Sun Java™ System

Identity Manager 7.1
配備ツール

Sun Microsystems, Inc.
4150 Network Circle
Santa Clara, CA 95054
U.S.A.

Part No: 820-2530

目 次

図目次

表目次

はじめに

本書『Sun Java™ System Identity Manager 配備ツール』では、さまざまな Identity Manager 配備ツールを使用するのに役立つ参照情報および手順に関する情報を提供します。

対象読者

『Sun Java™ System Identity Manager 配備ツール』は、製品配備のさまざまな段階で Identity Manager を顧客インストール用にカスタマイズするのに必要なワークフロー、画面、規則、システム設定、およびその他の設定ファイルを作成および更新するデプロイヤおよび管理者に向けて作成されました。

デプロイヤは、プログラミングに関する予備知識があり、XML、Java、Emacs や IDE (Eclipse または NetBeans など) に精通していることが望まれます。

管理者は、プログラミングの予備知識がなくてもかまいませんが、LDAP、Active Directory、または SQL など 1 つ以上のリソースドメインに非常に精通していることが望まれます。

内容の紹介

『Identity Manager 配備ツール』は、次の章で構成されています。

- 第 1 章「Identity Manager IDE の使用」- Identity Manager Integrated Development Environment (Identity Manager IDE) を紹介し、このアプリケーションをインストールする手順を説明します。
- 第 2 章「規則の操作」- 一般に XML または JavaScript で作成される、XPRESS ラッパー内の関数について説明します。規則は、頻繁に使用される XPRESS ロジックや、フォーム、ワークフロー、およびロール内で手軽に再利用できる、静的な変数を格納するためのメカニズムを提供します。
- 第 3 章「変数名前空間の操作」- 一般的な Identity Manager のタスクやプロセスの概要、その一般的な使用法、および実行環境となる名前空間について説明します。
- 第 4 章「アダプタの開発」- 会社や顧客に合わせて調整したカスタム Identity Manager リソースアダプタを作成する方法を説明します。
- 第 5 章「ファイアウォールまたはプロキシサーバーの操作」- ファイアウォールやプロキシサーバーを導入している場合の、Identity Manager での URL の使われ方および正確な URL データを入手する方法について説明します。
- 第 6 章「辞書サポートの設定」- Identity Manager の辞書サポートを構成および設定する方法を説明します。
- 第 7 章「Identity Manager Web サービスでの SPML 1.0 の使用」- Identity Manager サーバーが提供する SOAP ベースの Web サービスインタフェースの使用に関する詳細を説明します。要求メッセージの形式設定と応答メッセージの解析に使用する SPML 1.0 クラスについて説明します。
- 第 8 章「Identity Manager Web サービスでの SPML 2.0 の使用」- Identity Manager サーバーが提供する SOAP ベースの Web サービスインタフェースの使用に関する詳細を説明します。要求メッセージの形式設定と応答メッセージの解析に使用する SPML 2.0 クラスについて説明します。
- 付録 A「ビジネスプロセスエディタの使用方法」- Identity Manager ビジネスプロセスエディタ (BPE) について説明し、このアプリケーションを使用する際の手順を示します。

表記上の規則

この項の表は、本書で使用する次の表記規則について説明しています。

表記上の規則

次の表は、本書で使用する表記上の規則について説明しています。

表 1 表記上の規則

字体または記号	意味	例
AaBbCc123 (モノスペース)	API および言語の要素、HTML タグ、Web サイトの URL、コマンド名、ファイル名、ディレクトリパス名、画面出力の表示、サンプルコードを示します。	.login ファイルを編集します。 ls -a を使用してすべてのファイルを表示します。 % You have mail.
AaBbCc123 (太字のモノスペース)	ユーザーが入力する文字を、画面上のコンピュータ出力とは区別して示します。	%**su** Password:
AaBbCc123 (イタリック)	実際の名前または値によって置き換えられるコマンドまたはパス名の可変部分。	このファイルは、*install-dir*/bin ディレクトリにあります。

記号

次の表は、本書で使用する記号の表記規則を示しています。

表 2 記号の表記規則

記号	説明	例	意味
[ ]	省略可能なコマンドオプションが入ります。	ls [-l]	-l オプションは省略可能です。
-	同時に押すキーを連結します。	Control-A	Ctrl キーと A キーを同時に押します。
+	連続して押すキーを連結します。	Ctrl+A+N	Ctrl キーを押し、離してから、以後のキーを続けて押します。

表 2　記号の表記規則 (続き)

記号	説明	例	意味
>	グラフィカルユーザーインタフェースで選択するメニュー項目を示します。	「ファイル」>「新規」>「テンプレート」	「ファイル」メニューから「新規」を選択します。「新規」サブメニューから、「テンプレート」を選択します。

シェルプロンプト

次の表は、本書で使用するシェルプロンプトを示しています。

表 3　シェルプロンプト

シェル	プロンプト
UNIX または Linux の C シェル	*machine-name*%
UNIX または Linux の C シェルのスーパーユーザー	*machine-name*#
UNIX または Linux の Bourne シェルおよび Korn シェル	$
UNIX または Linux の Bourne シェルおよび Korn シェルのスーパーユーザー	#
Windows のコマンド行	C:¥

関連ドキュメントとヘルプ

Sun Microsystems は、Identity Manager をインストール、使用、および設定する際に役立つ次のような追加のドキュメントと情報を提供しています。

- 『Identity Manager インストール』: Identity Manager と関連ソフトウェアをインストールおよび設定する手順と参照情報が記載されています。
- 『Identity Manager Upgrade』: Identity Manager と関連ソフトウェアをアップグレードおよび設定する手順と参照情報が記載されています。
- 『Identity Manager 管理ガイド』: Identity Manager を使用して企業情報システムへのセキュリティー保護されたユーザーアクセスを実現するために、手順、チュートリアル、実例を説明します。
- 『Identity Manager の配備に関する技術概要』– Identity Manager 製品の概念に関する概要 (オブジェクトアーキテクチャーを含む) および基本的な製品コンポーネントの紹介が記載されています。
- 『Identity Manager ワークフロー、フォーム、およびビュー』: Identity Manager のワークフロー、フォーム、および画面の使用方法を示す参照情報と手順が記載されています。この中には、これらのオブジェクトをカスタマイズするのに必要なツールに関する情報が含まれます。
- 『Identity Manager リソースリファレンス』: アカウント情報をリソースから Sun Java™ System Identity Manager に読み込んで同期する方法を示す参照情報と手順が記載されています。
- 『Identity Manager Tuning, Troubleshooting, and Error Messages』: Sun Java™ System Identity Manager のチューニングに関するガイダンス、問題の追跡とトラブルシューティングの手順、およびこの製品を操作したときに発生する可能性があるエラーメッセージと例外についての説明を提供する参照情報と手順が記載されています。
- 『Identity Manager Service Provider Edition Deployment』: Sun Java™ System Identity Manager Service Provider Edition の計画と実装の方法を示す参照情報と手順が記載されています。
- Identity Manager ヘルプ : Identity Manager の完全な手順、参照情報、用語の説明を記載したオンラインガイダンス、オンライン情報です。ヘルプにアクセスするには、Identity Manager メニューバーの「ヘルプ」リンクをクリックします。主要なフィールドには、ガイダンス (フィールド固有の情報) があります。

オンライン上の Sun リソースへのアクセス

製品のダウンロード、プロフェショナルサービス、パッチとサポート、および開発者向け追加情報については、次の Web サイトにアクセスしてください。

- ダウンロードセンター
  http://wwws.sun.com/software/download/
- プロフェショナルサービス
  http://www.sun.com/service/sunps/sunone/index.html
- Sun Enterprise サービス、Solaris パッチ、およびサポート
  http://sunsolve.sun.com/
- 開発者向け情報
  http://developers.sun.com/prodtech/index.html

Sun テクニカルサポートへのお問い合わせ

製品のドキュメントで解決できない、本製品に関する技術的な質問については、次のいずれかの方法でカスタマサポートにお問い合わせください。

- オンラインサポート Web サイト http://www.sun.com/service/online/us
- 保守契約に基づいて提供されるサポート電話番号

関連するサードパーティー Web サイト

Sun は、本書に記載されているサードパーティー Web サイトの利用について責任を負いません。Sun は、このようなサイトまたはリソースで得られるあらゆる内容、広告、製品、およびその他素材を保証するものではなく、責任または義務を負いません。Sun は、このようなサイトまたはリソースで得られるあらゆるコンテンツ、製品、またはサービスによって生じる、または生じたと主張される、または使用に関連して生じる、または信頼することによって生じる、いかなる損害または損失についても責任または義務を負いません。

ご意見、ご要望の送付先

Sun ではマニュアルの品質向上のため、お客様のご意見、ご要望をお受けしております。

コメントをお送りになる場合は、http://docs.sun.com にアクセスして「コメントの送信」をクリックしてください。オンラインフォームで、ドキュメントのタイトルと Part No. を入力します。Part No. は、マニュアルのタイトルページまたは最上部に記載されている 7 桁または 9 桁の番号です。

たとえば、本書のタイトルは『Identity Manager 配備ツール』であり、Part No. は 820-2530 です。

ご意見、ご要望の送付先

第 1 章

Identity Manager IDE の使用

Identity Manager Integrated Development Environment (Identity Manager IDE) とは、使用している配備で Sun Java™ System Identity Manager (Identity Manager) オブジェクトを表示、カスタマイズ、およびデバッグできる Java アプリケーションのことです。

ここでは、Identity Manager IDE の使用方法について説明します。この章で説明する内容は次のとおりです。

概要

Identity Manager IDE は、ご使用の環境で、オブジェクトをグラフィカルに XML ベースで表示します。このアプリケーションを使用して次のタスクを実行できます。

- 特定のリポジトリに関連付けられたプロジェクトの作成
- 設定オブジェクト、電子メールテンプレート、フォーム、汎用オブジェクト、ライブラリ、規則、ワークフロープロセス、およびワークフローサブプロセスの表示、作成、および編集
- リポジトリからのオブジェクトのダウンロードおよびリポジトリへのオブジェクトのアップロード
- フォーム、規則、およびワークフローのデバッグ

この節の残りの部分では、Identity Manager IDE の全般的な高レベルの概要を説明します。

主な機能

Identity Manager IDE によって提供される主な機能は、次のとおりです。

- プロジェクトのプロジェクトベース、ディレクトリベース、または実行時の表示が可能な統合されたエクスプローラウィンドウ
- Identity Manager IDE プロジェクトが設定ビルド環境 (CBE) と統合されている
- ドキュメント変更のためのアクションメニュー
- カスタムエディタ。次のものがあります。
  - XML オブジェクトプロパティーを列挙し、XML を入力せずに基本的なオブジェクトタイプ、XPRESS、および XML オブジェクトを編集するためのオブジェクトプロパティーシートとグラフィカル値エディタ
  - XML を入力せずに承認、ワークフローサービス、ワークフロータスク、ユーザーなどを XML ソースに追加するためのドラッグ＆ドロップパレット
  - XML 要素と属性に対する構文の強調表示と自動補完を可能にする登録済みの `waveset.dtd` 定義ファイル
- フォーム、規則、およびワークフローのための統合されたデバッガ
- スタンドアロンおよびライブラリの規則を確認するための規則テスター
- 外部ブラウザ内でフォームをテストするためのフォームテスター
- Identity Manager ビュー (ユーザービューなど) をチェックアウト、変更、チェックインできるビューのチェックアウト機能

- CVS の統合

BPE と比較した場合の Identity Manager IDE

Identity Manager IDE は、完全に統合された NetBeans プラグインで、Identity Manager のビジネスプロセスエディタ (BPE) アプリケーションに代わるよう設計されています。Identity Manager IDE には、BPE と比べて次の利点があります。

- 配備のベストプラクティスにより近づきます。次のことが可能です。
  - リポジトリの外にあるオブジェクトの変更
  - Ant や CVS などの一般的な配備ツールの使用
- エディタ、パレット、デバッガ、およびナビゲーションにプラットフォームサポートを提供します
- XML 要素および属性にコード補完を適用することにより、未完コードを削減します。
- コンテキスト情報が提供され、統合ツールを使用することにより、編集が容易になります。

注 BPE は現在も下位互換性のサポート対象です。
BPE の使用方法の詳細については、次を参照してください。付録 A 「ビジネスプロセスエディタの使用方法」

Identity Manager IDE のインストール

ここでは、Identity Manager IDE アプリケーションのインストールと設定の手順を説明します。

注 Identity Manager のプロジェクトが version 7.1 リリースで大幅に変更されたため、version 7.1 の .nbm は、JDK 1.5 と NetBeans 5.5 を必要とします。したがって、version 7.0 の .nbm で作成されたすべてのプロジェクトをアップグレードする必要があります。手順については、4 ページの「version 7.0 プロジェクトのアップグレード」を参照してください。

開始する前に

Identity Manager IDE を実行するには、次のものが必要になります。

- ローカルシステムにインストールされた JDK 1.5 の下で動作する NetBeans 5.5。NetBeans 5.5 は、次のサイトからダウンロードできます。

  http://www.netbeans.org/

- Identity Manager に対する Configurator レベルのアクセス権。

注 デフォルトの Identity Manager プロジェクトでは、mail.jar、activation.jar、および jms.jar が custom/WEB-INF/lib ディレクトリ内に格納されます。これらの JAR ファイルは、バンドルされている Tomcat (NetBeans にデフォルトで付属) で必要になります。

別のターゲットアプリケーションサーバーを使用する場合には、これらの JAR ファイルを削除する必要がある可能性があります。アプリケーションサーバー製品の具体的な JAR 要件については、その製品のマニュアルを参照してください。

version 7.0 プロジェクトのアップグレード

7.0 の .nbm から 7.1 の .nbm にアップグレードするには、次の手順を実行する必要があります。

1. JDK インストールを JDK 1.5 にアップグレードします。
2. 現在の NetBeans インストールを NetBeans 5.5 にアップグレードします。
3. Identity Manager 7.1 の .nbm を NetBeans 5.5 インストール内にインストールします。

4. 新しい Identity Manager プロジェクトを作成します。

   ここでは、必要に応じて「通常」、「リモート」のいずれかのプロジェクトを作成できます。これらのプロジェクトタイプの詳細については、27 ページの「プロジェクトとは何か」を参照してください。

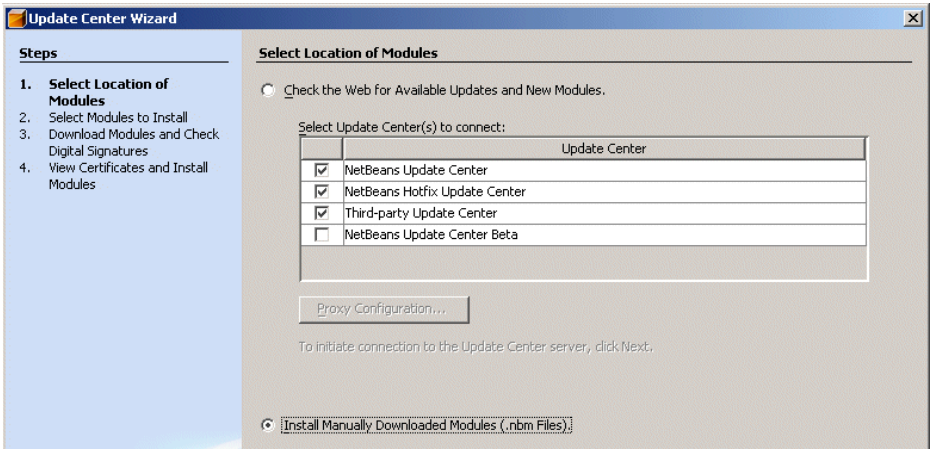
5. 古いプロジェクトの `objects` ディレクトリの内容を、新しいプロジェクトの `src/objects` ディレクトリにコピーします。

モジュールのインストール

Identity Manager IDE は、NetBeans に登録する必要のあるモジュールプラグインとしてパッケージ化されています。モジュールをインストールして登録するには、次の手順に従います。

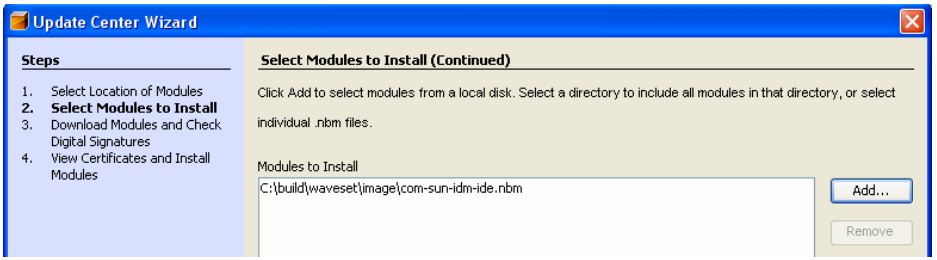
1. NetBeans 5.5 を開始します。
2. NetBeans メニューバーで、「Tools」>「Update Center」の順に選択します。
3. アップデートセンターウィザードが表示されたら (図 1-1)、「Install Manually Downloaded Modules」を選択して、「Next」をクリックします。

図 1-1 モジュールの場所を選択



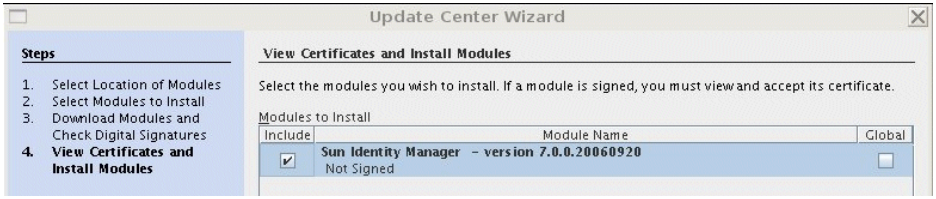
4. 「Select Modules to Install」パネル (図 1-2) で「Add」をクリックし、Identity Manager IDE NetBeans プラグインファイル (`com-sun-idm-ide.nbm`) を特定して選択します。このファイルは、Identity Manager の `image` ディレクトリ内にあります。

図 1-2　　インストールするモジュールの指定



5. 「Next」をクリックすると、「Select Modules to Install」パネルが表示されます。「Next」を再度クリックしてモジュールをダウンロードします。
6. Identity Manager IDE 使用許諾契約書が表示されたら、「Accept」をクリックします。
7. 「Download Modules」パネルが表示され、ダウンロード中のモジュールの状態が表示されます。「Next」をクリックします。
8. 「View Certificates and Install Modules」パネルが表示され (図 1-3)、ダウンロードしたモジュールが一覧で示されます。モジュール名の横にあるチェックボックスを有効にします。

図 1-3　　証明書を表示するモジュールの選択



9. 署名のないモジュールをインストールするかどうかを確認するポップアップが表示されます。
   - モジュールのインストールを続行する場合は、「Yes」をクリックします。
   - インストールを続行しない場合は、「No」をクリックします。
10. 「Finish」をクリックします。

NetBeans は Identity Manager IDE モジュールをインストールします。

これで、次の作業が可能になります。

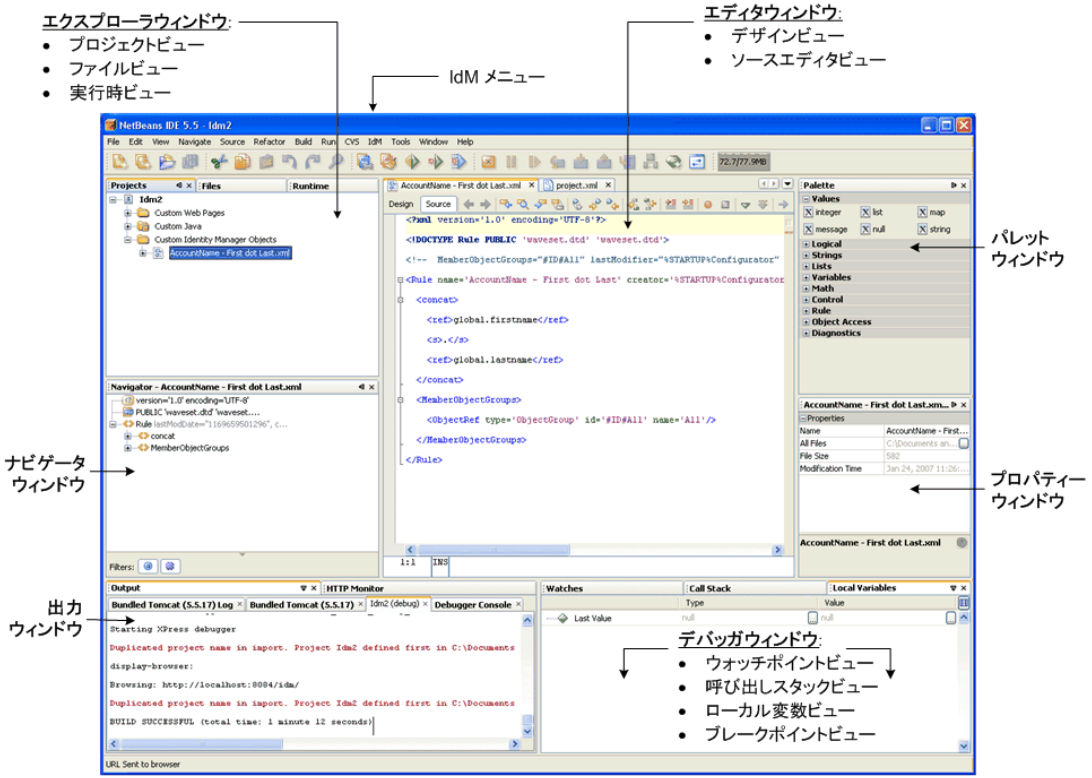
- NetBeans 内での Identity Manager プロジェクトの作成

- Identity Manager インストール環境からのオブジェクトのダウンロード
- アップロードする新規 XML オブジェクトファイルの作成

Identity Manager IDE インタフェースの操作

Identity Manager IDE のインタフェース (図 1-4 に示す) は、IdM メニュー (NetBeans のトップレベルメニューバー上に配置) と、次の各ウィンドウで構成されています。

図 1-4　　Identity Manager IDE ユーザーインタフェース



注

- Identity Manager IDE を最初に開いたときに、これらすべてのウィンドウが表示されるわけではありませんが、アプリケーションのさまざまな機能を使用するときには、ウィンドウが自動的に開くか、または開くことができるようになります。
- メニューオプションなど、NetBeans ユーザーインタフェースの一部となっている機能も表示されます。それらの機能については、このマニュアルでは説明しません。それらの機能については、NetBeans のオンラインヘルプや製品マニュアルを参照してください。

この節では、IdM メニューと Identity Manager IDE の各ウィンドウを紹介します。説明する内容は次のとおりです。

- IdM メニュー
- エクスプローラウィンドウ
- エディタウィンドウ
- パレットウィンドウ
- プロパティーウィンドウ
- 出力ウィンドウ
- デバッガウィンドウ

IdM メニュー

Identity Manager IDE アプリケーションのインストールと設定が完了すると、IdM メニューが NetBeans のトップレベルメニューバーから利用可能になります。

IdM メニューを使えば、NetBeans Explorer で現在選択されているノードが何であれ、そのノードに適したアクションを選択できます。次に例を示します。

- フォルダを選択した場合、オブジェクトのアップロードまたはダウンロードができる。
- Identity Manager XML ファイルを選択した場合、オブジェクトのアップロードまたは再読み込みができる。

表 1-1 では、IdM メニューのすべてのオプションについて説明します。

ヒント

- IdM メニューのほとんどのオプションは、プロジェクトツリー内のオブジェクトを右クリックしたときに表示されるポップアップメニューからも選択可能です。
- これらの IdM オプションの使用方法に関する追加情報については、「組み込みリポジトリの管理」、「リポジトリオブジェクトの操作」、および「Identity Manager IDE デバッガの操作」の各節を参照してください。

表 1-1 IdM メニューのオプション

選択	実行されるアクション
Repository > Explore	「Explore Repository」ウィンドウを開き、Identity Manager IDE リポジトリの内容を参照できるようにします。1 つ以上のオブジェクトを選択したあと、次の操作を行えます。 • 「Download」ボタンをクリックしてオブジェクトをローカルファイルシステムにダウンロードする。 • 「Delete」ボタンをクリックして選択されたオブジェクトをリポジトリから削除する。
Repository > Open Object	あるオブジェクトをエディタウィンドウ内で開きます。 **注 :** ローカルファイルシステム上のオブジェクトしか開けません。リポジトリからまだダウンロードされていないオブジェクトを開こうとすると、Identity Manager IDE によって自動的に、そのオブジェクトがローカルファイルシステムにダウンロードされ、その後にそのオブジェクトがエディタ内で開かれます。
Repository > Checkout View	編集のためにユーザービューなどの特定のビューをリポジトリからチェックアウトします。
Repository > Upload Objects	ローカルファイルシステムのオブジェクトをリポジトリにアップロードします。詳細については、49 ページを参照してください。
Repository > Diff Objects	ローカルファイルシステムのあるディレクトリ内のすべてのオブジェクトを、リポジトリ内の対応するオブジェクトと比較します。 Identity Manager IDE は、そのディレクトリ内のすべてのオブジェクトに対する差分処理をファイルシステム内で下方に再帰的に繰り返したあと、ローカルとリモートの XML コードを構文強調表示と行番号付きで左右に並べて比較します。

表 1-1　IdM メニューのオプション (続き)

