XRD_ERROR、XRD_URGENTWARNING、XRD_WARNING、XRD_INFO となります)。
- トレース・メッセージの場合は、XRD_TRACE 定数を使用します。

メッセージ・タイプ定数は、ODKConstant クラスで定義されています。

パラメーター値の使用: 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、{*number*} のように中括弧で囲まれた番号として表示されます。メッセージに追加したい各パラメーターには、以下のように中括弧で囲んだ番号をメッセージのテキストに挿入します。

```
message text {number} more message text.
```

180 ページの表 60 に示すメッセージ生成メソッドでは、メッセージ・パラメーターとしてオプションの数だけ値を指定できます。メソッド呼び出し内のパラメーター値の数は、メッセージ・テキスト内に定義されているパラメーターの数と同じでなければなりません。メッセージ生成メソッドは、各パラメーターの値を提供する必要があります。例として、メッセージ 1003 を再度見てみましょう。

```
1003
The agent configuration properties are {1}, {2}, {3}.
[EXPL]
This is a trace message that provides startup properties.
```

このメッセージを送信するコードには次のような行があります。

```
Vector params = new Vector(3);
for(int i=0; i<3; i++)
   params.add(agtProperties[i].propName);
Util.trace(ODKConstant.TRACELEVEL2, 1003, ODKConstant.XRD_TRACE,
   params);
```

trace() メソッドは、メッセージ・ファイル内のメッセージ・テキストにこれらのパラメーター値を組み合わせてメッセージを作成します。メッセージをトレース・ファイルに書き込む前に、trace() は、メッセージ・パラメーターを params 変数の値で置き換えます。

例えば、メッセージ 1003 は、トレース・ファイル内で次のように表示されるでしょう。

```
The agent configuration properties are Username, Password, Url.
```

メッセージ・テキストではパラメーターを使って特定のプロパティー・タイプを指定するので、ハードコーディングされた文字ストリングとして組み込む代わりに、欠落したプロパティーの任意のセットに対して同じメッセージを使用できます。

メッセージ・ファイルの保守

ユーザー・サイトでは、ODA 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管理者が設定する場合があります。そのため、エラー番号とその番号に対応する意味が Object Discovery Agent の最初のリリースの前後で変わらないようにすることが重要です。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割り当て直すことは推奨できません。

ただし、それでもエラー番号に対応する意味を変更するという場合は、必ずその変更を文書化して Object Discovery Agent のユーザーに通知してください。

Object Discovery Agent の実行中は、Object Discovery Agent のメッセージ・ファイルを変更することができます。ただし、この変更は、次に Object Discovery Agent

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。Object Discovery Agent の実行中に InterChange Server Express に障害が発生すると、サーバーは、以前に実行中であったすべての Object Discovery Agent のメッセージ・ファイルを、メモリー内に自動的に読み込みます。

例外処理

ODK API のメソッドは、事前に定義した特定の条件を示す例外をスローできます。このセクションでは、Java コネクターの例外の処理方法に関して、次のトピックを取り上げます。

- 『ODK 例外とは』
- 『ODK API ライブラリーからの例外』

注: エラー・ログとメッセージ・ログを使用すれば、エラーの条件およびメッセージをコネクターで処理することもできます。詳しくは、175 ページの『トレース・メッセージとエラー・メッセージの処理』を参照してください。

ODK 例外とは

ODK API のメソッドが例外をスローする場合、この*例外オブジェクト* は、ODKException クラス、またはそのサブクラスからのオブジェクトです。これらのクラスは、Java Exception クラスから拡張されたものです。 ODK 例外を作成するには、ODKException() コンストラクターを使用します。表 62 は、例外オブジェクトの情報を取得するために ODKException クラスが提供する accessor メソッドを示しています。

表 62. 例外オブジェクト内の情報

メンバー	accessor メソッド
メッセージ・テキスト	getMsg()

注: ODKException クラス内のメソッドについては、289 ページの『第 23 章 ODKException クラス』を参照してください。

ODKException クラスは、特定のエラー条件を示すサブクラスをいくつか提供します(290 ページの表 103 を参照)。

ODK API ライブラリーからの例外

ODA 用のコードを作成する際に Java の try ステートメントと catch ステートメントを含めれば、ODK API のメソッドがスローした特定の例外を処理することができます。大半の ODK API メソッドのリファレンスの説明には、そのメソッドがスローする例外をリストした、「Exceptions」というタイトルの付いたセクションがあります。

図 76 は、ArmyAgent4 クラス内のサンプル Roman Army ODA からのコード・フラグメントで、getClientFile() メソッドがスローした例外をキャッチします。

```
try
   {
   remotefile = ODKUtility.getODKUtility().getClientFile(filePath, this);
   }
catch (IOException ex)            //file was not found
   {
    return null;
   }
//agent doesn't implement IGeneratesBinFiles, so "getClientFile" failed.
catch (UnsupportedContentException ex)
   {                              //We'll return a random Son instance for now.
    return new Son("X" + ("" + new Date().hashCode()).substring(1),
       new Date().hashCode() % 10  + 2);
   }
```

図 76. getClientFile() からの例外のキャッチ

一般に ODK API メソッドは、例外をスローする際に、メッセージおよび状況の情報を例外オブジェクトに含めません。ただし、必要な場合はメッセージを例外オブジェクトに含めることもできます。

第 6 章 ビジネス・インテグレーション・システムへの Object Discovery Agent の追加

ユーザーが開発した Object Discovery Agent (ODA) に WebSphere Business Integration システムがアクセスできるようにするには、以下のステップを実行する必要があります。

1. ODA 名およびその命名規則を確立します。
2. ODA クラスを JAR ファイルにコンパイルします。
3. ODA の始動スクリプトを作成します。

ODA の名前付け

この章では、ODA の開発で使用されるファイルおよびディレクトリー用の推奨される命名規則について説明します。命名規則は、ODA のコードの配置および識別を容易にする手段になります。表 63 は、ODA 用の推奨される命名規則に関する要約です。

表 63. ODA 用の推奨される命名規則

ODA 名	ODA パッケージとクラス名	ODA 始動スクリプト	ODA ライブラリー・ファイル	ODA ランタイム・ディレクトリー
*srcDataName*ODA	com.ibm.oda.*srcDataName*.*ODAname*	start_*ODAname*	*ODAname*.jar	ODA¥*srcDataName*

各 ODA には、WebSphere Business Integration システム内部で一意に識別される名前を付ける必要があります。命名規則では、ODA 名 (*ODAname*) は以下の形式をとります。

*srcDataName*ODA

ここで、*srcDataName* は、ODA により変換されるソース・データを識別する一意の文字列です。例えば、ODA が HTML オブジェクトをビジネス・オブジェクト定義に変換した場合、そのソース・データは HTML 形式になるため、ODA 名は HTMLODA になります。また、この ODA 名で、ODA が関連付けられているアダプターを識別することもできます。例えば、WebSphere Business Integration Adapter for PeopleSoft 用のビジネス・オブジェクト定義を生成する ODA は、PeopleSoftODA のような名前になります。

ODA のコンパイル

ODA をコンパイルするには、以下のステップを実行します。

- JDK 開発環境を使用します。JDK のインストール方法については、 113 ページの『開発環境の設定』を参照してください。

- Object Discovery Agent Development Kit (ODK) API 用のライブラリー・ファイルが、製品ディレクトリーの lib サブディレクトリーにあることを確認してください。メインの ODK API ライブラリー・ファイルの名前は、以下のようになります。

  CwODK.jar

  その他に、以下の ODK ライブラリー・ファイルがあります。

  xrmi.jar, xerces.jar

- Java コンパイラーを使用して ODA ソース (.java) ファイルをクラス (.class) ファイルにコンパイルします。

  これらのファイルには、ODA クラス (ODAAgentBase2 クラスの拡張) のソース、および ODA が使用するその他のクラスのソースを組み込みます。 ODK クラス・ファイルの名前付けについては、 116 ページの『ODA 基底クラスの拡張』を参照してください。

- ODA のライブラリー・ファイルを作成します。このライブラリー・ファイルは、コンパイル済み Java コードを含んだ Java アーカイブ (JAR) ファイルです。

  命名規則では、JAR ファイル名は以下の形式をとります。

  *srcDataName*ODA.jar

  ここで、*srcDataName* は ODA のソース・データ (またはアダプター) を一意に識別します。ODA 名について詳しくは、 185 ページの『ODA の名前付け』を参照してください。

  例えば HTML で動作する ODA には HTMLODA のような ODA 名を付けることができるため、この ODA の JAR ファイルは

  HTMLODA.jar

  のような名前になります。

新規の ODA の始動

ODA を開始するには、*ODA 始動スクリプト* を実行してください。この始動スクリプトによって ODA ランタイムが開始します。この始動スクリプトは、ODA ランタイムを開始するバッチ・ファイルです。命名規則では、始動スクリプト名は以下の形式をとります。

start_*ODAName*.bat

ここで、ODAname は ODA の一意名 (ODA のソース・データ名) であり、この名前には接尾語としてストリング「ODA」が付加されます。例えば、ODA のソース・データが HTML 形式である場合、ODA には HTMLODA のような ODA 名を付けることができるため、この ODA の始動スクリプトは

start_HTMLODA.bat

のような名前になります。

作成した ODA を起動する前に、新規 ODA をサポートする始動スクリプトが存在することを確認しておく必要があります。始動スクリプトにユーザー独自の ODA を始動させるには、以下のステップを実行する必要があります。

1. 使用する ODA 用に ODA ランタイム・ディレクトリーを作成します。
2. ODA の始動スクリプトを作成します。Windows システムの場合、ODA 始動用のショートカットも作成してください。
3. 始動スクリプトを Windows サービスとしてセットアップします (オプショナル)。

これらの各ステップについては、以降のセクションで説明します。

ODA ランタイム・ディレクトリーの作成

ODA ランタイム・ディレクトリー には、ODA 用のランタイム・ファイルが格納されます。ODA ランタイム・ディレクトリーを作成するには、以下のステップを実行します。

1. 製品ディレクトリーの ODA サブディレクトリーの下に、新規 ODA 用の ODA ランタイム・ディレクトリーを作成します。

   ProductDir¥ODA¥*srcDataName*

   命名規則では、ディレクトリー名は ODA のソース・データ名 (*srcDataName*) と一致します。ソース・データ名は、ODA が連携動作するソース・データ (またはアダプター) を一意に識別する文字列です。詳しくは、185 ページの『ODA の名前付け』を参照してください。
2. この ODA ランタイム・ディレクトリーに ODA のライブラリー・ファイルを移動します。

   ODA のライブラリー・ファイルは Java アーカイブ (JAR) ファイルです。この JAR ファイルは ODA のコンパイル時に作成されています。詳しくは、185 ページの『ODA のコンパイル』を参照してください。

始動スクリプトの作成

76 ページの『システム始動ファイル』に記載されているように、ODA の開始を可能にするには、ODA 始動スクリプトが必要です。ODA では、システム管理者が ODA ランタイム・プロセスを開始するための始動スクリプトが必要です。WebSphere Business Integration Adapters インストーラーで Windows システムにアダプターをインストールするときには、ODA 用に以下のステップが実行されます。

- start_*ODAname*.bat 始動スクリプトを製品ディレクトリーの ODA¥*srcDataName* サブディレクトリーにインストールします。
- 各 ODA 用のメニュー・オプションを 「プログラム」>「IBM WebSphere Business Integration Adapters」>「Adapters」>「Object Discovery Agents」メニューの下に作成します。各メニュー項目は、各 ODA 用の Windows 始動スクリプト start_*ODAname*.bat を起動するショートカットです。

ユーザー独自の ODA を始動できるようにするには、始動スクリプトを生成し、この始動スクリプトを起動するためのショートカットを用意しておく必要があります。

始動スクリプトの作成

この start_*ODAname*.bat ファイル内で、以下のステップが必ず実行されるようにしてください。

- 始動スクリプト内に、以下の変数を設定します。

変数名	値
PATH	ODA のランタイム・ディレクトリーのパスを PATH 変数の前に追加します (これにより、そのランタイムが ODA の JRE を検索できるようになります)。 PATH="%CROSSWORLDS%"¥ODA¥*ODAruntimeDir*;%PATH% ここで、*ODAruntimeDir* は ODA のランタイム・ディレクトリーであり、*srcDataName* という形式をとります。詳細については、 187 ページの『ODA ランタイム・ディレクトリーの作成』を参照してください。
AGENTNAME	ODA 用の ODA 名 (*ODAname*) を指定します。ODA 名は以下の形式をとります。 *srcDataName*ODA ここで、*srcDataName* はソース・データの名前です。詳しくは、 185 ページの『ODA の名前付け』を参照してください。
AGENT	ODA のライブラリー・ファイル (ODA クラスを含んだ JAR ファイル) のフル・パス名を指定します。このパス名は以下の形式をとります。 "%CROSSWORLDS%"¥ODA¥*ODAruntimeDir*¥*ODAlibrary*.jar ここで: • *ODAruntimeDir* は ODA のランタイム・ディレクトリーであり、*srcDataName* という形式をとります。詳細については、 187 ページの『ODA ランタイム・ディレクトリーの作成』を参照してください。 • *ODAlibrary* は *ODAname*.jar という形式の ODA 用ライブラリー・ファイルです。詳しくは、 185 ページの『ODA のコンパイル』を参照してください。
AGENTCLASS	ODA パッケージおよびクラスの名前を指定します。この名前は以下の形式をとります。 com.ibm.oda.*srcDataName*.*ODAname* ここで: • *srcDataName* は ODA のソース・データの名前 (すべて小文字) です。詳しくは、 185 ページの『ODA の名前付け』を参照してください。 • *ODAname* は、ODA のクラス (ODA の基底クラスである ODKAgentBase2 を ODA クラスにより拡張したもの) の名前です。

変数名	値
JCLASSES	この変数に ODA 固有の JAR ファイルを追加します。JAR ファイル同士は、セミコロン (;) で区切ります。この変数は、最小限でも以下のクラスを組み込むように設定する必要があります。 • ODK ライブラリー・ファイル: CwODK.jar、xrmi.jar、xerces.jar • AGENT 変数 (上記参照) に格納されている ODA ライブラリー・ファイル

- 始動スクリプトに必要とされる ODA 固有の変数を、さらに定義して設定します。

  リリースごとに変更される可能性のある情報に合わせて、変数を定義します。さらに、このリリースに適切な値に変数を設定してから、始動スクリプトの適切なコマンド行の中に変数を含めることができます。今後情報に変更があった場合は、変数の値を変更するだけで済みます。この情報を使用するすべてのコマンド行を見つける必要はありません。
- ODA ランタイムを起動する行 (始動スクリプトの最後の行) に、適切な始動パラメーターを入力します。以下のパラメーターを含めてください。
  - 必要なすべての始動パラメーター: -l および -c
  - ODA の起動時に毎回適用するオプショナルの始動パラメーター:

    -v。このパラメーターの後ろに ODA のバージョンを続けます。
- ODA ランタイムを起動する行は、以下の形式でなければなりません。

```
"%CROSSWORLDS%¥bin¥java" -Duser.home="%CROSSWORLDS%" -mx128m
-classpath %JCLASSES% com.crossworlds.ODKInfrastructure.XRmiAgent
-l%AGENTNAME% -c%AGENTCLASS%
```

  注: 始動スクリプト内では ODA ランタイムを起動する行を必ず *1 行に収める* ようにしてください。つまり、サンプルの始動行の中に示されている改行の位置に復帰改行文字が存在していてはなりません。

ショートカットの作成

ショートカットを作成しておくと、「プログラム」>「IBM WebSphere Business Integration Adapters」>「Adapters」>「Object Discovery Agents」 内のメニュー項目から ODA を起動できます。Windows 上で稼働する ODA を起動するためのショートカットを容易に作成する方法は、既存の ODA 用ショートカットをコピーして、ショートカットのプロパティーの編集 (コネクター名の変更やその他の始動パラメーターの追加) を行うことです。

第 3 部 ODK クラスの解説

第 7 章 ODK API の概要

Object Discovery Agent (ODA) の開発に必要なクラス・ライブラリーは、Object Discovery Agent Development Kit (ODK) アプリケーション・プログラミング・インターフェース (API) に組み込まれています。この ODK API には、ODA 用の定義済みクラスが含まれています。これらのクラス・ライブラリーを使用して、ODA のクラスおよびメソッドを導出できます。 ODK API はそのほか、ユーティリティー (例えば、トレース・サービスやロギング・サービスを実装するメソッドなど) も提供しています。

IBM 提供の Java JAR ファイル (Java アーカイブ・ファイル) である `CwODK.jar` には、ODK API の定義済みクラスおよびインターフェースが収録されています。この JAR ファイルは、製品ディレクトリーの `lib` サブディレクトリーにあります。

注: Windows 2000 上で稼働する ODA をビルドする手順については、185 ページの『ODA のコンパイル』を参照してください。

クラスとインターフェース

ODK API のクラスおよびインターフェースは、以下のパッケージに収録されています。

`com.crossworlds.ODK`

表 64 は、ODK API のクラスおよびインターフェースの一覧です。

表 64. ODK API のクラスとインターフェース

クラスまたはインターフェース	説明	ページ
`AgentMetaData`	始動プロパティーまたはビジネス・オブジェクトのプロパティーを表すことのできる、エージェント・プロパティー・オブジェクトを表します。	195
`AgentProperty`	属性タイプ定数を定義します。	201
`BusObjAttr`	ビジネス・オブジェクト定義の内部にある属性を表します。	211
`BusObjAttrType`	属性タイプ定数を定義するインターフェースです。	
`BusObjDef`	ビジネス・オブジェクト定義 (ビジネス・オブジェクトについての記述) を表します。	229
`BusObjVerb`	ビジネス・オブジェクトの動詞 (ビジネス・オブジェクトに対して有効なアクションまたは操作についての記述内容) を表します。	241
`CompleteCondition`		245
`ContentMetaData`	ODA のコンテンツ・メタデータ (ODA が生成したコンテンツについての記述内容) を表します。	249
`ContentType`	ODA がサポートしているコンテンツ・タイプを表します。	255

表 64. ODK API のクラスとインターフェース (続き)

クラスまたはインターフェース	説明	ページ
DependentCondition		259
IGeneratesBinFiles	ソース・データを基にしたバイナリー・ファイルの生成をサポートするために ODA によって実装されるインターフェースです。	263
IGeneratesBoDefs	ソース・データを基にしたビジネス・オブジェクト定義の生成をサポートするために ODA によって実装されるインターフェースです。	267
IGeneratesContent	2 つのコンテンツ生成インターフェース (IGeneratesBinFiles および IGeneratesBoDefs) 用の基底クラスです。getContentProtocol() メソッドを定義します。 **注:** このマニュアルには、このインターフェースに関する独立した章を設けてありません。getContentProtocol() について詳しくは、IGeneratesBinFiles インターフェースまたは IGeneratesBoDefs インターフェースの解説を参照してください。	なし
InputCondition		273
ODKAgentBase	ODA 基底クラス (ODKAgentBase2) 用の基底クラスです。ODKAgentBase2 が継承するメソッドをいくつか定義します。 **注:** 本書には、このクラスに関する独立した章を設けてありません。詳しくは、ODKAgentBase2 クラスの解説を参照してください。	なし
ODKAgentBase2	ODA の基底クラスを表します。ODA クラスを定義して必要なメソッドを実装するには、このクラスを拡張します。	277
ODKConstant	ODK API で使用される以下の定数を定義するインターフェースです。 • 結果ステータス定数 • 動詞定数	283
ODKException	ODK API の例外オブジェクトを表します。	289
ODKUtility	ODA で使用するための各種ユーティリティー・メソッドを提供します。これらのユーティリティー・メソッドは、以下の一般カテゴリーに分類されます。 • メッセージを生成、ロギングするための静的メソッド • ビジネス・オブジェクトを作成するための静的メソッド • コネクター構成プロパティーを取得するための静的メソッド • ロケール情報を取得するためのメソッド	291
TreeNode		307

第 8 章 AgentMetaData クラス

Object Discovery Agent Development Kit (ODK) API には、Object Discovery Agent (ODA) のメタデータの格納を目的とした AgentMetaData クラスが用意されています。このクラスのメンバー変数は、ODA のメタデータに対応しています。ビジネス・オブジェクト・ウィザードは、ODA のクラスに含まれる getMetaData() メソッドを呼び出すことにより、ODA のメタデータにアクセスできます。

AgentMetaData クラスには、次のものが定義されています。

- 『メンバー変数』
- 198 ページの『メソッド』

AgentMetaData クラスには、ODKConstant インターフェースが実装されています。したがって、AgentMetaData オブジェクトでは、ODKConstant に定義されている定数をすべて使用できます。ODKConstant インターフェースにより定義される定数の一覧については、283 ページの『第 22 章 ODKConstant インターフェース』を参照してください。

メンバー変数

表 65 に、AgentMetaData クラスのメンバー変数について要約します。

表 65. AgentMetaData クラスのメンバー変数

メンバー変数	説明	ページ
agentVersion	ODA のバージョンを指定します。	202
searchableNodes	ツリー・ノード内の展開可能なノードの子を、ユーザー指定のパターンで検索できるかどうかを決定します。	203
searchPatternDesc	有効な検索パターンの基準を説明するためにユーザーに対して表示する説明を指定します。	203
supportedContent	ODA が (サポートするコンテンツ・タイプごとに) サポートするコンテンツ・プロトコルの説明が格納されます。	203

agentVersion

ODA のバージョンを指定します。

タイプ

```
public String agentVersion
```

注記

agentVersion メンバー変数は、AgentMetaData() コンストラクターの 2 番目の形式の構文により初期化できます。agentVersion は、初期化しない場合、空ストリングになります。どの ODA の場合も、ODA のメタデータを初期化する getMetaData() メソッドの中で、ODA のバージョンを初期化する必要があります。

searchableNodes

ツリー・ノード内の展開可能なノードの子を、ユーザー指定の検索パターンで検索できるかどうかを示します。

タイプ

```
public boolean searchableNodes
```

注記

searchableNodes メンバー変数には、ユーザーがビジネス・オブジェクト・ウィザードの「ソースの選択」ダイアログ・ボックスでツリー・ノード内の展開可能なノードの子を検索できるかどうかを決定する Boolean 値が格納されます。

- この変数が true の場合、ユーザーは、ビジネス・オブジェクト・ウィザードで展開可能なノードの名前を右マウス・ボタンでクリックしたときに、**「項目を検索」**メニュー項目を使用できます。このメニュー項目をクリックすると、「検索パターンの入力」ダイアログ・ボックスを表示できます。このダイアログでは、検索パターンを指定できます。

  ビジネス・オブジェクト・ウィザードは、getTreeNodes() メソッドを呼び出し、ユーザー指定の検索パターンを渡して、親ノード内を検索します。getTreeNodes() メソッドは、名前がこの検索パターンに一致する子をデータ・ソース内で検索し、実際に一致した子のみを戻します。ビジネス・オブジェクト・ウィザードは、親ノードが展開されると、これらの子をユーザーに対して表示します。
- この変数が false の場合、ユーザーは、展開可能なノードの名前を右マウス・ボタンでクリックしても**「項目を検索」**メニュー項目を利用できません。この場合、getTreeNodes() メソッドでは、ユーザー指定の検索パターンを処理する必要がありません。

AgentMetaData() コンストラクターは、searchableNodes メンバー変数を初期化しません。searchableNodes は、初期化しない場合、値が false になります。ODA で検索パターン機能をサポートする場合は、ODA のクラスに含まれる getMetaData() メソッドの中で、searchableNodes メンバー変数を初期化する必要があります。詳しくは、 132 ページの『検索パターン機能の実装』を参照してください。

searchPatternDesc

ユーザーに対して表示する、有効な検索パターンの基準を示す説明を指定します。

タイプ

```
public String searchPatternDesc
```

注記

searchPatternDesc メンバー変数には、検索パターンについての説明が格納されます。この説明は、「検索パターンの入力」ダイアログ・ボックスに表示されます。ユーザーがソース・ノードを右クリックして**「項目を検索」**をクリックすると、ビジネス・オブジェクト・ウィザードはこのダイアログ・ボックスを表示します。こ