選択	実行されるアクション
Repository > Manage Embedded Repository **注 :** このメニューオプションは、Identity Manager Project (Remote) の場合や独自のリポジトリを指定する場合は使用できません。	「Manage Embedded Repository」ダイアログを開きます。このダイアログでは、現在のプロジェクトに対する次のリポジトリ設定を変更できます。 • Repository Location: 別のリポジトリ場所を指定します。 • Initialize Repository: これが有効になっていると、Identity Manager IDE は既存のデータファイルを削除し、`idm.war` から `init.xml` ファイルをインポートします。単にリポジトリの場所をすでに初期化された別のリポジトリに変更する場合には、このオプションを有効にしないでください。 • Automatically Publish Identity Manager Objects: これが有効になっていると、ユーザーがプロジェクトを実行またはデバッグするたびに、Identity Manager IDE はプロジェクト内のすべての Identity Manager オブジェクトをリポジトリに自動的にアップロードします。
Repository > Reload Object	現在のプロジェクト内のオブジェクトを、リポジトリにある同じオブジェクトの新しいバージョンで置き換えるまたは上書きします。
Run LH command	Identity Manager IDE 内から `lh` コマンド (`lh console` や `lh setup` など) を実行します。
Set Identity Manager Instance	「Set Identity Manager Instance」ダイアログを開きます。このダイアログでは、ある特定のプロジェクトのコンテキストで次のアクションをどの Identity Manager インスタンスに適用するかを制御できます。 Debug Project Repository > Explore Repository > Open Project Checkout View Upload Object(s) Diff Object(s) Reload Object Test Form Test Rule 詳細は、37 ページの「Identity Manager インスタンスの設定」を参照してください。
Clear Credentials Cache	ユーザーが「Remember Password」オプションを有効にしている場合に Identity Manager IDE が記憶しているパスワードが削除されます。

表 1-1　IdM メニューのオプション (続き)

選択	実行されるアクション
Options	• Identity Manager IDE がリポジトリからオブジェクトをダウンロードする前に、自動生成されたリポジトリ ID のうち、指定された表現に一致するものをすべて削除します。ただし、ハードコードされた定義済みの ID は削除されません。 この機能は、あるリポジトリから別のリポジトリにオブジェクトを移動する場合に役立ちます。 • 「Exclude lastModDate」、「Exclude Ids」、または「Apply pattern substitution」オプションを有効または無効にすることで差分オプションを指定します。オブジェクトの差分処理の詳細については、60 ページの「差分オプションの使用」を参照してください。
Test Form	ブラウザ内でフォームをテストします。
Test Rule	スタンドアロンおよびライブラリの規則に対して行なった変更を検証します。

エクスプローラウィンドウ

エクスプローラウィンドウは Identity Manager IDE の右上にあり、次の 3 つのウィンドウで構成されています。

- プロジェクトウィンドウ
- ファイルウィンドウ
- 実行時ウィンドウ

プロジェクトウィンドウ

プロジェクトウィンドウは、Identity Manager IDE を開いたときにエクスプローラ内にデフォルトで表示され、開いているすべてのプロジェクトを垂直な論理ビューで示します。

プロジェクトノードを展開すると、プロジェクト内の次の XML オブジェクトが階層ビューで示されます。

- 設定オブジェクト
- 電子メールテンプレート
- フォーム
- 汎用オブジェクト
- ライブラリ

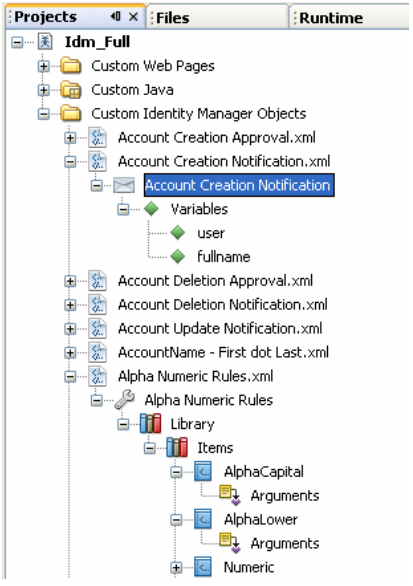
- 規則
- ワークフロープロセス
- ワークフローサブプロセス

XML オブジェクトを構成するほとんどの要素も、プロジェクトツリーのノードで表されます。

注	デフォルトでは、オブジェクトノードは、XML オブジェクト内と同じ順序でプロジェクトツリー内に表示されます。ただし、「Change Order」、「Insert Before」、「Insert After」、「Move Up」、または「Move Down」ポップアップメニューオプションを使えば、ツリー内 (および XML オブジェクト内) でのそれらの順序を変更できます。これらのオプションについては、表 1-2 を参照してください。

図 1-5 に、さまざまなトップレベルの XML ファイル、およびいくつかの変数ノード、ライブラリノード、規則ノード、および引数ノードを示します。

図 1-5　プロジェクトウィンドウ



このウィンドウで任意のノードを右クリックすると、そのノードに関係する各種タスクの実行を可能にするオプションを含む、ポップアップメニューが表示されます。

注 メニューオプションのいくつかは NetBeans アプリケーションによって提供されていますが、このマニュアルではそれらのオプションについては説明しません。
それらのオプションについては、NetBeans のオンラインヘルプや製品マニュアルを参照してください。

表 1-2 に、Identity Manager IDE に関係するポップアップメニューオプションの概要を示します。

表 1-2 Identity Manager IDE ポップアップメニューのオプション

ノード	メニューオプション	説明
Project	Debug Project	プロジェクトをデバッグするときに選択します。
	Close Project	プロジェクトを閉じるときに選択します。
	Repository >	リポジトリを参照したり、オブジェクトのオープン、アップロード、または差分処理を行なったり、ビューをチェックアウトしたり、組み込みリポジトリを管理したりする場合に選択します。
Object Node	Add	このプロジェクトの新規オブジェクトを作成するときに選択します。
	CVS	Identity Manager IDE 内から CVS バージョン制御ファイルを管理するときに選択します。
	Repository	リポジトリからローカルファイルシステムにオブジェクトをダウンロードするとき、ローカルファイルシステムからリポジトリにオブジェクトをアップロードするとき、およびビューをチェックアウトするときに選択します。
	Find	プロジェクト内でテキスト、オブジェクト名、オブジェクトタイプ、および日付を検索するときに選択します。プロジェクトウィンドウですべてのプロジェクトを検索することも、特定の場所を選択して検索することも可能です。
	Cut、Copy、Paste	オブジェクトをカットアンドペーストやコピーアンドペーストするときに選択します。
	Delete	選択したオブジェクトをプロジェクトから削除するときに選択します。
	Refactor	オブジェクトのコード構造を変更し、この変更が反映されるように残りのコードを更新するときに選択します。
	Tools	JUnit テストの作成、お気に入りへの追加、国際化の管理、または相違パッチの適用を行うときに選択します。
	Properties	選択されたオブジェクトのプロパティー (TaskDefinition プロパティー、設定プロパティー、リポジトリプロパティーなど) を表示および編集するときに選択します。

表 1-2　Identity Manager IDE ポップアップメニューのオプション (続き)

ノード	メニューオプション	説明
	Add *element*	アクション、引数、アクティビティー、フォーム、オブジェクト参照、プロセス、プロパティー、結果、戻り値、規則、遷移、または変数をオブジェクトに追加するときに選択します。 **注 :** オブジェクトノードのタイプによって、どの要素を追加できるかが決まります。たとえば、アクションオブジェクトには、引数、フォーム、結果、戻り値、および変数を追加できます。
	Change Order	複数のオブジェクトの順序を一度に変更するときに選択します。 「Change Order」ダイアログが表示されます。このダイアログでは、複数のオブジェクトを上方または下方に移動することで、プロジェクトツリー内および XML オブジェクト内でのそれらの位置を変更できます。
	Insert Before	選択されたオブジェクトノードを、プロジェクトツリー内の指定されたオブジェクトノードの上に移動するときに選択します。
	Insert After	選択されたオブジェクトノードを、プロジェクトツリー内の指定されたオブジェクトノードの下に移動するときに選択します。
	Move Up	選択されたオブジェクトノードをプロジェクトツリー内の上方に移動するときに選択します。
	Move Down	選択されたオブジェクトノードをプロジェクトツリー内の下方に移動するときに選択します。

ファイルウィンドウ

ファイルウィンドウを選択してノードを展開すると、プロジェクトウィンドウでは表示されないファイルやフォルダを含め、プロジェクトがディレクトリベースで表示されます。

プロジェクトビルドスクリプトやプロパティーファイルなどのプロジェクト設定ファイルを、開いたり、編集したりすることができます。

注　新しいプロジェクトを作成すると、Identity Manager IDE によって自動的に、`README.txt` ファイルが作成されます。

プロジェクトのファイルシステムの詳しい概要を表示する (あるいは別の設定ビルド環境 (CBE) で動作するようにプロジェクトの構造を変更する) には、ファイルウィンドウで `README.txt` をダブルクリックします。すると、エディタウィンドウに情報が表示されます。

実行時ウィンドウ

実行時ウィンドウは、DTD および XML スキーマカタログの下にある Identity Manager カタログ (`waveset.dtd`) に読み取り専用アクセスする場合に選択します。

エディタウィンドウ

エディタウィンドウにはツールバーと表示エリアがあり、デザインビューとソースエディタビューでオブジェクトを操作できます。

- **デザインビュー** : このビューは、次のものを操作する場合に使用します。
  - 規則オブジェクトを操作するための式ビルダー。追加情報については、15 ページの「規則のデザインビュー」を参照してください。
  - ワークフロープロセスおよびワークフローサブプロセスオブジェクトのグラフィカル表現。追加情報については、16 ページの「ワークフローのデザインビュー」を参照してください。

  注 規則とワークフローを操作する場合には、デザインビューがデフォルトのビューになります。

- **ソースエディタビュー** : このビューは、オブジェクトの XML ソースを操作する場合に使用します。
  追加情報については、18 ページの「ソースエディタビュー」を参照してください。

ツールバー上のほかのボタンは、どちらのビューを選択したかに応じて変わります。

注 パレットウィンドウからエディタウィンドウには、項目をドラッグすることができます。たとえば、次のようにします。

- 新規の Workflow サービス、承認、ユーザー、および Workflow タスクをエディタにドラッグして、ワークフローに使用できます。
- 新規のフィールドをエディタにドラッグして、フォームに使用できます。

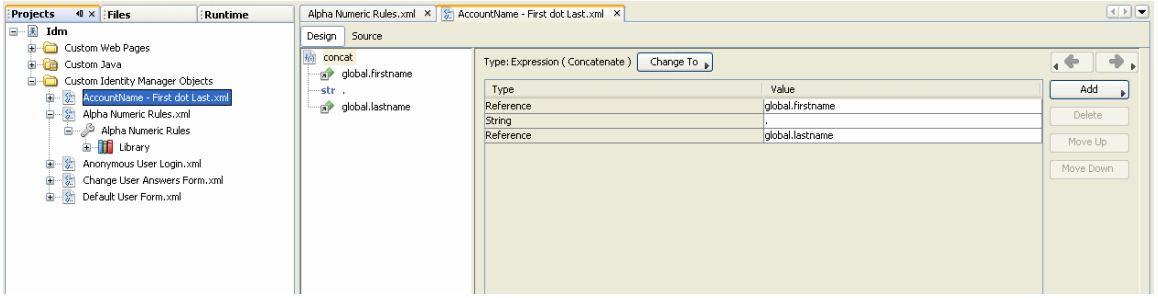
項目をデザインビューにドラッグすると、プロジェクトおよびファイルの両方のビューの XML ソースとノードツリーが更新されます。詳細については、22 ページの「パレットウィンドウ」を参照してください。

規則のデザインビュー

この節では、規則とワークフローのデザインビューエディタについて説明します。

規則のデザインビューエディタは、規則の論理構造を確認したり規則のプロパティーを変更したりしやすくするための式ビルダー (図 1-6) を提供します。

図 1-6 規則のデザインビューの例



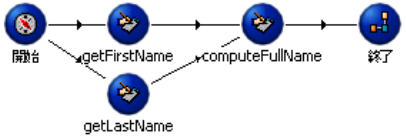
ある規則をデザインビューで開くには、その規則をプロジェクトウィンドウでダブルクリックするか、その規則のノードを選択してから「Design」ボタンをクリックします。

式ビルダーを使用する手順については、54 ページの「「Expression Builder」ダイアログの使用」を参照してください。

ワークフローのデザインビュー

ワークフローのデザインビューエディタにはワークフローが図示され (図 1-7)、図中の各ノードは特定のプロセスアクティビティーを、線は遷移を表現します。

図 1-7 ワークフローのデザインビューの例



ワークフローのデザインビューを操作するには、あるワークフローをプロジェクトウィンドウでダブルクリックするか、そのワークフローのノードを選択してから「Design」ボタンをクリックします。

デザインビューでは、ツールバーを使用して次のタスクを実行できます。

表 1-3　デザインビューのツールバーボタン

クリックするボタン	実行されるタスク
 Add Activity:	ワークフローにアクティビティーを追加する場合にクリックします。
 Remove Activity:	1 つ以上のアクティビティーをワークフローから削除する場合にクリックします。 複数のアクティビティーを削除するには、Ctrl キーを押したままノードを選択します。
 Layout:	ワークフローのレイアウトを設定する場合にクリックします。アクティビティーが無秩序に配置されている場合には、それらをリセットして編成します。
 Validate XML:	ワークフローの妥当性検査を行う場合にクリックします。妥当性検査情報は出力ウィンドウに表示され、通常はソースエディタビュー内の特定の行に移動するハイパーリンクが提供されます。このリンク先で XML を表示または修正できます。

これらのボタンに加え、ワークフローセレクタメニューを使用して (このツールバーにも用意されている)、デザインビューでワークフローやワークフローのサブプロセスを選択して作業できます。

ワークフローをデザインビューで編集すると、ワークフローライブラリから読み込まれる項目が、パレットウィンドウに表示されます。これらの項目をパレットからドラッグしてデザインビューにドロップし、図の中にアクティビティーノードを作成することができます。このとき、プロジェクトビューとファイルビューの XML ソースとノードツリーも更新されます。詳細については、22 ページの「パレットウィンドウ」を参照してください。

ソースエディタビュー

ソースエディタは完全な機能を備えたテキストエディタであり、選択されたオブジェクトのフィルタリングされていない XML を表示します。さらに、ソースエディタは、エクスプローラプロジェクトウィンドウおよびデバッガと統合されています。

ソースエディタビューを開くには、次のいずれかを実行します。

- プロジェクトウィンドウで編集可能なオブジェクトをダブルクリックする
- 利用可能な Identity Manager IDE テンプレートから編集可能な新しいオブジェクトを作成する (ソースエディタが自動的に開く)
- エディタウィンドウの最上部にある「Source」ボタンをクリックする (規則とワークフローでのみ利用可能)

図 1-8　　ソースエディタビューの例

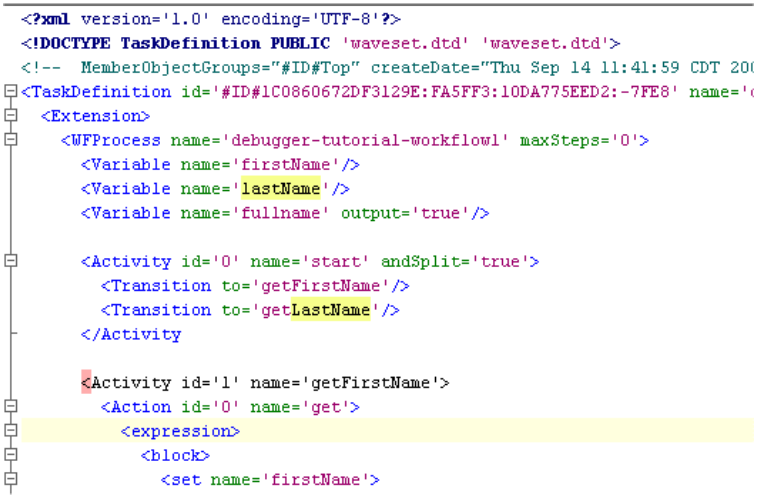


表 1-4 では、ソースエディタビューで提供されるツールバーボタンについて説明しています。各タスクのキーボードショートカットも示しています。

表 1-4 ソースエディタビューのツールバーボタン (およびショートカット)

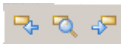
使用するボタン	実行されるタスク
	「Back」(Alt+K) および「Forward」(Alt+L): これらのボタンをクリックすると、「Source Editor」ウィンドウで行なった直前の編集に戻るか、または次の編集に進みます。
	これらのボタンを使用して、カーソルの挿入ポイントを次のように移動します。 • 「Find Previous Occurrence」(Shift+F3): このボタンをクリックすると、直前に検索したテキストを検出して、そこに挿入ポイントを移動します。 • 「Find Selection」(Ctrl+F3): このボタンをクリックすると、現在カーソルが挿入されている項目を検索します。 • 「Find Next Occurrence」(F3): このボタンをクリックすると、検索するテキストの次の出現を検出して、そこに挿入ポイントを移動します。
	これらのボタンを使用して、次のように機能をオンまたはオフに切り替えます。 • 「Toggle Highlight Search」(Alt+Shift+H): このボタンをクリックして、検索テキストの強調表示をオン / オフにします。 • 「Toggle Bookmark」(Ctrl+F2): 行を強調表示してからこのボタンを押すと、現在の行にブックマークが挿入されるか、または現在の行からブックマークが削除されます。
	「Next Bookmark」(F2) および「Previous Bookmark」(Shift+F2): これらのボタンをクリックすると、XML ソース内の次のブックマークまたは直前のブックマークを検出します。
	「Next Matching Word」(Ctrl+L) および「Previous Matching Word」(Ctrl+K): XML ソース内で単語を選択してからこれらのボタンをクリックすると、その単語の次または直前の出現を検出します。
	「Shift Line Left」(Ctrl+D) および「Shift Line Right」(Ctrl+T): これらボタンをクリックすると、選択した行のインデントがタブ一段分減少または増加します。
	「Start Macro Recording」(Ctrl+J S) および「Stop Macro Recording」(Ctrl+J E): これらのボタンをクリックして、キーストロークやカーソルの移動を含む、マクロの記録を開始または停止します。

表 1-4　ソースエディタビューのツールバーボタン (およびショートカット) (続き)

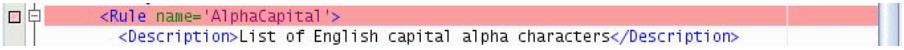
使用するボタン	実行されるタスク
	これらのボタンを使用して、次のように XML ソースを検査または妥当性検査します。 • 「Check XML」(Alt+F9): このボタンをクリックすると、オブジェクトの XML が正しく整形式であるかを検査します。出力ウィンドウには XML ソースのエラーが特定され、通常はソースエディタ内のその行へのハイパーリンクが提供されて、問題を訂正することができます。 • 「Validate XML」(Alt+Shift+F9): このボタンをクリックすると、オブジェクトの XML を DTD に対して妥当性検査します。妥当性検査情報は出力ウィンドウに表示され、通常はソースエディタビュー内の特定の行に移動するハイパーリンクが提供されます。このリンク先で XML を表示または修正できます。

- ブレークポイントの設定

ソースエディタ内でブレークポイント (プロセス実行内での個別の停止ポイント) を設定するには、XML タグのすぐ横の左マージンをクリックします。たとえば、`<WFProcess...>` タグの横のマージンをクリックすると、ワークフローの開始位置にブレークポイントを設定できます。

図 1-9 にブレークポイントの例を示します。

図 1-9　ブレークポイントの例



パレットからのドラッグ

編集のためにオブジェクトを選択すると、パレットウィンドウ (22 ページを参照) が通常 Identity Manager IDE の右上に、ワークフローライブラリおよび XPRESS カテゴリの項目とともに自動的に表示されます。項目を「Palette」からソースエディタの行にドロップして、XML ソース内のそのポイントに XML テキストを作成できます。

呼び出しの挿入

ソースエディタで開かれた XML ファイル内で右クリックすると、次の挿入オプションを含むポップアップメニューが表示されます。

- Insert Instance Invoke: 第一引数に指定された Java オブジェクト上のインスタンスメソッドを呼び出します。クラス名属性が定義されていない呼び出しは、インスタンス呼び出しタイプになります。
- Insert Static Invoke: static Java メソッドを呼び出します。クラス名属性が定義された呼び出しは、static 呼び出しタイプになります。

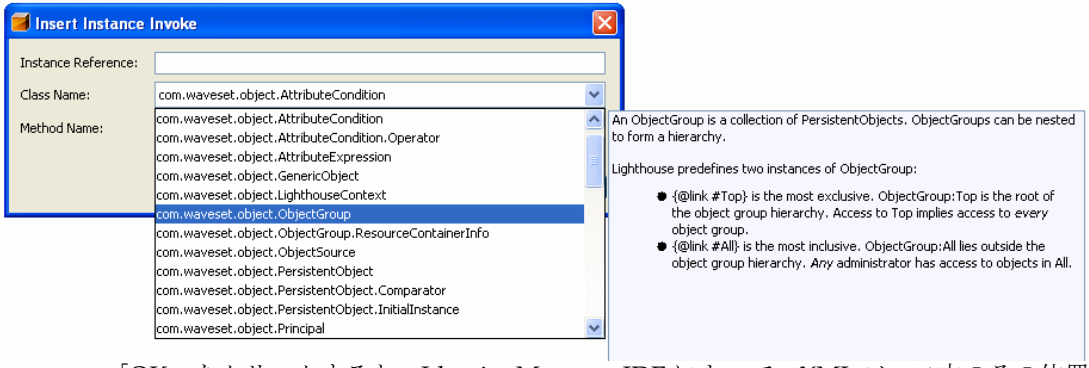
いずれかのオプションを選択すると「Insert Invoke」ダイアログが開くので、そこで次の情報を指定します。

- Instance Reference (インスタンス呼び出しの場合のみ)
- Class Name
- Method Name (選択したクラスによって利用可能なメソッドが決まる)

ヒント　「Class Name」メニュー、「Method Name」メニューのドロップダウン矢印をクリックすると、それぞれクラス、メソッドに対する JavaDoc が開きます。リスト内の名前の上でカーソルを移動させると、関連する JavaDoc を含むポップアップが、ダイアログの横に表示されます。

図 1-10 に、JavaDoc が表示された「Insert Invoke」ダイアログの例を示します。

図 1-10　JavaDoc を含む「Insert Instance Invoke」ダイアログの例



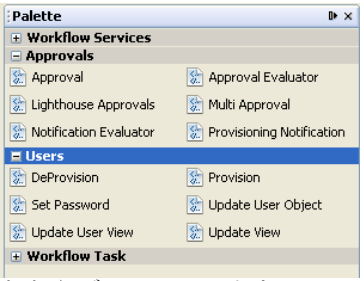
「OK」をクリックすると、Identity Manager IDE によって、XML ソース内のその位置に XML テキストが自動生成されます。

注　ワークフローのプロセスまたはサブプロセスを操作している場合、単純な計算を実行したり Java クラスや JavaScript を呼び出して複雑な処理を行なったりするために、「アクション」式を追加できます。

パレットウィンドウ

パレットウィンドウ (図 1-11 など) を使えば、XML を入力することなしに、エディタウィンドウ内に表示された電子メールテンプレート、フォーム、汎用オブジェクト、ライブラリ、ワークフロープロセス、またはワークフローサブプロセスオブジェクトに要素を「ドラッグ＆ドロップ」できます。

図 1-11　Delete User オブジェクトのパレットウィンドウの例



たとえば、フィールドをフォームオブジェクトにドラッグしたり、規則をライブラリオブジェクトにドラッグしたり、ワークフローサービス、承認、ユーザー、およびワークフロータスクをワークフロープロセスやワークフローサブプロセスオブジェクトにドラッグしたりできます。

プロジェクトツリーでオブジェクトをダブルクリックすると、パレットウィンドウが通常は Identity Manager IDE の右上に表示され、次のように編集ビューの種類に応じてさまざまな要素にアクセスできるようになります。

- **デザインビュー** : ワークフローライブラリから読み込まれた項目のグラフィカル表現を提供します。項目をパレットからデザインビューにドラッグすると、図にアクティビティーノードが作成されます。
- **ソースエディタビュー** : ワークフローライブラリおよび XPRESS カテゴリから読み込まれた項目の XML ソースを表示します。項目をパレットからソースエディタにドラッグすると、XML ソース内の項目をドロップした場所に、XML テキストが作成されます。

項目をいずれかのエディタウィンドウにドラッグすると、XML ソース、プロジェクト / ファイルビュー内のノードツリーの両方が更新されます。

注　オブジェクトに要素を追加したあと、プロジェクトツリー内および XML オブジェクト内におけるそれらの要素の順序を、各種ポップアップメニューオプションを使って変更できます。「Change Order」、「Insert Before」、「Insert After」、「Move Up」、「Move Down」の各オプションについては、表 1-2 を参照してください。

プロパティーウィンドウ

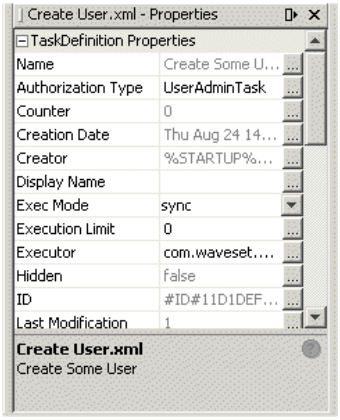
Identity Manager IDE のプロパティーウィンドウは、電子メールテンプレート、フォーム、汎用オブジェクト、ライブラリ、規則、ワークフロープロセス、およびワークフローサブプロセスオブジェクトに関連付けられた XML 要素に対応する、1 枚のプロパティーシートから構成されます。このプロパティーシートを使えば、オブジェクト名、ファイルサイズ、変更時刻、結果情報など、選択されたオブジェクトのプロパティーを表示および編集できます。

次のいずれかの方法で、プロパティーウィンドウを開きます。

- メインメニューバーから「Window」>「Properties」の順に選択します。
- プロジェクトウィンドウでノードを右クリックし、メニューから「Properties」を選択します。

プロパティーウィンドウが開いているときに、オブジェクトノードをダブルクリックしてウィンドウの内容を更新することができます。

図 1-12　　プロパティーウィンドウの例



注　　プロパティーシートを使ってオブジェクトの XML 要素を変更する手順については、51 ページの「オブジェクトの編集」を参照してください。

ヒント　　プロパティーウィンドウを使用しない方法として、ツリー内でオブジェクトノードを右クリックし、ポップアップメニューから「Properties」を選択する方法があります。表示される「Properties」ダイアログには、プロパティーウィンドウの場合と同じオブジェクトプロパティーシートが含まれます。

出力ウィンドウ

出力ウィンドウには、Identity Manager IDE がデバッグ中にエラーを検出したときに返すメッセージが表示されます。このメッセージには通常、エラーの原因となったソースコードの行へのハイパーリンクが含まれます。

出力ウィンドウはデバッガを実行すると自動的に開きますが、メインメニューバーで「Window」>「Output」の順に選択して開くこともできます。

図 1-13　出力ウィンドウの例

デバッガウィンドウ

Identity Manager IDE デバッガは、Java デバッガと BPE に備わっているデバッガの両方に似ています。コード内で行、グローバル、ビュー、またはフォームサイクルにブレークポイントを設定し、デバッガを開始して XML 内でコードを停止することができます。

さらに、Identity Manager IDE デバッガは、規則テスターおよびフォームテスターと統合されているほか、ウォッチポイントを追加して特定の変数の値を表示できるようにもなっています (これらの機能についてはこの章で後述する)。

デバッガには、XML ソースのトラブルシューティングに便利な次のウィンドウが提供されています。

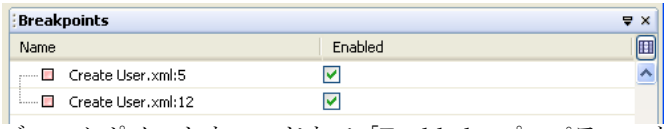
- ブレークポイントウィンドウ
- 呼び出しスタックウィンドウ
- ローカル変数ウィンドウ
- ウォッチポイントウィンドウ

ブレークポイントウィンドウ

ブレークポイントウィンドウには、現プロジェクトに設定したすべてのブレークポイントが、ブレークポイントの簡単な説明およびブレークポイントが現在有効か無効かを示すブール型フラグとともに一覧表示されます。

メインメニューバーで「Window」>「Debugging」>「Breakpoints」の順に選択すると、ブレークポイントウィンドウが開きます。

図 1-14　　ブレークポイントウィンドウの例



ブレークポイントウィンドウで「Enabled」プロパティーを変更することにより、ブレークポイントを有効化または無効化できます。

呼び出しスタックウィンドウ

デバッガを実行すると、Identity Manager IDE は自動的に呼び出しスタックウィンドウを起動します。

デバッガがブレークポイントに達すると、スレッドがブレークポイントに到達するまでに実行した一連の呼び出しである実行スタックが、呼び出しスタックウィンドウに表示されます。XPRESS では、これらの呼び出しは `BLOCK` や `CASE` などの XPRESS 操作です。

追加の関数が呼び出しチェーンに出現する場合、これらの関数は順番に一覧表示されます。このリストはスタックトレースと呼ばれ、リストにはプログラムのライフサイクルのその時点での実行スタックの構造が表示されます。

注　　呼び出しスタックウィンドウを閉じるかまたは別の理由でこのウィンドウが利用できなくなった場合は、メインメニューバーから「Window」>「Debugging」>「Call Stack」の順に選択することで、もう一度ウィンドウを開くことができます。

ローカル変数ウィンドウ

ローカル変数ウィンドウには、デバッガがブレークポイントで停止したときに、現時点の実行範囲内にあるすべての変数がリストされます。変数は、次のタイプの値を表します。

- 単純 (文字列など)
- 複合 (リストやマップなどその他の値を含む)

  複合タイプを展開して、それに含まれる要素を表示できます。

デバッグがアクティブでないときや、実行がブレークポイントで停止していないときは、ローカル変数ウィンドウは空白になります。

現在の要素が XPRESS 終了タグである場合、デバッガを停止すると、最後の評価の結果がローカル変数ウィンドウに表示されます。この評価値は、最後の値が意味を持つ他のタグにも適用されます。たとえば、Identity Manager が `<Argument>` タグをワークフローサブプロセスに対して評価すると、引数値がローカル変数ウィンドウに表示されます。

注

- XPRESS の実行中には、ローカル変数ウィンドウに変数が表示されません。
- デバッグが停止しているとき、Last Value エントリは空白となります。

ウォッチポイントウィンドウ

デバッガでは、ウォッチポイントを使用でき、デバッガ実行時に変数や式の値に加えられる変更を追跡することができます。有効な XPRESS 式であれば、ウォッチポイントになります。

ウォッチポイントを指定すると、プロジェクトをデバッグするときに監視対象になるすべての変数、式、タイプ、および値がウォッチポイントウィンドウに一覧表示されます。このリストには、XML ソース内のオブジェクトタイプに移動するハイパーリンクが含まれている場合もあります。

デバッガを起動すると、Identity Manager IDE はウォッチポイントウィンドウを自動的に起動します。または、メインメニューバーから「Window」>「Debugging」>「Watches」の順に選択することもできます。

キーボードショートカットの使用

NetBeans を使用することにより、Identity Manager IDE ではさまざまなキーボードキーの組み合わせを使って、コマンドを実行したり、Identity Manager IDE 内でナビゲートしたりすることができます。次の操作が可能です。

- メニューショートカットは、メニューバーでメニューオプションをナビゲート、起動、および選択するのに役立ちます。
- ソースエディタショートカットは、エディタ内でコマンドおよびアクションを実行するのに役立ちます。これらのショートカットは、すべてのファイルタイプに共通なものもあれば、特定のファイルタイプの編集中にのみ使用可能なものもあります。
- ウィンドウショートカットは、Identity Manager IDE ウィンドウ内で操作をナビゲート、起動、および選択するのに役立ちます。
- ヘルプショートカットは、Identity Manager IDE オンラインヘルプシステムをナビゲートするのに役立ちます。

使用可能なキーボードショートカットの詳細な説明、およびそれらの使用手順については、オンラインヘルプを参照してください。

注 ショートカットを割り当てるには、「Tools」>「Options」の順に選択し、「Options」ダイアログが表示されたら「Keymap」を選択します。

Identity Manager IDE プロジェクトの操作

Identity Manager IDE を効果的に使用するには、Identity Manager IDE プロジェクトの一部の基本的な概念について理解することも必要です。ここでは、次の内容を説明します。

プロジェクトとは何か

「プロジェクト」とは、ある配備のすべてのカスタマイズを制御するソースファイル (JSP、Java、および Identity Manager オブジェクト) やビルドスクリプトなどのコレクションのことです。

プロジェクトは特定のリポジトリに関連付けられます。複数のプロジェクトを 1 つのリポジトリに関連付けることは可能ですが、1 つのプロジェクトにつき 1 つのリポジトリにしか関連付けられません。

注 Identity Manager IDE プロジェクトを使用することにより、CVS と直接対話することができます。詳細については、NetBeans マニュアルを参照してください。

Identity Manager IDE 向けに次のいずれかのタイプのプロジェクトを作成できます。

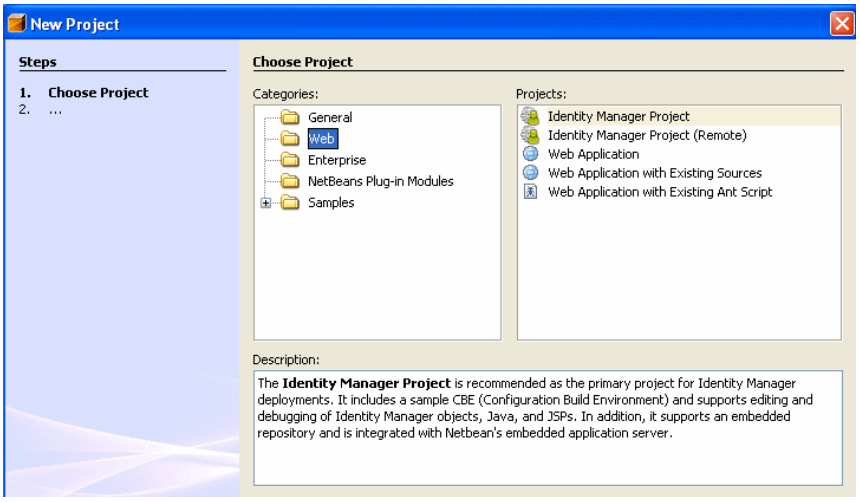
- Identity Manager Project: デプロイヤ向けの主要開発環境として提供される、完全な機能を備えたプロジェクト。主な機能を次に示します。
  - 完全なビルド環境
  - カスタム Java コードおよび JSP を管理する機能
  - NetBeans 組み込みアプリケーションサーバーと統合されているため、Identity Manager `war` ファイルのローカル配備が可能
  - 統合された Java、JSP、ワークフロー、フォーム、および XPRESS のデバッグ
  - 組み込みデータベースリポジトリ
- Identity Manager Project (Remote): 小規模な変更を加えたり外部サーバー上でデバッグを行なったりするための、機能限定版のプロジェクト。このプロジェクトには、フル機能版 Identity Manager プロジェクトの編集機能のすべてが備わっていますが、ビルド環境が欠如しており、war を起動することもできません。

プロジェクトの作成

新規プロジェクトを作成するには、次の手順に従います。

1. 「File」>「New Project」の順に選択します。
2. 新規プロジェクトウィザードが表示されたら (図 1-15)、「Categories」リストから「Web」を選択して、作成するプロジェクトのタイプを指定します。

図 1-15　新規プロジェクトウィザード : カテゴリの指定



3. 「Projects」リストから、次のサンプル Identity Manager プロジェクトの 1 つを選択します。
   - Identity Manager Project: 次のような完全な機能を備えた開発環境が必要な場合に、このプロジェクトを選択します。
     - サンプルの設定ビルド環境 (CBE)
     - カスタム Java コードおよび JSP を管理する機能
     - NetBeans 組み込みアプリケーションサーバーと統合されているため、Identity Manager `war` をローカルに配備できる
     - 統合された Java、JSP、ワークフロー、フォーム、および XPRESS のデバッグ
     - 組み込みデータベースリポジトリ

- Identity Manager Project (Remote): 小規模な変更を加えたり外部サーバー上でデバッグを行なったりする場合に、このプロジェクトを選択します。

  このタイプのプロジェクトには、完全な Identity Manager プロジェクトと同じ編集機能がすべて備わっていますが、ビルド環境は提供されておらず、Identity Manager.war ファイル (idm.war) を起動することもできません。

4. 完了したら、「Next」をクリックします。
   - 完全なプロジェクトを選択した場合は、「Identity Manager プロジェクトの作成」の節に進んでください。
   - リモートプロジェクトを選択した場合は、「Identity Manager Project (Remote) の作成」の節に進んでください。

Identity Manager IDE は、さまざまなバージョンのサーバーと連携動作するようになりました。

各 Identity Manager IDE プロジェクトはある特定の Identity Manager バージョンに関連付けられるため、Identity Manager IDE では、互換性バンドル (Identity Manager JAR ファイルと、バージョン固有の情報を含むいくつかの XML レジストリを提供する ZIP ファイル) が、サポートされている Identity Manager バージョンごとに必要となります。

- Identity Manager Project の場合 : 互換性バンドルは idm.war ファイル内に含まれており、Identity Manager IDE はプロジェクトの設定中にこのファイルに自動的にアクセスします。
- Identity Manager Projects (Remote) の場合 : リモートプロジェクトでは war ファイルの場所を指定しないため、次の互換性バンドルの場所を提供する必要があります。

  <*Identity Manager* のインストールルート >/sample/ide-bundle.zip

注

現時点では、複数の Identity Manager バージョンを同じプロジェクトから管理することはできません。別のバージョンのサーバーに対して、単純なデバッグを実行したり単純な調整を施したりする必要がある場合には、そのサーバー用に別のリモートプロジェクト (34 ページを参照) を作成します。

また、Identity Manager IDE を使って既存のプロジェクトを新しいバージョンにアップグレードすることもできません。このタスクは、次のようにして手動で実行する必要があります。

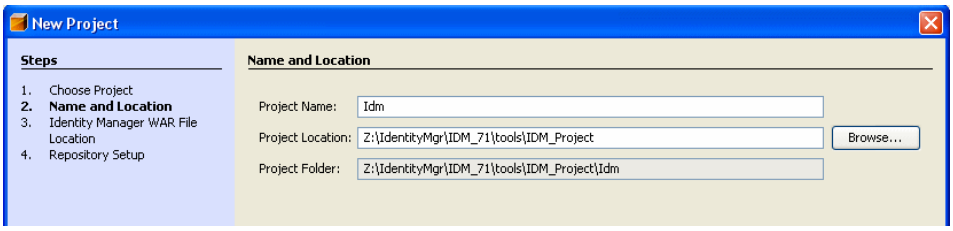
1. NetBeans を停止します。
2. すべてのオブジェクト、カスタム JSP、およびプロジェクトの idm-staging ディレクトリの内容を手動でアップグレードします。
3. 既存の nbproject/ide-bundle.zip を、アップグレード先のバージョンのもので置き換えます。

Identity Manager プロジェクトの作成

完全なプロジェクトの作成を完了するには、次の手順を実行します。

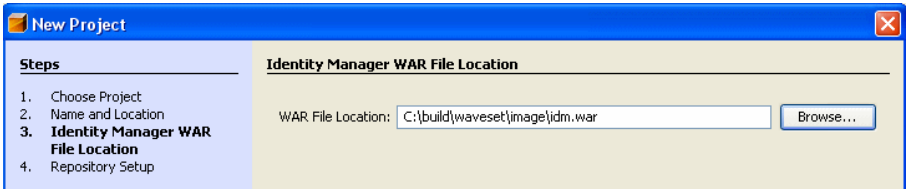
1. プロジェクトのデフォルトの名前、場所、および Identity Manager プロジェクトの格納先となるフォルダを含む「Name and Location」パネルが表示されます。
   - デフォルトの場所で問題ない場合は、「Next」をクリックします。
   - 別の場所を指定する場合は、デフォルトの情報を変更してから「Next」をクリックします。

図 1-16　新規プロジェクトウィザード : プロジェクト格納場所の指定



2. 「Identity Manager WAR File Location」画面が表示されたら、Identity Manager の `idm.war` ファイルへのパスを指定したあと、「Next」をクリックします。

図 1-17　新規プロジェクトウィザード : WAR ファイルの場所の指定



3. 「Repository Setup」画面 (図 1-18) が表示されたら、次のいずれかのリポジトリ設定オプションを選択します。

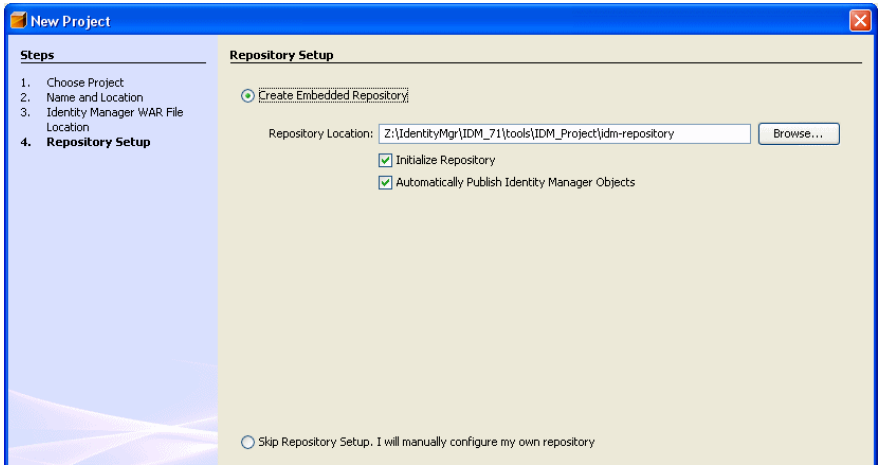
   - 組み込みリポジトリを Identity Manager IDE に作成させる場合は、「Create Embedded Repository」オプションを有効にし、手順 4 に進みます。

     注 「Create Embedded Repository」を選択すると、Identity Manager IDE によって、デフォルトの「sandbox」CBE ターゲットに対する組み込みリポジトリが作成されます。

     追加のターゲットを指定した場合、その追加ターゲットに対して「Manage Embedded Repository」も選択するか、あるいは `lh setup` または `setRepo` コマンドを実行する必要があります。

   - 既存のリポジトリを別のプロジェクトから再利用する場合には、「Skip Repository Setup」オプションを有効にし、33 ページの「リポジトリの手動設定」の手順に進みます。

図 1-18 新規プロジェクトウィザード : リポジトリの設定



4. 組み込みリポジトリのデータファイル用のディレクトリを指定します。

5. 既存のデータファイルを削除し、idm.war から init.xml ファイルをインポートするには、「Initialize Repository」を有効にします。

注	別のプロジェクトのリポジトリを再利用する場合は、このオプションを必ず無効にしてください。

6. プロジェクトを実行またはデバッグするたびにプロジェクト内のすべての Identity Manager オブジェクトをリポジトリに自動的にパブリッシュするには、「Automatically Publish Identity Manager Objects」を有効にします。

注	このオプションは、Identity Manager Project (Remote) の場合や独自のリポジトリを指定する場合は使用できません。

7. 「Finish」をクリックします。

   Identity Manager IDE がリポジトリの設定を行なっている間、「Creating Project」ダイアログが表示されます。Identity Manager IDE がデータファイルを削除し、init.xml ファイルをインポートするため、この処理には多少の時間がかかります。

   ダイアログが閉じると設定が完了し、BUILD SUCCESSFUL というメッセージが Identity Manager IDE の出力ウィンドウに表示されます。

エクスプローラウィンドウで「Files」タブを選択し、トップレベルのノードを展開すれば、新しいプロジェクトに関連付けられたファイルを確認できます。

注	新しいプロジェクトを作成すると、Identity Manager IDE によって自動的に、README.txt ファイルが作成されます。 プロジェクトのファイルシステムの詳しい概要を表示する (あるいは別の CBE で動作するようにプロジェクトの構造を変更する) には、ファイルウィンドウで README.txt をダブルクリックします。すると、エディタウィンドウに情報が表示されます。

リポジトリの手動設定

Identity Manager でサポートされている任意のタイプのリポジトリを使用するようにプロジェクトを設定することができます。

注 リポジトリの手動設定は、現在選択されている CBE ターゲットのリポジトリに影響を与えます。詳細は、37 ページの「Identity Manager インスタンスの設定」を参照してください。

設定手順はリポジトリのタイプごとに異なるため、次の例は、そのプロセスを示すために提供されたものです。

MySQL リポジトリを設定するには、次の手順を実行します。

1. `mysqljdbc.jar` をプロジェクトの `custom/WEB-INF/lib` ディレクトリにコピーします。
2. Identity Manager IDE のプロジェクトウィンドウで、*project name* を右クリックし、ポップアップメニューから「Run LH Command」を選択します。
3. 「Enter LH Command to Run」ダイアログが表示されたら、「LH Command」フィールドに **`setup`** と入力したあと、「OK」をクリックします。

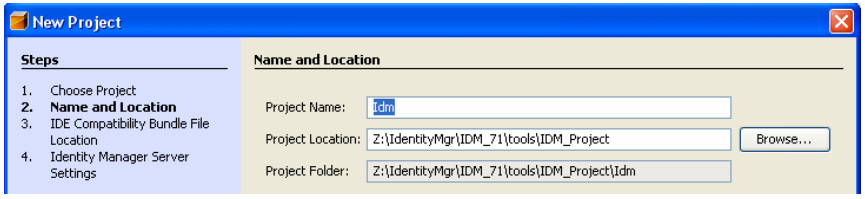
   出力ウィンドウの下に「Input」フィールドが表示され、いくつかの概要情報を含む Sun 設定ウィザードが表示されます。
4. 情報を読み終わったら、「Next」をクリックします。
5. 「Locate the Repository」画面が表示されたら、次の情報を入力したあと、「Next」をクリックします。
   a. メニューからリポジトリタイプを選択します。
   b. URL、JDBC ドライバ、および接続情報を確認し、必要であれば変更します。
6. 「Setup Demo?」画面が表示され、デモ環境の設定を継続するかどうかを尋ねられます。「No, I will configure Identity Manager myself」をクリックしてから「Next」をクリックします。
7. 「Save Configuration」画面が表示されたら、「Execute」をクリックすることで、`init.xml` ファイルのインポートと Identity Manager 設定の保存を行います。
8. インポートが完了したら、「Done」をクリックします。

Identity Manager Project (Remote) の作成

リモートプロジェクトの作成を完了するには、次の手順を実行します。

1. プロジェクトのデフォルトの名前、場所、および Identity Manager プロジェクトの格納先となるフォルダを含む「Name and Location」パネルが表示されます。
   - デフォルトの場所で問題ない場合は、「Next」をクリックします。
   - 別の場所を指定する場合は、デフォルトの情報を変更してから「Next」をクリックします。

図 1-19　新規プロジェクトウィザード : プロジェクト格納場所の指定



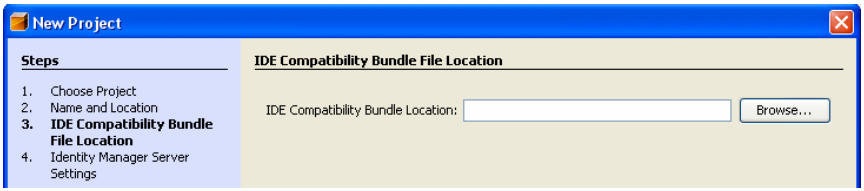
「Identity Manager IDE Compatibility Bundle File Location」画面が表示されます。

注　Identity Manager IDE はさまざまなバージョンのサーバーと連携動作でき、各 Identity Manager IDE プロジェクトはある特定の Identity Manager バージョンに関連付けられるため、IDE 互換性バンドル (Identity Manager JAR ファイルと、バージョン固有の情報を含むいくつかの XML レジストリを提供する ZIP ファイル) を、サポートされている Identity Manager バージョンごとに指定する必要があります。

2. 接続先サーバーのバージョンに対応する IDE 互換性バンドルを指定したあと、「Next」をクリックします。

   リモートプロジェクトの場合、IDE 互換性バンドルファイルは *<Identity Manager のインストールルート >*/sample/ide-bundle.zip 内に格納されています。

図 1-20　新規プロジェクトウィザード : IDE 互換性バンドルファイルの場所の指定



3. 「Identity Manager Server Settings」画面 (図 1-21) が表示されたら、次の情報を入力して開発用 Identity Manager インスタンスへの接続方法を定義したあと、「Finish」をクリックします。

注

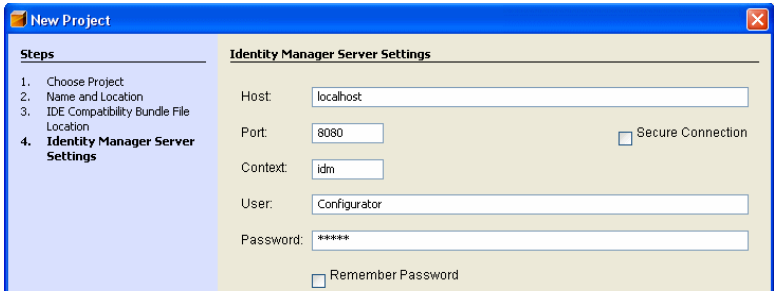
- Identity Manager IDE はリモート (SOAP) 接続をサポートします。
- リモート接続を設定する際、Identity Manager IDE はこの接続を使用してオブジェクトをアップロードまたはダウンロードします。

- 「Host」- プロジェクトが実行されているホストの名前を入力します。
- 「Port」: ディレクトリサーバーが待機する TCP ポートの番号を入力します。

  Identity Manager IDE で接続を開くときに SSL を使用する場合は、「Secure Connection」ボックスを有効にします。
- 「Context Path」: Identity Manager インストールのコンテキストルート (`/idm` など) を入力します。
- 「User」: 管理者のユーザー名を入力します。
- 「Password」: 管理者のパスワードを入力します。

  Identity Manager IDE がパスワードを記憶して、将来のセッションでパスワードが自動的に入力されるようにする場合は、「Remember Password」ボックスを有効にします。

図 1-21 新規プロジェクトウィザード : サーバー設定の指定



4. 「Finish」をクリックします。
   - リモートサーバーへの接続が成功すると、プロジェクトのノードが Identity Manager IDE のプロジェクトウィンドウ (左上端) に表示されます。
   - 接続に失敗すると、ウィザードのウィンドウ内にエラーメッセージが表示され、失敗に関する情報が提供されます。問題を解決し、「Finish」を再度クリックしてください。

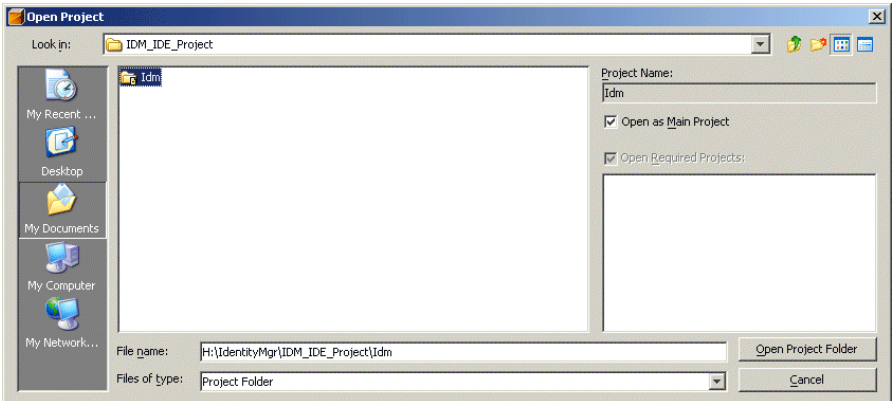
ヒント
- 組み込みサーバーに接続しているのではないことに注意してください。接続が失敗する場合は、リモートサーバーが動作していることを確認してください。
- 既存のプロジェクトのプロパティーを編集する際にも、Identity Manager IDE の接続をテストできます。

既存プロジェクトの選択

既存のプロジェクトを開くには、次の手順に従います。

1. 「File」>「Open Project」の順に選択します。
2. 「Open Project」ダイアログが表示されたら (図 1-22)、使用するプロジェクトフォルダを参照して「Open Project Folder」をクリックします。