の説明は、ユーザーが検索基準を指定するときに配慮する必要があるセマンティクスについての情報を提供します。つまり、ODA に実装されている検索の条件を示します。このメンバー変数には、searchableNodes メンバー変数が true の場合に限り、有効な値が格納されます。ODA で検索パターン機能をサポートする場合は、ODA のクラスに含まれる getMetaData() メソッドの中で、searchPatternDesc メンバー変数を初期化する必要があります。詳しくは、132 ページの『検索パターン機能の実装』を参照してください。

supportedContent

ODA が (サポートするコンテンツ・タイプごとに) サポートするコンテンツ・プロトコルを示す Vector が格納されます。

タイプ

```
public Vector supportedContent
```

注記

supportedContent メンバー変数には、ODA がサポートする生成コンテンツを説明する ContentProtocol オブジェクトで構成された、Java の java.util.Vector が格納されます。各 ContentProtocol オブジェクトに格納される情報は次のとおりです。

コンテンツ生成情報	説明
コンテンツ・タイプ	サポートされるコンテンツ・タイプを示す、ContentType オブジェクト。次のいずれかです。 • BusinessObject • BinaryFile
コンテンツ・プロトコル	指定されたコンテンツ・タイプ向けにサポートされているコンテンツ・プロトコルを示す、コンテンツ・プロトコル定数のマスク。次のいずれかです。 • CONTENT_PROTOCOL_ONREQUEST • CONTENT_PROTOCOL_CALLBACK コンテンツ・プロトコル定数は、ODKConstant インターフェースに定義されています。

注: ContentProtocol クラスは、ODA ランタイムとビジネス・オブジェクト・ウィザードが使用するクラスが含まれている ODAInfrastructure パッケージの一部です。このパッケージは、ODA 開発者が直接使用できるものではありません。ContentProtocol オブジェクトへのアクセスは、いずれも、ODA ランタイムまたはビジネス・オブジェクト・ウィザードによって処理されます。ODA がこのクラスのオブジェクトに直接アクセスすることはありません。

AgentMetaData() コンストラクターは、引き数として受け取った ODA オブジェクトに対して照会を行って、supportedContent メンバー変数を初期化します。このメンバー変数は、明示的に初期化する必要がありません。

メソッド

表 66 に、AgentMetaData クラスのメソッドについて要約します。

表 66. AgentMetaData クラスのメンバー・メソッド

メンバー・メソッド	説明	ページ
AgentMetaData()	エージェント・メタデータ・オブジェクトを作成します。	199
toXml()	指定されたプロパティーを現在の AgentProperty オブジェクト内にコピーします。	200

AgentMetaData()

エージェント・メタデータ・オブジェクトを作成します。

構文

```
public AgentMetaData(ODKAgentBase2 ODAobject);
public AgentProperty(ODKAgentBase2 ODAobject, String version);
```

パラメーター

ODAobject ODA を表す ODA オブジェクトへの参照です。AgentMetaData() コンストラクターは、作成する AgentMetaData オブジェクトの supportedContent メンバー変数の初期化のため、このパラメーターに指定されたオブジェクトに照会します (197 ページの『supportedContent』)。

version ODA のバージョンを指定します。AgentMetaData オブジェクトの agentVersion メンバー変数は、このパラメーターの値で初期化されます (195 ページの『agentVersion』)。

戻り値

新規にインスタンス化された AgentMetaData オブジェクト。

注記

AgentMetaData() メソッドは、*ODAobject* に指定された ODA でサポートされるコンテンツを、この ODA に照会します。このコンストラクターの新規 AgentMetaData オブジェクト生成構文について、以下に説明します。

- 最初の構文形式は、新しい AgentMetaData オブジェクトを定義し、サポートされるコンテンツに限り初期化します。この構文形式は、ODA にバージョンがないことを前提とした構文形式です。
- 2 番目の構文形式は、新しい AgentMetaData オブジェクトを定義し、そのオブジェクトに含まれる、サポートされるコンテンツとバージョンの両方を初期化します。

このコンストラクターの構文形式は、両方とも、*ODAobject* に指定された参照を使用して、ODA でサポートされるコンテンツを ODA に照会します。コンストラクターは、得られた情報を使用して、supportedContent メンバー変数を初期化します。

注: AgentMetaData() コンストラクターは、検索パターン機能をサポートするためのメンバー変数については初期化しません。ODA が検索パターンをサポートできるようにするには、searchableNodes メンバー変数および searchPatternDesc メンバー変数を、AgentMetaData オブジェクトがインスタンス化された後で明示的に初期化する必要があります。searchableNodes は、初期化しない場合、値が false になります。

toXml()

ODA メタデータを XML 形式に変換します。

構文

```
public String toXml();
```

パラメーター

なし。

戻り値

現在の AgentMetaData オブジェクトを XML 形式にしたものを内容とするストリング

第 9 章 AgentProperty クラス

Object Discovery Agent Development Kit (ODK) API には、エージェント・プロパティー・オブジェクト を表現することを目的とした AgentProperty クラスが用意されています。各エージェント・プロパティー・オブジェクトには、Object Discovery Agent (ODA) が必要とする以下のようなプロパティーに関する情報が格納されています。

- *構成プロパティー*。ODA が初期化の際に必要とする値を提供します。
- *ビジネス・オブジェクト・プロパティー*。ODA がビジネス・オブジェクト定義の生成の際に必要とする追加情報を提供します。

AgentProperty クラスには、次のものが定義されています。

- 『プロパティー・タイプ定数』
- 『メンバー変数』
- 208 ページの『メソッド』

プロパティー・タイプ定数

AgentProperty クラスには、プロパティー・タイプ定数を表す静的メンバー変数が定義されています。表 67 に、これらのプロパティー・タイプ定数について要約します。これらは、エージェント・プロパティーのデータ型として有効な値を表しています。プロパティー・タイプ定数は、いずれも、整数型 (int) です。

表 67. AgentProperty クラスのプロパティー・タイプ定数

プロパティー・タイプ定数	説明
TYPE_BOOLEAN	プロパティーのタイプが Boolean であることを示します。
TYPE_DOUBLE	プロパティーのタイプが Double であることを示します。
TYPE_FLOAT	プロパティーのタイプが Float であることを示します。
TYPE_INTEGER	プロパティーのタイプが Integer であることを示します。
TYPE_STRING	プロパティーのタイプが String であることを示します。

メンバー変数

表 68 に、AgentProperty クラスのメンバー変数について要約します。

表 68. AgentProperty クラスのメンバー変数

メンバー変数	説明	ページ
allDefaultValues	エージェント・プロパティーに表示されるデフォルト値を指定します。	202

表 68. *AgentProperty* クラスのメンバー変数 (続き)

メンバー変数	説明	ページ
allDependencies	現在のエージェント・プロパティーとその他の従属プロパティーの間の依存関係を規定する条件を指定します。	203
allValidValues	エージェント・プロパティーの有効値として表示する値を指定します。	203
allValues	エージェント・プロパティーとしてユーザーが選択した値を格納します。	203
cardinality	エージェント・プロパティーに単一値または複数値の保持を許可するかどうかを指定します。	204
description	エージェント・プロパティーのテキスト形式の説明文を入力します。また、その他の関連情報の保持も可能です。	205
isHidden	エージェント・プロパティーの値を暗号化して表示する必要があるかどうかを指定します。	205
isMultiple	ユーザーが複数の値をエージェント・プロパティーの値として入力できる手段をビジネス・オブジェクト・ウィザードに用意するかどうかを決定します。	205
isReadOnly	ユーザーがエージェント・プロパティーの値を指定できるか、またはプロパティーの値の表示のみ行うことができるかを決定します。	206
isRequired	エージェント・プロパティーに必ず値が指定されている必要があるかどうかを決定します。	207
propName	エージェント・プロパティーの名前を指定します。	207
type	エージェント・プロパティーのデータ型を指定します。	208

allDefaultValues

エージェント・プロパティーに表示されるデフォルト値を指定します。

タイプ

```
public java.lang.Object[] allDefaultValues
```

注記

allDefaultValues メンバー変数には、エージェント・プロパティーのデフォルト値の配列が格納されます。この配列の Object 要素の数は、次のように、そのプロパティーのカーディナリティー数に一致していなければなりません。

- 単一カーディナリティーのプロパティー (ODKConstant.SINGLE_CARD) の場合、allDefaultValues 配列に含まれる要素は、*1 つだけ* でなければなりません。
- 複数カーディナリティーのプロパティー (ODKConstant.MULTI_CARD) の場合、allDefaultValues 配列には、*1 つ以上* の要素を含めることができます。

詳しくは、 168 ページの『デフォルト値の指定』を参照してください。

allDependencies

現在のエージェント・プロパティーとその他の従属プロパティーの間の依存関係を規定する条件のリストを指定します。

タイプ

```
public CompleteCondition[] allDependencies
```

注記

allDependencies メンバー変数には、CompleteCondition オブジェクトの配列である条件配列に含まれる条件のリストが格納されます。各 CompleteCondition オブジェクトには、エージェント・プロパティーの値に関する条件が 1 つ格納されています。各条件は、入力条件と従属条件で構成されています。詳しくは、169 ページの『プロパティー値に対する条件の設定』を参照してください。

allValidValues

エージェント・プロパティーの有効値として表示する値を指定します。

タイプ

```
public java.lang.Object[] allValidValues
```

注記

allValidValues メンバー変数には、エージェント・プロパティーのドロップダウン・リストの初期設定に使用される値のリストが格納されます。ユーザーは、このドロップダウン・リストから、1 つ (単一カーディナリティーの場合)、または複数 (複数カーディナリティーの場合) の値をプロパティーの値として選択することができます。

allValidValues に値のリストが指定されている場合、ビジネス・オブジェクト・ウィザードは、isMultiple メンバー変数が true であるエージェント・プロパティーのすべてにおいて、それらの値をドロップダウン・リストに表示します。isHidden が true で allValidValues が null の場合、ビジネス・オブジェクト・ウィザードは、ユーザーが値を指定するサブグリッドを表示します。

注: isMultiple メンバー変数が false の場合、allValidValues メンバー変数は null でなければなりません。

詳しくは、166 ページの『表示コントロールのタイプの選択』を参照してください。

allValues

エージェント・プロパティーの値としてユーザーが指定した値が格納されます。

タイプ

```
public java.lang.Object[] allValues
```

注記

allValues メンバー変数は出力変数です。つまり、ユーザーの入力が完了した後、ビジネス・オブジェクト・ウィザードによって値が格納されます。この変数には、

ビジネス・オブジェクト・ウィザードの「エージェントの構成」のステップでユーザーが「値」列から選択した値が格納されます。この変数は、エージェント・プロパティーがユーザーに対して表示される前に初期化しておく必要がない *唯一の* メンバー変数です。

allValues 配列内の値の数は、エージェント・プロパティーのカーディナリティーによって決まります。

- エージェント・プロパティーが単一カーディナリティーを持つ場合 (エージェント・プロパティーの cardinality 変数が ODKConstant.SINGLE_CARD である場合)、allValues 配列に格納される値は 1 つだけです。
- エージェント・プロパティーが複数カーディナリティーを持つ場合 (エージェント・プロパティーの cardinality 変数が ODKConstant.MULTI_CARD である場合)、allValues 配列には、複数の値が格納されます (ユーザー指定値のそれぞれにつき 1 つずつ値が格納されます)。

cardinality

エージェント・プロパティーに単一値または複数値の保持を許可するかどうかを指定します。

タイプ

```
public java.lang.String cardinality
```

注記

cardinality メンバー変数は、エージェント・プロパティーの値が 1 つの値で構成されるか複数の値で構成されるかをを決定します。したがって、この変数は、ユーザーが特定のエージェント・プロパティーの値として指定できる値の数を決定します。

カーディナリティー	ユーザーが指定できるエージェント・プロパティー値の数	cardinality メンバー変数の値
単一	1 つ	ODKConstant.SINGLE_CARD
複数	複数	ODKConstant.MULTIPLE_CARD

ビジネス・オブジェクト・ウィザードに表示されるプロパティー用のコントロールのタイプは、そのプロパティーのカーディナリティーによって異なります。詳しくは、166 ページの『表示コントロールのタイプの選択』を参照してください。

次の例では、AgentProperty() コンストラクターの 3 番目の形式の構文を呼び出し、6 番目の引き数にカーディナリティーを表すストリング値を指定して、エージェント・プロパティーのカーディナリティーを初期化しています。

```
AgentProperty agt = new AgentProperty("Username",
        AgentProperty.TYPE_STRING,
        "User Id for logging into the database", true, false,
        ODKConstant.SINGLE_CARD, null, null);
```

注: エージェント・プロパティーのカーディナリティーを決定する値は、AgentProperty() コンストラクターの 2 番目の形式の構文でも、8 番目の引き数を使用して指定することができます。

description

エージェント・プロパティーのテキスト形式の説明文を入力します。また、その他の関連情報の保持も可能です。

タイプ

```
public java.lang.String description;
```

注記

description メンバー変数に格納されている内容は、ビジネス・オブジェクト・ウィザードの「エージェントの構成」のステップで、「説明」列に表示されます。次の例では、AgentProperty() コンストラクターの 3 番目の形式の構文を呼び出し、3 番目の引き数に説明を表すストリング値を指定して、エージェント・プロパティーの説明を初期化しています。

```
AgentProperty agt = new AgentProperty("Username",
         AgentProperty.TYPE_STRING,
         "User Id for logging into the database", true, false,
         ODKConstant.SINGLE_CARD, null, null);
```

注: エージェント・プロパティーの説明となる値は、AgentProperty() コンストラクターの 2 番目の形式の構文でも、6 番目の引き数を使用して指定することができます。

isHidden

エージェント・プロパティーの値を暗号化して表示する必要があるかどうかを指定します。

タイプ

```
public boolean isHidden;
```

注記

isHidden メンバー変数は、ビジネス・オブジェクト・ウィザードにエージェント・プロパティーの値をそのまま表示するかどうかを決定する Boolean 値です。isHidden が true の場合、エージェント・プロパティーの値は、表示されるときに暗号化されます。つまり、値はアスタリスク文字 (*) のストリングとして表示されます。エージェント・プロパティーの値を暗号化するかどうかを指定するには、次のように、AgentProperty() コンストラクターの 2 番目の形式の構文の、4 番目の引き数に Boolean 値を指定します。

```
AgentProperty agt = new AgentProperty("Username",
         AgentProperty.TYPE_STRING, true, false, true,
         "User Id for logging into the database", true,
         ODKConstant.SINGLE_CARD, null, null);
```

isMultiple

複数の値をエージェント・プロパティーの値として指定できる手段をビジネス・オブジェクト・ウィザードに用意するかどうかを決定します。

タイプ

```
public boolean isMultiple;
```

注記

isMultiple メンバー変数は、ユーザーが複数の値をエージェント・プロパティーの値として入力できる手段をビジネス・オブジェクト・ウィザードに用意する必要があるかどうかを決定する Boolean 値です。

- isMultiple が true の場合、ビジネス・オブジェクト・ウィザードには、allValidValues メンバー変数に格納されている値のリストを示すドロップダウン・リストが表示されます。ユーザーは、このリストで、エージェント・プロパティーに割り当てる値をクリックします。ユーザーがドロップダウン・リストから選択できる値の数は、cardinality メンバー変数の値によって決まります。allValidValues 配列が用意されていない場合、ビジネス・オブジェクト・ウィザードは、ユーザーが値を入力する複数の行で構成されたサブグリッドを表示します。
- isMultiple が false の場合、ビジネス・オブジェクト・ウィザードは、ユーザーに対して複数値の入力を許可しません。ただし、その代わりに、空のフィールドまたはデフォルト値 (指定されている場合) が表示されます。ユーザーは、このフィールドに、エージェント・プロパティー値を入力します。cardinality メンバー変数は、ODKConstant.SINGLE_CARD でなければなりません。

注: 詳しくは、 166 ページの『表示コントロールのタイプの選択』を参照してください。

次の例では、AgentProperty() コンストラクターの 3 番目の形式の構文を呼び出し、5 番目の引き数 (isMultiple 変数の値) に Boolean 値 true を指定することにより、エージェント・プロパティーを複数の値から成るリストで初期化して、ユーザーにその中から値を選択させることができるようにしています。

```
AgentProperty agt = new AgentProperty("Username",
         AgentProperty.TYPE_STRING,
         "User Id for logging into the database", true, true,
         ODKConstant.SINGLE_CARD, null, null);
```

注: isMultiple の値は、AgentProperty() コンストラクターの 2 番目の形式の構文でも、7 番目の引き数を使用して指定することができます。

isReadOnly

ユーザーがエージェント・プロパティーの値を指定できるか、または値の表示のみ行うことができるかを決定します。

タイプ

```
public boolean isReadOnly;
```

注記

isReadOnly メンバー変数は、ビジネス・オブジェクト・ウィザードにエージェント・プロパティーが表示されるときに、ユーザーがそのプロパティーの値を変更できるかどうかを決定する Boolean 値です。エージェント・プロパティーの値を変更

可能にするかどうかを指定するには、次のように、AgentProperty() コンストラクターの 2 番目の形式の構文の、5 番目の引き数に Boolean 値を指定します。

```
AgentProperty agt = new AgentProperty("Username",
         AgentProperty.TYPE_STRING, true, false, true,
         "User Id for logging into the database", true,
         ODKConstant.SINGLE_CARD, null, null);
```

isRequired

エージェント・プロパティーに値が必要であるかどうかを決定します。

タイプ

```
public boolean isRequired;
```

注記

isRequired メンバー変数は、エージェント・プロパティーに必ず値を指定する必要があるか、あるいはユーザーがそのプロパティーの値を空のままにできるかを決定する Boolean 値です。isRequired が true の場合、ユーザーはそのプロパティーの値を指定しなければなりません。次の例では、AgentProperty() コンストラクターの 3 番目の形式の構文を呼び出し、4 番目の引き数に Boolean 値 true を指定することにより、エージェント・プロパティーに値が必要であることを示しています。

```
AgentProperty agt = new AgentProperty("Username",
         AgentProperty.TYPE_STRING,
         "User Id for logging into the database", true, false,
         ODKConstant.SINGLE_CARD, null, null);
```

注: isRequired の値は、AgentProperty() コンストラクターの 2 番目の形式の構文でも、3 番目の引き数を使用して指定することができます。

propName

エージェント・プロパティーの名前を指定します。

タイプ

```
public java.lang.String propName;
```

注記

propName メンバー変数には、エージェント・プロパティーの名前 (Username、Password、DatabaseUrl など) を含むストリングが格納されます。propName メンバー変数の値は、ビジネス・オブジェクト・ウィザードの「エージェントの構成」のステップで、「プロパティー」列に表示されます。次の例では、AgentProperty() コンストラクターの 3 番目の形式の構文を呼び出し、最初の引き数に名前を指定して、エージェント・プロパティーの名前を初期化しています。

```
AgentProperty agt = new AgentProperty("Username",
         AgentProperty.TYPE_STRING,
         "User Id for logging into the database", true, false,
         ODKConstant.SINGLE_CARD, null, null);
```

注: AgentProperty() コンストラクターのどの構文形式でも、プロパティー名を指定して propName メンバー変数を初期化する必要があります。

type

エージェント・プロパティーのタイプを指定します。

タイプ

```
public int type;
```

注記

type メンバー変数には、エージェント・プロパティーのデータ型を表す整数値が格納されます。 201 ページの表 67 に、有効なプロパティー・タイプを表すために使用されるプロパティー・タイプ定数を示します。type メンバー変数の値のストリング表現は、ビジネス・オブジェクト・ウィザードの「エージェントの構成」のステップで、「タイプ」列に表示されます。エージェント・プロパティーのデータ型を初期化するには、次のように、AgentProperty() コンストラクターの 2 番目の引き数にプロパティー・タイプ定数を指定します。

```
AgentProperty agt = new AgentProperty("Username",
        AgentProperty.TYPE_STRING,
        "User Id for logging into the database", true, false,
        ODKConstant.SINGLE_CARD, null, null);
```

メソッド

表 69 に、AgentProperty クラスのメソッドについて要約します。

表 69. AgentProperty クラスのメンバー・メソッド

メンバー・メソッド	説明	ページ
AgentProperty()	エージェント・プロパティー・オブジェクトを作成します。	208
copy()	指定されたプロパティーを現在の AgentProperty オブジェクト内にコピーします。	210

AgentProperty()

エージェント・プロパティー・オブジェクトを作成します。

構文

```
public AgentProperty(String name);

public AgentProperty(String name, int type, boolean isReqd, boolean isHid,
   boolean isRdOnly, String desc, boolean isMult, String cardinality,
   Object[] validValues, Object[] defaultValues);

public AgentProperty(String name, int type, String desc, boolean isReqd,
   boolean isMult, String cardinality, Object[] validValues,
   Object[] defaultValues);
```

パラメーター

cardinality プロパティーに複数の値を保持できるようにするかどうかを指定します。エージェント・プロパティー・オブジェクトの cardinality メンバー変数は、このパラメーターの値で初期化されます (204 ページの『cardinality』)。

defaultValues プロパティーのデフォルト値を指定します。エージェント・プロパティー・オブジェクトの `allDefaultValues` メンバー変数は、このパラメーターの値で初期化されます (202 ページの『allDefaultValues』)。

desc プロパティーの説明を指定します。エージェント・プロパティー・オブジェクトの `description` メンバー変数は、このパラメーターの値で初期化されます (205 ページの『description』)。

isHid プロパティーの値を暗号化する必要があるかどうかを指定します。エージェント・プロパティー・オブジェクトの `isHidden` メンバー変数は、このパラメーターの値で初期化されます (205 ページの『isHidden』)。

isMult プロパティーに複数の値を用意して、ユーザーがその中から値を選択できるようにするかどうかを指定します。エージェント・プロパティー・オブジェクトの `isMultiple` メンバー変数は、このパラメーターの値で初期化されます (205 ページの『isMultiple』)。

isRdOnly ユーザーにプロパティー値の入力を許可するか、または表示のみ許可するかを指定します。エージェント・プロパティー・オブジェクトの `isReadOnly` メンバー変数は、このパラメーターの値で初期化されます (206 ページの『isReadOnly』)。

isReqd プロパティーに値が必要であるかどうかを指定します。エージェント・プロパティー・オブジェクトの `isRequired` メンバー変数は、このパラメーターの値で初期化されます (207 ページの『isRequired』)。

name プロパティーの名前を指定します。エージェント・プロパティー・オブジェクトの `propName` メンバー変数は、このパラメーターの値で初期化されます (207 ページの『propName』)。

type プロパティーのタイプを指定します。エージェント・プロパティー・オブジェクトの `type` メンバー変数は、このパラメーターの値で初期化されます (208 ページの『type』)。

validValues プロパティーの有効値を指定します。エージェント・プロパティー・オブジェクトの `allValidValues` メンバー変数は、このパラメーターの値で初期化されます (203 ページの『allValidValues』)。

戻り値

新規にインスタンス化された `AgentProperty` オブジェクト。

例外

`IllegalArgumentException`

name パラメーターの値が `null` の場合、または *type* パラメーターが有効なプロパティー・タイプ定数 (201 ページの表 67 を参照) ではない場合にスローされます。

注記

`AgentProperty()` メソッドの新規エージェント・プロパティー・オブジェクト生成構文について、以下に説明します。

- 最初の構文形式は、新しいエージェント・プロパティー・オブジェクトを定義し、そのオブジェクトのプロパティー名に限り 初期化します。このエージェント・プロパティーのタイプは、String になります。このプロパティーは、ユーザーに対して複数の値を表示しない、単一カーディナリティーのプロパティーになります。
- 2 番目の構文形式は、新しいエージェント・プロパティー・オブジェクトを定義し、そのオブジェクトのすべての メンバー変数を初期化します。プロパティーのメンバー変数に適切な値を指定することにより、プロパティーのメタデータをカスタマイズすることができます。
- 3 番目の構文形式は、新しいエージェント・プロパティー・オブジェクトを定義し、そのオブジェクトのメンバー変数を、isHidden と isReadOnly を除いて すべて初期化します。この場合、isHidden 変数と isReadOnly 変数は false になります。

copy()