図 1-22　「Open Project」ダイアログ



すでに 1 つまたは複数のプロジェクトが開かれている場合は、右上の「Open as Main Project」チェックボックスがアクティブになります。

3. このプロジェクトをメインプロジェクトとして指定する場合は、このボックスを有効にします。

4. 「Open Project Folder」をクリックすると、選択したプロジェクトがエクスプローラウィンドウに追加されます。

Identity Manager インスタンスの設定

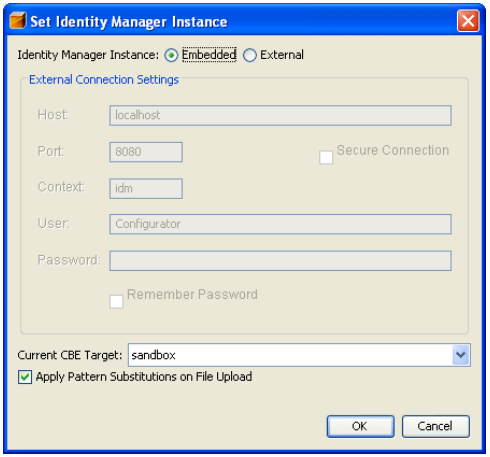
ある特定のプロジェクトのコンテキストで次のアクションをどの Identity Manager インスタンスに適用するかを制御するには、「Set Identity Manager Instance」機能を使用します。

- Debug Project
- Repository > Explore
- Repository > Open Project
- Checkout View
- Upload Object(s)
- Diff Object(s)
- Reload Object
- Test Form
- Test Rule

「Set Identity Manager Instance」ダイアログ (図 1-23) を開くには、次のいずれかの方法を使用します。

- NetBeans のメインメニューバーから、「IdM」>「Set Identity Manager Instance」の順に選択します。
- プロジェクトウィンドウからプロジェクトノードを右クリックし、ポップアップメニューから「Set Identity Manager Instance」を選択します。

図 1-23 「Set Identity Manager Instance」ダイアログ



プロジェクトの接続設定の確認

プロジェクトの外部接続設定を表示するには、「Identity Manager Instance」の次のオプションのいずれかを有効にします。

- Embedded (リモートオブジェクトでは利用不可能): 次のすべてのアクションを、NetBeans 組み込みアプリケーションサーバーに適用します。
  - Checkout View
  - Debug Project
  - Diff Object(s)
  - Reload Object
  - Repository > Explore
  - Repository > Open Project
  - Test Form
  - Test Rule
  - Upload Object(s)
- External: 上記アクションのすべてを、外部接続設定で指定されたサーバーに適用します。

現在の CBE ターゲットの設定

すべてのビルドアクション (「Build Project」、「Run Project」、および「Debug Project」) に対する現在の CBE ターゲットを指定するには、次の手順に従います。

1. 「Current CBE Target」メニューからターゲットを選択します。

注

選択されたターゲットは、各 `*-target.properties` ファイル内の `%%CUSTOM_SERVER_REPOSITORY%%` を指定することで、どのリポジトリが使用または設定されるかも制御します。

たとえば、CBE ターゲットを `dev` に設定し、「IdM」>「Repository」>「Manage Embedded Repository」の順に選択した場合 (あるいは「IdM」>「Run LH Command」の順に選択し、`setRepo` または `setup` と入力した場合)、Identity Manager IDE は `dev-target.properties` 内を検索して `%%CUSTOM_SERVER_REPOSITORY%%` を取得しますが、その内容は `nbproject/private/dev-ServerRepository.xml` です。次に Identity Manager IDE は、`nbproject/private/dev-ServerRepository.xml` を指定されたリポジトリに設定します。

- 「Build Project」アクションを実行した場合、生成された `war` ファイルには、`nbproject/private/dev-ServerRepository.xml` に等しい `ServerRepository.xml` が含まれています。
- ユーザーが「Run Project」または「Debug Project」アクションを実行すると、Identity Manager IDE は、`nbproject/private/dev-ServerRepository.xml` に等しいサーバーリポジトリを使って、NetBean の組み込みアプリケーションサーバー内の `war` ファイルを起動します。

---

2. 1 つ以上のファイルをリポジトリにアップロードするたびに、<*target-name*>`.properties` 内で定義されたパターン置換を Identity Manager IDE に適用させる場合は、「Apply Pattern Substitutions on File Upload」ボックスを有効にします (ここで、*target-name* は「Current CBE Target」で選択されたターゲットである)。

ターゲットが複数存在する場合のこの機能がどのように動作するかを確認するために、次のようにして、2 つのリポジトリを使って 2 つのターゲットを作成します。

1. 新しい (フル機能版の) Identity Manager プロジェクトを作成し、「Repository Setup」画面が表示されたら「Create Embedded Repository」を選択します。
2. 「Finish」をクリックすることで、リポジトリを作成し、現在のターゲットを「sandbox」に設定します。
3. アプリケーションサーバーを起動します。

「Projects」タブからプロジェクトノードを右クリックし、メニューから「Run Project」を選択します。

4. Identity Manager のブラウザが開いたらログインし、**`sandbox-user`** という名前のユーザーを作成します。

   アカウントの作成後に「List Accounts」タブを選択すると、`sandbox-user` が表示されるはずです。

5. アプリケーションサーバーを停止します。

   Identity Manager IDE の「Runtime」タブから「Servers」ノードを展開し、「Bundled Tomcat」を右クリックしたあと、ポップアップメニューから「Stop」を選択します。

6. サーバーが停止したら、Identity Manager IDE の「Projects」タブに戻ってプロジェクトノードを右クリックし、「Set Identity Manager Instance」を選択します。

7. 「Set Identity Manager Instance」ダイアログが表示されたら、「Current CBE Target」を *dev* に変更したあと、「OK」をクリックします。

8. プロジェクトのクリーンアップを確認するプロンプトが表示されたら、「Yes」を選択します。

9. 次に、dev ターゲット用のリポジトリを作成します。

   a. プロジェクトノードを右クリックし、メニューから「Repository」>「Manage Embedded Repository」の順に選択します。

   b. 最初のリポジトリで使用したのとは異なる場所を、dev ターゲット用として指定します。

   c. 「Initialize Repository」オプションを有効にしてから「OK」をクリックします。

10. 出力ウィンドウに `BUILD SUCCESSFUL` メッセージが表示されたら、プロジェクトノードを右クリックし、「Run Project」を選択します。

    NetBeans のログに、`Preparing image for environment target: dev` と記録されます。

11. Identity Manager にログインし、**`dev-user`** という名前の新しいユーザーを作成します。

    ここで「List Accounts」タブを選択すると、新しい `dev-user` が表示されるはずですが、`sandbox-user` は表示されません。

12. 元の `sandbox` ターゲットに切り替えます。

    a. サーバーを停止します。

    b. プロジェクトノードを右クリックし、「Set Identity Manager Instance」>「Current Target」>「sandbox」の順に選択します。

c. プロジェクトのクリーンアップを確認するプロンプトが表示されたら、「Yes」を選択します。

d. プロジェクトノードを右クリックし、メニューから「Run」を選択します。

13. Identity Manager にログインし、「List Accounts」を選択すると、`sandbox-user` が利用可能になっていることがわかります。

組み込みリポジトリの管理

現在のプロジェクトに対する次のリポジトリ設定を変更するには、「IdM」>「Repository」>「Manage Embedded Repository」の順に選択します。

注 組み込みリポジトリの管理は、現在選択されている CBE ターゲットのリポジトリに影響を与えます。詳細は、37 ページの「Identity Manager インスタンスの設定」を参照してください。

- Repository Location: 別のリポジトリ場所を指定します。
- Initialize Repository: これが有効になっていると、Identity Manager IDE は既存のデータファイルを削除し、`idm.war` から `init.xml` ファイルをインポートします。

  既存のプロジェクトのリポジトリを再利用する場合は、このオプションを無効にしてください。
- Automatically Publish Identity Manager Objects: これが有効化されると、ユーザーがプロジェクトを実行またはデバッグするたびに、Identity Manager IDE はプロジェクト内のすべての Identity Manager オブジェクトをリポジトリに自動的にアップロードします。

  注 このオプションは、Identity Manager Project (Remote) の場合や独自のリポジトリを指定する場合は使用できません。

リポジトリオブジェクトの操作

Identity Manager IDE では、BPE の場合のように直接リポジトリ内で作業するのではなく、ローカルファイルシステム上でリポジトリオブジェクトを操作できます。

ここでは、次の内容を説明します。

サポートされているオブジェクトタイプ

Identity Manager IDE では、次のオブジェクトタイプを操作できます。

- **設定オブジェクト** : フォームとワークフロープロセスを含む持続オブジェクト。
- **電子メールテンプレート** : さまざまな変更やアクションの通知をユーザーや管理者に送信するために使用されるテンプレート。たとえば、保留中のリクエストを承認者に通知したり、パスワードがリセットされたことをユーザーに通知したりするために、電子メールテンプレートを使用することができます。
- **フォーム** : Web ページに関連付けられ、ブラウザでユーザー表示属性をそのページにどのように表示するかについての規則が含まれているオブジェクト。フォームにはビジネスロジックを組み込むことができ、通常は、ユーザーに表示する前に、表示データを処理するために使用します。
- **汎用オブジェクト** : 名前と値のペアの単純なコレクションなどのビューを表すために通常使用されるオブジェクト。汎用オブジェクトは、タイプ `<Object>` の `<Extension>` を取ります。
- **ライブラリ** : 密接に関連するオブジェクト (通常は規則) をリポジトリで単一オブジェクトに編成するために使用されるオブジェクト。オブジェクトをライブラリに編成することで、ワークフローやフォームの設計者は、有用なオブジェクトを識別しやすくなります。
- **メタビュー** : Identity Manager において、メタビューはすべてのリソースの統合画面であり、一群のリソースを表示し、リソース上の属性がどのようなフローで処理されるかを表示するための共通データモデルを提供します。

- **規則** : XPRESS、XML オブジェクト、または JavaScript 言語で作成された関数を含む Identity Manager リポジトリ内のオブジェクト。Identity Manager 内では、規則は頻繁に使用されるロジックや、フォーム、ワークフロー、およびロール内で再利用される静的な変数を格納します。
- **ワークフロープロセス** : ワークフローは論理的で反復可能なプロセスであり、ドキュメント、情報、またはタスクが、ある関与者から別の関与者に渡されます。Identity Manager ワークフローは、アカウントの作成、更新、有効化、無効化、および削除を管理する複数のプロセスで構成されています。
- **ワークフローサブプロセス** : ワークフローに組み込むワークフローサブプロセスを作成するために使用するオブジェクト。

ビューのチェックアウト

Identity Manager IDE を使用することにより、編集のためにユーザービューなどの特定のビューをリポジトリからチェックアウトできます。

注	通常は、ビューのチェックアウト機能を使用してユーザーを更新しないでください。ビューの動作を理解する助けとして、この機能を紹介しているにすぎません。Identity Manager のユーザーを更新するには、Identity Manager Web インタフェースを使用します。

ビューをチェックアウトするには、次の手順に従います。

1. プロジェクトウィンドウで「Custom Identity Manager Objects」を右クリックし、ポップアップメニューから「Repository」>「Checkout View」の順に選択します。
2. 「Checkout View」ダイアログが表示されたら、メニューからビュータイプを選択し、ビュー名を入力したあと、「OK」をクリックします。

Identity Manager IDE で、ビューのコンテンツがツリー形式で開かれます。

編集後に、ほかの XML オブジェクトと同様に、ビューを右クリックしてリポジトリにチェックインします。

リポジトリからのオブジェクトの取得

Identity Manager IDE を使用することの利点の 1 つは、リポジトリの外でオブジェクトをダウンロードおよび変更できることです。また、必要であれば、リポジトリからオブジェクトを再読み込みし、ローカルファイルシステム上で変更したオブジェクトをそのオブジェクトで置き換えることもできます。

この節では、リポジトリからオブジェクトを取得するための方法をいくつか説明します。具体的には次のとおりです。

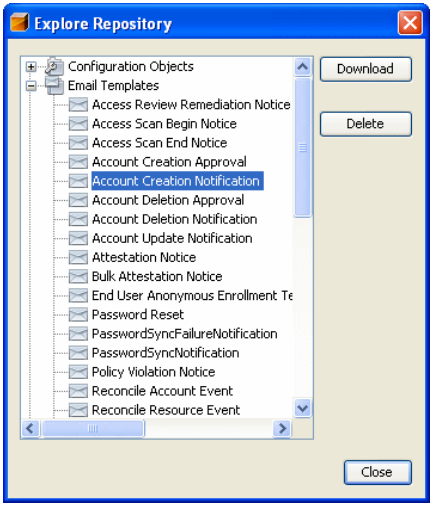
- オブジェクトのダウンロード
- オブジェクトを開く
- オブジェクトの再読み込み

オブジェクトのダウンロード

リポジトリからローカルファイルシステムへ初めてオブジェクトをダウンロードする場合は、次の手順に従ってください。

1. 「Custom Identity Manager Objects」ノードを右クリックし、「Repository」>「Explore」の順に選択すると、「Explore Repository」ダイアログが表示されます (図 1-24)。

図 1-24　「Explore Repository」ダイアログ



2. ノードを展開してリポジトリ内のオブジェクトを参照し、ダウンロードするオブジェクトをツリーから選択します。

ヒント	リストからオブジェクトを選択するときに Ctrl キーを押すことで、リポジトリから複数のオブジェクトをダウンロードできます。

3. 「Download」をクリックします。

選択したオブジェクトノード、およびオブジェクト内の個々の要素を表す子が、Identity Manager IDE によってエクスプローラ内の「Custom Identity Manager Objects」ツリーに追加されます。子ノードをダブルクリックすると、その XML 要素がエディタ内に、そのプロパティーシートがプロパティーウィンドウ内に、それぞれ表示されます。

オブジェクトを開く

ローカルファイルシステム上で開くオブジェクトの名前がすでにわかっている場合には、「Open Objects」オプションを使って大量のオブジェクト (リポジトリなど) の中から単一のオブジェクトをすばやく探してダウンロードします。

「Open Objects」オプションを使用するには、次の手順に従います。

1. IdM メニューから「IdM」>「Repository」>「Open Object」の順に選択します (あるいは、「Custom Identity Manager Objects」ノードを右クリックし、「Open Object」を選択する)。

   「Open」ダイアログが表示されます。

2. このダイアログ上のオプションを使ってオブジェクトの検索条件を指定します。

   a. Object Type: 検索するオブジェクトのタイプを選択します。

注	「Common Configuration Object Types」がデフォルトで選択されますが、これを使えば、デフォルトでシステム内に読み込まれるすべての設定オブジェクト (設定、電子メールテンプレート、フォーム、汎用オブジェクト、ライブラリ、規則、ワークフロープロセス、およびワークフローサブプロセス) を検索できます。 一般に、異なるオブジェクトタイプを検索したり、検索結果が最大結果数を超えたりしないかぎり、オブジェクトタイプを変更する必要はありません。

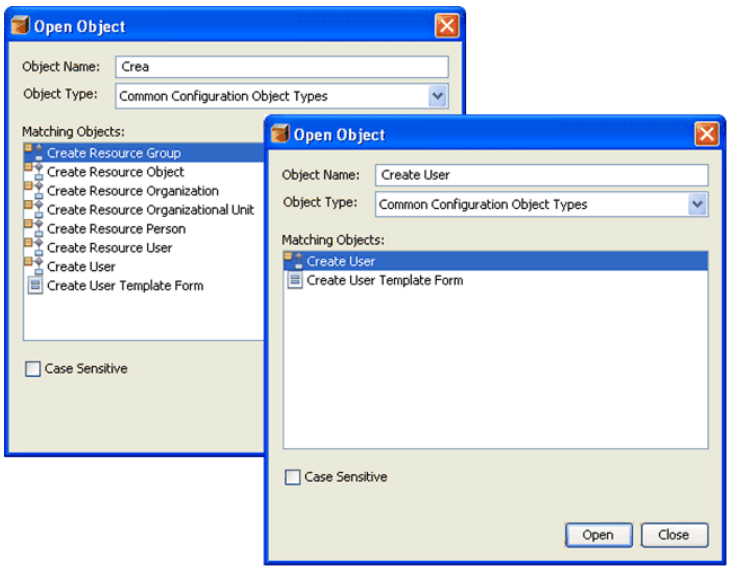
b. Object Name: オブジェクト名の一部または全部をこのテキストフィールドに入力します。

オブジェクト名としてアスタリスク (*) を指定すれば、リポジトリ内の特定タイプのすべてのオブジェクトにアクセスできます。

オブジェクト名の入力を始めると、一致する項目のリストが「Matching Objects」フィールドに表示されます。新しい文字を入力するたびに、表示される結果が絞られていきます。たとえば、「Create User」の一部として **`Creat`** と入力すると、多数の結果が表示されます (例については図 1-25 を参照)。

`Create` のようにエントリをさらに入力すると、条件がより具体的になるため、結果が絞られます。

図 1-25 「Open Object」ダイアログ



c. Case Sensitive: 「Object Name」フィールドに一致するオブジェクト名を検索する際に、大文字と小文字を区別する制約を加える場合に、このボックスを有効にします。

3. 「Matching Objects」リストから開くオブジェクトを選択したあと、「Open」をクリックするか Enter キーを押すと、Identity Manager IDE 内でそのオブジェクトが開きます。

注

- 「Matching Objects」リストは、最大で 1000 件のオブジェクトを返すことができます。検索結果がこの数を超えると、検索条件の見直しを促すメッセージが表示されます。
- リポジトリからまだダウンロードされていないオブジェクトを開こうとすると、Identity Manager IDE によって自動的に、そのオブジェクトがローカルファイルシステムにダウンロードされ、その後にそのオブジェクトがエディタ内で開かれます。

オブジェクト参照を開く

プロジェクトツリー内でノードを右クリックすると、特定の参照 (ObjectRef、FormRef、FieldRef、Workflow Subprocess など) の参照先となっているオブジェクトを開くことができます。ツリー内でオブジェクトを表現するオブジェクトノードを右クリックすると、ポップアップメニューから「Open *object name*」(ここで、*object name* はそれぞれ Reference、Form、Field、External Process のいずれか) を選択できます。

- 参照の参照先オブジェクトがローカルファイルシステム上に存在している場合、Identity Manager IDE は、そのローカルファイルシステム上のソースコードをソースエディタ内で開きます。
- オブジェクトのソースコードがまだローカルファイルシステム上に存在していない場合、Identity Manager IDE はそのオブジェクトをリポジトリから取得してから、そのソースコードをソースエディタ内で開きます。

たとえば、Default User Library オブジェクト参照を Tabbed User Form に追加するには、次の手順に従います。

1. Tabbed User Form をダウンロードし、「Includes」ノードが表示されるまでノードを展開します。
2. 「Includes」ノードを右クリックし、「Add an Object Reference」を選択します。

   ローカルファイルシステム上にまだオブジェクト参照が存在していないので、「Add Object Reference」ダイアログが表示されます。
3. あるオブジェクトを参照して選択することも、「Name」フィールドにオブジェクト名を直接入力することもできます。Default User Library オブジェクトを選択してから「OK」をクリックすると、その新しいオブジェクトがツリー内の「Includes」ノードの下に追加されます。
4. ここで、新しい Default User Library オブジェクト参照ノードを右クリックして「Open Reference」を選択すると、Identity Manager IDE によって、Default User Library オブジェクト参照のフォームがダウンロードされ、その XML ソースがソースエディタ内で開かれます。
5. ソースエディタ内のオブジェクト参照フォームを閉じます。

6. オブジェクト参照ノードを再度右クリックし、Default User Library 参照を開きます。

   Default User Library フォームがソースエディタ内で開かれますが、このフォームはすでにローカルファイルシステム上に存在しているため、ダウンロードされることはありません。

オブジェクトの再読み込み

現在のプロジェクト内のオブジェクトを、リポジトリにある同じオブジェクトの新しいバージョンで置き換えるまたは上書きするには、次の手順に従います。

1. オブジェクトノードを右クリックし、ポップアップメニューが表示されたら、「Repository」>「Reload Objects」を選択します。

   ヒント リストからオブジェクトを選択するときに Ctrl キーを押すことで、リポジトリから複数のオブジェクトをダウンロードできます。

2. 「Overwrite Files?」ダイアログが表示され、ローカルファイルシステムにそのオブジェクトがすでにあることを知らせるとともに、リポジトリから最新のコピーを読み込むかどうかが確認されます。
   - 「Yes」または「Yes for All」をクリックして続行します。
   - 続行しない場合は、「No」または「No for All」をクリックします。
3. 準備が完了したら、「Reload」ボタンをクリックします。

選択したオブジェクトノード、およびオブジェクト内の個々の要素を表す子が、エクスプローラに表示されます。子ノードをダブルクリックして、ノードの XML 要素に移動し、要素をプロパティーシートに表示することができます。

オブジェクト ID の削除

あるリポジトリから別のリポジトリへのオブジェクトの移動が容易に行えるよう、リポジトリからオブジェクトをダウンロードする前に自動生成されたすべてのリポジトリ ID を削除するように Identity Manager IDE を設定することができます。

注 一般に、正規表現パターンを変更する必要はなく、デフォルトで十分なはずです。この機能は、デフォルトパターンが破損した場合のために提供されています。

Identity Manager IDE は指定された表現に一致するすべてのオブジェクト ID とオブジェクト参照 ID を検索して削除しますが、その際、ハードコードされた定義済みの ID が削除されることはありません。

この機能を設定するには、次の手順に従います。

1. 「IdM」>「Options」の順に選択して「Repository Options」ダイアログを開きます。
2. 「ID Removal Pattern」フィールドに正規表現を入力したあと、「Use ID Removal」ボックスを有効にします。

注
- 「Use ID Removal」ボックスはデフォルトで有効になっています。
- Identity Manager IDE は正規表現を格納し、最後に使用された表現を「ID Removal Pattern」フィールドにデフォルトで表示します。このため、ユーザーは以前に使用した表現をすばやく再利用できます。

リポジトリへのオブジェクトのアップロード

新規または変更したオブジェクトをローカルファイルシステムからリポジトリにアップロードするには、次の手順に従います。

1. プロジェクトウィンドウで 1 つ以上のオブジェクトを選択します。

   ヒント リストからオブジェクトを選択するときに Ctrl キーを押すことで、アップロードするオブジェクトを複数選択できます。

2. ノードを右クリックし、ポップアップメニューから「Repository」>「Upload objects」の順に選択します。

   Identity Manager IDE は、選択されたオブジェクトを即座にアップロードします。

オブジェクトの自動アップロード

新しいプロジェクトを作成するときに、「Automatically Publish Identity Manager Objects」オプションを有効にすることができますが、これを有効にすると、プロジェクトを実行またはデバッグするたびに、Identity Manager IDE によってすべての Identity Manager オブジェクトがリポジトリに自動的にアップロードされます。

手順については、30 ページの「Identity Manager プロジェクトの作成」を参照してください。

プロジェクトの作成後にこのオプションを有効にするには、「Manage Embedded Repository」ダイアログ (「IdM」>「Repository」>「Manage Embedded Repository」) を使用します。

注	このオプションは、Identity Manager Project (Remote) の場合や独自のリポジトリを指定する場合は使用できません。

CBE パターン置換の使用

Identity Manager IDE では、組み込みまたは外部の Identity Manager インスタンスにオブジェクトをアップロードする際に、CBE (設定ビルド環境) パターン置換ファイルを使用できます。

メインメニューバーから「IdM」>「Set Identity Manager Instance」の順に選択すると、「Set Identity Manager Instance」ダイアログが表示されます。「Apply Pattern Substitutions on File Upload」を有効にすると、1 つ以上のオブジェクトをアップロードするたびに、*target-name*.properties (ここで、*target-name* は「Current CBE Target」で選択されたターゲット) 内に定義されたパターン置換が適用されます。

たとえば、sandbox-target.properties はデフォルトのターゲットであり、%%SMTP_HOST%%=mail.xyzcompany.com を定義しています。ユーザーは、sandbox-target.properties 内の mail.xyzcompany.com を自身の SMTP サーバーの名前で置き換えたあと、Identity Manager オブジェクトファイル内で %%SMTP_HOST%% 変数を必要に応じて使用できます。

注	• Identity Manager Project はサンプルの CBE を提供し、nbm は CBE パターン置換ファイルを認識します。*-target.properties ファイルは、CBE ビルドや publishIDMObjects ant ターゲットによっても使用されます。 • *-target.properties ファイルを別のディレクトリに移動してもかまいませんが、その場合、build.xml と idm-project.xml 内の cbeConfigDir 属性を更新してその変更が考慮されるようにする必要があります。

新しいオブジェクトの作成

プロジェクトに新規オブジェクトを作成するには、次の手順に従います。

1. プロジェクトウィンドウでオブジェクトノード (またはオブジェクトタイプノード) を右クリックして、「New」>「File/File Folder」の順に選択します。
2. 新規ファイルウィザードで、「Sun Identity Manager Objects」カテゴリを選択したあと、作成するオブジェクトタイプを選択します。「Next」をクリックします。
3. 「Name and Location」画面で、ファイル名を入力し、ファイルの格納先を指定します。

注	現在のプロジェクト、プロジェクトディレクトリ、およびオブジェクトフォルダがデフォルトで選択されますが、別の場所を指定することができます。

4. 「Finish」をクリックしてウィザードを閉じます。

Identity Manager IDE は、選択されたオブジェクトタイプノードの下の子として、新しいオブジェクトをツリーに追加します。また、新規オブジェクトの XML ソースがソースエディタビューに表示されます。

注	Identity Manager IDE では、各 XML ソースファイルにただ 1 つのオブジェクトだけが含まれていることを前提にしています。 そのため、複数のオブジェクトを含む XML ファイルを使用すると (`wfresource.xml` サンプルなど Identity Manager とともに配布されているファイルを含む)、オブジェクトはプロジェクトウィンドウにノードとして表示されず、パレット内で使用可能になりません。さらに、そのコンテキストメニューには Identity Manager IDE アクションが一切含まれません。 しかし、これらの XML ファイルをプロジェクトの「Objects」ディレクトリに配置すると、Identity Manager IDE はそれらのファイルをプレーンテキストとして扱うため、「File」ツリーで表示することができ、直接ファイルを開いて編集することもできます。