指定されたプロパティーを現在の AgentProperty オブジェクト内にコピーします。

構文

```
public void copy(AgentProperty prop);
```

パラメーター

prop コピーするプロパティーの名前を指定します。

第 10 章 BusObjAttr クラス

Object Discovery Agent Development Kit (ODK) API には、ビジネス・オブジェクト定義内の属性を表現することを目的とした BusObjAttr クラスが用意されています。 BusObjAttr インスタンスは、*属性オブジェクト* を表しています。このクラスには、次のものが定義されています。

- 『属性定数』
- 『メソッド』

注: ObjectEventId 属性に対応する属性オブジェクトは、ビジネス・オブジェクト定義 (BusObjDef オブジェクト) に自動的に定義されます。この属性は、この属性の特殊な目的を示す BusObjAttr.OBJECT_EVENT_ID 定数で自動的にマークされます。

属性定数

BusObjAttr クラスには、属性定数を表す静的メンバー変数が定義されています。表 70 に、属性定数について要約します。属性定数は、いずれも、整数型 (int) です。

表 70. BusObjAttr クラスの属性定数

属性定数	説明
カーディナリティー定数	
CARD_MULTIPLE	属性が子ビジネス・オブジェクトの配列を表すこと、すなわち属性のカーディナリティーが複数であることを示します。
CARD_SINGLE	属性が 1 つの値または 1 つの子ビジネス・オブジェクトを表すこと、すなわち属性のカーディナリティーが単一であることを示します。
ObjectEventId 定数	
OBJECT_EVENT_ID	属性が ObjectEventId であることを示します。

メソッド

表 71 に、BusObjAttr クラスのメンバー・メソッドについて要約します。

表 71. BusObjAttr クラスのメンバー・メソッド

メンバー・メソッド	説明	ページ
BusObjAttr()	ビジネス・オブジェクト属性のオブジェクトを作成します。	213
getAppText()	属性のアプリケーション固有情報を取得します。	214
getAttrType()	単純属性のタイプを取得します。	214

表 71. *BusObjAttr* クラスのメンバー・メソッド *(続き)*

メンバー・メソッド	説明	ページ
`getAttrTypeName()`	属性が子ビジネス・オブジェクトまたは子ビジネス・オブジェクト配列を表す場合、子ビジネス・オブジェクトのタイプを属性タイプとして取得します。	215
`getBOVersion()`	属性が子ビジネス・オブジェクトまたは子ビジネス・オブジェクト配列を表す場合、ビジネス・オブジェクト定義のバージョン番号を取得します。	215
`getCardinality()`	属性が子ビジネス・オブジェクトまたは子ビジネス・オブジェクト配列を表す場合、属性のカーディナリティーを取得します。	216
`getComments()`	属性に関連付けられているコメントを取得します。	216
`getDefault()`	属性のデフォルト値を取得します。	217
`getMaxLength()`	この属性の最大長を取得します。	217
`getName()`	属性の名前を取得します。	217
`getRelationType()`	属性の関係タイプ (属性が子ビジネス・オブジェクトまたは子ビジネス・オブジェクト配列を表す場合は containment) を取得します。	218
`isForeignKey()`	この属性がビジネス・オブジェクトの外部キーの一部となっているかどうかを判別します。	218
`isKey()`	この属性がビジネス・オブジェクトのキーの一部となっているかどうかを判別します。	218
`isRequiredKey()`	この属性がビジネス・オブジェクトの必須キーの一部となっているかどうかを判別します。	219
`isRequiredServerBound()`	ビジネス・オブジェクトがトリガー・イベントを表すときに属性を必要とするかどうかを判別します。	219
`isSimpleType()`	属性が単純タイプ (例えば、String、Integer、Float など) かどうかを判別します。あるいは、属性が子ビジネス・オブジェクトまたは子ビジネス・オブジェクトの配列を表すかどうかを判別します。	219
`setAppText()`	属性のアプリケーション固有情報を設定します。	220
`setAttrType()`	属性のタイプを設定します。	220
`setBOVersion()`	属性が子ビジネス・オブジェクトまたは子ビジネス・オブジェクト配列を表す場合、属性で表される子ビジネス・オブジェクトまたはオブジェクトのバージョンを設定します。	221
`setCardinality()`	属性が子ビジネス・オブジェクトまたは子ビジネス・オブジェクト配列を表す場合、属性のカーディナリティーを設定します。	221
`setComments()`	属性に関連付けられているコメントを設定します。	222
`setDefault()`	属性のデフォルト値を設定します。	222
`setIsForeignKey()`	属性を、その属性が外部キーの一部となっているかどうかを示す Boolean 値に設定します。	223
`setIsKey()`	属性を、その属性がキーの一部となっているかどうかを示す Boolean 値に設定します。	223

表 71. *BusObjAttr* クラスのメンバー・メソッド (続き)

メンバー・メソッド	説明	ページ
setIsRequiredKey()	属性を、その属性が必須キーの一部となっているかどうかを示す Boolean 値に設定します。	223
setMaxLength()	属性の最大長を設定します。	224
setName()	属性の名前を設定します。	224
setRelationType()	属性が子ビジネス・オブジェクトまたは子ビジネス・オブジェクト配列を表す場合、属性の関係タイプを containment に設定します。	225

BusObjAttr()

ビジネス・オブジェクト属性のオブジェクトを新規に作成します。

構文

```
public BusObjAttr(String name, int type);

public BusObjAttr(String name, int type, String typeName);

public BusObjAttr(String name, int type, String typeName, boolean isKey,
   boolean isForeignKey, boolean isReqd, String appSpecInfo, int maxLen
   String defaultValue, String BOversion, String cardinality, String relType,
   boolean isReqdServerBound, String comments);
```

パラメーター

appSpecInfo 属性のアプリケーション固有情報を指定します。

BOversion 属性が子ビジネス・オブジェクトまたは子ビジネス・オブジェクト配列を表す場合、子ビジネス・オブジェクトまたはオブジェクトのバージョンを指定します。

cardinality 属性が子ビジネス・オブジェクトまたは子ビジネス・オブジェクト配列を表す場合、属性のカーディナリティーを指定します。

comments 属性に関連付けるコメント (オプション) を指定します。

defaultValue 属性のデフォルト値を指定します。

isForeignKey 属性がビジネス・オブジェクトの外部キーの一部であるかどうかを指定します。

isKey 属性がビジネス・オブジェクトのキーの一部であるかどうかを指定します。

isReqd 属性に値が必要であるかどうかを指定します。

isReqdServerBound
ビジネス・オブジェクトがトリガー・イベントを表すときに、属性に値が必要であるかどうかを指定します。

maxLen 属性の値の最大長を指定します。

name 属性の名前を指定します。

relType 属性が子ビジネス・オブジェクトまたは子ビジネス・オブジェクト配列を表す場合、関係タイプとして containment を指定します。

type 属性のタイプを指定します。

typeName 属性が子ビジネス・オブジェクトまたは子ビジネス・オブジェクト配列を表す場合、子ビジネス・オブジェクトのタイプを属性タイプとして指定します。

戻り値

新規にインスタンス化された `BusObjAttr` オブジェクト。

getAppText()

属性のアプリケーション固有情報を取得します。

構文

```
public String getAppText();
```

パラメーター

なし。

戻り値

属性のアプリケーション固有情報が格納されている `String`。

参照

setAppText()

getAttrType()

属性のタイプを取得します。

構文

```
public int getAttrType();
```

パラメーター

なし。

戻り値

属性のタイプを表す整数。この整数値を属性タイプ定数の 1 つと比較します。

`BusObjAttrType.BOOLEAN`
属性のデータ型は Boolean です。

`BusObjAttrType.CIPHERTEXT`
属性のデータ型は Cipher Text です。

`BusObjAttrType.DATE`
属性のデータ型は Date です。

`BusObjAttrType.DOUBLE`
属性のデータ型は Double です。

`BusObjAttrType.FLOAT`
属性のデータ型は Float です。

BusObjAttrType.INTEGER
属性のデータ型は Integer です。

BusObjAttrType.INVALID_TYPE
属性のデータ型は無効です。

BusObjAttrType.LONGTEXT
属性のデータ型は Long Text です。

BusObjAttrType.OBJECT
属性のデータ型は Object です (別のビジネス・オブジェクトが格納されています)。

BusObjAttrType.STRING
属性のデータ型は String です。

参照

getAttrTypeName(), setAttrType()

getAttrTypeName()

属性のデータ型の名前を取得します。

構文

```
public String getAttrTypeName();
```

パラメーター

なし。

戻り値

子ビジネス・オブジェクト (属性に子ビジネス・オブジェクトが含まれている場合) のタイプであるビジネス・オブジェクト定義の名前が格納されている String。

注記

getAttrTypeName() メソッドは、子ビジネス・オブジェクトの属性タイプの名前を取得します。属性が子ビジネス・オブジェクト (または子ビジネス・オブジェクトの配列) を表している場合、その属性の属性タイプは BusObjAttrType.OBJECT であり、属性タイプの名前は子ビジネス・オブジェクトのビジネス・オブジェクト定義の名前です。

参照

getAttrType(), setAttrType()

getBOVersion()

属性が子ビジネス・オブジェクトまたは子ビジネス・オブジェクト配列を表す場合、ビジネス・オブジェクト定義のバージョン番号を取得します。

構文

```
public String getBOVersion();
```

パラメーター

なし。

戻り値

属性によって表される子ビジネス・オブジェクト定義のバージョン番号が格納されている `String`。

参照

setBOVersion()

getCardinality()

属性が子ビジネス・オブジェクトまたは子ビジネス・オブジェクト配列を表す場合、属性のカーディナリティーを取得します。

構文

```
public String getCardinality();
```

パラメーター

なし。

戻り値

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表す属性のカーディナリティーが格納されている `String`。このストリング値を以下のカーディナリティー定数と比較します。

`BusObjAttr.CARD_SINGLE`
属性は単一カーディナリティーを持ちます。

`BusObjAttr.CARD_MULTIPLE`
属性は複数カーディナリティーを持ちます。

参照

setCardinality()

getComments()

属性に関連付けられているコメントを取得します。

構文

```
public String getComments();
```

パラメーター

なし。

戻り値

属性のコメントが格納されている `String`。

getDefault()

属性のデフォルト値を取得します。

構文

```
public String getDefault();
```

パラメーター

なし。

戻り値

属性のデフォルト値が格納されている `String`。

参照

setDefault()

getMaxLength()

この属性の最大長を取得します。

構文

```
public int getMaxLength();
```

パラメーター

なし。

戻り値

属性値の最大長を表す整数。

参照

setMaxLength()

getName()

属性の名前を取得します。

構文

```
public String getName();
```

パラメーター

なし。

戻り値

属性の名前が格納されている `String`。

参照

setName()

getRelationType()

属性の関係タイプ (属性が子ビジネス・オブジェクトまたは子ビジネス・オブジェクト配列を表す場合は containment) を取得します。

構文

```
public String getRelationType();
```

パラメーター

なし。

戻り値

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表す属性の関係タイプ (「containment」) が格納されている String。

参照

setRelationType()

isForeignKey()

この属性がビジネス・オブジェクトの外部キーの一部となっているかどうかを判別します。

構文

```
public boolean isForeignKey();
```

パラメーター

なし。

戻り値

属性が外部キー、または外部キーの一部となっている場合は true を戻します。それ以外の場合は false を戻します。

参照

setIsForeignKey()

isKey()

この属性がビジネス・オブジェクトの基本キーの一部となっているかどうかを判別します。

構文

```
public boolean isKey();
```

パラメーター

なし。

戻り値

属性がキー、またはそのキーの一部となっている場合は `true` を戻します。それ以外の場合は `false` を戻します。

参照

setIsKey()

isRequiredKey()

この属性がビジネス・オブジェクトの必須キーの一部となっているかどうかを判別します。

構文

```
public boolean isRequiredKey();
```

パラメーター

なし。

戻り値

属性が必須キー、または必須キーの一部となっている場合は `true` を戻します。それ以外の場合は `false` を戻します。

参照

setIsRequiredKey()

isRequiredServerBound()

ビジネス・オブジェクトがトリガー・イベントを表すときに属性を必要とするかどうかを判別します。

構文

```
public boolean isRequiredServerBound();
```

パラメーター

なし。

戻り値

ビジネス・オブジェクトがコラボレーション・オブジェクト要求を表す場合に、属性が必要であれば `true`、それ以外の場合は `false` を戻します。

isSimpleType()

属性が単純タイプ (例えば、String、Integer、Float など) かどうかを判別します。あるいは、属性が子ビジネス・オブジェクトまたは子ビジネス・オブジェクトの配列を表すかどうかを判別します。

構文

```
public boolean isSimpleType();
```

パラメーター

なし。

戻り値

属性が単純タイプである場合は `true`、それ以外の場合は `false`。

参照

getAttrType(), setAttrType()

setAppText()

属性のアプリケーション固有情報を設定します。

構文

```
public void setAppText(String appInfo);
```

パラメーター

appInfo 属性に割り当てられるアプリケーション固有の情報です。

戻り値

なし。

参照

getAppText()

setAttrType()

属性のタイプを設定します。

構文

```
public void setAttrType(int type);
public void setAttrType(int type, String typeName);
```

パラメーター

type 次のいずれかの属性タイプ定数として表される、属性のタイプです。

```
BusObjAttrType.BOOLEAN
BusObjAttrType.CIPHERTEXT
BusObjAttrType.DATE
BusObjAttrType.DOUBLE
BusObjAttrType.FLOAT
BusObjAttrType.INTEGER
BusObjAttrType.LONGTEXT
BusObjAttrType.OBJECT
BusObjAttrType.STRING
```

typeName 子ビジネス・オブジェクト、または子ビジネス・オブジェクトの配列を表す属性に対するビジネス・オブジェクトの名前です。この場合、属性のタイプは、子ビジネス・オブジェクトのタイプと同じで、`type` の値は `OBJECT` です。

戻り値

なし。

例外

BusObjInvalidAttrException

type が無効な場合、つまり、属性タイプ定数で表される値の 1 つに該当しない場合、スローされます。

注記

`setAttrType()` メソッドには、以下の構文形式があります。

- 最初の構文形式では、`BusObjAttrType` クラスに定義されているいずれかの属性タイプ定数を指定することにより、単純属性の属性タイプを設定できます。
- 2 番目の構文形式では、子ビジネス・オブジェクトまたは子ビジネス・オブジェクト配列の属性タイプを設定できます。この構文形式を使用すると、属性タイプを (属性タイプ定数 `BusObjAttrType.OBJECT` に) 設定し、子ビジネス・オブジェクトのビジネス・オブジェクト定義名を指定することができます。

参照

getAttrType(), getAttrTypeName()

関連する参照情報については、 227 ページの『第 11 章 BusObjAttrType インターフェース』および 289 ページの『第 23 章 ODKException クラス』を参照してください。

setBOVersion()

属性が子ビジネス・オブジェクトまたは子ビジネス・オブジェクト配列を表す場合、ビジネス・オブジェクト定義のバージョン番号を設定します。

構文

```
public void setBOVersion(String version);
```

パラメーター

version この属性によって表された子ビジネス・オブジェクトまたはオブジェクトに対するビジネス・オブジェクト定義のバージョンです。

戻り値

なし。

参照

getBOVersion()

setCardinality()

属性が子ビジネス・オブジェクトまたは子ビジネス・オブジェクト配列を表す場合、属性のカーディナリティーを設定します。

構文

```
public void setCardinality(String cardinality);
```

パラメーター

cardinality この属性に割り当てられるカーディナリティーです。カーディナリティーは、以下のカーディナリティー定数の 1 つによって表されます。

```
BusObjAttr.CARD_SINGLE
BusObjAttr.CARD_MULTIPLE
```

戻り値

なし。

例外

BusObjInvalidAttrException

cardinality が有効な値でない場合、つまり、有効なカーディナリティー定数が指定されていない場合にスローされます。

参照

getCardinality()

setComments()

属性に関連付けられるコメントを設定します。

構文

```
public void setComments(String comment);
```

パラメーター

comment 属性に関する追加情報を提供するコメント・ストリングです。

戻り値

なし。

参照

getComments()

setDefault()

属性のデフォルト値を設定します。

構文

```
public void setDefault(String defaultValue);
```

パラメーター

defaultValue 属性に割り当てられるデフォルト値です。

戻り値

なし。

参照

getDefault()

setIsForeignKey()

属性が外部キーの一部となっているかどうかを示す属性プロパティーを設定します。

構文

```
public void setIsForeignKey(boolean fKey);
```

パラメーター

fKey この属性が外部キーの一部となっているかどうかを示します。

戻り値

なし。

参照

isForeignKey()

setIsKey()

属性が基本キーの一部となっているかどうかを示す属性プロパティーを設定します。

構文

```
public void setIsKey(boolean key);
```

パラメーター

key この属性がキーの一部となっているかどうかを示します。

戻り値

なし。

参照

isKey()

setIsRequiredKey()

属性を、その属性が必須キーの一部となっているかどうかを示す Boolean 値に設定します。

構文

```
public void setIsRequiredKey(boolean isReqd);
```

パラメーター

isReqd この属性が必須キーとなっているかどうかを示します。

戻り値

なし。

参照

isRequiredKey()

setMaxLength()

属性の最大長を設定します。

構文

```
public void setMaxLength(int maxLength);
```

パラメーター

maxLength 属性に割り当てられる最大長です。

戻り値

なし。

例外

BusObjInvalidAttrException

最大長が `maxLength < 0` または `maxLength > 2^31-1` に該当する場合、スローされます。

参照

getMaxLength()

setName()

属性の名前を設定します。

構文

```
public void setName(String name);
```

パラメーター

name 属性に割り当てられる名前です。

戻り値

なし。

参照

getName()

setRelationType()

属性が子ビジネス・オブジェクトまたは子ビジネス・オブジェクト配列を表す場合、属性の関係タイプを containment に設定します。

構文

```
public void setRelationType(String relType);
```

パラメーター

relType この属性に割り当てられる関係タイプです。

戻り値

なし。

参照

getRelationType()

第 11 章 BusObjAttrType インターフェース

Object Discovery Agent Development Kit (ODK) API には、ビジネス・オブジェクト定義内の属性の有効なデータ型を表現することを目的とした BusObjAttrType インターフェースが用意されています。 BusObjAttrType インターフェースが実装されているクラスは、いずれも、このインターフェースに定義されている定数に直接アクセスできます。例えば、ODKAgentBase2 クラスに BusObjAttrType インターフェースを実装すると、次のようにして、このクラスのメソッドから BOOLEAN 定数にアクセスできます。

```
int bool_type = BOOLEAN;
```

BusObjAttrType インターフェースには、次のものが定義されています。

- 『属性タイプ定数』
- 『静的メンバー変数』

属性タイプ定数

BusObjAttrType インターフェースには、属性タイプ定数を表す静的メンバー変数が定義されています。表 72 に、属性タイプ定数について要約します。属性タイプ定数は、いずれも、整数型 (int) です。

表 72. BusObjAttrType インターフェースの属性タイプ定数

属性タイプ定数	説明
BOOLEAN	属性タイプが Boolean であることを表します。
CIPHERTEXT	属性タイプが CipherText であることを表します。
DATE	属性タイプが Date であることを表します。
DOUBLE	属性タイプが Double であることを表します。
FLOAT	属性タイプが Float であることを表します。
INTEGER	属性タイプが Integer であることを表します。
INVALID_TYPE	無効な属性タイプであることを表します。
LONGTEXT	属性タイプが Long Text であることを表します。
OBJECT	属性タイプが Object であることを表します。
STRING	属性タイプが String であることを表します。

静的メンバー変数

BusObjAttrType インターフェースには、属性タイプ定数 (静的メンバー変数として定義されています) に加えて、表 73 に示す静的メンバー変数も定義されています。

表 73. BusObjAttrType クラスの静的メンバー変数

静的メンバー変数	説明
AttrTypes	異なる複数の属性タイプの名前が格納される String 配列。この配列は、属性タイプを指標に指定することができます。例えば、次のコードは、Integer 属性タイプのタイプ名を取得します。 `BusObjAttrType.AttrTypes[BusObjAttrType.INTEGER]`

第 12 章 BusObjDef クラス

Object Discovery Agent Development Kit (ODK) API には、Object Discovery Agent (ODA) が生成するビジネス・オブジェクト定義を表現することを目的とした BusObjDef クラスが用意されています。表 74 に、BusObjDef クラスのメソッドについて要約します。

表 74. BusObjDef クラスのメンバー・メソッド

メンバー・メソッド	説明	ページ
BusObjDef()	ビジネス・オブジェクト定義のオブジェクトを作成します。	230
addDefaultVerbs()	サポートされている動詞のリストにデフォルトの動詞 (Create、Retrieve、Update、および Delete) を追加します。	230
getAppInfo()	ビジネス・オブジェクト定義に関するアプリケーション固有の情報を取得します。	231
getAttrCount()	ビジネス・オブジェクト定義の属性リスト内にある属性 (ObjectEventID を含む) の数を取得します。	231
getAttribute()	ビジネス・オブジェクト定義内にある該当する名前の属性、または指定された位置にある属性を取得します。	231
getAttributeIndex()	その属性名を与えて、ビジネス・オブジェクト定義内にある属性の位置序数を取得します。	232
getAttributeList()	ビジネス・オブジェクト定義内の属性リストが格納されているベクトルを取得します。	233
getName()	ビジネス・オブジェクト定義の名前を取得します。	233
getVerb()	指定された動詞名に対応する動詞オブジェクトを取得します。	233
getVerbCount()	動詞リスト内の動詞の数を取得します。	234
getVerbList()	ビジネス・オブジェクト定義内の動詞リストが格納されているベクトルを取得します。	234
getVersion()	ビジネス・オブジェクト定義のバージョンを取得します。	235
insertAttribute()	ビジネス・オブジェクトの属性リストに、指定された属性を挿入します。	235
insertVerb()	指定された動詞をビジネス・オブジェクトの動詞リストに挿入します。	236
removeAttribute()	属性リスト内にある指定された位置の属性を削除します。	237
removeVerb()	動詞リスト内にある指定された名前の動詞を削除します。	238
setAppInfo()	ビジネス・オブジェクト定義に関するアプリケーション固有の情報を設定します。	238
setAttributeList()	ビジネス・オブジェクト定義用の属性リストを設定します。	239
setVerbList()	ビジネス・オブジェクト定義用の動詞リストを設定します。	239

BusObjDef()

ビジネス・オブジェクト定義のオブジェクトを作成します。

構文

```
public BusObjDef(String name);

public BusObjDef(String name, Vector attrList, String[] verbNames,
   String appSpecInfo);

public BusObjDef(String name, Vector attrList, Vector verbList,
   String appSpecInfo);
```

パラメーター

appSpecInfo ビジネス・オブジェクト・レベルのアプリケーション固有の情報を指定します。

attrList ビジネス・オブジェクト定義の属性リストを格納する Java Vector を指定します。

name ビジネス・オブジェクト定義の名前を指定します。

verbList ビジネス・オブジェクトの動詞の Vector を指定します。

verbNames ビジネス・オブジェクトの動詞の名前が格納される String 配列を指定します。

戻り値

新規にインスタンス化された BusObjDef オブジェクト。

例外

BusObjInvalidDefException
定義

BusObjInvalidVerbException
定義

addDefaultVerbs()

ビジネス・オブジェクト定義の動詞リストにデフォルトの動詞 (Create、Retrieve、Update、および Delete) を追加します。

構文

```
public void addDefaultVerbs();
```

パラメーター

なし。

戻り値

なし。

getAppInfo()