オブジェクトの編集

この節では、プロジェクト内のオブジェクトを編集する方法について説明します。この情報には次のものが含まれます。

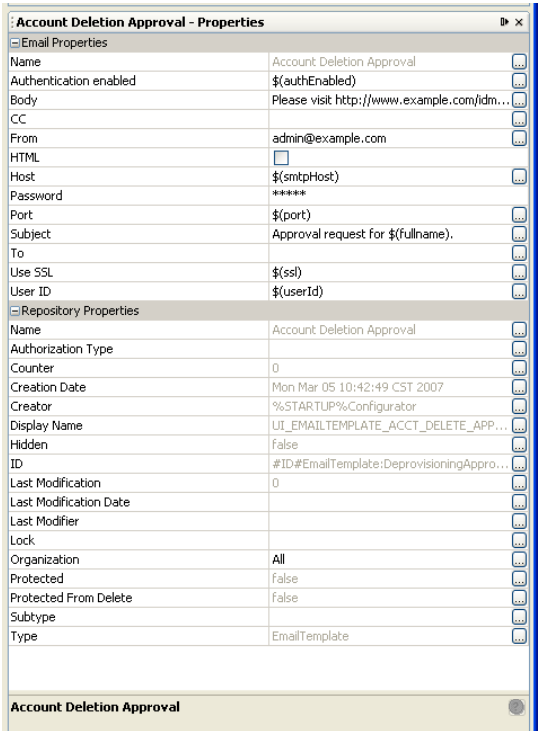
- オブジェクトプロパティーの編集
- オブジェクトへの要素の追加

オブジェクトプロパティーの編集

電子メールテンプレート、フォーム、ライブラリ、およびワークフローオブジェクトのプロパティーを変更するには、Identity Manager IDE プロパティーウィンドウを使用します。

たとえば、「Account Deletion Approval」オブジェクトノードをクリックすると、次のプロパティーシート (図 1-26) がプロパティーウィンドウ内に表示されます。

図 1-26　プロパティーの例



ヒント　プロパティーウィンドウを使用しない方法として、オブジェクトノードを右クリックし、ポップアップメニューから「Properties」を選択する方法があります。表示される「Properties」ダイアログには、プロパティーウィンドウの場合と同じオブジェクトプロパティーシートが含まれます。

プロパティーシートには、次の機能の一部または全部が含まれます。

- **テキストフィールド**: 単純値または複合値を次のように編集できるようにします。
  - **単純値フィールド**: 新しい文字列または整数値をテキストフィールドに直接入力します。値を変更したり新しい値を追加したあとで、次のようにします。
    - 変更を適用するには、Enter キーを押すか別のテキストフィールドをクリックします。
    - 変更を取り消すには、Esc キーを押します。
  - **複合値フィールド**: 既存の値をクリックすると「Expression Builder」ダイアログ (54 ページの「「Expression Builder」ダイアログの使用」を参照) が開き、式を編集できるようになります。

注	グレーのテキストは読み取り専用のフィールドです。

- **省略記号 (...) ボタン**: テキストフィールドに表示されたオブジェクト値を編集するための代替手段を提供します。
  - **単純値フィールド**: ボタンをクリックするとダイアログが開き、新しい文字列または整数値を入力できるようになります。値を変更したり新しい値を追加したあとで、次のようにします。
    - 変更を適用してダイアログを閉じるには、「OK」をクリックします。
    - 変更を取り消してダイアログを閉じるには、「Cancel」をクリックします。
  - **複合値フィールド**: ボタンをクリックすると「Expression Builder」ダイアログが開き、式を編集できるようになります。

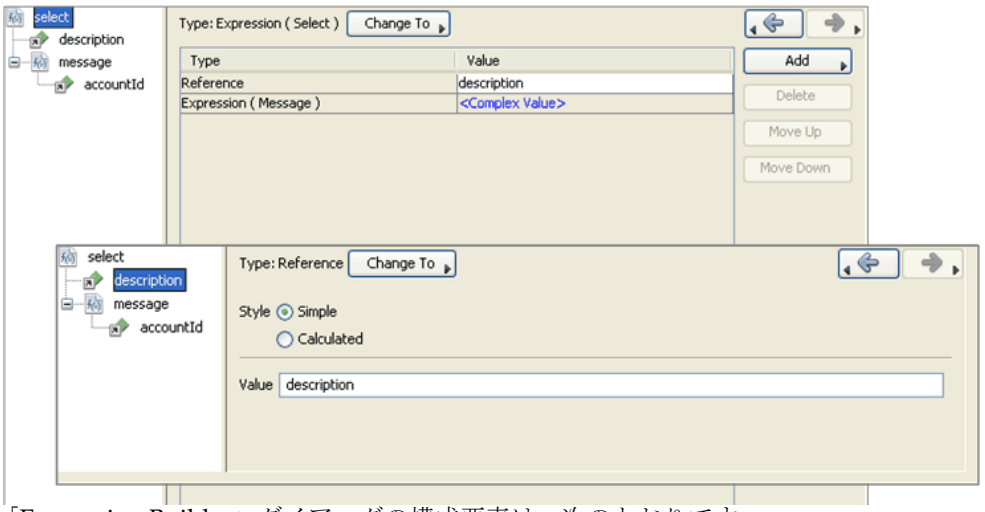
注	グレーのテキストは読み取り専用のフィールドです。

- **ドロップダウンメニュー**: いくつかの具体的な値を含むリストから値を選択するか、プロパティー名をダブルクリックして使用可能な値を順番に切り替えます。
- **チェックボックス**: クリックしてプロパティーを有効または無効にします。

「Expression Builder」ダイアログの使用

プロパティーウィンドウ内での操作中に複合値フィールドをクリックするか省略記号ボタンをクリックすると、図 1-27 に示すような、オブジェクトの XML 式を編集するための「Expression Builder」ダイアログが開きます。

図 1-27 「Expression Builder」ダイアログの 2 つの例



「Expression Builder」ダイアログの構成要素は、次のとおりです。

- オブジェクトの XML 式の階層表示を提供するツリービュー区画。

  また、このビュー内で要素を右クリックして式を編集するためのオプションにアクセスすることもできます。

- 選択された式要素に対する一連の編集オプションを提供する編集オプション区画。
- 選択された式に関係する一般情報や JavaDoc 情報を提供する情報区画 (ほとんどの編集オプション区画の最下部に配置される)。

注 規則のデザインビューでも式ビルダー (図 1-6 を参照) が提供されていますが、これは、ユーザーが規則の論理構造を確認したり規則のプロパティーを変更したりしやすくするためです。

表 1-5 では、さまざまな編集オプションについて説明します。

注 この表に記載されている式タイプの詳細については、『Identity Manager ワークフロー、フォーム、およびビュー』を参照してください。

表 1-5 式ビルダーのオプション

編集オプション	説明
• 「Add」ボタン	次のいずれかの式タイプを選択して新しい式要素を追加します。
• 「Add」メニューオプション (ツリービューで要素を右クリックした場合にのみ利用可能)	• XPRESS Object > Select String、List、Map、または Integer。 • Reference • Expression > Select Logical > *Logical type* 、Strings > *String type*、Lists > *List type*、Variables > *Define a variable*、Math > *Math type*、Control > *Control type*。 • Object Access > Select New、Invoke (Static または Instance)、または Get。 • Diagnostics > Select Script、Trace、または Print。 • Rule • XML Object > Select String、List、Map、または Integer。
• 「Delete」ボタン • 「Delete」メニューオプション (ツリービューで要素を右クリックする)	選択された式を削除します。
「Move Up」/「Move Down」ボタン	要素を XML 式内で上方または下方に移動します。

表 1-5　式ビルダーのオプション (続き)

編集オプション	説明

「Wrap Object」オプション (ツリービューで要素を右クリックした場合にのみ利用可能)	ある式要素 (`string` など) を別の式要素 (`concat` など) の内側にラップします。
	このオプションは、すでに定義済みの引数を必要とする関数呼び出しを含めるのを忘れた場合に役立ちます。たとえば、`firstname` 属性への参照を定義し、その参照を `concat` 内にラップした場合、`secondname` 属性への参照を `concat` の引数として定義でき、その結果は次のようになります。
	`<concat>`
	`<ref>firstname</ref>`
	`<ref>secondname</ref>`
	`</concat>`

• 「Change To」ボタン	要素の式タイプを次のいずれかに変更します。
• 「Change To」メニューオプション (ツリービューで要素を右クリックする)	• XPRESS Object > Select String、List、Map、または Integer。 • Reference • Expression > Select Logical > *Logical type* 、Strings > *String type*、Lists > *List type*、Variables > *Define a variable*、Math > *Math type*、Control > *Control type*。 • Object Access > Select New、Invoke (Static または Instance)、または Get。 • Diagnostics > Select Script、Trace、または Print。 • Rule • XML Object > Select String、List、Map、または Integer。
矢印ボタン 	最近アクセスした式編集パネル間を移動します (ブラウザの「戻る」/「進む」ボタンに類似)。
「Type」/「Value」テーブル	選択された式内で現在使用されている式タイプを表示および編集します。 • **「Type」列** (読み取り専用): 式タイプを一覧表示します。 • **「Value」列** (編集可能): 式タイプの値を一覧表示します。 単純な文字列または整数を編集するには、「Value」フィールドに新しい値を入力します。 「<Complex Value>」を編集するには、フィールドをクリックしてその式に対する「Expression Builder」ダイアログを開きます。
「Style」ボタン	式のスタイルを「Simple」、「Calculated」のいずれかに設定します。
「Value」フィールドと「<> Name」フィールド	現在の単純な文字列または整数を置換します。

表 1-5　式ビルダーのオプション (続き)

編集オプション	説明
「Trace」ボックス	IDM 内で XPRESS が動作している間、トレース診断を出力します
「Browse JavaDoc」ボタン	呼び出し式の作成時に JavaDoc 情報を表示します。 このボタンをクリックすると「JavaDoc」ダイアログが開き、「Class Names」メニューと「Method Names」メニューから項目を選択できるようになります。あるメニュー項目をクリックするか、その項目上でマウスを静止させると、関連する JavaDoc 情報を含むポップアップが表示されます。

式を編集するには、次のいずれかを行います。

- ツリービューで要素を選択したあと、編集オプション区画の各種オプションを使用します。

  編集オプションの中には、「Change To」ボタンのように大部分の式要素に共通するものもあれば、特定の要素の編集時にしか利用できないものもあります。

- ツリービューで要素を右クリックしたあと、ポップアップメニューからオプションを選択します。

オブジェクトへの要素の追加

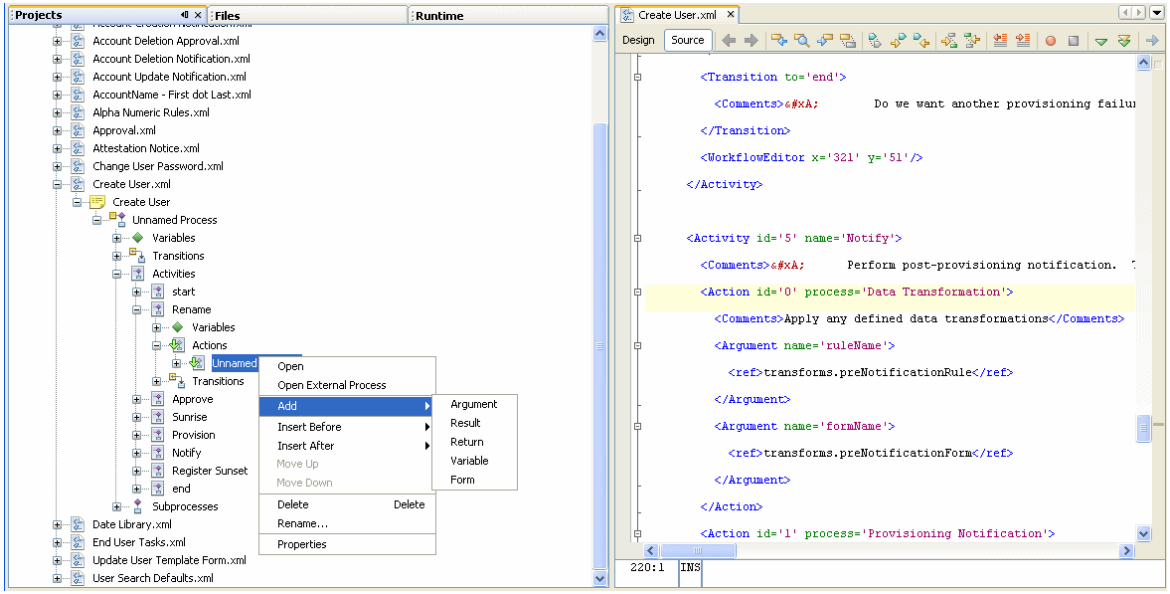
オブジェクトに新しい要素 (子要素) を追加するには、XML ソースを直接編集するか、ポップアップメニューの「Add」オプションを使用します。

注　プロパティーウィンドウでは、オブジェクトプロパティーの編集は行えますが、要素の追加は行えません。

「Add」オプションを使用するには、次の手順に従います。

1. 子要素の追加先となるオブジェクトノードを右クリックし、ポップアップメニューから「Add *element_type*」を選択します。

図 1-28　アクションへの要素の追加

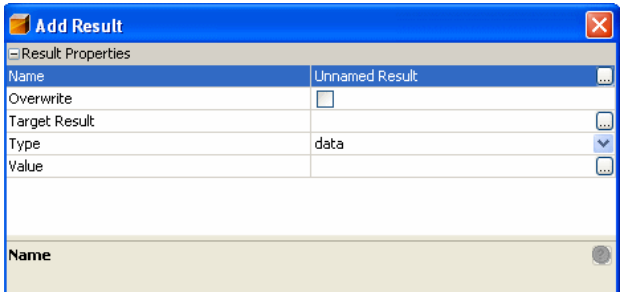


注　オブジェクトタイプによって、どの *element_type* を追加できるかが決まります。たとえば、ワークフローのプロセスまたはサブプロセスを操作している場合であれば、単純な計算を実行したり Java クラスや JavaScript を呼び出して複雑な処理を行なったりするために、「アクション」を追加できます。続いて、そのアクションオブジェクトに引数、フォーム、結果、戻り値、または変数を追加できます。

2. 「Add *element_type*」ダイアログが表示されたら、プロパティーシートを使って新しい要素の名前やその他のプロパティーを指定します (51 ページの「オブジェクトプロパティーの編集」を参照)。

   たとえば、アクションオブジェクトに結果を追加する場合には、「Add Result」ダイアログが表示されます。

図 1-29　「Add Result」ダイアログ



3. 「OK」をクリックすると、Identity Manager IDE によって、選択されたオブジェクトノードの下に新しい要素が追加されます。また、新規オブジェクトの XML ソースがソースエディタビューに表示されます。

注

あるフォームにインクルード要素を追加する場合、フォームの名前を参照して選択することも、名前を直接入力することもできます。

フォーム名を入力する場合は、次の構文を使用する必要があります。そうしないとエラー (`Error creating Object Reference`) が発生します。

<FormType>`:`*<SomeFormName>*

たとえば、AIX User Form を Default User Form に追加するには、次の手順に従います。

1. `Default User Form.xml` をダウンロードし、「Includes」ノードが表示されるまでツリーを展開します。
2. 「Includes」ノードを右クリックし、「Add Object Reference」を選択します。
3. 「Add Object Reference」ダイアログが表示されたら、「Name」フィールドに AIX User Form を入力することも、「Browse」をクリックしてフォームを特定および選択することもできます。
4. 名前を入力する場合は、**`UserForm:AIX User Form`** と入力する必要があります。

オブジェクトの削除

ローカルファイルシステム上のプロジェクトからオブジェクトを削除するには、次の手順に従います。

1. プロジェクトウィンドウで 1 つ以上のオブジェクトを右クリックします。
2. ポップアップメニューから「Delete」を選択します。

   オブジェクトが即座にプロジェクトから削除されます。

Identity Manager IDE リポジトリからオブジェクトを削除するには、次の手順に従います。

1. NetBeans メニューバーで「IdM」>「Repository」>「Explore」の順に選択します。
2. 「Explore Repository」ウィンドウが表示されたら、オブジェクトノードを展開して削除するオブジェクトを見つけます。
3. 1 つ以上のオブジェクトを選択したあと、「Delete」ボタンをクリックします。

注	この「Delete」ボタンを使って個々の規則をライブラリから削除することはできません。1 つ以上の規則をライブラリから削除するには、ライブラリの全体をローカルファイルシステムにダウンロードし、規則を削除したあと、ライブラリを元のリポジトリにアップロードする必要があります。

削除されたすべてのオブジェクトは、リポジトリから即座に削除されます。

差分オプションの使用

Identity Manager IDE では、ローカルファイルシステム上のディレクトリまたは単一オブジェクトとリポジトリ内の対応するオブジェクトとの比較 (差分処理) を行なったあと、その差分を左右に並べて表示することができます。

グローバルな差分オプションの設定

「IdM」>「Options」を選択すると、「Repository Options」ダイアログが開きます。このダイアログでは、次のグローバルな差分処理オプション (デフォルトではすべて有効) を指定できます。

- Exclude lastModDate: オブジェクトの差分処理時に Identity Manager IDE が `lastModDate` の変更を除外するかどうかを制御します。

- Exclude ids: オブジェクトの差分処理時に Identity Manager IDE が自動生成されたすべてのリポジトリ ID を除外するかどうかを制御します。
- Apply pattern substitution: オブジェクトの差分処理時に Identity Manager IDE が `*-target.properties` ファイル (設定ビルド環境 (CBE) パターン置換ファイル) を適用するかどうかを制御します。

注

- 「Exclude lastModDate」および「Exclude ids」オプションは、リポジトリ側のオブジェクトにしか影響を与えません。
- 上記オプションのいずれかを変更すると、Identity Manager IDE のアクティブなウィンドウがユーザーの選択に基づいて自動更新されます。

  たとえば、「Identity Differences」タブの「Options」をクリックして「Repository Options」ダイアログを開き、「Exclude ids」オプションを無効にした場合、「Identity Differences」タブに表示された情報は Identity Manager IDE によって自動更新されますが、エディタ内の差分ウィンドウは更新されません。

オブジェクトの差分処理

オブジェクトに対するすべての変更を保存し終わると、次の各節で説明するように、あるディレクトリまたは単一オブジェクトの内容に対して差分処理を行えます。

- ディレクトリの差分処理
- 単一オブジェクトの差分処理

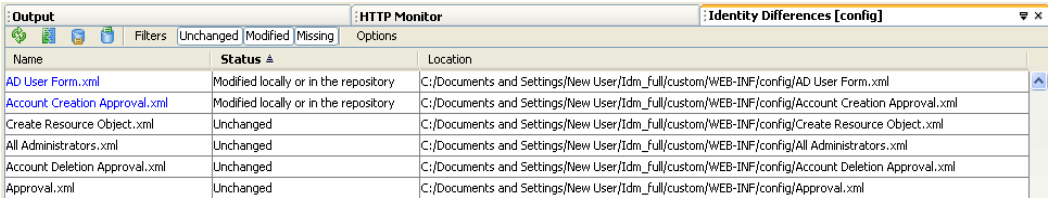
ディレクトリの差分処理

あるディレクトリの内容に対して差分処理を行うには、次のようにします。

- メニューバーから「IdM」>「Repository」>「Diff Objects」の順に選択します。
- 「Custom Identity Manager Objects」などのディレクトリノードを右クリックし、「Repository」>「Diff Objects」の順に選択します。

Identity Manager IDE によってすべてのオブジェクトの差分処理が、ファイルシステム内で下方に再帰的に繰り返されたあと、その結果が「Identity Differences」タブ (図 1-30) に表示されます。

図 1-30 「Identity Differences」タブ



このタブの構成要素は次のとおりです。

- 差分処理されたすべてのファイルの名前、状態、および場所の一覧を含むテーブル。

  このテーブルでは、色分けされたファイル名を使って、変更されたファイル (青色) や新規でまだサーバー上に存在していないファイル (緑色) が示されます。

ヒント このテーブル内の変更された項目をダブルクリックすると差分出力ウィンドウが開き、差分を左右に並べて確認することができます。

- 表 1-6 で説明するアイコンとボタン

表 1-6 「Identity Differences」タブ : 差分アイコン / ボタン

クリックするアイコンまたはボタン	実行されるタスク
Refresh	ローカルファイルシステムからオブジェクトのすべての変更内容を取得し、それらのオブジェクトの差分処理を再実行したあと、「Identity Differences」タブの差分情報を再表示します。 **重要 :** 「Reload All」、「Upload All」、または「Diff All」オプションを使用する前に、「Refresh」をクリックして「Identity Differences」タブに一覧表示されたオブジェクトの最新の変更内容のすべてを取り込む必要があります。
Diff All	選択されたディレクトリに含まれるすべてのオブジェクトの差分処理を行います。 あるディレクトリの内容に対する差分処理を行うと、その追加処理として、Identity Manager IDE によって、ローカルまたはリポジトリ内で変更されたすべてのオブジェクトの名前がエディタのドロップダウンメニューに設定されます。このリストからあるオブジェクトを選択すると、その差分ウィンドウがエディタ内に表示されます。
Reload All	「Identity Differences」タブに一覧表示されたすべての変更済みオブジェクトを、リポジトリ内の対応するオブジェクトで置き換えます。
Upload All	「Identity Differences」タブに一覧表示されたすべての変更済みオブジェクトを、リポジトリにアップロードします。

表 1-6　「Identity Differences」タブ : 差分アイコン / ボタン (続き)

クリックするアイコンまたはボタン		実行されるタスク
		フィルタボタン : これらのボタンの任意の組み合わせを選択することで、「Identity Differences」テーブルにおける結果の表示方法を制御します。
Unchanged		• Unchanged: 変更されていないすべての結果を、リストからフィルタリングします。
Modified		• Modified: ローカルまたはリポジトリ内で変更されたすべての結果を、リストからフィルタリングします。
Missing		• Missing: リポジトリ内に存在しないすべての結果を、リストからフィルタリングします。
Options	**Options**	60 ページの「グローバルな差分オプションの設定」で説明したグローバルな差分処理オプションを指定する場合にクリックします

注　すべてのオブジェクトの再読み込み、アップロード、または差分処理を行う前に、「Refresh」をクリックして「Identity Differences」タブに一覧表示されたオブジェクトの最新の変更内容を取得する必要があります。「Refresh」を実行しなかった場合、「Identity Differences」タブに表示された古いバージョンのオブジェクトが Identity Manager IDE によって使用されることになります。

「Identity Differences」タブのいずれかのアイコンまたは「Options」ボタンを使用すると、「Identity Differences」タブに表示された情報は Identity Manager IDE によって自動更新されますが、エディタ内の差分ウィンドウは更新されません。

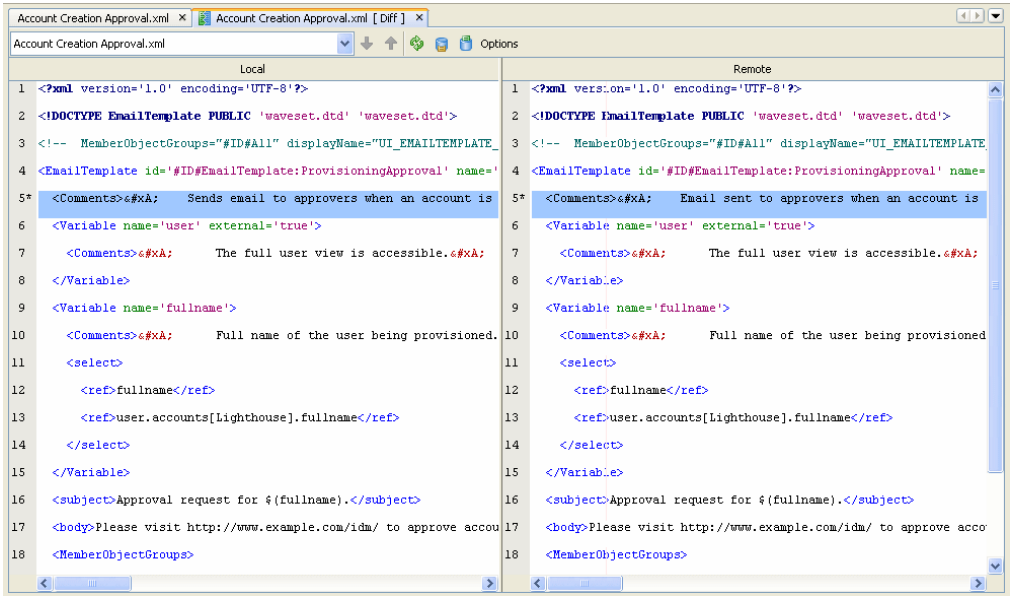
単一オブジェクトの差分処理

ある単一オブジェクトの内容に対して差分処理を行うには、次のようにします。

- メニューバーから「IdM」>「Repository」>「Diff Object」の順に選択します。
- オブジェクトを右クリックし、「Repository」>「Diff Object」の順に選択します。

Identity Manager IDE のエディタウィンドウが更新され、図 1-31 に示すように、そのオブジェクトのローカルビューとリモートビューが表示されます。

図 1-31　左右に並べて表示した例



また、エディタウィンドウには、表 1-7 で説明するアイコンとボタンも備わっています。

表 1-7　エディタの差分アイコン / ボタン

クリックするアイコンまたはボタン		実行されるタスク
	Refresh	ローカルファイルシステムからオブジェクトのすべての変更内容を取得し、オブジェクトの差分処理を再実行したあと、エディタ内の差分情報を再表示します。 **重要**：「Replace local with repository」または「Upload local to repository」オプションを使用する前に、「Refresh」をクリックしてそのオブジェクトに対する最新の変更内容を含めておく必要があります。
	Replace local with repository	変更済みのローカルオブジェクトをリポジトリ内の対応するオブジェクトで置き換えます。
	Upload local to repository	変更済みのローカルオブジェクトをリポジトリにアップロードします。
Options	**Options**	60 ページの「グローバルな差分オプションの設定」で説明したグローバルな差分処理オプションを指定する場合にクリックします