ビジネス・オブジェクト定義に関するアプリケーション固有の情報を取得します。

構文

```
public String getAppInfo();
```

パラメーター

なし。

戻り値

ビジネス・オブジェクト・レベルのアプリケーション固有情報が格納されている String。

参照

setAppInfo()

getAttrCount()

ビジネス・オブジェクト定義の属性リスト内にある属性の数を取得します。

構文

```
public int getAttrCount();
```

パラメーター

なし。

戻り値

ビジネス・オブジェクト定義内にある属性 (ObjectEventId 属性を含む) の数。

getAttribute()

ビジネス・オブジェクト定義の属性リスト内にある該当する名前の属性、または指定された位置にある属性を取得します。

構文

```
public BusObjAttr getAttribute(String attrName);
public BusObjAttr getAttribute(int pos);
```

パラメーター

attrName ビジネス・オブジェクト定義の属性リストから取得する属性の名前。

pos ビジネス・オブジェクト定義の属性リスト内にある属性の位置序数を指定する整数。

戻り値

ビジネス・オブジェクト定義内にある指定の属性に対応する属性オブジェクト (BusObjAttr)。

例外

BusObjNoSuchAttrException
指定された属性が存在しない場合、または指定された属性リスト内の位置が無効な場合にスローされます。

注記

getAttribute() メソッドは、ビジネス・オブジェクト定義の属性リストから属性を取得します。このメソッドは、取得した属性を、属性オブジェクト (BusObjAttr) として戻します。この属性に関する情報を取得するには、BusObjAttr クラスのメソッドを使用します。

参照

getAttributeIndex(), getAttributeList()

---

getAttributeIndex()

その属性名を与えて、ビジネス・オブジェクト定義内にある属性の位置序数を取得します。

構文

```
public int getAttributeIndex(String attrName);
```

パラメーター

attrName 位置序数を取得する属性の名前。

戻り値

ビジネス・オブジェクト定義の属性リスト内にある属性の位置 (整数)。

例外

BusObjNoSuchAttrException
ビジネス・オブジェクト定義内に指定された属性が存在しない場合にスローされます。

参照

getAttribute()

getAttributeList()

ビジネス・オブジェクト定義内にある属性のリストを取得します。

構文

```
public Vector getAttributeList();
```

パラメーター

なし。

戻り値

ビジネス・オブジェクト定義内の属性ごとに属性オブジェクト (BusObjAttr) を 1 つずつ持つ java.util.Vector オブジェクト。

注記

getAttributeList() メソッドは、ビジネス・オブジェクト定義の属性リストを、属性オブジェクトが格納された Java Vector として戻します。この Vector オブジェクトから属性オブジェクトを取得するには、java.util.Vector クラスのメソッドを使用します。属性オブジェクトから情報を取得するには、BusObjAttr クラスのメソッドを使用します。

参照

setAttributeList()

getName()

ビジネス・オブジェクト定義の名前を取得します。

構文

```
public String getName();
```

パラメーター

なし。

戻り値

ビジネス・オブジェクト定義の名前が格納されている String。

getVerb()

指定された動詞をビジネス・オブジェクト定義の動詞リストから取得します。

構文

```
public BusObjVerb getVerb(String verb);
```

パラメーター

verb ビジネス・オブジェクト定義の動詞リストから取得する動詞の名前。

戻り値

ビジネス・オブジェクト定義の動詞リスト内にある指定された動詞に対応する動詞オブジェクト (BusObjVerb)。

例外

BusObjNoSuchVerbException
指定された動詞が存在しない場合にスローされます。

注記

getVerb() メソッドは、ビジネス・オブジェクト定義の動詞リストから動詞を取得します。このメソッドは、取得した動詞を動詞オブジェクト (BusObjVerb) として戻します。この動詞に関する情報を取得するには、BusObjVerb クラスのメソッドを使用します。

参照

getVerbCount(), getVerbList()

---

getVerbCount()

ビジネス・オブジェクト定義の動詞リスト内にある動詞の数を取得します。

構文

```
public int getVerbCount();
```

パラメーター

なし。

戻り値

ビジネス・オブジェクト定義の動詞リスト内にある動詞の数 (整数)。

参照

getVerb()

---

getVerbList()

ビジネス・オブジェクト定義内にある動詞のリストを取得します。

構文

```
public Vector getVerbList();
```

パラメーター

なし。

戻り値

ビジネス・オブジェクト定義内のサポートされる動詞ごとに動詞オブジェクト (BusObjVerb) を 1 つずつ持つ java.util.Vector オブジェクト。

注記

getVerbList() メソッドは、ビジネス・オブジェクト定義の動詞リストを、動詞オブジェクトが格納された Java Vector として戻します。この Vector オブジェクトから動詞オブジェクトを取得するには、java.util.Vector クラスのメソッドを使用します。動詞オブジェクトから情報を取得するには、BusObjVerb クラスのメソッドを使用します。

参照

setVerbList()

---

getVersion()

ビジネス・オブジェクト定義のバージョンを取得します。

構文

```
public String getVersion();
```

パラメーター

なし。

戻り値

ビジネス・オブジェクト定義のバージョンが格納されている String。

---

insertAttribute()

指定された属性をビジネス・オブジェクト定義の属性リストに挿入します。

構文

```
public void insertAttribute(BusObjAttr attrObj);
public void insertAttribute(BusObjAttr attrObj, int pos);
```

パラメーター

attrObj ビジネス・オブジェクト定義の属性リストに追加される属性オブジェクト。

pos 属性リストに追加される属性の位置を示す序数。

戻り値

なし。

例外

BusObjInvalidAttrException
属性オブジェクトが表す属性が無効である場合にスローされます。

注記

insertAttribute() メソッドには、以下の構文形式があります。

- 最初の構文形式では、属性名を使用して、追加する属性を指定します。この構文形式を使用した場合、指定した属性は、insertAttribute() によって、ビジネス・オブジェクトの属性リスト内の ObjectEventId 属性の*真上* の位置に挿入されます。
- 2 番目の構文形式では、追加する属性のほか、その属性の追加先となる属性リスト内の位置を示す序数を指定します。位置序数を指定すると、指定した属性が insertAttribute() によってビジネス・オブジェクト定義の属性リスト内 (*pos* により指定した位置) に挿入され、リスト内の後続の属性がすべて 1 つ下の位置に移動します。

重要: 位置序数を指定するときは、指定する位置が ObjectEventId 属性よりも*上*にあることを確認してください。

参照

removeAttribute()

---

insertVerb()

指定された動詞をビジネス・オブジェクト定義の動詞リストに挿入します。

構文

```
public void insertVerb(BusObjVerb verbObj);
public void insertVerb(String verbStrng, String appSpecInfo);
```

パラメーター

appSpecInfo 動詞リストに追加される動詞に関するアプリケーション固有の情報。

verbObj 動詞リストに追加する動詞オブジェクト。

verbStrng 動詞リストに追加する動詞の名前。

例外

BusObjInvalidVerbException
動詞オブジェクトが表す動詞が既存の動詞と重複する場合にスローされます。

注記

`insertVerb()` メソッドの構文形式について以下に説明します。ビジネス・オブジェクト定義の動詞リストへの動詞オブジェクトの挿入は、次のいずれかの方法で行うことができます。

- 最初の構文形式では、初期化済みの動詞オブジェクト (`BusObjVerb` インスタンス) を、追加する動詞として指定します。動詞オブジェクトを初期化するには、`BusObjVerb` クラスのメソッドを使用します。
- 2 番目の構文形式では、動詞に関する情報 (動詞の名前およびその動詞のアプリケーション固有の情報) を指定します。

参照

removeVerb()

---

removeAttribute()

指定された属性をビジネス・オブジェクト定義の属性リストから削除します。

構文

```
public BusObjAttr removeAttribute(int pos);
public BusObjAttr removeAttribute(String attrName);
```

パラメーター

attrName ビジネス・オブジェクト定義の属性リストから削除する属性の名前。

pos 削除される属性の位置を示す序数。

戻り値

削除される属性が格納されている属性オブジェクト (`BusObjAttr`)。

例外

BusObjNoSuchAttrException
指定された属性が存在しない場合にスローされます。

BusObjInvalidAttrException
削除対象の属性が削除不可の属性 (`ObjectEventId` 属性など) である場合にスローされます。

注記

`removeAttribute()` メソッドには、以下の構文形式があります。

- 最初の構文形式では、ビジネス・オブジェクト定義の属性リスト内での属性の位置を示す序数を使用して、削除する属性を指定します。
- 2 番目の構文形式では、属性名およびこの属性を追加する属性リストの中での属性の位置によって、削除する属性を指定します。

重要: 位置序数を指定するときは、指定する位置が ObjectEventId 属性の位置ではない ことを確認してください。

参照

insertAttribute()

removeVerb()

指定された動詞をビジネス・オブジェクト定義の動詞リストから削除します。

構文

```
public BusObjVerb removeVerb(String verb);
```

パラメーター

verb ビジネス・オブジェクト定義の動詞リストから削除する動詞オブジェクトに対応する動詞の名前。

戻り値

削除される動詞が格納されている動詞 (BusObjVerb) オブジェクト。

例外

BusObjNoSuchVerbException
指定された動詞が存在しない場合にスローされます。

参照

insertVerb()

setAppInfo()

ビジネス・オブジェクト定義に関するアプリケーション固有の情報を設定します。

構文

```
public void setAppInfo(String appSpecInfo);
```

パラメーター

appSpecInfo ビジネス・オブジェクト・レベルのアプリケーション固有の情報。

戻り値

なし。

参照

getAppInfo()

setAttributeList()

ビジネス・オブジェクト定義用の属性リストを設定します。

構文

```
public void setAttributeList(Vector attrList);
```

パラメーター

attrList ビジネス・オブジェクト定義の属性リスト内に格納する属性オブジェクトが格納されている java.util.Vector オブジェクト。

例外

BusObjInvalidAttrException
attrList に含まれる属性オブジェクトに重複がある場合や、null のものがある場合にスローされます。

注記

setAttributeList() メソッドは、*attrList* 属性リストを、属性オブジェクトが格納された Java Vector として渡します。属性オブジェクトに情報を格納するには、BusObjAttr クラスのメソッドを使用します。この Vector オブジェクトに属性オブジェクトを格納するには、java.util.Vector クラスのメソッドを使用します。

参照

getAttributeList()

setVerbList()

ビジネス・オブジェクト定義用の動詞リストを設定します。

構文

```
public void setVerbList(Vector verbList);
```

パラメーター

verbList ビジネス・オブジェクト定義の動詞リスト内に格納する動詞オブジェクトが格納されている java.util.Vector オブジェクト。

戻り値

なし。

例外

BusObjInvalidVerbException

verbList に含まれる動詞オブジェクトに重複がある場合や、null のものがある場合にスローされます。

注記

setVerbList() メソッドは、*verbList* 動詞リストを、動詞オブジェクトが格納された Java Vector として渡します。動詞オブジェクトに情報を格納するには、BusObjVerb クラスのメソッドを使用します。この Vector オブジェクトに動詞オブジェクトを格納するには、java.util.Vector クラスのメソッドを使用します。

参照

getVerbList()

第 13 章 BusObjVerb クラス

Object Discovery Agent Development Kit (ODK) API には、ビジネス・オブジェクト定義内の動詞を表現することを目的とした BusObjVerb クラスが用意されています。 BusObjVerb インスタンスは、*動詞オブジェクト* を表しています。表 75 に、BusObjVerb クラスのメソッドについて要約します。

表 75. BusObjVerb クラスのメンバー・メソッド

メンバー・メソッド	説明	ページ
BusObjVerb()	ビジネス・オブジェクト動詞のオブジェクトを作成します。	241
clone()	動詞オブジェクトをクローンします。	241
getAppInfo()	動詞のアプリケーション固有情報を取得します。	242
getName()	動詞の名前を取得します。	242
setAppInfo()	動詞のアプリケーション固有情報を設定します。	243
setName()	動詞の名前を設定します。	243

BusObjVerb()

ビジネス・オブジェクト動詞のオブジェクトを作成します。

構文

```
public BusObjVerb(String verb, String appSpecInfo);
```

パラメーター

appSpecInfo 動詞のアプリケーション固有情報を指定します。

verb ビジネス・オブジェクト定義でサポートされている動詞を指定します。

戻り値

新規にインスタンス化された BusObjVerb オブジェクト。

例外

BusObjInvalidVerbException
指定された動詞が無効である場合にスローされます。

clone()

動詞オブジェクトをクローンします。

構文

```
public Object clone();
```

パラメーター

なし。

戻り値

なし。

注記

この clone() メソッドを指定すると、java.lang.Object クラス内の clone() メソッドがオーバーライドされます。

---

getAppInfo()

動詞のアプリケーション固有情報を取得します。

構文

```
public String getAppInfo();
```

パラメーター

なし。

戻り値

動詞のアプリケーション固有の情報が格納されている String。

参照

setAppInfo()

---

getName()

動詞の名前を取得します。

構文

```
public String getName();
```

パラメーター

なし。

戻り値

動詞の名前が格納されている String。

参照

setName()

setAppInfo()

動詞のアプリケーション固有情報を設定します。

構文

```
public void setAppInfo(String appSpecInfo);
```

パラメーター

appSpecInfo
動詞オブジェクトに格納する、動詞レベルのアプリケーション固有情報。

戻り値

なし。

参照

getAppInfo()

setName()

動詞の名前を設定します。

構文

```
public void setName(String verb);
```

パラメーター

verb　動詞オブジェクトに格納する、動詞の名前。

戻り値

なし。

例外

BusObjInvalidVerbException
指定された動詞が無効である場合にスローされます。

参照

getName()

第 14 章 CompleteCondition クラス

Object Discovery Agent Development Kit (ODK) API には、エージェント・プロパティー (AgentProperty オブジェクトで表現されます) の値に関する条件を表現することを目的とした CompleteCondition クラスが用意されています。各条件は、入力条件と従属条件という 2 種類の副条件で構成されています。エージェント・プロパティーの条件は、すべて、そのエージェント・プロパティーの allDependencies メンバー変数に格納されています。

注: 入力条件については、273 ページの『第 20 章 InputCondition クラス』を参照してください。従属条件については、259 ページの『第 17 章 DependentCondition クラス』を参照してください。

CompleteCondition クラスには、次のものが定義されています。

- 『演算子定数』
- 246 ページの『メンバー変数』
- 246 ページの『メソッド』

演算子定数

CompleteCondition クラスには、演算子定数を表す静的メンバー変数が定義されています。表 76 に、これらの演算子定数について要約します。これらは、条件に使用できる有効な演算子を表しています。演算子定数は、いずれも、String 型です。

表 76. CompleteCondition クラスの演算子定数

演算子定数	説明
OP_EQUAL	Equals (=) 演算子を表す String が含まれています。
OP_EXISTS	Exists 演算子を表す String が含まれています。
OP_GREATER_THAN	Greater Than (>) 演算子を表す String が含まれています。
OP_GREATER_THAN_EQUAL	Greater Than または Equal To (>=) 演算子を表す String が含まれています。
OP_LESS_THAN	Less Than (<) 演算子を表す String が含まれています。
OP_LESS_THAN_EQUAL	Less Than or Equal To (<=) 演算子を表す String が含まれています。
OP_NOT_EQUAL	Not Equal (!=) 演算子を表す String が含まれています。

メンバー変数

表 77 に、CompleteCondition クラスのメンバー変数について要約します。

表 77. CompleteCondition クラスのメンバー変数

メンバー変数	説明	ページ
allDependentConditions	プロパティー用のすべての従属条件を指定します。	246
allInputConditions	プロパティー用のすべての入力条件を指定します。	246

allDependentConditions

現在の完全条件に含まれるすべての従属条件が格納されている配列を指定します。

タイプ

```
public DependentCondition[] allDependentConditions
```

注記

allDependentConditions メンバー変数には、従属条件配列に含まれる従属条件のリストが格納されます。従属条件配列は、DependentCondition オブジェクトの配列です。各 DependentCondition オブジェクトには、従属条件が 1 つ格納されています。従属条件は、関連する入力条件が true であると評価された場合に、従属プロパティーの値を制限します。詳しくは、 169 ページの『プロパティー値に対する条件の設定』 を参照してください。

allInputConditions

現在の完全条件に含まれるすべての入力条件が格納されている配列を指定します。

タイプ

```
public InputCondition[] allInputConditions
```

注記

allInputConditions メンバー変数には、入力条件配列に含まれる条件のリストが格納されます。入力条件配列は、InputCondition オブジェクトの配列です。各 InputCondition オブジェクトには、入力条件が 1 つ格納されています。入力条件は、現在のエージェント・プロパティーの値に対して行う必要がある比較を指定するものです。詳しくは、 169 ページの『プロパティー値に対する条件の設定』を参照してください。

メソッド

表 78 に、CompleteCondition クラスのメソッドについて要約します。

表 78. CompleteCondition クラスのメンバー・メソッド

メンバー・メソッド	説明	ページ
CompleteCondition()	完全条件オブジェクトを作成します。	247

表 78. *CompleteCondition* クラスのメンバー・メソッド *(続き)*

メンバー・メソッド	説明	ページ
copy()	現在の完全条件を、指定された完全条件オブジェクトにコピーします。	247

CompleteCondition()

完全条件オブジェクトを作成します。

構文

```
public CompleteCondition();

public CompleteCondition(InputCondition[] allInputConds,
   DependentCondition[] allDepConds);
```

パラメーター

allDepConds 従属条件の配列を指定します。allDependentConditions メンバー変数は、このパラメーターの値で初期化されます (246 ページの『allDependentConditions』)。

allInputConds 入力条件の配列を指定します。allInputConditions メンバー変数は、このパラメーターの値で初期化されます (246 ページの『allInputConditions』)。

戻り値

新規にインスタンス化された CompleteCondition オブジェクト。

copy()

現在の完全条件を、指定された完全条件オブジェクトにコピーします。

構文

```
public void copy(CompleteCondition completeCond);
```

パラメーター

completeCond 現在の完全条件のコピー先となる完全条件オブジェクトの名前を指定します。

戻り値

なし。

第 15 章 ContentMetaData クラス

Object Discovery Agent Development Kit (ODK) API には、Object Discovery Agent (ODA) の生成コンテンツのメタデータの格納を目的とした ContentMetaData クラスが用意されています。このクラスのメンバー変数は、ODA のコンテンツ・メタデータに対応しています。ODA は、コンテンツを生成すると、生成されたコンテンツを表すコンテンツ・メタデータ・オブジェクトを戻す必要があります。コンテンツ・メタデータ・オブジェクトを戻すメソッドは、以下のように ODA がサポートするコンテンツ・プロトコルによって異なります。

- ODA が特定のコンテンツ・タイプ (ビジネス・オブジェクト定義またはファイル) 向けに要求時プロトコルをサポートしている場合、コンテンツ・メタデータは、適切なコンテンツ生成メソッドによって、ビジネス・オブジェクト・ウィザードに戻されます。
- ODA がコールバック・プロトコルをサポートしている場合 (コンテンツ・タイプがファイルのときのみ)、コンテンツ・メタデータは、ユーザー定義のメソッドによって、ODKUtility.contentComplete() メソッド経由でビジネス・オブジェクト・ウィザードに戻されます。

注: 詳しくは、 109 ページの『生成済みコンテンツの提供』を参照してください。

Business Object Designer Express は、ODA がサポートするコンテンツ・タイプごとに生成されたコンテンツに関する情報を、コンテンツ・メタデータ・オブジェクトを使用して取得します。Business Object Designer Express は、サポートされている生成プロトコルを判別するため、ODA の getContentProtocol() メソッド (IGeneratesContent クラスから) を呼び出します。

ContentMetaData クラスには、次のものが定義されています。

- 『メンバー変数』
- 251 ページの『メソッド』

メンバー変数

表 79 に、ContentMetaData クラスのメンバー変数について要約します。

表 79. ContentMetaData クラスのメンバー変数

メンバー変数	説明	ページ
contentType	生成コンテンツのコンテンツ・タイプを示します。	250
count	要求されたコンテンツのコンテンツ要素の総数を指定します。	250
length	要求されたコンテンツの全長をバイト単位で指定します。	250

contentType

生成コンテンツのコンテンツ・タイプを示します。

タイプ

```
public ContentType contentType
```

注記

contentType メンバー変数は、コンテンツ・メタデータが表す生成コンテンツのコンテンツ・タイプを示す ContentType オブジェクトです。このメンバー変数は、生成コンテンツに対応する適切なコンテンツ・タイプに設定されていなければなりません (表 80 を参照)。

表 80. コンテンツ・タイプ値

コンテンツ・タイプ	contentType メンバー変数の値
ビジネス・オブジェクト定義 バイナリー・ファイル	ContentType.BusinessObject ContentType.BinaryFile

例えば、ODA は、コンテンツ生成を完了すると、生成されたコンテンツのタイプに応じた contentType メンバー変数を持つコンテンツ・メタデータ・オブジェクトを戻す必要があります。

count

要求されたコンテンツのコンテンツ要素の総数を指定します。この count の値は、ゼロ (0) よりも大きくなければなりません。

タイプ

```
public long count
```

length

要求されたコンテンツの合計サイズをバイト単位で指定します。コンテンツの長さが不明な場合は、長さゼロ (0) を割り当てます。

重要

現在、ビジネス・オブジェクト・ウィザードでは length メンバー変数を使用していません。したがって、このメンバー変数は、ゼロ (0) や -1 などの「null」値で初期化する必要があります。

タイプ

```
public long length
```

メソッド

表 81 に、ContentMetaData クラスのメソッドについて要約します。

表 81. ContentMetaData クラスのメンバー・メソッド

メンバー・メソッド	説明	ページ
ContentMetaData()	コンテンツ・メタデータ・オブジェクトを作成します。	251
badContent()	指定されたタイプのコンテンツが ODA によって生成されなかったことを示すコンテンツ・メタデータ・オブジェクトを戻します。	251
contentNotReady()	ODA でのコンテンツ生成が完了していないことを示すコンテンツ・メタデータ・オブジェクトを戻します。	252
contentUnavailable()	適切なインターフェースが実装されているにもかかわらず ODA が指定されたコンテンツを生成しないことを通知する、コンテンツ・メタデータ・オブジェクトを戻します。	252

ContentMetaData()

コンテンツ・メタデータ・オブジェクトを作成します。

構文

```
public ContentMetaData(ContentType contentType, long length, long count);
```

パラメーター

contentType コンテンツ・メタデータ・オブジェクトが表す生成コンテンツのコンテンツ・タイプを示す ContentType オブジェクト。コンテンツ・メタデータ・オブジェクトの contentType メンバー変数は、このパラメーターの値で初期化されます (250 ページの『contentType』)。

count 要求されたコンテンツのコンテンツ要素の数を指定します。コンテンツ・メタデータ・オブジェクトの count メンバー変数は、このパラメーターの値で初期化されます (250 ページの『count』)。

length 要求されたコンテンツの合計サイズをバイト単位で指定します。コンテンツ・メタデータ・オブジェクトの length メンバー変数は、このパラメーターの値で初期化されます (250 ページの『length』)。現在、ビジネス・オブジェクト・ウィザードでは length メンバー変数を使用していません。

戻り値

新規にインスタンス化された ContentMetaData オブジェクト。

badContent()

ODA が生成したコンテンツが不完全であること、あるいは生成コンテンツにエラーがあることをビジネス・オブジェクト・ウィザードに通知します。

構文

```
public static ContentMetaData badContent(ContentType contentType);
```

パラメーター

contentType 正常に生成されていないコンテンツのコンテンツ・タイプを示す ContentType オブジェクト。

戻り値

正常に生成されていないコンテンツを表す ContentMetaData オブジェクト。

contentNotReady()

ODA が指定されたコンテンツの生成を完了していないことを、ビジネス・オブジェクト・ウィザードに通知します。

構文

```
public static ContentMetaData contentNotReady(ContentType contentType);
```