注 差分処理ウィンドウが開いていて、かつユーザーがエディタのいずれかのアイコンまたは「Options」ボタンをクリックした場合、差分処理ウィンドウ内に表示された情報は Identity Manager IDE によって自動更新されますが、「Identity Differences」タブは更新されません。

XML の操作

この節では、ソースエディタウィンドウでの XML の操作方法について説明します。これらの情報は、次のように構成されています。

- XML の編集
- 自動補完の使用
- 不正な形式の XML の識別と修正
- XML の妥当性検査

XML の編集

あるオブジェクトの XML を編集するには、次の手順に従います。

1. プロジェクトウィンドウまたはファイルウィンドウでオブジェクトノードをダブルクリックします。
2. デフォルトでソースエディタビューが表示されない場合は、「Source」ボタンをクリックします。

オブジェクトのフィルタリングされていない XML がソースエディタに表示されたら、必要に応じて XML を編集、検査、および妥当性検査できます。

注 プロジェクトウィンドウまたはファイルウィンドウにまだ表示されていないオブジェクトを編集するには、「IdM」>「Repository」>「Open Objects」の順に選択します。「Open Objects」ダイアログでオブジェクト名とオブジェクトタイプの全部 (または一部) を指定することで、ローカルファイルシステム上でそのオブジェクトを検索します。

リポジトリからダウンロードされていないオブジェクトを開こうとすると、Identity Manager IDE によって自動的に、そのオブジェクトがローカルファイルシステムにダウンロードされ、その後にそのオブジェクトがエディタ内で開かれます。

Identity Manager IDE には次のように、オブジェクトの XML を編集するためのオプションがいくつか用意されています。

- ソースエディタウィンドウで情報を直接入力する。
- プロジェクトウィンドウでオブジェクトノードを右クリックし、ポップアップメニューから「Add」ダイアログにアクセスし、そこで必要な情報を入力する。
- ワークフローライブラリおよび XPRESS カテゴリの項目が表示されたパレットウィンドウを使用する。項目を「Palette」からソースエディタの行にドロップして、XML ソース内のそのポイントに XML テキストを作成できます。
- ソースエディタに表示された XML ファイル内で右クリックし、インスタンスまたは static 呼び出し文をを挿入する。詳細については、20 ページの「呼び出しの挿入」を参照してください。

自動補完の使用

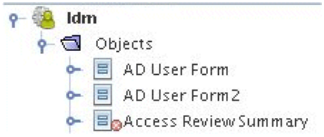
Identity Manager IDE はインストール時に `waveset.dtd` 定義ファイルを登録し、これによって XML 要素と属性の自動補完が可能になります。

自動補完機能を使用するには、XML 要素または属性の入力を開始し、それから Ctrl+Spacebar を押します。NetBeans は、要素を自動補完するために使用できる要素と属性のリストを表示します。

不正な形式の XML の識別と修正

Identity Manager IDE に読み込んだファイルに不正な形式の XML が含まれている場合、またはファイルに加えた編集が定義 (DTD) ファイルに従っていない場合には、プロジェクトウィンドウとファイルウィンドウでオブジェクトの最上位の親に赤色のエラーアイコンが表示されます。

図 1-32　エラーアイコン



プロパティーウィンドウが開いている場合、そのウィンドウの説明領域にエラーメッセージが表示されます。あるいは、無効なファイルのツールヒント情報を使ってエラーメッセージを表示させることもできます。それには、そのファイルのノードの上でマウスを静止させます。

オブジェクトに不正な形式の XML が含まれていると、次のようになります。

- オブジェクトの子ノードが表示されません。
- 「Add」、「Clone」、および「Upload」コマンド、一部のコンテキストメニューとボタンが無効になります。
- 「Form」、「Rule」、または「Library」ノードに対して、フォームのテストまたは規則のテスト機能が使用できません。
- XML エディタに直接追加された XML は失われませんが、追加されたオブジェクトのノードは XML を修正するまで表示されません。

ソースエディタウィンドウで右クリックして、ポップアップメニューから「Validate XML」を選択すると、形式が不正な XML ソースへのリンクを含む結果が出力ウィンドウに表示されます。このリンクをクリックすると、Identity Manager IDE がソースエディタウィンドウで不正な形式の XML の次の行を強調表示するため、容易に問題を検出して修正することができます。

XML の妥当性検査

ソースエディタで右クリックしてコンテキストメニューから「Validate XML」を選択することにより、オブジェクトの XML をただちに妥当性検査できます。

出力ウィンドウで結果を確認します。次のようなメッセージが表示されるはずです。

```
XML validation started.

file:/H:/IdentityMgr/IDM_IDE_Project/Idm/objects/
System%20Configuration.xml...

XML validation finished.
```

Identity Manager IDE デバッガの操作

Identity Manager IDE には、Identity Manager のフォーム、規則、およびワークフローのデバッグに使用できるグラフィカルなデバッガが用意されています。このデバッガを使用して、ブレークポイントとウォッチポイントの設定、コードのステップスルー、変数の検査と変更、クラスと呼び出しスタックの検査、スレッドの監視、および複数セッションの実行を行えます。

ここでは、Identity Manager IDE デバッガの使用法を説明します。手続き型プログラミング言語のコードデバッガを使用した経験があれば、この節で使用されている用語の理解は難しくありません。説明する内容は次のとおりです。

- デバッガの開始
- ブレークポイントの設定
- ウォッチポイントの使用
- 実行プロセスのステップスルー
- フォームのデバッグ
- 規則のテスト
- ワークフローのデバッグ
- テスト環境の外部でのデバッガの実行
- Identity Manager IDE チュートリアルの使用 : フォーム、規則、およびワークフローのデバッグ
- デバッガの停止

警告

Identity Manager IDE デバッガはデフォルトで有効になっていますが、デバッグ終了後には必ずデバッガを無効にし、他のユーザーが誤って本稼働環境のアプリケーションサーバーに接続することのないようにします。

デバッガを無効にする手順については、100 ページの「デバッガの無効化」を参照してください。

注 Identity Manager IDE デバッガを使用するときには、次の事柄に注意します。

- 本稼働環境でデバッガを使用しない。

  ブレークポイントの設定はグローバルな設定です。したがって、Identity Manager IDE はブレークポイントに達すると、着信リクエストスレッドを中断します。

- プロジェクトのプロパティーに指定されるユーザーは、Identity Manager 管理者機能を持っている必要がある。

  ただし、デバッガは、ほかのユーザーをシステムからロックアウトし、ほかのユーザーのセッションからの重要なデータを含む変数を表示するスレッドを中断できることに注意してください。この権限を悪用すると多大な影響があることを考慮して、権限を割り当てるときには十分に注意してください。

- デバッガを実行するユーザーにアプリケーションサーバーのプライベートコピーを割り当てる必要がある。

  複数のユーザーが同じアプリケーションサーバー上で開発作業を行なっており、あるユーザーがデバッガをそのサーバーに接続した場合、それ以外のユーザーはブレークポイントに到達してロックアウトされます。

- デバッガはクラスタをサポートしません。

- ワークフロー、フォーム、およびビューの詳細については、『Sun Java™ SystemIdentity Manager ワークフロー、フォーム、およびビュー』のマニュアルを参照してください。

デバッガの開始

Identity Manager IDE デバッガを開始するには、メインメニューバーから「Run」>「Debug Main Project」の順に選択します。

デバッグセッションを開始すると、Identity Manager IDE はプログラムに関するランタイム情報を表示する一連のデバッガウィンドウを自動的に開きます。これらのウィンドウで次の操作ができます。

- デバッグプロセスの開始と停止
- プロセス実行のナビゲーション
- ブレークポイントの設定 (プロセス実行での個別の停止ポイント)

ブレークポイントの設定

コードの特定の行を実行する前にオブジェクトの実行を停止するには、デバッガの *breakpoint* コマンドを使用します。

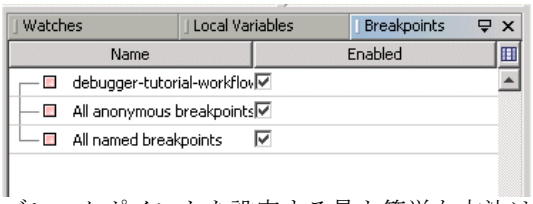
注 ブレークポイントの設定は、グローバルな設定となることに注意してください。デバッガは指定されたブレークポイントに達すると、着信リクエストスレッドを中断します。このアクションは、どのユーザーがリクエストを作成したかにかかわらず実行されます。

デバッガ使用中は、フォームやワークフローをどこで起動したかにかかわらず、ブレークポイントが適用されます。ほとんどのデバッガではソース上の位置にしかブレークポイントを設定できませんが、Identity Manager IDE のデバッガでは「Refresh view」などの概念的な実行ポイントにもブレークポイントを設定できます。この場合、デバッガはビューの更新操作が発生するたびに操作を中断します。そこで、「Refresh view」にステップインし、進行中の基礎となるフォーム処理を監視できます。

すべてのソースブレークポイントの要約をブレークポイントウィンドウに表示できます。通常、このウィンドウは Identity Manager IDE の左下隅にあります。ブレークポイントウィンドウでブレークポイントをクリックして、ソースエディタ内の対応するブレークポイントに移動することもできます。

ブレークポイントで停止すると、デバッガは現在の実行ポイントの範囲内にある変数も表示します。

図 1-33 ブレークポイントウィンドウ



ブレークポイントを設定する最も簡単な方法は、ブレークポイントを追加するタグの真横の、ソースエディタの左マージン部分をクリックすることです。

メインメニューから「Run」>「New Breakpoint」の順に選択することもできます。「New Breakpoint」ダイアログが表示されたら、「Debugger」メニューから「XPRESS」を選択します。ダイアログの内容が変更され、次のオプションが提供されます。

- **「Breakpoint Type」メニュー**: このメニューから選択可能なオプションは「Identity Manager」のみで、デフォルトで選択済みになっています。

- 「Global」タブ (デフォルト): 次のオプションのいずれかまたは両方を選択します。
  - 「All anonymous breakpoints」- 匿名ソースにブレークポイントを設定します。
  - 「All named breakpoints」- 特定のページで使用されているフォームが分からない場合にこのオプションを使用します。このブレークポイントを設定して、そのページに移動できます。その後、「All named breakpoints」オプションを無効化して、そのフォーム内でブレークポイントをより具体的に絞り込むようにします。このオプションを無効化しない場合、デバッガはすべてのブレークポイントで停止します。

注	両方のオプションを有効にすると、デバッガはすべてのブレークポイントをチェックします。

- 「View Cycle tab」タブ : 「Checkin View」、「Checkout View」、「Get View」、または「Unlock View」など、一般に使用されるビューに関連するブレークポイントを設定します。

  プロセス実行中に発生するビューの処理に基づいて、コードブレークポイントを設定することができます。呼び出される最も一般的なビュー操作がこのダイアログに一覧表示され、ビューごと選択できます。

- 「Form Cycle」タブ : フォーム処理の各段階に関連するブレークポイントを設定する場合に選択します。

  フォーム処理の指定した段階に基づいて、コードブレークポイントを設定できます。フォーム処理の段階については、『Sun Java™ System Identity Manager ワークフロー、フォーム、およびビュー』を参照してください。

Identity Manager に添付されているチュートリアルサンプルには、ブレークポイントの使い方の例が示されています。80 ページの「Identity Manager IDE チュートリアルの使用 : フォーム、規則、およびワークフローのデバッグ」を参照してください。

ウォッチポイントの使用

Identity Manager IDE デバッガでは、有効な XPRESS 式であればどの式でもウォッチポイントとして使用することができます。ウォッチポイントには 2 つの目的があります。

- 簡単な `<refs>` を使用して、特定の変数セットを評価できます。
- より複雑な XPRESS 式を使用して、プロジェクトに特定の変更を加えた場合にどうなるかを判別できます。

デバッガを最初に起動したとき、およびステップ実行中に中断するたび、またはブレークポイントで、デバッガはその時点のコンテキストでウォッチポイントを評価します。また、ウォッチポイントを追加または変更するたびに、デバッガはウォッチポイントを再評価します。

Identity Manager に添付されているチュートリアル (`debugger-tutorial-workflow1.xml`) には、ウォッチポイントの使い方の例が示されています。81 ページの「例 1: ワークフローと規則をデバッグする」を参照してください。

実行プロセスのステップスルー

ステップスルーとは、実行中のプロセスの関数を逐次、計画的に分析する処理のことです。

用語

ステップイン、ステップオーバー、およびステップアウトは、言語の構造によって実行順が暗黙的に決定される手続き型プログラミング言語のデバッガに由来する用語です。しかし、Identity Manager のフォームとワークフローでは、コード内で要素が出現する順序はその実行順に影響しません。

したがって、これらの用語が Identity Manager IDE やビジネスプロセスエディタ (BPE) で使用される場合、多少異なった意味を持ちます。

- **ステップイン** : 現在のスレッド上の次の実行ポイントに移動することを指します。ステップインは常に、デバッガの XML 表示でプロセス内を進むことのできる最小の単位です。
- **ステップオーバー** : 現在の `begin` タグから現在の `end` タグまで中間の要素で止まらずに移動することです。ステップオーバーでは、`start` タグと `end` タグの間のほとんどすべての要素をスキップできます。ただし、現在の要素の `start` タグと `end` タグの間に次の実行ポイントが出現しない場合、デバッグはそこで停止します。

たとえば、複数のアクティブな仮想スレッドを含むワークフローで、アクションの start タグにステップできますが、実行される次の要素は別のアクションです。この場合、プロセスは別のポイントで実行を停止します。そのため、重要な可能性がある要素を誤ってスキップすることを回避できます。

- **ステップアウト**: 実行スタックが現在よりも 1 少なくなるまで、増分的に移動することを指します。ステップオーバーに類似しています。次の実行ポイントが、異なる親の実行スタックを持つ場合は、代わりにそこで停止します。

ヒント　次のヒントは、実行中のプロセスに正常にステップスルーさせるのに役立つ情報です。

- デバッガでのステップインを、デバッグタスクのコンテキストで実現可能な範囲で細かく設定します。これは、デバッグにとって重要な可能性がある要素を空過するのを避けるために役立ちます。
- ステップ実行は、プログラムの実行順を変更しません。プログラムの実行順は、デバッガを接続しない場合と同じです。目に見える実行部分をスキップ可能です (ただし、それでも実行自体は行われる)。
- コードのステップスルーをできるだけ小さくしたい場合は、ステップインを使用します。
- start タグと end タグの間で、内容に関して問題が発生しそうにないと思われるときは、ステップオーバーを使用します。デバッガはこの要素をスキップしますが、これらのタグ内のコードは引き続き実行されます。

表 1-8 は、Identity Manager IDE デバッガによる、次のコードサンプルの処理方法のスナップショットを示します。

```
<A>
   <B/>
</A>
<D/>
(A、B、および D は何らかの XML 要素 )
```

表 1-8　デバッグプロセスの例

実行順	結果
<A>, <B/>, </A>, <D/>	「**step-into**」をクリックすると、デバッガはこの実行順で行を強調表示します。 「**step-over**」をクリックすると、デバッガは <A>、</A> を強調表示します。 (B をスキップして)、<D/> を強調表示します。
<A>, <D/>, <B/>, </A>	「**step-over**」をクリックすると、<A>、<D/>、<B/>、</A> の順でコード行が表示されます (この場合、ステップオーバーはステップインと同じ)。

フォームのデバッグ

ここでは、匿名ソースの操作方法を説明し、フォームテスターユーティリティーの設定および起動の手順を示します。

Identity Manager フォームエンジンがフォームを処理する方法の詳細については、『Sun Java™ System Identity Manager ワークフロー、フォーム、およびビュー』を参照してください。

注　呼び出しスタックウィンドウで実行スタックをチェックして、どのパスが現在処理中であるかを確認できます。

匿名ソースの操作

フォームをステップスルーするときに、デバッガは匿名ソースを識別できます。匿名ソースとは、Login フォームや MissingFields など、一時的に作成されるフォームまたはフォームの一部のことで、Identity Manager リポジトリにある持続的フォームは該当しません。

これらのフォームはリポジトリに存在しせず、固有の識別子を持たないため、匿名ソースには個別のブレークポイントを設定することができません。ただし、70 ページの「ブレークポイントの設定」で説明されている方法で「All anonymous sources breakpoint」を設定すると、匿名ソースに対してステップスルーはできます。

この設定により、デバッガは直接の XPRESS ソースを持たないポイントで実行を停止することができます。結果として、デバッガは匿名ソースからの行を検出するたびに中断します。

フォームテスターの設定および起動

Identity Manager IDE には、外部ブラウザでのフォームのテストを可能にするフォームテスターが用意されています。フォームテスターはデバッガに統合されているため、フォームのトラブルシューティングも可能です。

フォームをテストするには、次の手順に従います。

1. 次のいずれかの方法で、フォームテスターを起動します。
   - プロジェクトウィンドウでフォームノードを選択し、NetBeans メニューバーから「IdM」>「Test Form」の順に選択します。
   - プロジェクトウィンドウでフォームを右クリックし、ポップアップメニューから「Form Preview」を選択します。
   - ソースエディタで `<Form>` 要素内またはその任意の子の内側で右クリックし、ポップアップメニューから「Test Form」を選択します。

注 警告メッセージが表示され、Identity Manager IDE がユーザーのフォームをリポジトリにアップロードするときに、既存のコピーが置き換えられる (上書きされる) ことが通知されます。テスト処理を続行するか停止するかを指定する必要があります。

2. プロンプトが表示されたら、Identity Manager にログインします。

ヒント フォームをテストするたびにログインしないですむように、次の手順に従って System Configuration オブジェクトの `allowInterAppAuthenication` プロパティーを変更します。

1. 「IdM」>「Repository」>「Explore」の順に選択します。
2. 「Explore Repository」ウィンドウから「Generic Objects」ノードを展開し、「System Configuration」をダブルクリックしてこのオブジェクトをダウンロードします。
3. `System Configuration.xml` をダブルクリックしてソースエディタ内に表示させたあと、下方にスクロールして (あるいは「Edit」>「Find」を使って) `allowInterAppAuthentication` 属性を見つけます。
4. この属性の値を **`true`** に変更し、変更を保存します。

Identity Manager IDE は、ブラウザウィンドウに新規 Identity Manager セッションを起動し、Identity Manager 管理者ユーザーインタフェースのコンテキストでフォームを表示します。さらに、アップロードされたフォームに関する情報が Identity Manager IDE リポジトリの出力ウィンドウに表示されます。

注	Identity Manager には、フォームをデバッグする方法を理解するのに役立つチュートリアル (`debugger-tutorial-workflow1.xml`) が提供されています。手順については、80 ページの「Identity Manager IDE チュートリアルの使用 : フォーム、規則、およびワークフローのデバッグ」を参照してください。

規則のテスト

Identity Manager IDE が提供する規則テスターを使えば、スタンドアロン規則やライブラリ規則をソースエディタ内で編集する際に、それらの規則を検証できます。ソースエディタで規則のすべての引数の値を指定したあと、規則を「実行」します (トレース文を組み込むこともできる)。

次のいずれかの方法で、フォームテスターを起動できます。

- プロジェクトウィンドウで規則ノードを選択し、NetBeans メニューバーから「IdM」>「Test Rule」の順に選択します。
- プロジェクトウィンドウで規則を右クリックし、ポップアップメニューから「Test Rule」を選択します。
- ソースエディタで `<Rule>` 要素内またはその任意の子の内側で右クリックし、ポップアップメニューから「Test Rule」を選択します。

次の例は、規則テスターの動作方法を示すために提供されています。

- 例 1: スタンドアロン規則をテストする
- 例 2: ライブラリ規則をテストする

例 1: スタンドアロン規則をテストする

この例では、単純なスタンドアロン規則をテストする方法を示します。

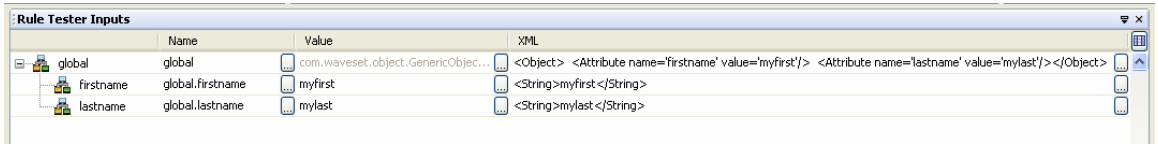
1. 「Custom Identity Manager Object」を右クリックし、「Repository」>「Explore」の順に選択することで、リポジトリ内のオブジェクトを表示します。
2. 「Explore Repository」ダイアログで「Rules」ノードを展開し、`Accountname First dot Last` をダブルクリックして、この規則を開きます。
3. 「Rule Tester Inputs」ウィンドウを右クリックして、メニューから「Add value」を選択します。
4. 「New Value」ダイアログが表示されたら、次の情報を入力し、「OK」をクリックします。
   - 「Name」: `global.firstname`

- 「Type」: `String`
- 「Value」: `myfirst`

5. もう一度、「Rule Tester Inputs」ウィンドウを右クリックし、次の情報でもう 1 つの新しい値を入力します。
   - 「Name」: `global.lastname`
   - 「Type」: `String`
   - 「Value」: `mylast`

   「Rule Tester Inputs」ウィンドウは、図 1-34 のようになります。

図 1-34 「Rule Tester Inputs」ウィンドウ



6. これで、規則をテストできます。

   NetBeans メニューバーから「IdM」>「Test Rule」の順に選択するか、あるいはソースエディタウィンドウで右クリックし、ポップアップメニューから「Test Rule」を選択します。

   注 規則テストが規則を自動的に保存し、レポジトリに公開することを警告するメッセージが表示されます。「Yes」または「Always」をクリックして続行します。

   次のような出力が、「Rule Tester Output」ウィンドウに表示されるはずです。

   ```
   <String>myfirst.mylast</String>

   <String>myfirst.mylast</String>

   <String>myfirst.mylast</String>
   ```

7. 次に、規則のテストとデバッガの実行を同時に試行します。
   デバッガがまだ実行されていない場合は、メインメニューバーの「Debug Main Project」ボタン をクリックします。
8. ソースエディタウィンドウに移動し、XML 内で `<Rule>` タグの横のマージンをクリックして、ブレークポイントを追加します。
9. 「Window」>「Debugging」>「Breakpoints」の順に選択すると、ブレークポイントウィンドウが開きます。
10. ソースエディタで、`<Rule>` 要素内の任意の位置で右クリックし、ポップアップメニューから「Test Rule」を選択します。

ブレークポイントでデバッガが停止します (ブレークポイントウィンドウ内の結果を参照)。

11. 「Step Into」ボタン  を 7 回クリックして規則をステップスルーし、ローカル変数ウィンドウの結果を監視します。
12. 「Continue」をクリックして、「Rule Tester Output」ウィンドウの結果を監視します。

例 2: ライブラリ規則をテストする

ライブラリ規則のテストは、スタンドアロン規則のテストに非常に似ています。

1. 必要に応じて、リポジトリから Alpha Numeric Rules ライブラリをダウンロードし、ノードをダブルクリックしてソースエディタでこのライブラリ規則を開きます。
2. 「Alpha Numeric Rules」のノードを順次展開し、`stringToChars` 規則が表示されたらダブルクリックします。

   この規則は形式引数を宣言しているため、「Rule Tester Inputs」ウィンドウには `testStr` 引数が表示されます。
3. `testStr` 引数に値を指定します (「Rule Tester Inputs」ウィンドウで引数名を右クリックし、「Add value」を選択し、「New Value」ダイアログに値を入力する)。
4. ソースエディタで右クリックし、「Test Rule」を選択して規則を実行し、「Rule Tester Output」ウィンドウで結果を監視します。

注 Identity Manager には、規則をデバッグする方法を理解するのに役立つチュートリアル (`debugger-tutorial-workflow1.xml`) が提供されています。手順については、80 ページの「Identity Manager IDE チュートリアルの使用 : フォーム、規則、およびワークフローのデバッグ」を参照してください。

ワークフローのデバッグ

ワークフローは単一の Java スレッドによって実行され、呼び出しスタックウィンドウで単一の Java スレッドによって表現されます。ただし、ワークフローの内部で、各アクティビティーは個別の仮想スレッドになります。

ワークフロー実行の間、ワークフローエンジンは仮想スレッドのキューを循環的に処理します。各仮想スレッドは、次の表で説明する状態のいずれかになります。

表 1-9 仮想スレッドの状態

ワークフローアクティビティーの状態	定義
準備完了	遷移したばかりのアクティビティーを特定します (この状態はごく一時的であり、アクションは通常、準備完了と指定された直後に実行を開始する)。
実行中	現在実行中であるか、まだ実行されていない 1 つ以上のアクションを含むアクティビティーを特定します。 これは論理状態であり、Java スレッドがその時点でそのアクションを実行していることを意味しません。現在実行中のアクションは、必ず太字または通常のフォントで表示されます。 実行中ではないアクションは、斜体で表示されます。
保留中のアウトバウンド	アクティビティー内のすべてのアクションが実行された直後のアクティビティーを特定します。このようなアクティビティーは、保留中のアウトバウンド状態に移行します。この状態のアクションは、アウトバウンド遷移の発生を待機します。OR 分岐の場合、アクションは 1 つの遷移が発生するまでこの状態です。AND 分岐の場合、その条件が true と評価されるすべての遷移が発生するまで、アクションはこの状態です。
非アクティブ	すべての遷移が発生済みのアクティビティーを特定します。
保留中のインバウンド	そのアクティビティーが AND 合流である仮想スレッドを特定します。これは、この仮想スレッドへの 1 回の遷移が発生したが、プロセスはまだほかの遷移を待機していることを意味します。

すべての遷移が完了したあとで、ワークフロープロセスは実行を開始します。

変更のあとでワークフローのリビジョンを妥当性検査するには、プロジェクトウィンドウでオブジェクトまたはプロセスを選択してから、「Tools」>「Validate」の順に選択してテストします。出力ウィンドウでメッセージを確認します。

注 Identity Manager には、ワークフローをデバッグする方法を理解するのに役立つチュートリアル (debugger-tutorial-workflow1.xml) が提供されています。手順については、80 ページの「Identity Manager IDE チュートリアルの使用 : フォーム、規則、およびワークフローのデバッグ」を参照してください。