パラメーター

contentType 完全に生成されていないコンテンツのコンテンツ・タイプを示す ContentType オブジェクト。

戻り値

完全に生成されていないコンテンツを表す ContentMetaData オブジェクト。

contentUnavailable()

適切なインターフェースが実装されているにもかかわらず、ODA が指定されたコンテンツの生成をサポートしないことを、ビジネス・オブジェクト・ウィザードに通知します。

構文

```
public static ContentMetaData contentUnavailable(ContentType contentType);
```

パラメーター

contentType 生成できないコンテンツのコンテンツ・タイプを示す ContentType オブジェクト。

戻り値

生成できないコンテンツを表す ContentMetaData オブジェクト。

注記

contentUnavailable() メソッドは、*contentType* が示すコンテンツ・タイプのコンテンツを ODA が生成できないことを通知します。例えば、ODA がある特定のコンテンツ・タイプについてコールバック・コンテンツ・プロトコルのみ をサポートしている場合、ビジネス・オブジェクト・ウィザードは、ODA のコンテンツ生成メソッド (ビジネス・オブジェクト定義コンテンツ用の generateBoDefs() またはバイナリー・ファイル・コンテンツ用の generateBinFiles()) を呼び出してはいけま

せん。そのため、コンテンツ生成メソッドは、ビジネス・オブジェクト・ウィザードに対する戻り値として、`contentUnavailable()` を呼び出すことができるようになっています。

第 16 章 ContentType クラス

Object Discovery Agent Development Kit (ODK) API には、Object Discovery Agent (ODA) が生成できるコンテンツのタイプ (有効なコンテンツ・タイプ) を表現することを目的とした ContentType クラスが用意されています。 ContentType クラスには、次のものが定義されています。

- 『メンバー変数』
- 256 ページの『メソッド』

メンバー変数

表 82 に、ContentType クラスのメンバー変数について要約します。

表 82. ContentType クラスのメンバー変数

メンバー変数	説明	ページ
BinaryFile	ODA がバイナリー・ファイルをコンテンツとして生成することを示します。	255
BusinessObject	ODA がビジネス・オブジェクト定義をコンテンツとして生成することを示します。	255

BinaryFile

ODA がバイナリー・ファイルをコンテンツとして生成することを示します。

タイプ

```
public static final ContentType BinaryFile
```

注記

BinaryFile メンバー変数は、ODA がバイナリー・ファイルをコンテンツとして生成できることを示します。つまり、ODA は IGeneratesBinFiles インターフェースを実装しています。ファイル・コンテンツは、次のいずれかのコンテンツ・プロトコルを使用して生成できます。

- 要求時コンテンツ・プロトコル (ファイル生成処理を行う generateBinFiles() メソッドが ODA に実装されていなければなりません)。
- コールバック・コンテンツ・プロトコル (ファイル生成処理を行うユーザー定義のメソッドが ODA に実装されていなければなりません)。

詳しくは、 153 ページの『コンテンツとしてのバイナリー・ファイルの生成』を参照してください。

BusinessObject

ODA がビジネス・オブジェクト定義をコンテンツとして生成することを示します。

タイプ

```
public static final ContentType BusinessObject
```

注記

BusinessObject メンバー変数は、ODA がビジネス・オブジェクト定義をコンテンツとして生成できることを示します。つまり、ODA は IGeneratesBoDefs インターフェースを実装しています。ビジネス・オブジェクト定義コンテンツは、要求時コンテンツ・プロトコルを使用して生成しなければなりません。そのためには、ビジネス・オブジェクト定義の生成を処理する generateBoDefs() メソッドが ODA に実装されていることが必要です。詳しくは、128 ページの『コンテンツとしてのビジネス・オブジェクト定義の生成』を参照してください。

メソッド

表 83 に、ContentType クラスのメソッドについて要約します。

表 83. ContentType クラスのメンバー・メソッド

メンバー・メソッド	説明	ページ
ContentType()	コンテンツ・タイプ・オブジェクトを作成します。	256
equals()	2 つのコンテンツ・タイプ・オブジェクトを比較します。	256
from_int()	指定された序数値に対応するコンテンツ・タイプ・オブジェクトを生成します。	257
toString()	現在のコンテンツ・タイプ・オブジェクトのリテラル表現を戻します。	257
value()	現在のコンテンツ・タイプに対応する序数値を戻します。	258
xmlObject()	現在のコンテンツ・タイプ・オブジェクトを表現する XML オブジェクトを生成します。	258

ContentType()

コンテンツ・タイプ・オブジェクトを作成します。

構文

```
public ContentType(int contTypeOrdValue);
```

パラメーター

contTypeOrdValue
コンテンツ・タイプを表す序数値。

戻り値

新規にインスタンス化された ContentType オブジェクト。

equals()

2 つのコンテンツ・タイプ・オブジェクトを比較します。

構文

```
public boolean equals(Object contentTypeObj);
```

パラメーター

contentTypeObj
現在の ContentType オブジェクトと比較する ContentType オブジェクトへの参照。

戻り値

2 つのコンテンツ・タイプ・オブジェクトが等しいかどうかを示す boolean 値。

注記

この equals() メソッドを指定すると、java.lang.Object クラス内の equals() メソッドがオーバーライドされます。

from_int()

指定された序数値に対応するコンテンツ・タイプ・オブジェクトを生成します。

構文

```
public static ContentMetaData from_int(int contTypeOrdValue);
```

パラメーター

contTypeOrdValue
現在のコンテンツ・タイプを表す序数値。

戻り値

指定された序数値が示すコンテンツ・タイプを表す ContentMetaData オブジェクト。

参照

value()

toString()

現在のコンテンツ・タイプ・オブジェクトのリテラル表現を戻します。

構文

```
public String toString();
```

パラメーター

なし。

戻り値

現在のコンテンツ・タイプ・オブジェクトのリテラル表現が格納された String オブジェクト。

注記

この toString() メソッドを指定すると、java.lang.Object クラス内の toString() メソッドがオーバーライドされます。

value()

現在のコンテンツ・タイプに対応する序数値を戻します。

構文

```
public int value();
```

パラメーター

なし。

戻り値

現在のコンテンツ・タイプを表す序数値 (整数)。

参照

from_int()

xmlObject()

現在のコンテンツ・タイプ・オブジェクトを表現する XML オブジェクトを生成します。

構文

```
public XMLObject xmlObject();
```

パラメーター

なし。

戻り値

現在のコンテンツ・タイプ・オブジェクトを表現する com.crossworlds.ODK.XMLObject オブジェクト。

第 17 章 DependentCondition クラス

Object Discovery Agent Development Kit (ODK) API では、DependentCondition クラスを使用して従属条件を表現します。従属条件とは、依存エージェント・プロパティーの値を制限する条件を定義したものです。従属条件は、関連する入力条件が true であると評価されたときに、従属プロパティーに適用されます。従属条件および関連する入力条件は、完全条件オブジェクト (CompleteCondition) に格納されます。

注: 完全条件については、 245 ページの『第 14 章 CompleteCondition クラス』を参照してください。

DependentCondition クラスには、次のものが定義されています。

- 『メンバー変数』
- 261 ページの『メソッド』

メンバー変数

表 84に、DependentCondition クラスのメンバー変数について要約します。

表 84. DependentCondition クラスのメンバー変数

メンバー変数	説明	ページ
isDynamic	ビジネス・オブジェクト・ウィザードにおいて、従属条件の比較の前に特定値プロパティーの値の検査が必要になるかどうかを指定します。	259
operatorType	従属条件の演算子タイプを指定します。	260
propertyName	表示する従属プロパティーの名前を指定します。	260
specificValue	従属プロパティーの値と比較する値を指定します。	260
typeOfSpecificValue	従属条件の特定値のデータ型を指定します。	261

isDynamic

ビジネス・オブジェクト・ウィザードにおいて、従属条件の比較の前に特定値プロパティーの値の検査が必要になるかどうかを指定します。

タイプ

```
public boolean isDynamic
```

注記

isDynamic メンバー変数が true である場合、ビジネス・オブジェクト・ウィザードは、従属プロパティーの値に関する比較を実行する*前* に、specificValue メンバー変数に指定されているプロパティーの値を取得します。specificValue に定数が格納される場合、isDynamic は false に設定されなければなりません。

operatorType

従属条件の演算子タイプを指定します。

タイプ

```
public String operatorType
```

注記

operatorType は、ビジネス・オブジェクト・ウィザードによって行われる従属プロパティー (propertyName メンバー変数に指定されたプロパティー) の値と specificValue の値の比較の種類を指定します。operatorType 変数に有効な値は、CompleteCondition クラスに定義されている演算子定数です。詳しくは、245 ページの表 76 を参照してください。

propertyName

従属プロパティーの名前を指定します。

タイプ

```
public String propertyName
```

注記

propertyName メンバー変数には、従属プロパティーの名前が格納されます。従属条件によって制限されるのは、この従属プロパティーの値です (関連する入力条件が true であると評価された場合に制限されます)。

specificValue

従属プロパティーの値と比較する値を指定します。

タイプ

```
public String specificValue
```

注記

specificValue には、従属条件の値が保持されます。この値は、ビジネス・オブジェクト・ウィザードによって、従属プロパティー (propertyName メンバー変数に指定されたプロパティー) の値と比較されます。比較の種類は、operatorType 変数によって決定されます。この変数に保持される特定値は次のいずれかです。

- 従属プロパティーと同じ型の定数

  例えば、従属条件として Less Than 演算子 (CompleteCondition.OP_LESS_THAN) が operatorType に指定され、値 5 が specificValue に指定されているときに、関連する入力条件が true であると評価された場合、従属プロパティーの値は 5 よりも小さくなければなりません。

- 別のエージェント・プロパティーの名前

  例えば、従属条件として Greater Than 演算子 (CompleteCondition.OP_GREATER_THAN) が operatorType に指定され、「Property1」プロパティーの名前が specificValue に指定されているときに、

関連する入力条件が true であると評価された場合、従属プロパティーの値は Property1 エージェント・プロパティーの値よりも大きくなければなりません。

specificValue 変数は、どの種類の値でも保持できるように、String 型の変数として宣言されます。ただし、比較を正しく行うには、typeOfSpecificValue メンバー変数に格納される特定値の実際のデータ型が、ビジネス・オブジェクト・ウィザードに認識されなければなりません。

typeOfSpecificValue

従属条件の特定値のデータ型を指定します。

タイプ

```
public int typeOfSpecificValue
```

注記

typeOfSpecificValue には、従属条件の特定値のデータ型が保持されます。specificValue 変数は、どの種類の値でも保持できるように、String 型の変数として宣言されます。ただし、比較が正常に実行されるためには、ビジネス・オブジェクト・ウィザードが特定値の実際のデータ型を認識していることが必要です。typeOfSpecificValue 変数の値として有効な値は、AgentProperty クラスに定義されているプロパティー・タイプ定数です。詳しくは、201 ページの表 67 を参照してください。

例えば、従属条件の特定値が整数定数 5 である場合、以下のようになります。

- specificValue 変数には、ストリング「5」が保持されます。
- typeOfSpecificValue 変数には、AgentProperty.TYPE_INTEGER プロパティー・タイプ定数が保持されます。

メソッド

表 85 に、DependentCondition クラスのメソッドについて要約します。

表 85. DependentCondition クラスのメンバー・メソッド

メンバー・メソッド	説明	ページ
DependentCondition()	従属条件オブジェクトを作成します。	261
copy()	現在の従属条件を、指定された DependentCondition オブジェクトにコピーします。	262

DependentCondition()

従属条件オブジェクトを作成します。

構文

```
public DependentCondition();

public DependentCondition(String name, String op,
   boolean isDyn, int type, String specificVal);
```

パラメーター

isDyn 特定値プロパティーの値を動的に取得する必要があるかどうかを指定します。isDynamic メンバー変数は、このパラメーターの値で初期化されます (259 ページの『isDynamic』)。

name 従属プロパティーの名前です。propertyName メンバー変数は、このパラメーターの値で初期化されます (260 ページの『propertyName』)。

op 実行する必要がある比較の種類を指定する演算子です。operatorType メンバー変数は、このパラメーターの値で初期化されます (260 ページの『operatorType』)。

specificVal 従属条件の特定値を指定します。specificValue メンバー変数は、このパラメーターの値で初期化されます (260 ページの『specificValue』)。

type 特定値のデータ型を指定します。typeOfSpecificValue メンバー変数は、このパラメーターの値で初期化されます (261 ページの『typeOfSpecificValue』)。

戻り値

新規にインスタンス化された DependentCondition オブジェクト。

copy()

現在の従属条件を、指定された従属条件オブジェクトにコピーします。

構文

```
public void copy(DependentCondition depCond);
```

パラメーター

depCond 現在の従属条件のコピー先となる従属条件オブジェクトへの参照です。

戻り値

なし。

第 18 章 IGeneratesBinFiles インターフェース

Object Discovery Agent (ODA) がそのコンテンツとしてのバイナリー・ファイルを生成するのに必要な機能は、Object Discovery Agent Development Kit (ODK) API により定義されます。その定義の際に使用されるのが IGeneratesBinFiles インターフェースです。 ODA によるバイナリー・ファイルの生成を可能にするために ODA 開発者が実装する必要のある一連のメソッドは、このインターフェースによって定義されます。ビジネス・オブジェクト・ウィザードは IGeneratesBinFiles インターフェースのメソッドを呼び出し、コンテンツ (ファイル・オブジェクト) の生成およびアクセスを行います。ファイル・オブジェクトとは、バイナリーのオペレーティング・システム・ファイルを表す Java File オブジェクトです。

注: ODA のコンテンツとしてのビジネス・オブジェクト定義の生成も、ODA によってサポートされていなければなりません。ODA がソース・データからビジネス・オブジェクト定義を生成できるようにするには、IGeneratesBoDefs インターフェースを実装しておく必要があります。詳しくは、267 ページの『第 19 章 IGeneratesBoDefs インターフェース』を参照してください。

ODA がファイル・オブジェクトを生成できるよう、ODA 開発者は以下のステップを実行する必要があります。

- ODA クラス (ODKAgentBase2 クラスの拡張) を定義する際には、ODA により実装されるインターフェースとしての IGeneratesBinFiles をインクルードしてください。
- ODA クラスの内部には、IGeneratesBinFiles インターフェースのメソッドを実装します。IGeneratesBinFiles はインターフェースであるため、ODA 開発者は表 86 のメソッドをすべて 実装する必要があります。

表 86. IGeneratesBinFiles インターフェースのメンバー・メソッド

メンバー・メソッド	説明	ページ
generateBinFiles()	データ・ソースから選択されたソース・ノード用のファイル・オブジェクトを生成します。	263
getBinFile()	生成済みのファイル・オブジェクトを取得します。	265
getContentProtocol()	このバイナリー・ファイルのコンテンツ・タイプ用にサポートされているコンテンツ・プロトコルを示します。	266

generateBinFiles()

ファイル・オブジェクトを生成します。

構文

```
public ContentMetaData generateBinDefs(String[] strNames);
```

パラメーター

strNames [ ] String オブジェクトの配列です。この引き数は現在使用されていません。

戻り値

ContentMetaData オブジェクト (生成されたファイル・オブジェクトについての記述内容) です。

例外

ODKException バイナリー・ファイルの生成に失敗した場合にスローされます。

注記

generateBinFiles() メソッドの用途は、ODA がファイル (ContentType.BinaryFile) 内容の生成に使用するコンテンツ・プロトコルに応じて以下のように異なります。

- ODA が「要求時に」ファイルを生成するときは、ビジネス・オブジェクト・ウィザードが generateBinFiles() メソッドを明示的に呼び出してファイルを生成します。
- ODA がコールバックを介してファイルを生成する場合、ビジネス・オブジェクト・ウィザードは generateBinFiles() メソッドを明示的には*呼び出しません*。代わりに、ODA はそれ以外のなんらかの方法を使用して、ビジネス・オブジェクト・ウィザードが後でアクセスできるようにファイルを生成します。

ODA が「要求時に」ファイルを生成するときは、generateBinFiles() メソッドは IGeneratesBinFiles インターフェース用のコンテンツ生成メソッドとなります。このメソッドは、ビジネス・オブジェクト定義生成プロセスに関する情報を格納するファイル・オブジェクトを作成することが出来ます。ビジネス・オブジェクト・ウィザードは、generateBinFiles() メソッドを呼び出してコンテンツ (ODA がファイル・コンテンツの生成をサポートしている場合) を生成します。このメソッドはウィザード開始のステップ 5 (ビジネス・オブジェクトの生成) で呼び出されます。

要求時プロトコルの場合、このメソッドにより生成済みコンテンツが戻されることは実際にはありません。代わりに戻されるのは、生成済みコンテンツについて記述した情報を包含する、コンテンツ・メタデータ (ContentMetaData) オブジェクトです。ビジネス・オブジェクト・ウィザードは、この戻されたコンテンツ・メタデータ・オブジェクトに基づいて、コンテンツ生成処理が完了したかどうかを判断できます。生成が完了すると、ビジネス・オブジェクト・ウィザードは getBinFiles() メソッドを使用して、生成済みのファイル・オブジェクトを取得します。
generateBinFiles() の実装方法について詳しくは、157 ページの『ファイルの生成』を参照してください。

参照

generateBoDefs(), getBinFile()

getBinFile()

生成されたコンテンツの構造体から生成済みファイル・オブジェクトを取得します。

構文

```
public File[] getBinFile(long index);
```

パラメーター

index 生成されたコンテンツの構造体から取得するファイル・オブジェクトを指定します。

例外

ODKException ビジネス・オブジェクト・ウィザードが、生成されたコンテンツの構造体から生成済みファイル・オブジェクトを取得しようとして問題が発生した場合にスローされます。

注記

getBinFile() メソッドは、IGeneratesBinFiles インターフェース用の内容検索メソッドです。このメソッドは、ODA 用の生成されたコンテンツの構造体 (生成済みファイル・オブジェクトが ODA により取り込まれた構造体) から、生成済みファイル・オブジェクトを検索します。生成されたコンテンツの構造体への取り込みを行うメソッドは、次のように、ODA がファイル生成用にサポートしているコンテンツ・プロトコルに応じて異なります。

- ODA が「要求時に」ファイルを生成するときは、生成されたコンテンツの構造体への取り込みが generateBinFiles() メソッドによって実行されます。
- ODA がコールバックを介してファイルを生成する場合、生成されたコンテンツの構造体への取り込みは、ユーザー定義のメソッドによって実行されます。

getBinFile() が生成済みファイル・オブジェクトを 1 つ戻すかまたはすべて戻すかは、*index* 引き数の値によって決定されます(表 87 を参照)。

表 87. 戻されるファイル・オブジェクトの指定

index 引き数の値	getBinFile() の動作
0 から *count* までの範囲内 (ここで *count* は、生成されたコンテンツの構造体の中のファイル・オブジェクトの数を指定する、コンテンツ・メタデータ・オブジェクト内のメンバー変数です) ODKConstant.GET_ALL_OBJECTS	*1* つ のファイル・オブジェクト Java File (生成されたコンテンツの構造体の中の、指定された *index* 位置にある File オブジェクト) が格納された配列を戻します。 生成されたコンテンツの構造体の中の生成済みのファイル・オブジェクトがすべて 格納された配列を戻します。

getBinFile() の実装方法について詳しくは、162 ページの『生成済みファイルへのアクセスの提供』を参照してください。

参照

generateBinFiles(), getBoDefs()

---

getContentProtocol()

ODA がサポートしている、指定されたコンテンツ・タイプ用のコンテンツ・プロトコルを示します。

構文

```
public long getContentProtocol(ContentType contentType);
```

パラメーター

contentType サポートされているコンテンツ・プロトコルを取得するメソッド用のコンテンツ・タイプを示します。

戻り値

ODA により実装されるコンテンツ・プロトコルを示す長整数 (long) 値。この long 値を以下のコンテンツ・プロトコル定数と比較してください。

ODKConstant.CONTENT_PROTOCOL_CALLBACK
ODA がコールバック・プロトコルをサポートしていることを示します。つまり、指定されたコンテンツの生成開始、および生成完了時のビジネス・オブジェクト・ウィザードへの通知は ODK により実行されます。

ODKConstant.CONTENT_PROTOCOL_ONREQUEST
ODA がオンデマンド・プロトコルをサポートしていることを示します。つまり、指定されたコンテンツ・タイプの生成開始はビジネス・オブジェクト・ウィザードにより実行されます。

注記

getContentProtocol() メソッドは、IGeneratesBoDefs インターフェースの拡張である IGeneratesContent インターフェース内に定義されている単一メソッドです。*contentType* コンテンツ・タイプ用に ODA がサポートしているコンテンツ・プロトコルを判別するための getContentProtocol() は、ビジネス・オブジェクト・ウィザードによって呼び出されます。詳しくは、126 ページの『実装されるコンテンツ・プロトコルの指定』を参照してください。

第 19 章 IGeneratesBoDefs インターフェース

Object Discovery Agent (ODA) がそのコンテンツとしてのビジネス・オブジェクト定義を生成するのに必要な機能は、Object Discovery Agent Development Kit (ODK) API により定義されます。その定義の際に使用されるのが IGeneratesBoDefs インターフェースです。 ODA がソース・データからビジネス・オブジェクト定義を生成するための一連のメソッドは、ODA 開発者が実装しておく必要があります。この一連のメソッドの定義は、この IGeneratesBoDefs インターフェースによって行われます。IGeneratesBoDefs インターフェースのメソッドは、ビジネス・オブジェクト・ウィザードにより呼び出されます。このメソッドの呼び出しにより、ソース・ノードの取得のほか、コンテンツであるビジネス・オブジェクト定義の生成、およびアクセスが可能になります。

注: ODA のコンテンツとしてのファイル・オブジェクトの生成も、ODA によりサポートすることができます。ODA がソース・データからバイナリー・ファイルを生成できるようにするには、IGeneratesBinFiles インターフェースを実装しておく必要があります。詳しくは、263 ページの『第 18 章 IGeneratesBinFiles インターフェース』を参照してください。

ODA がソース・データからビジネス・オブジェクト定義を生成できるよう、ODA 開発者は以下のステップを実行する必要があります。

- ODA クラス (ODKAgentBase2 クラスの拡張) を定義する際には、ODA により実装されるインターフェースとしての IGeneratesBoDefs をインクルードしてください。
- ODA クラスの内部には、IGeneratesBoDefs インターフェースのメソッドを実装します。IGeneratesBoDefs はインターフェースであるため、ODA 開発者は表 88 のメソッドをすべて 実装する必要があります。

表 88. IGeneratesBoDefs インターフェースのメンバー・メソッド

メンバー・メソッド	説明	ページ
generateBoDefs()	データ・ソースを基に、指定されたソース・ノード用のビジネス・オブジェクト定義を生成します。	267
getBoDefs()	生成済みのビジネス・オブジェクト定義を検索します。	269
getContentProtocol()	このビジネス・オブジェクト定義のコンテンツ・タイプ用にサポートされているコンテンツ・プロトコルを示します。	269
getTreeNodes()	ソース・ノードの階層を表す、ツリー・ノードの配列を作成します。	270

generateBoDefs()

指定されたソース・ノード用のビジネス・オブジェクト定義を生成します。

構文

```
public ContentMetaData generateBoDefs(String[] srcNodeNames);
```

パラメーター

srcNodeNames [ ]
ユーザーが選択したソース・ノードの名前が格納されている配列です。

戻り値

ContentMetaData オブジェクト (*srcNodeNames* 引き数に指定されたソース・ノード用の生成済みビジネス・オブジェクト定義についての記述内容) です。

例外

ODKException ビジネス・オブジェクト定義の生成に失敗した場合にスローされます。

注記

generateBoDefs() メソッドは、IGeneratesBoDefs インターフェース用のコンテンツ生成メソッドです。このメソッドは、*srcNodeNames* 配列内に指定されている各ソース・ノード用のビジネス・オブジェクト定義を作成します。これらのソース・ノードは、ビジネス・オブジェクト・ウィザードの「ソースの選択」ダイアログ・ボックスでユーザーにより選択されたものです。ユーザーがソース・ノードを選択し終えると、ビジネス・オブジェクト・ウィザードは generateBinFiles() メソッドを呼び出してコンテンツを生成します。このメソッドはウィザード開始のステップ 5 (ビジネス・オブジェクトの生成) で呼び出されます。

注: ビジネス・オブジェクト定義を生成するためには、要求時コンテンツ・プロトコルを ODA によりサポートする必要があるため、ビジネス・オブジェクト・ウィザードは generateBoDefs() を常時呼び出します。コンテンツ・プロトコルについて詳しくは、126 ページの『ODA コンテンツ・プロトコルの選択』を参照してください。

generateBoDefs() メソッドの目的は、ユーザーにより選択された各ソース・ノード用のビジネス・オブジェクト定義 (BusObjDef オブジェクト) を生成し、生成されたコンテンツの構造体に保存して、コンテンツ・メタデータ・オブジェクト ContentMetaData (生成済みコンテンツについての記述内容) を戻すことにあります。このメソッドでは、生成済みコンテンツがビジネス・オブジェクト・ウィザードに戻されることは事実上ありません。ビジネス・オブジェクト・ウィザードは、この戻されたコンテンツ・メタデータ・オブジェクトに基づいて、コンテンツ生成処理が完了したかどうかを判断できます。生成が完了すると、ビジネス・オブジェクト・ウィザードは getBoDefs() メソッドを使用して、生成済みのビジネス・オブジェクト定義を取得します。generateBoDefs() の実装方法について詳しくは、137 ページの『ビジネス・オブジェクト定義の生成』を参照してください。

参照

generateBinFiles(), getBoDefs()

getBoDefs()

生成済みのビジネス・オブジェクト定義を検索します。

構文

```
public BusObjDef[] getBoDefs(long index);
```

パラメーター

index 生成されたコンテンツの構造体から検索するビジネス・オブジェクト定義を指定します。

例外

ODKException ビジネス・オブジェクト・ウィザードが、生成されたコンテンツの構造体から生成済みビジネス・オブジェクト定義を取得しようとして問題が発生した場合にスローされます。

注記

getBoDefs() メソッドは、IGeneratesBoDefs インターフェース用の内容検索メソッドです。このメソッドは、ODA 用の生成されたコンテンツの構造体 (生成済みビジネス・オブジェクト定義が generateBoDefs() メソッドによって取り込まれた構造体) から生成済みビジネス・オブジェクト定義を検索します。getBoDefs() が生成済みビジネス・オブジェクト定義を 1 つ戻すかまたはすべて戻すかは、*index* 引き数の値によって決定されます (表 89 を参照)。

表 89. 戻されるビジネス・オブジェクト定義の指定

index 引き数の値	getBoDefs() の動作
0 から *count* までの範囲内 (ここで *count* は、生成されたコンテンツの構造体の中のビジネス・オブジェクト定義の数を指定する、コンテンツ・メタデータ・オブジェクト内のメンバー変数です)	*1* つ のビジネス・オブジェクト定義 BusObjDef (生成されたコンテンツの構造体の中の、指定された *index* 位置にある BusObjDef オブジェクト) が格納された配列を戻します。
ODKConstant.GET_ALL_OBJECTS	生成されたコンテンツ構造体中の生成済みビジネス・オブジェクト定義がすべて 格納された配列を戻します。

getBoDefs() の実装方法について詳しくは、152 ページの『生成済みビジネス・オブジェクト定義へのアクセスの提供』を参照してください。

参照

generateBoDefs(), getBinFile()

getContentProtocol()

ODA がサポートしている、指定されたコンテンツ・タイプ用のコンテンツ・プロトコルを示します。

構文

```
public long getContentProtocol(ContentType contentType);
```

パラメーター

contentType サポートされているコンテンツ・プロトコルを判別するメソッド用のコンテンツ・タイプを示します。

戻り値

ODA により実装されるコンテンツ・プロトコルを示す長整数 (long) 値。この long 値を以下のコンテンツ・プロトコル定数と比較してください。

ODKConstant.CONTENT_PROTOCOL_CALLBACK
ODA がコールバック・プロトコルをサポートしていることを示します。つまり、指定されたコンテンツの生成開始、および生成完了時のビジネス・オブジェクト・ウィザードへの通知は ODK により実行されます。

ODKConstant.CONTENT_PROTOCOL_ONREQUEST
ODA がオンデマンド・プロトコルをサポートしていることを示します。つまり、指定されたコンテンツ・タイプの生成開始はビジネス・オブジェクト・ウィザードにより実行されます。

注記

getContentProtocol() メソッドは、IGeneratesBoDefs インターフェースの拡張である IGeneratesContent インターフェース内に定義されている単一メソッドです。*contentType* コンテンツ・タイプ用に ODA がサポートしているコンテンツ・プロトコルを判別するための getContentProtocol() は、ビジネス・オブジェクト・ウィザードによって呼び出されます。詳しくは、126 ページの『実装されるコンテンツ・プロトコルの指定』を参照してください。

getTreeNodes()

ソース・ノードの階層内の 1 つのレベルを表す、ツリー・ノードの配列を作成します。

構文

```
public TreeNode[] getTreeNodes(String parentNodePath, String searchPattern);
```

パラメーター

parentNodePath
トップレベル・ノードから、ビジネス・オブジェクト・ウィザードに戻される子の親ソース・ノードまでの完全修飾パスを指定します。パスの中の各ノードはコロン (:) で区切ります。

searchPattern 拡張可能な *parentNodePath* ノード内の下位ノードに対するユーザー指定の検索パターンです。

戻り値

TreeNode オブジェクト (指定されたオブジェクト階層内の下位ノードとなっているツリー・ノード・オブジェクト) の配列です。

例外

ODKException Object Discovery Agent がツリー・ノードを取得するときに問題が発生した場合にスローされます。

注記

getTreeNodes() メソッドは、IGeneratesBoDefs インターフェース用のソース・ノード生成メソッドです。ビジネス・オブジェクト・ウィザードが getTreeNodes() を起動すると、ウィザードの「ソースの選択」(ステップ 3) ダイアログ・ボックスを初期化するツリー・ノードの配列が取得されます。このダイアログ・ボックスから、ビジネス・オブジェクト定義生成用の特定のソース・ノードを選択します。データ・ソース内のソース・ノードの階層を表すツリー・ノードは、getTreeNodes() メソッド内で構築しておく必要があります。getTreeNode() メソッドは、このソース・ノードの階層を TreeNode オブジェクトの配列として、呼び出し元のビジネス・オブジェクト・ウィザードに戻します。

getTreeNodes() が戻すツリー・ノード配列によって、ソース・ノード階層の特定レベルにソース・ノードができます。どのレベルにおいても、一部のソース・ノードを拡張可能な (下位ノードを持つ) ソース・ノードにし、ほかのソース・ノードをリーフ (終端) ノードにすることができます。ソース・ノードの左側にプラス (+) 記号が表示されているときは、そのソース・ノードを展開することによって、階層をトラバースすることができます。ユーザーがノードを展開すると、ビジネス・オブジェクト・ウィザードによって getTreeNodes() が再度呼び出され、ユーザーが展開するノードの名前がこのメソッドの *parentNodePath* 引き数として指定されます。このノード名は、パス内の各ノードの名前で構成されます。ノード名はそれぞれコロン (:) で区切られます。

注: ODA では、パスがオペレーティング・システムに依存しないようにするため、スラッシュや円記号ではなくコロンを使用します。

getTreeNodes() メソッドは、以下の基本タスクを実行して、ソース・ノードを生成します。

1. データ・ソース内にある検索対象の親オブジェクトが識別されるよう、*parentNodePath* を解析します。
2. 指定されたデータ・ソース親オブジェクトの子オブジェクトを検出します。

   ビジネス・オブジェクト・ウィザードが *searchPattern* 引き数を getTreeNodes() に渡すと、ユーザーによる検索基準の指定が完了します。結果として、getTreeNodes() によって戻されるノードは必然的に *parentNodePath* ノードの下位ノードのうち *searchPattern* の検索基準に適合した下位ノードだけに限られることになります。検索パターンをソース・ノードに適用可能にするためには、以下の条件が true でなければなりません。

   - ユーザー指定の *searchPattern* が、ODA によりサポートされている検索基準と一致していなければならない

ODA のメタデータ (AgentMetaData) オブジェクト内にある searchPatternDesc メンバー変数により、サポートされている検索基準の説明がユーザーに対して表示されます。ただし、ユーザー指定の *searchPattern* については、getTreeNodes() メソッドで解析して、サポートされている検索基準に適合することを確認しておく必要があります。

- ODA が検索パターンをサポートしている

  ODA のメタデータ (AgentMetaData) オブジェクト内の searchableNodes メンバー変数は trueでなければなりません。searchableNodes が false の場合、検索基準のユーザー入力を開始するための「項目を検索」メニュー項目は利用できません。この場合は、検索基準のユーザー入力はできません。

3. 子オブジェクト用のツリー・ノードを構築してツリー・ノード配列に格納します。

getTreeNodes() の実装方法について詳しくは、129 ページの『ソース・ノードの生成』を参照してください。

参照

関連する参照情報については、307 ページの『第 25 章 TreeNode クラス』を参照してください。

第 20 章 InputCondition クラス

Object Discovery Agent Development Kit (ODK) API には、入力条件を表す InputCondition クラスが提供されています。このクラスを使用して、エージェント・プロパティー値に関する条件を指定することができます。入力条件が true に評価された場合、関連する従属条件が従属プロパティーに適用されます。入力条件および関連する従属条件 (1 つまたは複数) は、完全条件 (CompleteCondition) オブジェクト内に格納されます。

注: 完全条件については、245 ページの『第 14 章 CompleteCondition クラス』を参照してください。

InputCondition クラスによって、以下が定義されます。

- 『メンバー変数』
- 275 ページの『メソッド』

メンバー変数

表 90 は、InputCondition クラスのメンバー変数についての要約です。

表 90. InputCondition クラスのメンバー変数

メンバー変数	説明	ページ
isDynamic	入力条件の比較の前にビジネス・オブジェクト・ウィザードによる特定値プロパティー値のチェックが必要かどうかを指定します。	273
operatorType	入力条件の演算子タイプを指定します。	274
specificValue	エージェント・プロパティーの値と比較する値を指定します。	274
typeOfSpecificValue	入力条件となる特定の値のデータ型を指定します。	274

isDynamic

入力条件の比較の前にビジネス・オブジェクト・ウィザードによる特定値プロパティー値のチェックが必要かどうかを指定します。

タイプ

```
public boolean isDynamic
```

注記

isDynamic メンバー変数が true の場合、specificValue メンバー変数がエージェント・プロパティーの値との比較を行う*前に* 指定したプロパティーの値が、ビジネス・オブジェクト・ウィザードによって取得されます。specificValue に定数が含まれている場合、isDynamic を false に設定する必要があります。

operatorType

入力条件の演算子タイプを指定します。

タイプ

```
public String operatorType
```

注記

operatorType には、ビジネス・オブジェクト・ウィザードによって行われるエージェント・プロパティー値と specificValue 間の比較の種類を指定します。operatorType 変数に有効な値は、CompleteCondition クラスに定義されている演算子定数です。詳しくは、245 ページの表 76 を参照してください。

specificValue

エージェント・プロパティーの値と比較する値を指定します。

タイプ

```
public String specificValue
```

注記

specificValue には、ビジネス・オブジェクト・ウィザードがエージェント・プロパティーの値と比較する、入力条件の値が保持されます。比較の種類は、operatorType 変数によって決定されます。特定値は、以下のいずれかです。

- (エージェント・プロパティーと同じタイプの) 定数

  例えば、ある入力条件に operatorType として Less Than 演算子 (CompleteCondition.OP_LESS_THAN) が指定されていて、specificValue として値 5 が指定されている場合、関連する従属条件はエージェント・プロパティーの値が 5 未満のときに適用されます。
- 別のエージェント・プロパティーの名前

  例えば、ある入力条件に operatorType として Greater Than 演算子 (CompleteCondition.OP_GREATER_THAN) が指定されていて、specificValue として「Property1」プロパティーの名前が指定されている場合、関連する従属条件はエージェント・プロパティーの値が 「Property1」プロパティーの値より大きいときに適用されます。

specificValue 変数を String 型で宣言しておくと、この変数にどのような種類の値も保持されるようになります。ただし、比較を正しく行うには、typeOfSpecificValue メンバー変数に格納される特定値の実際のデータ型が、ビジネス・オブジェクト・ウィザードに認識されなければなりません。

typeOfSpecificValue

入力条件となる特定の値のデータ型を指定します。

タイプ

```
public int typeOfSpecificValue
```

注記

typeOfSpecificValue には、入力条件となる特定の値のデータ型が保持されます。specificValue 変数を String 型で宣言しておくと、この変数にどのような種類の値も保持されるようになります。ただし、正確に比較を行うには、特定値の実際のデータ型がビジネス・オブジェクト・ウィザードに認識されていなければなりません。 typeOfSpecificValue 変数に有効な値は、AgentProperty クラスに定義されているプロパティー・タイプ定数です。詳しくは、 201 ページの表 67 を参照してください。

例えば、入力条件の特定値が 5 の整定数である場合は、以下のようになります。

- specificValue 変数には、ストリング「5」が保持されます。
- typeOfSpecificValue 変数には、AgentProperty.TYPE_INTEGER プロパティー・タイプ定数が保持されます。

メソッド

表 91 に、InputCondition クラスのメソッドについて要約します。

表 91. InputCondition クラスのメンバー・メソッド

メンバー・メソッド	説明	ページ
InputCondition()	入力条件オブジェクトを作成します。	275
copy()	指定された入力条件オブジェクト内に現在の入力条件をコピーします。	276

InputCondition()

入力条件オブジェクトを作成します。

構文

```
public InputCondition();

public InputCondition(String operator, boolean isDyn, int type,
   String specificVal);
```

パラメーター

isDyn 特定値プロパティーの値を動的に取得する必要があるかどうかを指定します。isDynamic メンバー変数は、このパラメーターの値で初期化されます (259 ページの『isDynamic』)。

operator 実行する必要がある比較の種類を指定する演算子です。operatorType メンバー変数は、このパラメーターの値で初期化されます (260 ページの『operatorType』)。

specificVal 入力条件の特定値です。specificValue メンバー変数 (274 ページの『specificValue』) は、このパラメーターの値で初期化されます。

type 特定値のデータ型を指定します。typeOfSpecificValue メンバー変数は、このパラメーターの値で初期化されます (261 ページの『typeOfSpecificValue』)。

戻り値

新規にインスタンス化された InputCondition オブジェクト。

copy()

指定された入力条件オブジェクト内に現在の入力条件をコピーします。

構文

```
public void copy(InputCondition inputCond);
```

パラメーター

inputCond 現在の入力条件のコピー先となる InputCondition オブジェクトへの参照です。

戻り値

なし。

第 21 章 ODKAgentBase2 クラス

Object Discovery Agent Development Kit (ODK) API には、Object Discovery Agent (ODA) の基底クラスとしての ODKAgentBase2 クラスが提供されています。 ODA 開発者は、このクラスから *ODA クラス* を導出して、ODA 用の抽象メソッドを実装する必要があります。

注: ODKAgentBase2 クラスは、低水準 ODA ライブラリーの ODKAgentBase クラスを拡張します。 ODKAgentBase2 クラスは、この ODKAgentBase クラスの getAgentProperties() メソッド、getVersion() メソッド、init() メソッド、および terminate() メソッドを継承します。また、この ODKAgentBase クラスの getTreeNodes() メソッドおよび generateDefs() メソッドを「使用不可」にします。なぜなら、これらのメソッドはこの時点で IGeneratesBoDefs インターフェースの getTreeNodes() メソッドおよび generateBoDefs() メソッド内に定義された機能で置き換えられるためです。

重要

この ODA 基底クラスは、各 ODA によって拡張されなければなりません。また、getVersion() を除くすべての ODA メソッドのインプリメンテーションは、各 ODAによって*提供されなければなりません*。

表 92 に、ODKAgentBase2 クラスのメソッドについて要約します。

表 92. ODKAgentBase2 クラスのメンバー・メソッド

メンバー・メソッド	説明	ページ
getAgentProperties()	ODA 構成プロパティーの配列をビジネス・オブジェクト・ウィザードに送信します。	277
getMetaData()	ODA のメタデータをビジネス・オブジェクト・ウィザードに送信します。	278
getVersion()	ODA のバージョンを検索します。	279
init()	ODA を初期化します。	279
terminate()	必要なクリーンアップ・タスクを実行して ODA を終了します。	280

getAgentProperties()

ODA 構成プロパティーの配列をビジネス・オブジェクト・ウィザードに送信します。

構文

```
public abstract AgentProperty[] getAgentProperties();
```

パラメーター

なし。

戻り値

AgentProperty オブジェクトの配列 (ODA 構成プロパティーごとに 1 つのオブジェクト)

例外

ODKException ODA が構成プロパティーの取得に失敗した場合にスローされます。

注記

ビジネス・オブジェクト・ウィザードは getAgentProperties() メソッドを起動して、「エージェントの構成」(ステップ 2) ダイアログ・ボックスを初期化するための ODA 構成プロパティーの配列を取得します。このダイアログ・ボックスから、これらのプロパティー値の入力や変更ができます。

重要: getAgentProperties() メソッドは、デフォルトのインプリメンテーションを持たない抽象メソッドです。したがって、この getAgentProperties() メソッドは ODA クラスで実装する必要があります。

getAgentProperties() メソッド内で、ODA 構成プロパティーごとにエージェント・プロパティー (AgentProperty) オブジェクトのインスタンス化および初期化を行い、構成プロパティー配列の中に各プロパティーを格納する必要があります。この構成プロパティー配列は、getAgentProperties() メソッドによって呼び出し元のビジネス・オブジェクト・ウィザードに戻されます。「エージェントの構成」ダイアログ・ボックスから構成プロパティーを設定し終えると、ユーザーが初期化したこれらのプロパティーをビジネス・オブジェクト・ウィザードが ODA ランタイム・メモリー内に読み込みます。ユーザーにより初期化されたプロパティーを取得するには、ODKUtility クラスの getAgentProperty() メソッドまたは getAllAgentProperties() メソッドを使用します。getAgentProperties() の実装方法について詳しくは、118 ページの『構成プロパティーの取得』を参照してください。

参照

getAgentProperty(), getAllAgentProperties()

---

getMetaData()

ODA のメタデータをビジネス・オブジェクト・ウィザードに送信します。

構文

```
public abstract AgentMetaData getMetaData();
```

パラメーター

なし。

戻り値

ODA 用のメタデータを包含する AgentMetaData オブジェクト。

注記

ビジネス・オブジェクト・ウィザードは、getMetaData() メソッドを起動して ODA のメタデータを取得します。 getAgentProperties() メソッドの呼び出し完了後は、getMetaData() が呼び出されます。getMetaData() メソッドによって戻されるのが、ODA 用の AgentMetaData オブジェクトです。この AgentMetaData オブジェクトは、ODA 用のメタデータ (例えば、サポートされているバージョンや生成済みコンテンツなど) を包含します。このメソッドの内部で AgentMetaData() コンストラクターを呼び出し、インスタンス化されたメタデータ・オブジェクトを戻してください。

重要: getMetaData() メソッドは、デフォルトのインプリメンテーションを持たない抽象メソッドです。したがって、この getMetaData() メソッドは ODA クラスで実装する必要があります。

getMetaData() の実装方法について詳しくは、120 ページの『ODA メタデータの初期化』を参照してください。

getVersion()

ODA ランタイムのバージョンを検索します。

構文

```
public String getVersion();
```

パラメーター

なし。

戻り値

ODA ランタイムのバージョンが格納されている String。

init()

ODA 初期化タスクを実行します。

構文

```
public abstract void init();
```

パラメーター

なし。

戻り値

なし。

例外

ODKException ODA の初期化が失敗した場合にスローされます。

注記

Business Object Designer Express は、init() メソッドを呼び出すことにより、Object Discovery Agent を初期化します。getAgentProperties() メソッドおよび getMetaData() メソッドの呼び出し完了後は、init() が呼び出されます。ODA の初期化では一般的に、ODA の構成プロパティー値の取得、データ・ソース (アプリケーション、データベース、XML ファイル、その他のビジネス・オブジェクト・ソースなど) への接続の確立、ODA に必要なリソースの割り当てが行われます。

重要: init() メソッドは、デフォルトのインプリメンテーションを持たない抽象メソッドです。したがって、この init() メソッドは ODA クラスで実装する必要があります。

init() の実装方法について詳しくは、122 ページの『ODA 開始の初期化』を参照してください。

terminate()

必要なクリーンアップ・タスクを実行して ODA を終了します。

構文

```
public abstract void terminate();
```

パラメーター

なし。

戻り値

なし。

注記

Business Object Designer Express は、ODA をシャットダウンするときに terminate() メソッドを呼び出します。このメソッドを実装する際は、メモリーをすべて解放してデータ・ソースから切断するのを習慣にしておくとよいでしょう。

重要: terminate() メソッドは、デフォルトのインプリメンテーションを持たない抽象メソッドです。したがって、この terminate() メソッドは ODA クラスで実装する必要があります。

使用すべきでないメソッド

ODKAgentBase2 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 *使用すべきでないメソッド* を使用してもエラーにはなりませんが使用しないようにしてください。また、既存のコードを新しいメソッドに移行しておくこともお勧めします。使用すべきでないメソッドは、今後のリリースで除外されることがあります。

ODKAgentBase2 クラスの使用すべきでないメソッドの一覧を下表に示します。新規に ODA を記述する (既存の ODA を修正しない) 場合は、このセクションを無視してかまいません。

表 93. ODKAgentBase2 クラスの使用すべきでないメソッド

使用すべきでないメソッド	代替メソッド
getTreeNodes() (ODKAgentBase から継承された場合)	IGeneratesBoDefs インターフェース内の getTreeNodes()
generateDefs() (ODKAgentBase から継承された場合)	IGeneratesBoDefs インターフェース内の generateBoDefs() (ビジネス・オブジェクト定義を生成するため) **注:** IGeneratesBinFiles インターフェース内で generateBinFiles() メソッドを使用してファイルを生成することもできます。

第 22 章 ODKConstant インターフェース

Object Discovery Agent Development Kit (ODK) API は、ODKConstant インターフェースを使用して、Object Discovery Agent (ODA) に一般定数を渡します。ODKConstant インターフェースを実装するクラスはすべて、定義済み定数に直接アクセスできます。例えば、TreeNode クラスが ODKConstant インターフェースを実装する場合、TreeNode クラスのメソッドが MSG_QUESTION 定数にアクセスできるようにするには、次のようにコーディングします。

```
int message_icon = MSG_QUESTION;
```

ODKConstant インターフェースは、以下のような定数を表す静的メンバー変数を定義します。

- 『ストリング値定数』
- 『ユーザー応答ダイアログ定数』
- 285 ページの『カーディナリティー定数』
- 285 ページの『トレース・レベル定数』
- 285 ページの『メッセージ・タイプ定数』
- 286 ページの『ノード種類定数』
- 286 ページの『コンテンツ・プロトコル定数』
- 287 ページの『コンテンツ・インデックス定数』

ストリング値定数

表 94 は、ODKConstant インターフェースのストリング値定数についての要約です。これらのストリング値定数は、Blank および Ignore という特殊な属性値を表します。ストリング値定数はすべて String 型です。

表 94. ODKConstant インターフェースのストリング値定数

ストリング値定数	説明
CW_EMPTY_STRING	空の String ("") に対応する定義の定数を指定します。
CW_NULL_STRING	null 値に対応する定義の定数を指定します。