Identity Manager IDE チュートリアルの使用 : フォーム、規則、およびワークフローのデバッグ

Identity Manager には、フォーム、規則、およびワークフローにデバッガを使用する方法を習得するのに役立つチュートリアル (debugger-tutorial-workflow1.xml) が提供されています。このチュートリアルには、サンプルのフォーム、規則、ワークフローが含まれており、この節を通して同じサンプルが使用されます。

警告 このチュートリアルは、本稼働環境で有効にしないでください。

この節は、次のように構成されています。

はじめに

debugger-tutorial-workflow1.xml チュートリアルを使用するには、次の操作を行います。

1. 次のいずれかの方法で、debugger-tutorial-workflow1.xml ファイルをインポートします。
   - Identity Manager で、「Configure」>「Import Exchange File」の順に選択します。「File to Upload」フィールドに「**sample/debugger-tutorial.xml**」と入力します。
   - コンソールから、次のように入力します。

     ```
     import -v sample/debugger-tutorial.xml
     ```

     ファイルが正常にインポートされたら、次のファイルが読み込まれたことを示す確認メッセージが表示されます。

```
debugger-tutorial-workflow1
debugger-tutorial-workflow2
compute-full-name
```

2. debugger-tutorial-workflow1 と debugger-tutorial-workflow2 をリポジトリからプロジェクトにダウンロードします。手順については、44 ページの「リポジトリからのオブジェクトの取得」を参照してください。
3. アプリケーションサーバーを再起動したあと、メインメニューバーで「Run」>「Debug Main Project」の順に選択して、Identity Manager IDE デバッガを起動します。

例 1: ワークフローと規則をデバッグする

この例では、ワークフローのデバッグと規則の実行をステップインおよびステップスルーする方法を含む、簡単なワークフローとワークフローを使用する規則とをデバッグする方法を示します。

この練習を完了するには、次のステップを実行する必要があります。

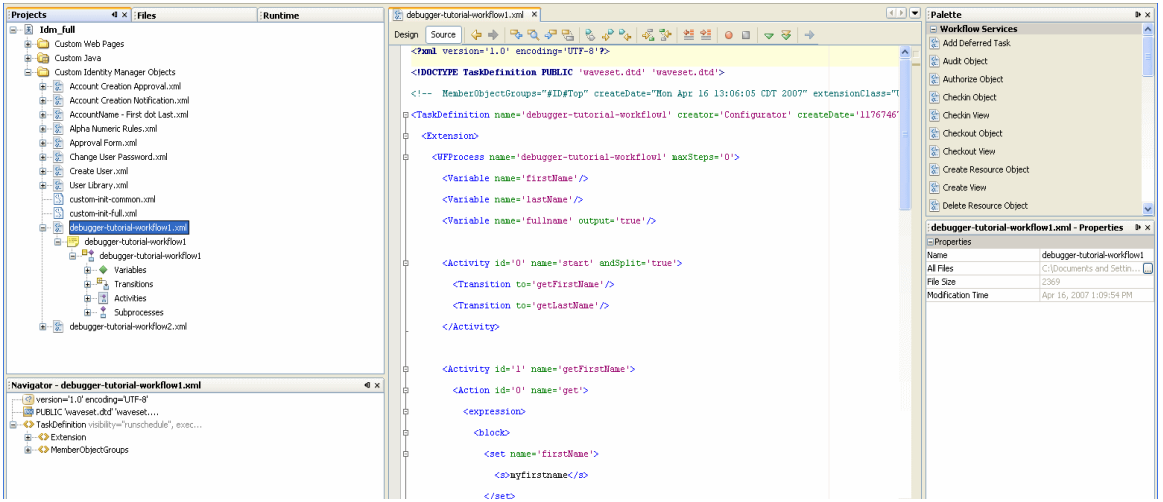
1. プロセスの起動
2. 実行の開始
3. getFirstName スレッドのステップスルー。
4. getlastname スレッドのステップインおよびステップオーバー。
5. computefullname 処理のステップイン。
6. 規則処理のステップスルー
7. ワークフロー処理の完了

ステップ 1: プロセスの起動

ワークフローのデバッグプロセスを起動するには、次の手順に従います。

1. プロジェクトウィンドウで debugger-tutorial-workflow1.xml ノードを展開します。
2. debugger-tutorial-workflow1.xml ノードをダブルクリックして、ソースエディタに XML を表示します (必要に応じて「Source」ボタンをクリックする)。

図 1-35　debugger-tutorial-workflow1.xml ソースを開いたところ



3. <WFProcess> タグの左のマージンを 1 回クリックして、ワークフローの開始位置にブレークポイントを設定します。
4. 必要に応じて、メインメニューバーで「Debug Main Project」ボタンをクリックして、Identity Manager IDE デバッガを起動します。
5. Identity Manager にログインしたあと、「Server Tasks」>「Run Tasks」の順に選択します。
6. 「Available Tasks」ページが表示されたら、「Name」列で debugger-tutorial-workflow1.xml をクリックします。

   Identity Manager IDE のソースエディタをチェックして、指定したブレークポイントでデバッグが停止したかを確認します。

   次の点にも注意してください。

   - 呼び出しスタックウィンドウの上部には、Thread [*thread name*] (suspended) と表示されるはずです。これは、このワークフローがここに示された名前のスレッドによって現在実行されており、設定したブレークポイントで中断していることを意味します。

     「Thread」の下には実行スタックが表示されます。このスタックは逆順のスタックトレースであり、呼び出し元の関数が上に、呼び出される関数が下に表示されます (これは、ほとんどのデバッガでの実行トレースの表示とは逆の順序)。

スタックの一番上のフレームは「Checkin View (ProcessViewer)」という名前であり、これは、ワークフローがその時点で ProcessViewer の `checkinView` メソッドによって呼び出されていることを示します。このスタックフレームの Java ソースコードにはアクセスできないため、このフレームをクリックしても新しい情報は表示されません。ただし、スタックフレームは、ワークフローがどの場所から起動されているかについてのコンテキストを提供します。

スタック内の次のフレームは、ワークフロープロセス (<*WFProcess*>) の開始位置である現在の実行ポイントに対応しているため、強調表示されています。

- ローカル変数ウィンドウには、現時点で現在の実行ポイントの範囲内にある、すべての変数のリストが表示されるはずです。次の変数が含まれます。

表 1-10 スコープ内の変数

変数	説明
Last Value	最後の評価の結果
Interactive	ビューによってプロセスへの入力として渡される変数。
WF_CASE_OWNER	暗黙のワークフロー変数
fullname	<*Variable*> 宣言を使用してワークフロー内で宣言される変数
WF_CONTEXT	暗黙のワークフロー変数
WF_CASE_RESULT	暗黙のワークフロー変数
firstName	<*Variable*> 宣言を使用してワークフロー内で宣言される変数
lastName	<*Variable*> 宣言を使用してワークフロー内で宣言される変数

これで、実行を開始する準備が完了しました。次のステップでは、実行手順を説明します。

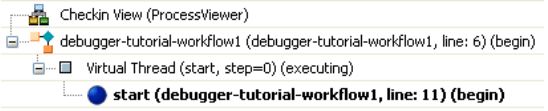
ステップ 2: 実行の開始

実行を開始するには、次の手順に従います。

1. Identity Manager IDE のメインメニューバーで「Step-Into」ボタン  をクリックします。

   デバッガが開始アクティビティーに移動します。実行スタックに「`Virtual Thread [start, step=0] (executing)`」が含まれていることを確認してください。これは、現在実行中の状態である開始アクティビティーの仮想スレッドがあることを示します。

図 1-36　開始アクティビティーの仮想スレッド



2. debugger-tutorial-workflow1 フレーム (2 レベル上) をダブルクリックして WFProcess を強調表示します。そこには、呼び出し側の位置が表示されます。
3. 呼び出しスタックウィンドウ内の太字のエントリをダブルクリックして、現在の行に戻ります。
4. 「step-into」をもう一度クリックします。

   デバッガはソースエディタの </Activity> 行に移動し、呼び出しスタックウィンドウが表示されている場合は、「Virtual Thread [start, step=0]」が保留中のアウトバウンドになります。

これで、次のステップ 3 に進むことができます。

ステップ 3: getFirstName スレッドのステップスルー

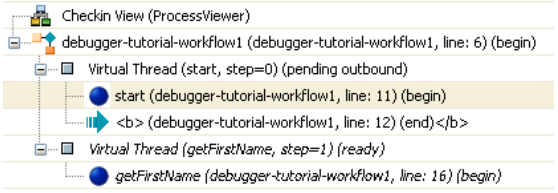
getFirstName スレッドをステップスルーするには、次の手順に従います。

1. 「step-into」をクリックします。

   デバッガでは、getFirstName への遷移が強調表示されます。
2. 「step-into」をもう一度クリックします。

   呼び出しスタックウィンドウをもう一度チェックし、デバッガがこの遷移の結果として getFirstName の新規仮想スレッドを作成し、現時点でこの仮想スレッドが準備完了の状態になっていることを確認してください。

図 1-37　開始アクティビティーの仮想スレッド



「Virtual Thread [start, step=0]」が保留中のアウトバウンドのままであることに注意してください。これは、この仮想スレッドが and-split 操作であるため、可能性のある遷移をすべて引き受ける必要があるからです。

3. 「step-into」をもう一度クリックします。

デバッガはソースエディタの getFirstName アクティビティーにジャンプし、呼び出しスタックウィンドウでは状態が準備完了から実行中に変わります。

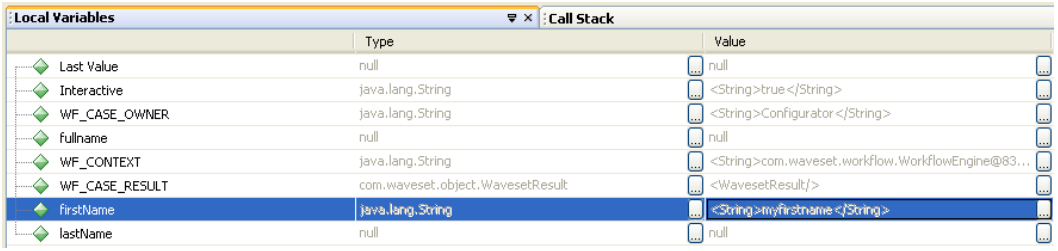
4. 「step-into」をもう一度クリックします。

   デバッガは get アクションに移動します。

5. ここで、「Step-Into」をあと 3 回、またはデバッガがソースエディタ内で </set> タグに達するまでクリックします。

   ローカル変数ウィンドウをチェックして、</set> の結果として firstName 値が <String>myfirstname</String> に設定されていることを確認します。

図 1-38　　firstName の新しい値



6. ウォッチポイント表現を追加します。

   a. ウォッチポイントウィンドウ (「Window」>「Debugging」>「Watches」) を開きます。

   b. ウィンドウを右クリックして、メニューから「New Watch」を選択します。

   c. 「New Watch」ダイアログが表示されたら、次の XPRESS 文を「Watch Expression」フィールドに入力して、「OK」をクリックします。

      ```
      <ref>firstName</ref>
      ```

   デバッガはウォッチポイント表現を評価しますが、その値は手順 5 で設定した値である <String>myfirstname</String> になるはずです。

ステップ 4: getLastName スレッドのステップインおよびステップオーバー

次の手順に従って、getLastName スレッドをステップインおよびステップオーバーします。

1. 「Step-Into」をあと 3 回、またはソースエディタ内でデバッガが getFirstName の </Activity> 行に達するまでクリックします。

   呼び出しスタックウィンドウで、Virtual Thread (getFirstName, step=1) が、保留中のアウトバウンドになっていることを確認します。

2. 「step-into」をクリックします。

   デバッガは、Virtual Thread (start, step=0) に戻り、getLastName への遷移の処理を開始しようとします。

3. 「step-into」をクリックします。

   Virtual Thread (start, step=0) は、すべての遷移の処理が完了したので、非アクティブになります。この遷移の結果として、getLastName が準備完了の状態になります。

4. 「step-into」をクリックします。

   この時点で、Virtual Thread (start, step=0) は非アクティブであるために消えて、デバッガは Virtual Thread (getLastName, step=2) に移動し、これが実行中の状態になります。

5. 「step-over」ボタン  をクリックして、getLastName の終わりまでスキップします。

   ローカル変数ウィンドウを確認すると、lastName 変数が <String>mylirstname</String> に設定されているはずです。getFirstName および getLastName の両方の仮想スレッドは、保留中のアウトバウンド状態です。

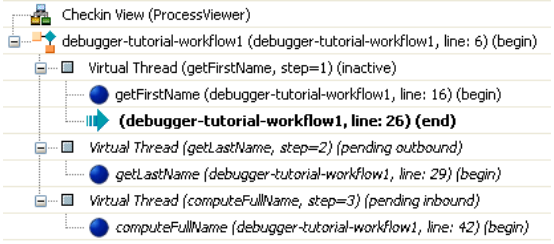
6. 「step-into」をクリックします。

   ソースエディタをチェックし、デバッガが getFirstName から computeFullName へ遷移しつつあることを確認してください。

7. 「step-into」をクリックします。

   呼び出しスタックウィンドウで、getFirstName は非アクティブになり、新しく「Virtual Thread (computeFullName, step=3)」が作成されます。

図 1-39　　新しい computeFullName 仮想スレッド



   このスレッドは、getLastName からのインバウンド遷移をまだ待機しているため、「pending inbound」状態です (待機が発生するのは、これが and-join 操作であるため。or-join 操作の場合は、プロセスの状態がただちに「ready」になる)。

8. 「step-into」をクリックします。

デバッガは getLastName から computeFullName への遷移に遷移します。

ステップ 5: computeFullName 処理のステップイン

次の手順を使用して、computeFullName 処理にステップインします。

1. 「step-into」をクリックします。

   この遷移により、「Virtual Thread (computeFullName, step=3)」の状態が、保留中のインバウンドから準備完了に変化します。

2. 「step-into」をクリックします。

   この時点で、Virtual Thread (computeFullName, step=3) の状態は、実行中です。

3. 「step-into」をあと 5 回クリックします。

   ソースエディタを確認すると、デバッガが firstName の </argument> タグ上にあり、ローカル変数ウィンドウで Last Value が <String>myfirstname</String> になっています。この値は firstName 引数に渡されます。

ステップ 6: 規則処理のステップスルー

規則処理をステップスルーするには、次の手順に従います。

1. 「step-into」をあと 3 回クリックします。

   デバッガが「compute-full-name」規則にステップインします。

2. 呼び出しスタックウィンドウで、フレームをクリックして 1 つ上のフレーム上に移動します。

   debugger-tutorial-workflow1 内の <rule> 呼び出しが強調表示され、規則の呼び出し元の場所を示します。

3. 太字の行をダブルクリックして、その行を再選択します。

4. 「Step-Into」をあと 3 回、またはデバッガが </ref> タグに達するまでクリックします。

   ここでローカル変数ウィンドウを確認すると、Last Value エントリは、<ref>firstName</ref> の結果である <String>myfirstname</String> になっています。

5. 「Step-Into」をあと 3 回、またはデバッガが </concat> タグに達するまでクリックします。

   Last Value エントリは、<concat> 式の結果である <String>myfirstname mylastname</String> になっています。

6. 「Step-Into」をあと 2 回クリックします。デバッガは </rule> タグに戻ります。

ステップ 7: ワークフロープロセスの完了

ワークフロープロセスを完了するには、次の手順に従います。

7. </set> 要素に達するまで「step-into」をクリックします。

   fullname 変数が <String>myfirstname mylastname</String> に更新されています。

8. 「step-into」をあと 2 回クリックします。

   この時点で、「Virtual Thread (computeFullName, step=3)」の状態は、保留中のアウトバウンドです。

9. 「Step-Into」をあと 5 回クリックします。

   end が準備完了、続いて実行中になり、そのあとデバッガは </WFProcess> タグに達します。これは、プロセスが完了したことを示します。

10. 「step-into」をクリックします。

    呼び出しスタックウィンドウには「After Checkin View」と表示されます。これは、ワークフローを呼び出した、ビューのチェックイン操作が完了したことを示します。

11. Identity Manager IDE メインメニューバーの「Continue」ボタンをクリックして、実行を再開します。

    ブラウザの要求がタイムアウトしていない場合、プロセスダイアグラムを伴う「Task Results」ダイアグラムが表示されます。

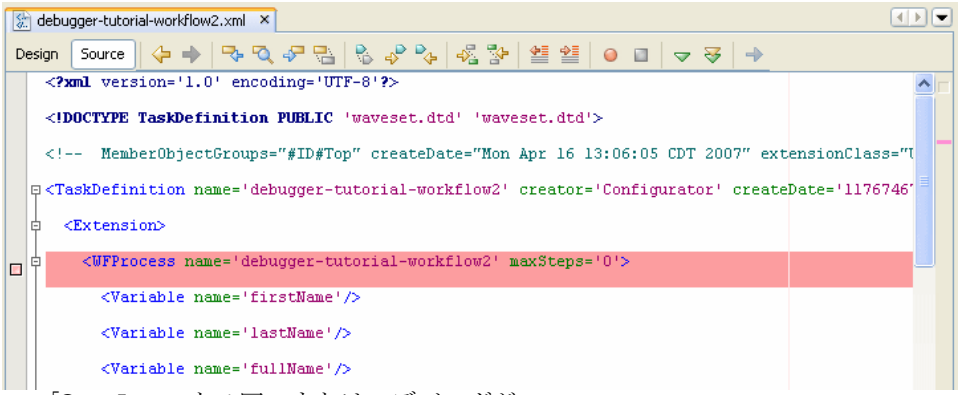
例 2: 手動アクションとフォームを含むワークフローのデバッグ

この例では、手動アクションとフォームを含む、サンプルワークフローのデバッグ方法を説明します。debugger-tutorial-workflow2 チュートリアルファイルを使用し、次の手順を実行します。

1. debugger-tutorial-workflow2.xml をダブルクリックして、ソースエディタに XML を表示します。
2. <WFProcess...> タグにブレークポイントを設定します。
3. 必要に応じて、Identity Manager IDE メインメニューバーで「Debug Main Project」ボタン  をクリックします。
4. Identity Manager にログインし、「Server Tasks」>「Run Tasks」の順にナビゲートします。
5. 「Available Tasks」ページが表示されたら、「Name」列から debugger-tutorial-workflow2 を選択します。

   設定したブレークポイントでデバッガが停止していることを確認します。

図 1-40　新しいブレークポイント



6. 「Step-Into」を 6 回、または、デバッガが <ManualAction... name='getNameAction'> に達するまでクリックします。
7. 「step-into」をクリックします。
8. 別のスレッドでフォーム処理が発生するという説明の「Stepping into Manual Action」ダイアログが表示されたら、「Yes」または「Always」をクリックします。
9. 処理が行われていることを確認するために、ブレークポイントを <Form> タグに設定します。

   フォーム処理が完了すると、ワークフローは別のスレッドでの実行を継続します。その結果、</ManualAction> にブレークポイントを設定して、フォームが処理を完了したあとのワークフロー処理を監視する必要があります。

   デバッガでは、指定どおりに <Form> タグと </ManualAction> タグにブレークポイントが設定されています。加えて、呼び出しスタックウィンドウには「After Checkin view」が示されます。ワークフロー処理は可能なかぎり (手動アクションが完了するまで) 進行済みであるため、ワークフロープロセスからのステップアウトが完了します。
10. 「Continue」をクリックします。デバッガは <Form> 要素に設定されたブレークポイントで処理を停止します。

    呼び出しスタックウィンドウで、次のエントリに注目します。

    - 「Derivation」- フォーム実行が「derivation」パス上にあることを示します。
    - 「Checkout View (WorkItem:...)」- 特定の作業項目に対し、ビューのチェックアウトのコンテキストで処理が発生していることを示します。
    - 「ManualAction forms」- 作業項目ビューに対して作用し、変数オブジェクトを通じてワークフロー変数を操作します。変数オブジェクトを展開して、null でないワークフロー変数を表示します。

11. このフォームには <Derivation> 式が含まれないため、「Continue」をクリックして次の処理フェーズに進みます。フォーム処理の「HTML Generation (root component)」パスが開始されます。

HTML 生成フェーズ (root コンポーネント)

root コンポーネントの HTML を生成するには、次の手順に従います。

1. 「step-into」を 2 回クリックします。

   デバッガは、タイトルの <Property> 要素を処理しましたが、このプロパティーの値がローカル変数ウィンドウの Last Value エントリに入ります。

2. 「step-into」をあと 3 回クリックします。

   このパスではページの root コンポーネントの構築のみを扱うため、デバッガはフォームフィールドをスキップして </Form> 要素に直接移動します。

3. 「Continue」をクリックして、フォーム処理の HTML 生成 (サブコンポーネント) パスを開始します。

HTML 生成 (サブコンポーネント)

サブコンポーネントの HTML を生成するには、次の手順に従います。

1. 「Step-Into」を 13 回、または、デバッガが </Form> タグに達するまでクリックします。

   デバッガはこれらの各フィールドを反復処理し、それらの表示プロパティーを評価します。

2. 「Continue」をクリックします。

   実行が再開されたため、デバッガには中断されたスレッドは表示されません。Identity Manager ブラウザウィンドウに制御が戻ります。

3. Identity Manager ブラウザウィンドウに戻り、入力を求められたら姓と名を入力して「Save」をクリックします。

   デバッガフレームに戻り、デバッガがブレークポイントで中断していることを確認します。

4. ローカル変数ウィンドウで、変数サブツリーを展開して、今入力した名前が firstName および lastName の値として表示されることを確認します。

   デバッガはこの時点で、フォーム処理の確認フェーズです。

確認

このフォームには確認フィールドがないため、処理は発生しません。「Continue」をクリックして、フォーム処理の検証フェーズを開始します。

検証と展開

このフォームには検証式が含まれないため、明示的な処理は発生しません。

1. 「Continue」をクリックして検証フェーズをスキップし、フォーム処理の展開フェーズに進みます。
2. 「step-into」を 6 回クリックします。

   デバッガは variables.fullName フィールドの <Expansion> の <rule> タグに移動します。
3. 「Step-Into」を 5 回クリックします。デバッガは <Rule> 要素にステップインします。
4. 「Step-Into」を 7 回、または、デバッガが </Rule> 要素に達するまでクリックします。

   「Last Value」に姓名が入ります。
5. 「step-into」をもう一度クリックすると、フォームでの処理が再開します。
6. 「step-into」をもう一度クリックします。

   トップレベルの variables.fullName には、実行されたばかりの展開式の値が格納されています。フォーム処理中はフォーム出力が独自の一時的な form_outputs データ構造に保持され、そこではパスの式が平坦化されるため、この値は variables データ構造の子ではなくトップレベルのエンティティーとなります。

   フォーム処理のあと、フォーム出力は元のビューに同化されます。暗黙的な変数 form_inputs および form_outputs において、form_inputs は未変更の作業項目ビューを示し、form_outputs は、フォーム処理の完了後にビューに同化される出力フィールドを示します。

   一般に、form_inputs はビューを特定し、form_outputs にはビューに同化されるデータが含まれます。ただし、Active Sync フォームのように、必ずしもすべてのフォームがビューに結び付けられるわけではありません。フォームエンジンは一般的なデータマッピングエンジンであり、フォーム入力からフォーム出力へのマッピングを行います。ビューハンドラは、フォームエンジンにビューを渡す処理と、出力をビューに戻して反映する処理を受け持ちます。
7. 「Continue」をクリックします。

   デバッガは </ManualAction> ブレークポイントに到達します。これは、デバッガが手動アクションにステップインしたときにすでに設定したブレークポイントです。firstName 変数および lastName 変数は、入力した値です。fullName 値は、実行されたばかりの展開式の結果です。
8. 「Step-Into」を 5 回、<ManualAction... name='displayNameAction'> に達するまでクリックします。

9. 「step-into」をもう一度クリックします (プロンプトが表示されたら「Yes」または「Always」をクリックする)。
10. 「Continue」をクリックします。

    この時点で、デバッガは displayNameForm の「Derivation」パスの位置です。

取得と HTML 生成 (root コンポーネント)

取得フェーズと HTML 生成フェーズを完了するには、次の手順に従います。

1. 「Continue」をクリックして、displayNameForm の HTML 生成 (root コンポーネント) 処理を開始します。
2. 「Step-Into」を 8 回、または、デバッガが subTitle の </Property> 要素に達するまでクリックします。
3. 「Continue」を 2 回クリックします。

    デバッガは次のメッセージを表示します。

    ```
    No suspended threads because execution has resumed.Control has now
    returned to the browser window.
    ```

4. Identity Manager ブラウザウィンドウに戻ります。

    表示される情報は、入力したものと同じです。

5. 「Save」をクリックして、デバッガフレームに戻ります。

    この時点で、デバッガは「Confirmation」パスの位置であり、displayNameForm を処理しています。

検証と展開

検証と展開を開始するには、次の手順に従います。

1. 「Continue」をクリックして検証パスを開始します。
2. 「Continue」をクリックして展開パスを開始します。
3. 「Continue」をもう一度クリックします。

    手動アクションが完了したため、この時点でデバッガは </ManualAction> タグの位置です。この時点で、ワークフロー処理は再開されています。

4. 「Step-Into」を 5 回、または、デバッガがワークフローの実行が完了したことを示す </WFProcess> タグに達するまでクリックします。
5. 「Continue」をクリックします。
6. Identity Manager ウィンドウに戻ると、ワークフロープロセスダイアグラムが表示されているはずです。「OK」をクリックします。

例 3: タブ付きユーザーフォームと更新ビューのデバッグ

このサンプルデバッグの手順では、フォームまたはワークフローを起動した場所に関係なく、デバッガのブレークポイントがどのように適用されるかを示します。

この手順を完了するには、次のステップを実行する必要があります。

1. ブレークポイントを設定します。
2. 新しいユーザーを作成します。
3. ビューの更新前の結果を表示します。
4. ビューの更新後の結果を表示します。
5. フォームをステップスルーします。
6. フォームの処理を終了します。