ユーザー応答ダイアログ定数

sendMsg() メソッドを ODKUtility クラスに定義しておくと、ODA 開発者にユーザー応答ダイアログ・ボックスを表示するための手段が提供されるようになります。sendMsg() をサポートする目的から、ODKConstant インターフェースには 表 95 に示すユーザー応答ダイアログ定数が用意されています。ユーザー応答ダイアログ定数はすべて、整数 (int) 型です。

表 95. ODKConstant インターフェースのユーザー応答ダイアログ定数

ユーザー応答ダイアログ定数	説明
ダイアログ・ボタン定数	
MSG_OK	ユーザー応答ダイアログ・ボックスの「OK」ボタンの表示を指定します。
MSG_OKCANCEL	ユーザー応答ダイアログ・ボックスの「OK」ボタンおよび「キャンセル」ボタンの表示を指定します。
MSG_RETRYCANCEL	ユーザー応答ダイアログ・ボックスの「再試行」ボタンおよび「キャンセル」ボタンの表示を指定します。
MSG_ABORTRETRYIGNORE	ユーザー応答ダイアログ・ボックスの「再試行」ボタン、「無視」ボタン、および「中止」ボタンの表示を指定します。
MSG_YESNO	ユーザー応答ダイアログ・ボックスの「はい」ボタンおよび「いいえ」ボタンの表示を指定します。
MSG_YESNOCANCEL	ユーザー応答ダイアログ・ボックスの「はい」ボタン、「いいえ」ボタン、および「キャンセル」ボタンの表示を指定します。
ダイアログ・アイコン定数	
MSG_ERROR	ユーザー応答ダイアログ・ボックスにエラー・アイコンを表示するように指定します。
MSG_CRITIALERROR	ユーザー応答ダイアログ・ボックスにクリティカル・エラー・アイコンを表示するように指定します。
MSG_WARNING	ユーザー応答ダイアログ・ボックスに警告アイコンを表示するように指定します。
MSG_INFORMATION	ユーザー応答ダイアログ・ボックスに情報アイコンを表示するように指定します。
MSG_QUESTION	ユーザー応答ダイアログ・ボックスに疑問符アイコンを表示するように指定します。
ユーザー応答定数	
ODK_OK	ユーザー応答ダイアログ・ボックスの「OK」ボタンを指定します。
ODK_CANCEL	ユーザー応答ダイアログ・ボックスの「キャンセル」ボタンを指定します。
ODK_RETRY	ユーザー応答ダイアログ・ボックスの「再試行」ボタンを指定します。
ODK_IGNORE	ユーザー応答ダイアログ・ボックスの「無視」ボタンを指定します。
ODK_ABORT	ユーザー応答ダイアログ・ボックスの「中止」ボタンを指定します。
ODK_YES	ユーザー応答ダイアログ・ボックスの「はい」ボタンを指定します。
ODK_NO	ユーザー応答ダイアログ・ボックスの「いいえ」ボタンを指定します。
ODK_CLOSE	ユーザー応答ダイアログ・ボックスの「閉じる」ボタンを指定します。
ODK_HELP	ユーザー応答ダイアログ・ボックスの「ヘルプ」ボタンを指定します。

カーディナリティー定数

表 96 に、ODKConstant インターフェースのカーディナリティー定数について要約します。これらの定数は、エージェント・プロパティーのカーディナリティーの有効値を表します。カーディナリティー定数はすべて String 型です。

表 96. ODKConstant インターフェースのカーディナリティー定数

カーディナリティー定数	説明
MULTIPLE_CARD	エージェント・プロパティーが値を複数 持つことができるように指定します。この場合、ユーザーはプロパティーの値を複数指定できます。
SINGLE_CARD	エージェント・プロパティーが値を *1* つのみ 持つことができるように指定します。この場合、ユーザーはプロパティーの値を 1 つのみ指定できます。

トレース・レベル定数

表 97 に、ODKConstant インターフェースのトレース・レベル定数について要約します。これらの定数は、トレース・メソッド trace() (ODKUtility クラスに定義済み) 用の有効なトレース・レベルを表しています。トレース・レベル定数はすべて、整数 (int) 型です。

表 97. ODKConstant インターフェースのトレース・レベル定数

トレース・レベル定数	説明
TRACELEVEL0	トレース・レベルが 0 (エラー・ロギングがオン、トレースがオフ) であることを表します。
TRACELEVEL1	トレース・レベルが 1 であることを表します。
TRACELEVEL2	トレース・レベルが 2 であることを表します。
TRACELEVEL3	トレース・レベルが 3 であることを表します。
TRACELEVEL4	トレース・レベルが 4 であることを表します。
TRACELEVEL5	トレース・レベルが 5 であることを表します。

各トレース・レベルで必要なコンテンツについては、 90 ページの表 15 を参照してください。

メッセージ・タイプ定数

表 98 に、ODKConstant インターフェースのメッセージ・タイプ定数について要約します。これらの定数は、trace() メソッド (ODKUtility クラスに定義済み) によって出力されるメッセージの重大度レベルを示しています。メッセージ・タイプ定数はすべて、整数 (int) 型です。

表 98. ODKConstant インターフェースのメッセージ・タイプ定数

メッセージ・タイプ定数	説明
XRD_FATAL	致命的エラーを表します。
XRD_ERROR	エラーを表します。
XRD_URGENTWARNING	緊急警告を表します。
XRD_WARNING	警告を表します。

表 98. ODKConstant インターフェースのメッセージ・タイプ定数 (続き)

メッセージ・タイプ定数	説明
XRD_INFO	通知メッセージを表します。
XRD_TRACE	トレース・メッセージを表します。

重要: ODKConstant インターフェースにも、メッセージ・タイプ定数が用意されています。このメッセージ・タイプ定数は、XRD_INT_*messageType* という形式です。また、未定義のメッセージ・タイプを表す XRD_UNKNOWN 定数も定義されています。これらのメッセージ・タイプ定数は内部でのみ使用されるため、ODA では*使用しないでください*。

ノード種類定数

表 99 に、ODKConstant インターフェースのノード種類定数について要約します。ビジネス・オブジェクト・ウィザードの「ソースの選択」ダイアログ・ボックスにツリー・ノードが表示されたときにそのツリー・ノードに対してユーザーが行える操作は、これらの定数で示されます。ノード種類定数はすべて、整数 (int) 型です。

表 99. ODKConstant インターフェースのノード種類定数

ノード種類定数	説明
NODE_NATURE_NORMAL	ツリー・ノードを「標準」ノードとして指定します。この指定では、ユーザーによるノード展開、またはノード選択 (リーフ・ノードの場合) が可能になります。
NODE_NATURE_FILE	ツリー・ノードをファイルに関連付けることができるように指定します。この指定では、ビジネス・オブジェクト・ウィザードはノード名のコンテキスト・メニューの「関連付けられたファイル」メニュー項目を使用可能にして、ノードに関連付けるファイルをユーザーが位置指定できるようにします。

コンテンツ・プロトコル定数

表 100 に、ODKConstant インターフェースのコンテンツ・プロトコル定数について要約します。これらの定数は、ODA のコンテンツ・プロトコルを表しています。コンテンツ・プロトコル定数はすべて byte 型です。

表 100. ODKConstant インターフェースのコンテンツ・プロトコル定数

コンテンツ・プロトコル定数	説明
CONTENT_PROTOCOL_ONREQUEST	ODA がコンテンツを「要求時に」生成するように指定します。この指定では、ビジネス・オブジェクト・ウィザードが ODA によるコンテンツ生成を明示的に要求します。そのようなコンテンツの準備ができると、ODA ランタイムはビジネス・オブジェクト・ウィザードに通知するため、通知を受けたウィザードは、適宜にコンテンツを検索できます。

表 100. ODKConstant インターフェースのコンテンツ・プロトコル定数 (続き)

コンテンツ・プロトコル定数	説明
CONTENT_PROTOCOL_CALLBACK	ODA がコンテンツを「自発的に」生成するように指定します。この指定では、ODA がいつコンテンツを生成するかについて予想も保証もできません。そのようなコンテンツの準備ができると、ODA は必ずビジネス・オブジェクト・ウィザードに通知するため、通知を受けたウィザードは、適宜にコンテンツを検索できます。

コンテンツ・インデックス定数

表 101 に、ODKConstant インターフェースのコンテンツ・インデックス定数について要約します。この定数は、生成済みのコンテンツをすべて戻すように指示する内容検索メソッドに対する特別な値を表しています。コンテンツ・インデックス定数は long 型です。

表 101. ODKConstant インターフェースのコンテンツ・インデックス定数

コンテンツ・インデックス定数	説明
GET_ALL_OBJECTS	ODA の内容検索メソッドへの引き数として渡されます。GET_ALL_OBJECTS コンテンツ・インデックス定数は、内容検索メソッドが生成済みコンテンツをすべて 戻すように指定する定数です。

詳しくは、 152 ページの『生成済みビジネス・オブジェクト定義へのアクセスの提供』および 162 ページの『生成済みファイルへのアクセスの提供』を参照してください。

第 23 章 ODKException クラス

ODKException クラスは、Object Discovery Agent Development Kit (ODK) API の例外を示す基底クラスです。 ODK API によって、Java Exception クラスは次の名前の独自の例外クラスを作成するように拡張されています。

```
com.crossworlds.ODK.ODKException
```

このクラスは、ODK API のメソッドがスローできる*例外オブジェクト* を表します。

注: 各 ODK API メソッドのリファレンスの説明には、そのメソッドによってスローされる例外がリストされています。

ODKException クラスによって、以下が定義されます。

- 『メソッド』
- 290 ページの『例外サブクラス』

メソッド

表 102 に、ODKException クラスのメソッドについて要約します。

表 102. ODKException クラスのメンバー・メソッド

メンバー・メソッド	説明	ページ
ODKException()	ODK 例外オブジェクトを作成します。	289
getMsg()	例外オブジェクトから例外メッセージを取得します。	289

ODKException()

例外オブジェクトを作成します。

構文

```
public ODKException(String msg);
```

パラメーター

msg 例外オブジェクトの例外メッセージ。

戻り値

新規にインスタンス化された ODKException オブジェクト。

getMsg()

例外オブジェクトから例外メッセージを取得します。

構文

```
public String getMsg();
```

パラメーター

なし。

戻り値

例外メッセージが格納されている String。

例外サブクラス

この ODKException クラス内には、ODK API のメソッド内に存在しうる特別な例外を識別するサブクラスが含まれています。サブクラス化された例外の一覧を表 103 に示します。

表 103. ODKException サブクラス

例外サブクラス	定義
BusObjInvalidAttrException	属性が無効な場合にスローされます。
BusObjInvalidDefException	ビジネス・オブジェクト定義が無効な場合にスローされます。
BusObjInvalidVerbException	動詞が無効な場合にスローされます。
BusObjNoSuchAttrException	ビジネス・オブジェクト定義内に属性が存在しない場合にスローされます。
BusObjNoSuchVerbException	動詞がビジネス・オブジェクト定義によってサポートされていない場合にスローされます。
ODKInvalidNodeException	ツリー・ノードの例外を指示するためにスローされます。
ODKInvalidPropException	例外が発生した原因が無効なプロパティーにあることを指示するためにスローされます。
UnsupportedContentException	要求された生成済みコンテンツを ODA がサポートできない場合にスローされます。

第 24 章 ODKUtility クラス

Object Discovery Agent Development Kit (ODK) API には、Object Discovery Agent (ODA) にさまざまなユーティリティー・メソッドを提供する ODKUtility クラスが用意されています。単一 ODKUtility オブジェクトへのハンドルを取得することによって、ODA が以下の機能性にアクセスできるようになります。

- ODA ランタイムのメモリー内の情報
  - ODA 構成プロパティーに対するユーザー指定の値
  - ビジネス・オブジェクト・プロパティーに対するユーザー指定の値
- 以下の機能を備えたユーティリティー・メソッド
  - ユーザー入力を必要とするメッセージの送信
  - 非ブロッキング・ステータス・メッセージの送信
  - トレースの実行

注: ODKUtility オブジェクトへのハンドルを取得するには、この ODKUtility クラスの getODKUtility() メソッドを使用します。

表 104 に、ODKUtility クラスのメソッドについて要約します。

表 104. ODKUtility クラスのメンバー・メソッド

メンバー・メソッド	説明	ページ
contentComplete()	コールバック・プロトコル使用時に、ODA によるコンテンツ生成が完了したことをビジネス・オブジェクト・ウィザードに通知します。	292
getAgentProperty()	指定された ODA 構成プロパティーを取得します。	293
getAllAgentProperties()	ODA 構成プロパティーをすべて取得します。	293
getAllBOSpecificProperties()	ビジネス・オブジェクト・プロパティーをすべて取得します。	294
getBOSpecificProperty()	指定されたビジネス・オブジェクト・プロパティーを取得します。	294
getBOSpecificProps()	指定されたビジネス・オブジェクト・プロパティーを、ユーザー入力用の「BO プロパティー」ダイアログ・ボックスに送信します。	295
getClientFile()	指定されたファイルを取得するようビジネス・オブジェクト・ウィザードに要求します。	296
getMsg()	ODA メッセージ・ファイルからメッセージを戻します。	298
getODKUtility()	ODKUtility オブジェクトへのハンドルを戻します。	299

表 104. *ODKUtility* クラスのメンバー・メソッド *(続き)*

メンバー・メソッド	説明	ページ
`sendMsg()`	メッセージやボタンが含まれるユーザー応答ダイアログ・ボックスを表示し、ユーザーに応答を要求します。	299
`sendStatusMsg()`	ユーザーにメッセージを表示します。	301
`trace()`	トレース・ファイルにメッセージを書き込みます。	302

contentComplete()

ODA がコールバック・コンテンツ・プロトコル用コンテンツの生成を完了したことを、ビジネス・オブジェクト・ウィザードに通知します。

構文

```
public void contentComplete(ContentMetaData contentMetaData);
```

パラメーター

contentMetaData
コンテンツ・メタデータ・オブジェクト (生成済みコンテンツの現在の状態についての記述内容)です。

戻り値

なし。

注記

`contentComplete()` メソッドは、ODA によるコンテンツ生成の完了を示します。コールバック・プロトコルの場合、適切なコンテンツ生成メソッドの呼び出しによるコンテンツ生成は、ビジネス・オブジェクト・ウィザードでは*開始されません*。代わりに ODA がコンテンツ生成を開始し、ビジネス・オブジェクト・ウィザードは ODA からのコンテンツ生成完了の通知を待ちます。この通知を実行するために、ODA は `contentComplete()` を呼び出します。ビジネス・オブジェクト・ウィザードは通知を受けた後、適切な内容検索メソッドを呼び出して、生成されたコンテンツを取得します。

重要

ODA は特定のコンテンツを生成する目的にコールバック・プロトコルを使用する際に、`contentComplete()` メソッドを呼び出して、コンテンツが使える状態にあることをビジネス・オブジェクト・ウィザードに*通知する必要があります*。そうしない限り、生成済みコンテンツが取得可能な状態にあることがビジネス・オブジェクト・ウィザードに認知されません。

生成済みコンテンツのタイプ、および生成済みコンテンツ内の項目数は、*contentMetaData* オブジェクトに示されているはずです。

getAgentProperty()

指定された ODA 構成プロパティーを取得します。

構文

```
public AgentProperty getAgentProperty(String propName);
```

パラメーター

propName 取得する構成プロパティーの名前。

戻り値

指定された構成プロパティー、または null (その名前の構成プロパティーが存在しない場合) を包含する AgentProperty オブジェクト。

注記

getAgentProperty() メソッドは、*propName* 構成プロパティーを ODA ランタイム・メモリーから取得します。「エージェントの構成」ダイアログ・ボックスでユーザーが構成プロパティーの値を指定した後、その構成プロパティーをビジネス・オブジェクト・ウィザードが ODA ランタイム・メモリーに読み込みます。このメソッドは、指定された構成プロパティーを AgentProperty オブジェクトとして戻します。プロパティーに関する情報を取得するには、オブジェクトのメンバー変数にアクセスします。

参照

getAgentProperties(), getAllAgentProperties()

getAllAgentProperties()

ODA 構成プロパティーをすべて取得します。

構文

```
public Hashtable getAllAgentProperties();
```

パラメーター

なし。

戻り値

AgentProperty オブジェクトとして表される ODA 構成プロパティーを包含する java.util.Hashtable オブジェクトへの参照。プロパティー名にキーが設定されています。

注記

getAllAgentProperties() メソッドは、ODA ランタイム・メモリーから ODA 構成プロパティーをすべて取得します。「エージェントの構成」ダイアログ・ボック

スでユーザーが構成プロパティーの値を指定した後、その構成プロパティーをビジネス・オブジェクト・ウィザードが ODA ランタイム・メモリーに読み込みます。このメソッドは、キーを値にマップする Hashtable オブジェクトとして、構成プロパティーを取得します。キーはプロパティーの名前であり、値は関連付けられたプロパティーの値です。この構造体から情報を取得するには、Hashtable クラスのメソッド (例えば、keys() や elements()) を使用してください。

参照

getAgentProperties(), getAgentProperty()

---

getAllBOSpecificProperties()

「BO プロパティー」ダイアログ・ボックスからビジネス・オブジェクト・プロパティーをすべて取得します。

構文

```
public Hashtable getAllBOSpecificProperties();
```

パラメーター

なし。

戻り値

AgentProperty オブジェクトとして表されるビジネス・オブジェクト・プロパティーを包含する java.util.Hashtable オブジェクトへの参照。プロパティー名にキーが設定されています。

注記

getAllBOSpecificProperties() メソッドは、ODA ランタイム・メモリーからビジネス・オブジェクト・プロパティーをすべて取得します。「BO プロパティー」ダイアログ・ボックス (ステップ 5 の一部) でユーザーがプロパティー値を指定した後、これらのプロパティーをビジネス・オブジェクト・ウィザードがメモリーに保存します。このメソッドは、キーを値にマップする Hashtable オブジェクトとして、ビジネス・オブジェクト・プロパティーを戻します。キーはビジネス・オブジェクト・プロパティーの名前であり、値は関連付けられたプロパティーの値です。この構造体から情報を取得するには、Hashtable クラスのメソッド (例えば、keys() や elements()) を使用してください。

参照

getBOSpecificProperty(), getBOSpecificProps()

---

getBOSpecificProperty()

指定されたビジネス・オブジェクト・プロパティーを取得します。

構文

```
public AgentProperty getBOSpecificProperty(String propName);
```

パラメーター

propName 取得されるビジネス・オブジェクト・プロパティーの名前。

戻り値

指定されたビジネス・オブジェクト・プロパティー、または null (その名前のビジネス・オブジェクト・プロパティーが存在しない場合) を包含する AgentProperty オブジェクト。

注記

getBOSpecificProperty() メソッドは、*propName* ビジネス・オブジェクト・プロパティーを ODA ランタイム・メモリーから取得します。「BO プロパティー」ダイアログ・ボックス (ステップ 5 の一部) でユーザーがプロパティー値を指定した後、これらのプロパティーをビジネス・オブジェクト・ウィザードがメモリーに保存します。このメソッドは、指定されたビジネス・オブジェクト・プロパティーを AgentProperty オブジェクトとして戻します。プロパティーに関する情報を取得するには、オブジェクトのメンバー変数にアクセスします。

参照

getAllBOSpecificProperties(), getBOSpecificProps()

---

getBOSpecificProps()

指定されたビジネス・オブジェクト・プロパティーを、ユーザー入力用の「BO プロパティー」ダイアログ・ボックスに送信します。

構文

```
public Hashtable getBOSpecificProps(AgentProperty[] properties,
   String titleBarText);
public Hashtable getBOSpecificProps(AgentProperty[] properties,
   String titleBarText, String propGridText);
```

パラメーター

properties ビジネス・オブジェクト・プロパティー (AgentProperty オブジェクト内にある各プロパティー) の配列。

titleBarText 「BO プロパティー」ダイアログ・ボックスのタイトル・バーに表示するテキスト。

propGridText 「BO プロパティー」ダイアログ・ボックスのプロパティー・グリッド上部のテキスト領域に表示するテキスト。

戻り値

プロパティー名に対してキー入力された (AgentProperty オブジェクトとしての) ビジネス・オブジェクト・プロパティーの、Java Hashtable オブジェクト。

例外

ODKInvalidPropException
プロパティーが無効 (例えば、名前がないプロパティーなど) の場合にスローされます。

XMLException プロパティーの XML 変換が失敗した場合にスローされます。

注記

getBOSpecificProps() メソッドは、ビジネス・オブジェクト・プロパティーの *properties* 配列をビジネス・オブジェクト・ウィザードに送信し、このウィザードによりこの配列が「BO プロパティー」ダイアログ・ボックスで表示されます。このダイアログ・ボックスから、これらのプロパティー値の入力や変更ができます。getBOSpecificProps() メソッドを呼び出す前に、ビジネス・オブジェクト・プロパティーごとにエージェント・プロパティー (AgentProperty) オブジェクトのインスタンス化および初期化を行い、*properties* ビジネス・オブジェクト・プロパティー配列の中に各プロパティーを格納しておく必要があります。このビジネス・オブジェクト・プロパティー配列は、getBOSpecificProps() メソッドによって呼び出し元のビジネス・オブジェクト・ウィザードに渡されます。

「BO プロパティー」ダイアログ・ボックスからビジネス・オブジェクト・プロパティーを設定し終えると、これらのユーザー指定のプロパティーを、ビジネス・オブジェクト・ウィザードが java.util.Hashtable オブジェクトおよび ODA ランタイム・メモリー内に保存します。ODA の内部では以下のいずれかの方法で、ユーザーにより初期化されたプロパティーを取得できます。

- ODA ランタイム・メモリーから

  ODKUtility クラスの getBOSpecificProperty() メソッドまたは getAllBOSpecificProperties() メソッドを使用します。ユーザーにより初期化されたプロパティー値は、エージェント・プロパティー (AgentProperty) オブジェクトの allValues メンバー変数の中に代入されます。

- getBOSpecificProps() により戻される Hashtable オブジェクトから

  エージェント・プロパティーを取得するには、Hashtable オブジェクトのメソッドを使用します。

getBOSpecificProps() の使用方法について詳しくは、138 ページの『ビジネス・オブジェクト・プロパティーの要求』を参照してください。

参照

getAllBOSpecificProperties(), getBOSpecificProperty()

---

getClientFile()

指定されたファイルを取得するようビジネス・オブジェクト・ウィザードに要求します。

構文

```
public byte[] getClientFile(String srcNodePath, ODKAgentBase2 ODAobj);
```

パラメーター

srcNodePath ビジネス・オブジェクト・ウィザードから要求されるファイルのソース・ノード・パス。

ODAobj ODA が操作の実行 (ファイル・コンテンツの生成) を許可されているかどうかの確認に使用される ODA (ODKAgentBase2) オブジェクト。

戻り値

指定されたオペレーティング・システム・ファイルのコンテンツ (バイト配列)。

例外

UnsupportedContentException
ODA がファイル・コンテンツの生成をサポートしていない (つまり、IGeneratesBinFiles インターフェースを実装しない) 場合にスローされます。

Java.io.IOException
ファイル取得中に、ファイルが見つからないなどのエラーが発生した場合にスローされます。

注記

getClientFile() メソッドは、*srcNodePath* に識別されるオペレーティング・システム・ファイルのコンテンツを戻すようビジネス・オブジェクト・ウィザードに要求します。この *srcNodePath* パスは、以下の形式をとります。

fileNodePath:*fileLocation*

ここで:

- *fileNodePath* は、ファイルに関連付けられているノードのソース・ノード名をコロン (:) で区切ったリストです (例: Apollo:Vulso:Flavius.xml)。
- *fileLocation* は、ファイルへの完全オペレーティング・システム・パスです (例: C:¥temp¥XMLFiles¥Flavius.xml)。

ソース・ノードで表されるオブジェクトの関連ファイルにアクセスするには、getClientFile() メソッドを使用します。ソース・ノードにファイルを関連付けることができる場合、ファイルのソース・ノード・パスを解釈し、このファイルのコンテンツを読み取れなければなりません。このことは以下の*両方* の点で必要です。

- ソース・ノード生成中に、getTreeNodes() メソッドが、ファイル内の下位ノードを「検出」できなければならないという点。
- コンテンツ生成中に、コンテンツを生成するメソッドが、ファイル内のノードに格納された情報にアクセスできなければならないという点。

詳しくは、 155 ページの『ソース・データ用ファイルの読み取り』を参照してください。

getMsg()

ODA 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 getMsg(int msgNum, int msgType);
public String getMsg(int msgNum, int msgType, msgParameters);
public String getMsg(int msgNum, int msgType, Vector paramArray);
```

パラメーター

msgNum メッセージ・ファイルからのメッセージ番号を指定します。

msgParameters 最大 3 つの String パラメーター値のオプショナル・リストです。各パラメーターはメッセージ・リスト内のパラメーターに対応しています。

msgType 以下のいずれかのメッセージ・タイプ定数として指定されるメッセージのタイプ。

```
ODKConstant.XRD_FATAL
ODKConstant.XRD_ERROR
ODKConstant.XRD_URGENTWARNING
ODKConstant.XRD_WARNING
ODKConstant.XRD_INFO
```

paramArray メッセージのパラメーターに挿入される、Java Vector としてのパラメーターのオプショナル・リスト。

例外

IllegalArgumentException
msgType 引き数が有効でない場合にスローされます。

戻り値

指定されたメッセージ番号に関連付けられたテキストが格納されている String。メッセージ・パラメーターが指定された場合、そのメッセージ・パラメーターの値はメッセージ内に適宜挿入されます。*msgNum* が有効でない場合、メソッドは null を戻します。

注記

getMsg() メソッドはメッセージをメッセージ・ファイルから取得します。getMsg() メソッドは、MessageFile 始動プロパティーからこのファイルの名前を識別します。この MessageFile 始動プロパティーは、ODK が自動的に ODA 始動プロパティーと一緒にインクルードします。 getMsg() メソッドには、以下の形式があります。

- 最初の形式を使用すると、指定されたメッセージ番号 (*msgNum*) のメッセージが ODA メッセージ・ファイルから取得されます。
- 2 番目の形式を使用した場合も、指定されたメッセージ番号 (*msgNum*) のメッセージが ODA メッセージ・ファイルから取得されます。この形式では、メッセージを取得する前に、メッセージに挿入する String メッセージ・パラメーター (*msgParameters*) を最大 3 つまで指定することもできます。

- 3 番目の形式を使用した場合も、ODA メッセージ・ファイルからメッセージが送信されてメッセージ・パラメーターが与えられます。ただしこの形式では、メッセージ・パラメーターが Java `Vector` *paramArray* 内の要素として送信されます。

ODA メッセージ・ファイルについては、179 ページの『メッセージ・ファイル』を参照してください。メッセージ・パラメーターについては、181 ページの『パラメーター値の使用』を参照してください。

参照

trace()

getODKUtility()

単一の `ODKUtility` オブジェクトへのハンドルを戻します。

構文

```
public static ODKUtility getODKUtility();
```

パラメーター

なし。

戻り値

`ODKUtility` オブジェクトへのハンドル。

注記

`getODKUtility()` メソッドを ODA コード内に使用すると、`ODKUtility` クラスのユーティリティーへのアクセスが可能になります。`ODKUtility` メソッドにアクセスする*前*に、`getODKUtility()` を使用してこのクラスの単一オブジェクトへのハンドルを取得しておく必要があります。

注: `getODKUtility()` への呼び出しは多くの場合、`getAgentProperties()` メソッドで実行されます。詳しくは、118 ページの『ODKUtility オブジェクトへのハンドルの取得』を参照してください。

参照

getAgentProperties()

sendMsg()

メッセージが含まれるユーザー応答ダイアログ・ボックスを表示し、ユーザーに応答を要求します。

構文

```
public int sendMsg(String msg, int dialogFlags);
```

パラメーター

msg ユーザー応答ダイアログ・ボックスに表示するメッセージ。

dialogFlags ユーザー応答ダイアログ・ボックスの一部として表示するボタンおよびアイコンを示す一連のフラグ。これらのボタンおよびアイコンは、表 105 に示すユーザー応答ダイアログ定数のマスクとして指定してください。

戻り値

ユーザー応答ダイアログ・ボックスを終了するためにユーザーがクリックしたボタンを示す整数。この整数値を以下のユーザー応答定数と比較してください。

`ODKConstant.ODK_OK`
ユーザーが「OK」ボタンを選択しました。

`ODKConstant.ODK_CANCEL`
ユーザーが「キャンセル」ボタンを選択しました。

`ODKConstant.ODK_RETRY`
ユーザーが「再試行」ボタンを選択しました。

`ODKConstant.ODK_IGNORE`
ユーザーが「無視」ボタンを選択しました。

`ODKConstant.ODK_ABORT`
ユーザーが「中止」ボタンを選択しました。

`ODKConstant.ODK_YES`
ユーザーが「はい」ボタンを選択しました。

`ODKConstant.ODK_NO`
ユーザーが「いいえ」ボタンを選択しました。

`ODKConstant.ODK_CLOSE`
ユーザーが「閉じる」ボタンを選択しました。

`ODKConstant.ODK_HELP`
ユーザーが「ヘルプ」ボタンを選択しました。

注記

`sendMsg()` メソッドは、ユーザー応答ダイアログ・ボックスがユーザーに表示されるよう、要求をビジネス・オブジェクト・ウィザードに送信します。このユーザー応答ダイアログ・ボックスの以下のコンポーネントを指定してください。

- *msg* ストリングには、ユーザーに対して表示する必要のある条件、質問、または情報を示すテキストを代入します。
- *dialogFlags* マスクには、ユーザー応答ダイアログ・ボックスの外観を表す以下の機能を代入します。
  - 表示するボタン

    これらのボタンの 1 つをユーザーがクリックすると、ユーザー応答ダイアログ・ボックスが終了します。これらのボタンを指定するには、表 105 の『表示するボタン』のセクションにあるダイアログ・ボタン定数を使用します。
  - 表示するアイコン

アイコンにより表示するユーザー応答ダイアログ・ボックスのタイプが判別されます。ダイアログ・ボックスのタイプを指定するには、表 105 の『表示するダイアログ・ボックス・アイコン』のセクションにあるダイアログ・アイコン定数の 1 つを使用します。

表 105. ユーザー応答ダイアログ・ボックスの外観の表示

ユーザー応答ダイアログ・ボックスの外観	ODKConstant ユーザー応答ダイアログ定数
表示するボタン:	
OK	MSG_OK
OK、キャンセル	MSG_OKCANCEL
再試行、キャンセル	MSG_RETRYCANCEL
再試行、無視、中止	MSG_ABORTRETRYIGNORE
はい、いいえ	MSG_YESNO
はい、いいえ、キャンセル	MSG_YESNOCANCEL
表示するダイアログ・ボックス・アイコン:	
エラー・アイコン	MSG_ERROR
クリティカル・エラー・アイコン	MSG_CRITICALERROR
警告アイコン	MSG_WARNING
情報アイコン	MSG_INFORMATION
疑問符 (?) アイコン	MSG_QUESTION

注: 表 105 のユーザー応答ダイアログ定数はすべて ODKConstant インターフェースに定義されています。

dialogFlags 引き数を指定するには、ユーザー応答ダイアログ・ボックスの外観の説明となるユーザー応答ダイアログ定数のマスクを作成します。例えば、以下の sendMsg() への呼び出しによって、「再試行」ボタンや「キャンセル」ボタンだけでなくエラー・アイコンも表示するユーザー応答ダイアログ・ボックスが作成されます。

```
String msg = new String(bdkUtil.getMsg(1002, ODKConstant.XRD_ERROR, params));
bdkUtil.sendMsg(msg, ODKConstant.MSG_RETRYCANCEL | ODKConstant.MSG_ERROR);
```

参照

sendStatusMsg()

sendStatusMsg()

ユーザーにメッセージを表示します。

構文

```
public void sendStatusMsg(String msg);
```

パラメーター

msg ユーザーに送信されるメッセージ。

戻り値

なし。

参照

sendMsg()

---

trace()

トレース・ファイルにメッセージを書き込みます。

構文

```
public void trace(int level, int msgType, String message);

public void trace(int level, int msgNum, int msgType);
public void trace(int level, int msgNum, int msgType, msgParameters);
public void trace(int level, int msgNum, int msgType, Vector paramArray);

public void trace(int level, int msgType, BusObjDef boDef);

public void trace(int level, int msgType, AgentProperty[] properties,
   String foreword);
```

パラメーター

boDef トレース・ファイルに書き込まれるビジネス・オブジェクト定義。

foreword メッセージを理解しやすくするために *properties* プロパティー配列の前に配置される String (例えば、「These are the properties for the Object Discovery Agent」など)。

level 以下のいずれかのトレース・レベル定数として指定されるトレース・レベル。

```
ODKConstant.TRACELEVEL0
ODKConstant.TRACELEVEL1
ODKConstant.TRACELEVEL2
ODKConstant.TRACELEVEL3
ODKConstant.TRACELEVEL4
ODKConstant.TRACELEVEL5
```

message トレース・ファイルに書き込まれる String メッセージ。

msgNum メッセージ・ファイル内のメッセージ番号を指定します。

msgParameters 最大 3 つの String パラメーター値のオプショナル・リストです。各パラメーターはメッセージ・リスト内のパラメーターに対応しています。

msgType 以下のいずれかのメッセージ・タイプ定数として指定されるメッセージのタイプ。

```
ODKConstant.XRD_FATAL
ODKConstant.XRD_ERROR
ODKConstant.XRD_URGENTWARNING
ODKConstant.XRD_WARNING
ODKConstant.XRD_INFO
ODKConstant.XRD_TRACE
```

paramArray メッセージに挿入されるパラメーターのベクトル。

properties トレース・ファイルに書き込まれるエージェント・プロパティー (AgentProperty) オブジェクトの配列。

戻り値

なし。

例外

IllegalArgumentException

properties 引き数が `null` の場合、または *msgType* 引き数が無効な場合にスローされます。

注記

トレース・レベル がシステム・トレース・レベル以下である場合、`trace()` メソッドは指定された情報をトレース・ファイルに送信します。システム・トレース・レベルは、`TraceLevel` 構成プロパティーを介して設定されます。この構成プロパティーはビジネス・オブジェクト・ウィザードによって自動的に ODA 構成プロパティーに組み込まれます。トレース・レベル がゼロ (0) の場合、エラー・ロギングがアクティブ化され、`trace()` によってエラー・メッセージがトレース・ファイルに送信されます。トレース・レベルがゼロ以外 (表 106 を参照) の場合、トレースがアクティブ化され、`trace()` によってトレース・メッセージがトレース・ファイルに送信されます。

表 106. ODA のトレース・レベル

トレース・レベル	説明	トレース・レベル定数
0	エラー・メッセージをログに記録します。	`TRACELEVEL0`
1	メソッドが開始されるたびにトレースをとります。通常は、ビジネス・オブジェクト定義ごとに状況メッセージおよびキー情報が提供されます。	`TRACELEVEL1`
2	エージェント・プロパティーおよび受け取った値のトレースをとります。	`TRACELEVEL2`
3	ビジネス・オブジェクト定義の名前をトレースします。通常は、ビジネス・オブジェクト・プロパティーおよび受け取った値が提供されます。	`TRACELEVEL3`
4	メソッドが開始および終了するたびにメッセージのトレースをとります。すべてのスレッドの作成に関する記録がとられます。	`TRACELEVEL4`
5	ODA の初期化を指示します。取得されたすべてのエージェント・プロパティーの値、ODA が生成した各スレッドの詳細状況、およびビジネス・オブジェクト定義のダンプが提供されます。	`TRACELEVEL5`

ODA のトレースの出力先の名前は、ユーザーが `TraceFileName` 構成プロパティーを介して設定してください。設定された構成プロパティーは ODK によって自動的に ODA 始動プロパティーに組み込まれます。その結果として、(初期化済み始動プロパティーを受け取る) `init()` メソッドの開始後に初めてトレースが開始されます。

trace() メソッドには、以下の形式があります。

- 最初の 4 つの形式を使用すると、テキスト・メッセージがトレース・ファイルに送信されます。
  - 最初の形式では、指定されたテキスト *message* がトレース・ファイルに送信されます。
  - 2 番目の形式を使用すると、指定されたメッセージ番号 (*msgNum*) のメッセージが ODA メッセージ・ファイルから送信されます。
  - 3 番目の形式を使用した場合も、指定されたメッセージ番号 (*msgNum*) のメッセージが ODA メッセージ・ファイルから送信されます。この形式では、メッセージをトレース先に送信する前に、メッセージに挿入する String メッセージ・パラメーター (*msgParameters*) を最大 3 つまで送信することもできます。
  - 4 番目の形式を使用した場合も、ODA メッセージ・ファイルからメッセージが送信されてメッセージ・パラメーターが与えられます。ただしこの形式では、メッセージ・パラメーターが Java Vector *paramArray* 内の要素として送信されます。

  ODA メッセージ・ファイルについては、 179 ページの『メッセージ・ファイル』を参照してください。メッセージ・パラメーターについては、 181 ページの『パラメーター値の使用』を参照してください。
- 5 番目の形式を使用すると、ビジネス・オブジェクト定義のダンプがトレース・ファイルに送信されます。このダンプは repos_copy ユーティリティーの形式でフォーマットされます。基本フォーマットは以下のとおりです。

```
[BusinessObjectDefinition]
Name=busObjName
AppSpecificInfo=business-object-level application-specific information
[Attribute]
Name=attribute1
Type=attribute type
Cardinality=n or 1
AppSpecificInfo=attribute-level application-specific information
other attribute properties
[End]
...
```

- 6 番目の形式を使用すると、指定されたエージェント *properties* のダンプがトレース・ファイルに送信されます。 *forward* 引き数によって、メッセージをわかりやすくするための前置きのテキストが与えられます。

参照

getMsg()

使用すべきでないメソッド

ODKUtility 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*使用すべきでないメソッド* を使用してもエラーにはなりませんが使用しないようにしてください。また、既存のコードを新しいメソッドに移行しておくこともお勧めします。使用すべきでないメソッドは、今後のリリースで除外されることがあります。

ODKUtility クラスの使用すべきでないメソッドを表 107 に示します。新規に ODA を記述する (既存の ODA を修正しない) 場合は、このセクションを無視してかま

いません。

表 107. ODKUtility クラスの使用すべきでないメソッド

使用すべきでないメソッド	代替メソッド
フィルター操作をサポートしているすべてのメソッド • `filterData()` • `getFilter()` • `setFilter()`	ODA は依然としてユーザー・レベルのフィルター操作をサポートしています。ただし、プログラム・レベルではサポートしていません。詳しくは、92 ページの『フィルターの使用』 を参照してください。プログラム・レベルでは、検索パターン機能を使用することにより、特定の親ノードに対応するの下位ノードの数の絞り込みができます。詳しくは、132 ページの『検索パターン機能の実装』を参照してください。

第 25 章 TreeNode クラス

Object Discovery Agent Development Kit (ODK) API には、ツリー・ノードを表す TreeNode クラスが用意されています。 Object Discovery Agent (ODA) では、ビジネス・オブジェクト・ウィザードがソース・ノードの階層をユーザーに表示するためのツリー・ノードの配列が生成されます。ユーザーは、このソース・ノード階層のノード間をナビゲートして、ODA により生成されるビジネス・オブジェクト定義を持つオブジェクトを選択できます。

TreeNode クラスによって、以下が定義されます。

- 『メンバー変数』
- 310 ページの『メソッド』

TreeNode クラスは ODKConstant インターフェースを実装します。そのため、ODKConstant に定義されている定数はすべて、TreeNode オブジェクトに使用できます。ODKConstant インターフェースにより定義される定数の一覧については、 283 ページの『第 22 章 ODKConstant インターフェース』を参照してください。

メンバー変数

表 108 は、TreeNode クラスのメンバー変数についての要約です。

表 108. TreeNode クラスのメンバー変数

メンバー変数	説明	ページ
description	ツリー・ノードの説明が含まれています。	307
isExpandable	ツリー・ノードを展開可能に (現在のレベルより下に要素が配置されるように) するかどうかを指定します。	308
isGeneratable	ツリー・ノードが生成可能であること、すなわちノードがビジネス・オブジェクト定義に変換可能であることを指定します。	308
name	ツリー・ノードの名前が含まれています。	308
nodes	展開されたツリー・ノードの階層が含まれています。	309
polymorphicNature	ノードの種類、つまりそのノードが「標準」(展開可能またはリーフ) であるか「ファイル」であるかを定義します。	309

description

ツリー・ノードの説明が含まれています。

タイプ

```
public String description
```

注記

description メンバー変数は、「ソースの選択」ダイアログ・ボックスの「説明」列に表示されます。

isExpandable

ツリー・ノードを展開可能に (現在のレベルより下にノードが配置されるように) するかどうかを指定します。

タイプ

```
public boolean isExpandable
```

注記

表 109 に示すように、isExpandable メンバー変数は、ノードが展開可能であるかどうかを示します。

表 109. ノードのタイプ

ノードのタイプ	説明	isExpandable の値
展開可能 リーフ (終端)	下位ノードを持つノード。 下位ノードを持たないノードですが、ソース・ノード階層のブランチの終端ポイントとなっています。	true false

isExpandable を true に設定できるのは、標準の種類のノード (polymorphicNature メンバー変数が NODE_NATURE_NORMAL に設定されたノード) のみです。

isGeneratable

ツリー・ノードを生成可能にするかどうか (ODA がコンテンツを生成するノードとしてこのノードを選択できるようにするかどうか) を指定します。

タイプ

```
public boolean isGeneratable
```

name

ツリー・ノードの名前が含まれています。

タイプ

```
public String name
```

注記

name メンバー変数は、「ソースの選択」ダイアログ・ボックスの「名前」列に表示されます。

nodes

展開された子ツリー・ノードの階層が含まれています。

タイプ

```
public TreeNode[] nodes
```

注記

nodes メンバー変数には TreeNode オブジェクト (この親ノードの子ごとに 1 つのオブジェクト) の配列が含まれています。下位ノードには、さらにその下位ノード (この親ノードの孫) を含めることができます。このメンバー変数が使用されるのは、ノードが展開可能である (リーフではない) 場合、つまり isExpandable メンバー変数が true のときのみです。

polymorphicNature

ツリー・ノードに対してユーザーが行える有効な操作を示します。

タイプ

```
public int polymorphicNature
```

注記

ビジネス・オブジェクト・ウィザードの「ソースの選択」ダイアログ・ボックスにノードが表示されたときにそのノードに対してユーザーが実行できる操作は、polymorphicNature メンバー変数で示されます。この変数には、ツリー・ノードの種類を示す、整数のノード種類定数が含まれています。 表 110 に示すように、これらのノード種類定数は ODKConstant インターフェースに定義されています。

表 110. ツリー・ノードの種類

ツリー・ノードの種類	説明	ノード種類定数
標準	ユーザーは以下のいずれの操作も行うことができます。 • ノードがリーフ (終端) ノードである場合、そのノードを選択することができます。リーフ・ノードに限り、コンテンツ内に生成するノードとして選択できます。 • ノードを展開すれば、さらにその他のノードも表示することができます。ビジネス・オブジェクト・ウィザードの展開可能ノードの名前の左側には、プラス記号 (+) が表示されます。	NODE_NATURE_NORMAL

表 110. ツリー・ノードの種類 (続き)

ツリー・ノードの種類	説明	ノード種類定数
ファイル	ユーザーがローカル・ファイル・システムからのファイルをノードに関連付けることができます。ノード名を右クリックしたときに表示されるコンテキスト・メニューの中のメニュー項目「関連付けられたファイル」は、ビジネス・オブジェクト・ウィザードによってアクティブ化されます。このメニュー項目を選択すると、システム・ファイルを参照するためのウィンドウが開きます。このウィンドウから、ノードに関連付けるファイルを選択することができます。 ファイル・ノード種類を持つツリー・ノードの場合、ODA は `getClientFile()` メソッド (`ODKUtility` クラスに定義済み) を使用して、ユーザーにより選択されたファイルの内容を取得できます。	`NODE_NATURE_FILE`

注: `TreeNode` クラスは `ODKConstant` インターフェースを実装するため、ノード種類定数を `ODKConstant` 名で修飾しなくても `polymorphicNature` メンバー変数に使えるようになります。

ノード種類について詳しくは、 134 ページの『ツリー・ノードの組み立て』を参照してください。

メソッド

表 111 は、`TreeNode` クラスのメソッドについての要約です。

表 111. TreeNode クラスのメンバー・メソッド

メンバー・メソッド	説明	ページ
`TreeNode()`	ツリー・ノード・オブジェクトを作成します。	307

TreeNode()

ツリー・ノード・オブジェクトを作成します。

構文

```
public TreeNode(String name, String desc, boolean isGen, boolean isExp);

public TreeNode(String name, String desc, boolean isGen, boolean isExp,
   TreeNode[] treeNodes);

public TreeNode(String name, String desc, boolean isGen, boolean isExp,
   TreeNode[] treeNodes, int nodeNature);
```

パラメーター

desc ノードの説明を指定します。description メンバー変数 (307 ページの『description』) は、このパラメーターの値で初期化されます。

isGen ノードが「生成可能」(つまり、ノードをビジネス・オブジェクト定義に変換可能) かどうかを指定します。isGeneratable メンバー変数 (308 ページの『isGeneratable』) は、このパラメーターの値で初期化されます。

isExp ノードが展開可能 (つまりノードがリーフ) かどうかを指定します。isExpandable メンバー変数 (308 ページの『isExpandable』) は、このパラメーターの値で初期化されます。

name ノードの名前を指定します。name メンバー変数 (308 ページの『name』) は、このパラメーターの値で初期化されます。

nodeNature ノードの種類を以下のノード種類定数のいずれかとして示します。

```
ODKConstant.NODE_NATURE_FILE
ODKConstant.NODE_NATURE_NORMAL
```

treeNodes 完全に展開されたノードの階層を指定します。nodes メンバー変数 (309 ページの『nodes』) は、このパラメーターの値で初期化されます。

戻り値

新規にインスタンス化された TreeNode オブジェクト。

注記

ツリー・ノードをインスタンス化する TreeNode() メソッドには、以下の形式があります。

- 最初の形式のコンストラクターでは、ツリー・ノードの名前および説明のほか、ツリー・ノードを生成可能にするかどうか、または展開可能にするかどうかを指定できます。この形式では、下位ノードの配列 (nodes メンバー変数) は、null に初期化され、ノード種類 (polymorphicNature メンバー変数) は「標準」に初期化されます。リーフ・ノードを初期化する場合は、この形式を使用してください。
- 2 番目の形式のコンストラクターを使用すると、(最初の形式で指定される値に加え) 下位ノードの配列の指定も可能になります。この形式を使用した場合、ノード種類が「標準」に初期化されます。展開可能なノードを初期化する場合は、この形式を使用してください。
- 3 番目の形式のコンストラクターを使用すると、(最初および 2 番目の形式で指定される値に加え) ノード種類の指定も可能になります。ファイル種類のノードを初期化する場合は、この形式を使用してください。

詳しくは、 134 ページの『ツリー・ノードの組み立て』を参照してください。

第 4 部 付録

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。
IBM WebSphere InterChange Server Express v4.3、IBM WebSphere Business Integration Toolset v4.3、IBM WebSphere Business Integration Adapters v2.4、IBM WebSphere Business Integration Collaborations v4.2.

IBM WebSphere InterChange Server Express v4.3、IBM WebSphere Business Integration Toolset Toolset v4.3

WebSphere Business Integration Adapter Framework v2.4.0

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]

[ハ行]

[マ行]

[ヤ行]

[ラ行]

A

B

C

D

E

F

G

I

J

L

M

N

O

P

R

S

T

U

V

X