ブレークポイントの設定

ブレークポイントを設定するには、次の手順に従います。

1. ブレークポイントウィンドウを右クリックして、メニューから「Breakpoints」を選択します。
2. 「Breakpoints」ダイアログが表示されたら、「Debugger」メニューから「XPRESS」を選択した後、「View」タブを選択します。
3. 「Refresh View」チェックボックスを有効にします。

   これで、実行中にビューが更新されるたびに、デバッガでブレークポイントが実行されるようになります。

新規ユーザーの作成

新規ユーザーを作成するには、次の手順に従います。

1. Identity Manager で、「Accounts」タブを選択し、左上のドロップダウンメニューから「New User」を選択します。
2. 「Create User」ページが表示されたら、**`jean faux`** のように名と姓を入力します。
3. 別のタブをクリックして、ビューの更新操作をトリガーします。

   Identity Manager がブレークポイントに到達して中断していることに注意してください。

「Before Refresh View」結果の表示

Identity Manager IDE に戻り、次の点を確認します。

- ソースエディタは、現在設定した「Refresh View」ブレークポイントで中断しています。

- 呼び出しスタックウィンドウには、「Before Refresh View」が表示されます。これは、更新操作が発生する直前のビューの状態を示します。
- ローカル変数ウィンドウには、更新される直前のビューが表示されます。

ローカル変数ウィンドウで、グローバルサブツリーを展開して、フォームで入力した `firstname` と `lastname` の値を探します。`fullname` の値は、この時点では「`null`」です。

「After Refresh View」結果の表示

「After Refresh View」の結果を表示する手順は、次のとおりです。

1. 「Continue」をクリックします。

   呼び出しスタックウィンドウには、「After Refresh View」が一覧表示されます。ここには、更新操作が発生した直後のビューの状態が表示されます。`fullname` の値はこの時点では「`jean faux`」です。

2. 「Continue」をもう一度クリックします。

   フォームが実行を再開します。Identity Manager ブラウザウィンドウに戻って、「First Name」を **`jean2`** に変更します。別のタブをクリックして、もう一度更新をトリガーします。

   Identity Manager IDE ソースエディタに戻ると、フォーム処理は「Before Refresh View」の箇所で中断しています。

フォームのステップスルー

フォームをステップスルーするには、次の手順に従います。

1. 「step-into」をクリックして、実行内の姓名の展開部分を表示します。

   呼び出しスタックウィンドウに「Before Expansion,」が一覧表示されます。これは、フォームの変数が展開されていないことを示します。

2. 「step-into」をもう一度クリックします。

   呼び出しスタックウィンドウに「Before Expansion, `iteration=0`」がリストされます。これは、最初の「Expansion」パスの前にフォーム変数が出現することを示します。

3. 「step-into」をもう一度クリックします。

   呼び出しスタックウィンドウに匿名ソース (Tabbed User Form (Anonymous, line: 3)(begin)) が一覧表示されます。匿名ソースは一時的に作成されるラッパーフォームであり、MissingFields フォームに関連します。

4. タブ付きユーザーフォームの先頭に達するまで、「step-into」をさらに 2 回クリックします。

5. <Field name='global.fullName'>」に達するまで「step-into」をクリックし続けます (約 20 ～ 30 回のステップイン操作)。
6. 15 回または </Field> 要素に達するまで「Step-Into」をクリックします。

   ステップ操作中に、</concat> タグの Last Value エントリが、jean2 faux になっており、form_outputs 値が global.fullname: jean2 faux になっていることを確認します。

フォーム処理の完了

フォーム処理を完了するには、次の手順に従います。

1. 「Step-Out」を 7 回クリックします。

   この時点で、呼び出しスタックウィンドウが示す内容は次のようになるはずです。

   ```
   Refresh View (User)
   After Expansion
   ```

   ローカル変数ウィンドウには、すべての展開が実行されたあとのフォーム変数の状態が表示されます。
2. 「Step-Out」をもう一度クリックします。

   これで、「After Refresh View」に到達しました。ローカル変数ウィンドウには、ビュー変数が表示されています。
3. グローバルサブツリーを展開します。

   この時点で、fullname の値は「jean2 faux」です。
4. 「Continue」をクリックします。

Java と XPRESS のデバッグ

次の例では、カスタム Java コードを作成およびコンパイルする方法と、XPRESS デバッガと Java デバッガの両方を同時に使用する方法を示します。

注 Identity Manager Project (Remote) は Java のデバッグをサポートしません。

ヒント この例を始める前に、NetBeans HTTP モニターを無効にして余分なウィンドウを削除してください。

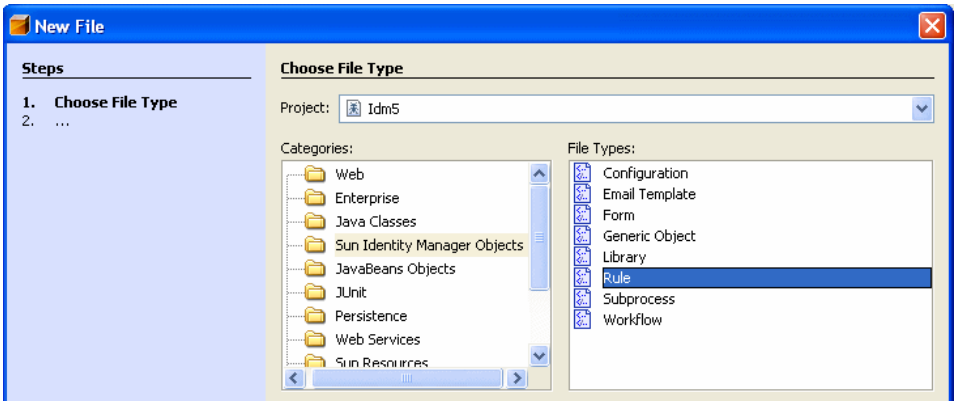
1. 実行時ウィンドウで「Servers」ノードを展開し、「Bundled Tomcat」を右クリックし、ポップアップメニューから「Properties」を選択します。
2. 「Server Manager」ダイアログが表示されたら、「Enable HTTP Monitor」ボックスを無効にしてから「Close」をクリックします。

1. 次の手順を実行してカスタム Java コードを作成します。
   a. 「Projects」タブで「Source Packages」を右クリックし、ポップアップメニューから「New」>「Java Class」を選択します。
   b. 「Class Name」フィールドに **`TestClass`** を、「Package」フィールドに **`testpackage`** を、それぞれ入力します。
   c. `testpackage` ノードが表示されたら、次のメソッドを追加します。

```
public static String concat(String s1, String s2)
{
   return s1+s2;
}
```

   d. このクラスを保存します。
2. 次の手順を実行してこの Java コードを呼び出す規則を作成します。
   a. プロジェクトウィンドウで、「Custom Identity Manager Objects」を右クリックし、ポップアップメニューから「New」>「File/Folder」を選択します。
   b. 「New File」ダイアログが表示されたら、「Categories」リストから「Sun Identity Manager Objects」を選択したあと、「File Types」リストから「Rule」を選択します。

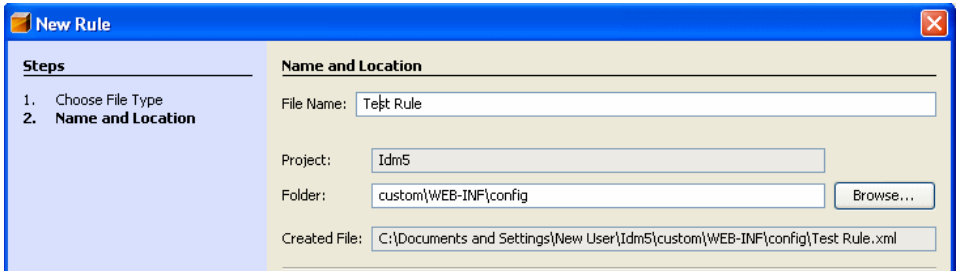
図 1-41　「Rule」ファイルタイプの選択



c. 完了したら、「Next」をクリックします。

d. 「New Rule」画面が表示されたら、「File Name」フィールドに **`Test Rule`** と入力してから「Finish」をクリックします。

図 1-42　「Rule」ファイルタイプの選択



プロジェクトウィンドウが更新され、`Test Rule.xml` が表示されます。

e. `Test Rule.xml` を選択したあと、エディタウィンドウの「Source」タブをクリックしてその XML を表示します。

f. 必要であれば、`<Rule name='New Rule'` を `<Rule name='Test Rule'` に変更します。

g. 次のメソッド呼び出しを挿入したあと、規則を保存します。

```
<invoke class='testpackage.TestClass' name='concat'>
        <ref>arg1</ref>
        <ref>arg2</ref>
</invoke>
```

3. 次の手順を実行することで、カスタム Java コードのビルドとアプリケーションサーバーの起動を行います。

   a. プロジェクトノードを右クリックし、「Debug Project」を選択します。

      Identity Manager IDE が Web アプリケーションをビルドし、サーバーを起動し終わるまで待ちます (所要時間は数分)。

   b. プロンプトが表示されたら、パスワードを入力します。

      ログファイルを確認すれば、この規則が自動的にアップロードされたことがわかります。

   c. ブラウザウィンドウが表示されたら、それは閉じてしまってかまいません。

      ブラウザを無効にするには、プロジェクトを右クリックし、「Properties」を選択します。「Project Properties」ウィンドウで「Display Browser on Run」のチェックを外します。

4. Java コード内と XPRESS コード内にブレークポイントを設定します。

   a. `Test Rule.xml` で、`<Rule` タグの左にあるマージンをクリックしてブレークポイントを設定します。

   b. `TestClass.java` で、`return s1+s2;` 文の左にあるマージンをクリックしてブレークポイントを設定します。

5. 規則テスターを使って両方のブレークポイントを呼び出します。

   a. NetBeans メニューバーから「Window」>「Rule Tester Inputs」の順に選択することで、「Rule Tester Inputs」ウィンドウを開きます。

   b. プロジェクトウィンドウで `Test Rule.xml` を選択します。

   c. 「Rule Tester Inputs」ウィンドウを右クリックして、「Add value」を選択します。「New Value」ダイアログが表示されたら、次の情報を入力し、「OK」をクリックします。

      - 「Name」: **`arg1`**
      - 「Type」: **`String`**
      - 「Value」: **`myvalue1`**

d. 「Rule Tester Inputs」ウィンドウをもう一度右クリックして、「Add value」を選択します。「New Value」ダイアログが表示されたら、次の情報を入力し、「OK」をクリックします。

- 「Name」: **`arg2`**
- 「Type」: **`String`**
- 「Value」: **`myvalue2`**

e. `Test Rule.xml` ノードを展開し、「Test Rule」を右クリックしたあと、ポップアップメニューから「Test Rule」を選択します。

f. 規則を保存するかどうかを確認するメッセージが表示されたら、「Always」をクリックします。

この時点で、規則のブレークポイントに達しています。

g. 「Step-Into」ボタン  を 6 回クリックして、Java コード内のブレークポイントに到達させます。

この時点で Java デバッガが有効になっており、ローカル変数ウィンドウに Java 変数が表示されるようになっています。

注 Java と XPRESS を同時にデバッグしている場合、実際には 2 つのデバッガが同時に動作しており、両者は互いを認識していません。どちらのデバッガも、独自のブレークポイントとステップイン状態を保持しています。

Java コード内にブレークポイントが存在していないかぎり、ステップインを実行しても、XPRESS コードから Java コードへのステップインが発生することはありません。

h. 「Continue」をクリックします。

XPRESS デバッガ内の `</invoke>` タグに戻っていることに注意してください。また、Last Value にも、呼び出し結果として `myvalue1myvalue2` と表示されているはずです。

注 Java コードを離れるときには、常に「Continue」をクリックして Java デバッガの実行を再開するようにしてください。Java デバッガをステップ状態で離れると、デバッグのパフォーマンスが大幅に低下します。

i. 「Continue」を再度クリックして、「Rule Tester Output」ウィンドウ内で規則の結果を確認します。

6. 規則を次のように変更したあと、再実行します。
   a. `<ref>arg1</ref>` を `<s>myprefix</s>` に変更します。
   b. 規則を右クリックし、「Test Rule」を選択します。
   c. 「Continue」を 2 回クリックし、「Rule Tester Output」ウィンドウで新しい結果を確認します。
7. Java コードを次のように変更したあと、再実行します。
   a. `return s1+s2;` を `return s1+s2+"mysuffix";` に変更します
   b. Java コードを変更したので、再ビルドと再配備を行う必要があります。プロジェクトを右クリックし、「Debug Project」を選択します。
   c. 「Test Rule」ノードを右クリックし、「Test Rule」を選択します。
   d. 「Continue」を 2 回クリックすると、「Rule Tester Output」ウィンドウ内に次の結果が表示されるはずです。

      ```
      <String>myprefixmyvalue2mysuffix</String>
      ```

デバッガの停止

Identity Manager IDE デバッガを停止するには、メインメニューバーから「Run」>「Finish Debugging Session」の順に選択します。

デバッガの無効化

デバッグ終了後には必ずデバッガを無効にし、他のユーザーが誤って本稼働環境のアプリケーションサーバーに接続することのないようにします。

デバッガを無効化するには、次の手順に従います。

1. 「Projects」タブから、「Generic Objects」ノードを展開し、「System Configuration」をダブルクリックして、ソースエディタウィンドウに XML を表示します。
2. 下方へスクロールするか「Edit」>「Find」を使用して `serverSettings.default.debugger.enabled` 属性を探し、その値を **`false`** に変更します。

```
<Attribute name='serverSettings'>
        <Object>
          <Attribute name='default'>
            <Object>
              <Attribute name='debugger'>
                <Object>
                  <Attribute name='enabled'>
                    <Boolean>false</Boolean>
                  </Attribute>
```

3. メインメニューバーで、「File」>「Save」の順に選択して、変更を保存します。
4. アプリケーションサーバーを再起動します。

テスト環境の外部でのデバッガの実行

デバッグが必要な問題が本稼働環境に見つかった場合は、その問題をテスト環境で再現してデバッグを試みるのが最善です。デバッガでブレークポイントを設定すると、大量のトラフィックが発生している本稼働環境内のアプリケーションサーバーを短時間のうちに停止させる可能性があります。ブレークポイントを設定する位置によっては、他者がシステムを使用できないようにブロックすることも可能です。

独立したテスト環境でデバッグを実行できない場合は、次の手順に従います。

1. クラスタ内のノードのうちの 1 つをオフラインにすることにより、すべての有効なトラフィックをクラスタのサブセットに振り分けます (以後、このタスクの説明では、このノードを *server-a* とする)。
2. Identity Manager IDE を使用し、`SystemConfiguration` `serverSettings.server-a.debugger.enabled` プロパティーを `true` に設定してシステム設定オブジェクトを編集します。
3. server-a を再起動し、システム設定オブジェクトのプロパティー設定の変更を有効にします。
4. 適切なホスト (server-a)、ポート、コンテキストパス、ユーザー、およびパスワードを指定して、プロジェクト設定を変更します。
5. デバッガを開始します。
6. デバッグが終了したら、`serverSettings.server-a.debugger.enabled` を `false` に設定し、server-a を再起動して、稼働中の本稼働環境にデバッガが接続しないようにします。
7. server-a をオンラインのクラスタに再統合します。

NetBeans からの Identity Manager IDE のアンインストール

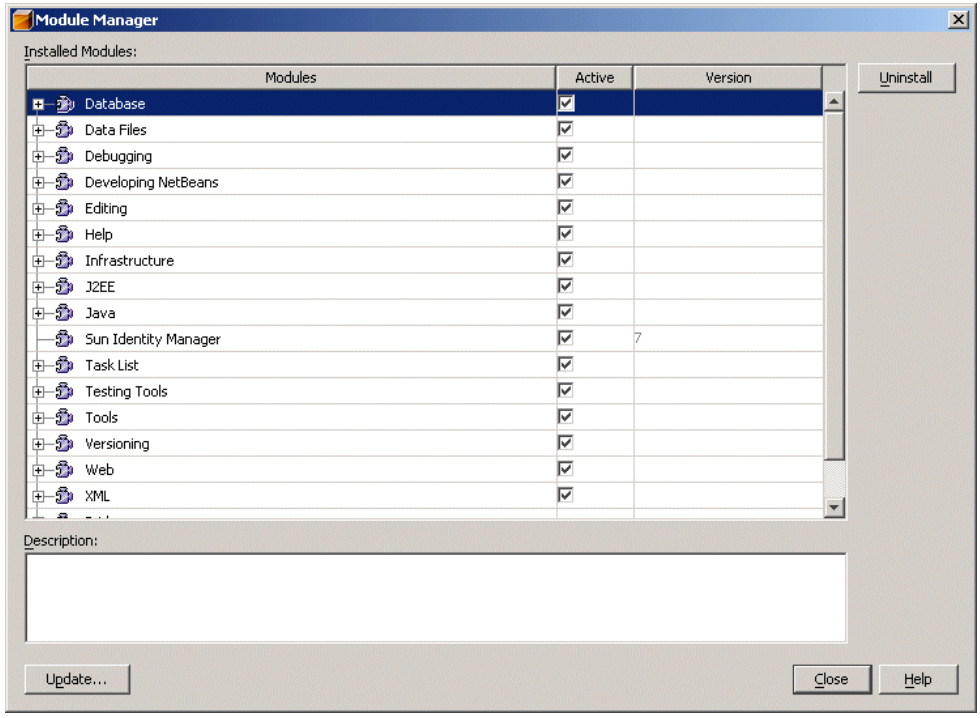
何らかの理由で、Identity Manager IDE モジュールを NetBeans からアンインストールする場合は、次の手順を実行します。

1. 必要に応じて、NetBeans を開きます。
2. NetBeans メニューバーで「Tools」>「Module Manager」の順に選択します。

   「Module Manager」ダイアログが表示され (図 1-43)、インストール済みのモジュールがすべてリストされます。「Active」列には、その中の有効なモジュールが示されます。

注	Identity Manager IDE では、必要のないモジュールを無効化することにより、起動時間を最小化し、メモリーを節約することができます。無効化されたモジュールはインストールディレクトリから削除されず、単に Identity Manager IDE によって無視されます。無効化されたモジュールは、いつでもふたたび有効にすることができます。

図 1-43　　アンインストールするモジュールの選択



3. モジュールのリストから Sun Identity Manager を選択します。
4. 「Uninstall」をクリックします。
5. Sun Identity Manager モジュールをアンインストールすることを確認するポップアップが表示されます。「OK」をクリックして、アンインストールプロセスを続行します。
6. プロセスが完了したら、モジュールマネージャーを閉じます。

Identity Manager IDE のトラブルシューティング

この節では、Identity Manager IDE の問題のトラブルシューティング時に必要となる可能性のある情報を提供します。

削除不能エラー

次の状況下では、削除不能を示すエラーメッセージが表示される可能性があります。

- サーバーの稼働中に「IdM」>「Manage Embedded Repository」を選択すると、次のエラーメッセージが表示されます。

  ```
  Unexpected Exception: 'Unable to delete
  [NetbeansHome]¥[Project]¥idm-repository¥idm.data'. See the logs for
  details.'
  ```

  このエラーを解決するには、サーバーを停止してからもう一度試します。

- サーバーの稼働中に「生成物を削除」を実行しようとすると、次の形式のエラーメッセージがいくつか発生します。

  ```
  Unable to delete file ../image/idm/WEB-INF/lib/*.jar
  ```

  このエラーを解決するには、サーバーを停止してからもう一度試します。

メモリー不足エラー

Identity Manager IDE を使用した作業中にメモリー不足エラーが発生する場合には、NetBeans のメモリー設定値を増やす必要があるかもしれません。対処方法については、NetBeans 製品マニュアルを参照してください。

「Tomcat Manager」ダイアログが表示され、ユーザー名とパスワードを要求される

標準の Identity Manager IDE プロジェクトの操作中にバンドル版の Tomcat インスタンスを起動すると「Tomcat Manager」ダイアログが表示される場合、それは一般に、次のいずれかの状態を示しています。

- 複数の Tomcat インスタンスが稼働している。

  ホストマシン上で稼働している Tomcat インスタンスが 1 つだけであることと、そのインスタンスがバンドル版の Tomcat と同じポート上で待機するように設定されていることを、確認する必要があります。

- 資格の不一致が発生した。

  バンドル版 Tomcat サーバーの一部として格納されている資格が、「Server Manager」の「Username」および「Password」フィールドに格納されている資格と一致する必要があります。フィールドの値の詳細については、次の Web サイトを参照してください。

  http://wiki.netbeans.org/wiki/view/FaqInstallationDefaultTomcatPassword

バンドル版 Tomcat のポート番号と格納された資格をチェックするには、次の手順に従います。

1. Identity Manager IDE の「Runtime」タブを選択し、「Servers」ノードと「Bundled Tomcat」ノードを展開します。
2. 「Bundled Tomcat」ノードを右クリックし、ポップアップメニューから「Properties」を選択します。
3. 「Server Manager」ダイアログが表示されたら、「Server Port」、「Username」、「Password」の各フィールドの値をチェックします。

第 2 章

規則の操作

この章では、Identity Manager の規則と規則ライブラリの操作方法について説明します。
これらの情報は、次のように構成されています。

この章は、次の節で構成されています。

- 規則と規則ライブラリについて
- デフォルト規則および規則ライブラリのカスタマイズ
- 新しい規則と規則ライブラリの作成
- 規則の参照
- 規則のセキュリティー保護

注

配備環境用に規則を作成、編集、およびテストするには、Sun Identity Manager Integrated Development Environment (Identity Manager IDE) を使用します。手順については、第 1 章「Identity Manager IDE の使用」を参照してください。

Identity Manager Business Process Editor (BPE) アプリケーションを使用して、規則を作成、編集、および妥当性検査することもできます。付録 A「ビジネスプロセスエディタの使用方法」を参照してください。

規則と規則ライブラリについて

ここでは、次の内容を説明します。

規則とは何か

規則とは、XPRESS、XML オブジェクト、または JavaScript 言語で記述された関数を含む Identity Manager リポジトリのオブジェクトです。Identity ManagerIdentity Manager 内で、規則は頻繁に使用されるロジックや静的な変数を、フォーム、ワークフロー、およびロール内で再利用できるように格納するためのメカニズムを提供します。

引数を規則に渡して、規則の動作を制御することができます。また、規則はフォームやワークフローによって保守される変数を、参照または変更することもできます。

注

XPRESS および XML オブジェクトは、両方とも XML で作成されるため、この章で紹介する XPRESS および XML オブジェクトのコード例はどちらも同じように見えます。

この章は、XPRESS 言語に精通していることを前提にしています。XPRESS の使用方法の詳細については、『Sun Java™ System Identity Manager ワークフロー、フォーム、およびビュー』を参照してください。

JavaScript での規則の記述については、144 ページの「JavaScript での規則の記述」を参照してください。

コード例 2-1 は単純な XML 規則です。規則名は `<Rule>` 要素の `name` 属性によって定義され、文字列値 `john` または `mary` を返します。規則本体は `<Rule>` 要素内の XPRESS 式です。

コード例 2-1 XML 規則の例

```
<Rule name='getApprover'>
   <cond><eq><ref>department></ref><s>sales</s></eq>
      <s>john</s>
      <s>mary</s>
   </cond>
</Rule>
```

規則を使用する理由

XPRESS を使用できる場所であればどこでも、規則を呼び出すことができます。特にワークフローやフォーム内でよく使用します。規則を使用することにより、ロジックのフラグメントや静的な値をカプセル化し、それを多くの場所で再利用できます。

XPRESS ロジックや静的な値を保存して再利用することには、次の利点があります。

- **メンテナンスが簡単**。規則を参照するフォームやワークフローすべてを変更する代わりに、1 つのオブジェクトを変更するだけで規則を変更できます。
- **開発を分散できる**。ユーザーは規則を参照するすべてのフォームやワークフローを意識する必要はなく、規則要件に集中して規則を作成することができます。
- **複雑さを感じさせない**。それほど技術のないユーザーでも、XML、ワークフロー、フォームの知識を必要としないインタフェースを使用して、単純な規則を修正できます。

フォーム内での規則の使用

通常、規則はフォーム内で呼び出して、`allowedValues` 表示プロパティーを算定したり、`<Disable>` 式内のフィールド可視性を制御したりします。フォーム内で次のものを格納して再利用するには、規則が最も効率的なメカニズムとなる場合があります。

- 企業内部門のリスト
- デフォルト値
- オフィスビルのリスト

フォームから規則を呼び出す場合は、これらのフォームのセキュリティー保護を適切に行うことが特に重要です。規則のセキュリティー保護については、153 ページの「規則のセキュリティー保護」を参照してください。

コード例 2-2 の規則は、役職のリストを返します。Identity Manager フォームでは、選択肢の名前のリストを計算するために、このような規則がよく使われます。新しい役職を追加または変更する場合は、この規則を変更するだけでよく、この規則を参照するすべてのフォームを修正する必要はありません。

コード例 2-2 役職リストを返す

```
<Rule name='Job Titles'>
   <list>
      <s>Sales</s>
      <s>Accounting Manager</s>
      <s>Customer Service Representative</s>
   </list>
</Rule>
```

コード例 2-3 のフィールドは、前の例で定義された規則を呼び出して、役職リストを選択ボックスで使用します。

注 この例では、`rule name` 要素に小文字の ***r*** が使用されていますが、これはこの要素が規則を定義するためではなく、規則を呼び出すために使用されているからです。

コード例 2-3 役職リストを選択ボックスで使用する

```
<Field name='global.jobTitle'>
   <Display class='Select'>
      <Property name='title' value='Job Title'/>
      <Property name='allowedValues'>
         <rule name='Job Titles'/>
      </Property>
   </Display>
</Field>
```

ロール内での規則の使用

規則を使用して、ロール定義に任意のリソース属性の値を設定することができます。規則が評価されると、規則はユーザービューの任意の属性を参照できるようになります。

コード例 2-4 の規則は、ユーザーのリソース記述の値を設定します (NT など)。この規則に関連付けられたロールを持つユーザーが作成されると、この規則に従って記述値が自動的に設定されます。

コード例 2-4 ユーザーのリソース記述の値の設定

```
<Rule name='account description'>
    <concat>
      <s>Account for </s>
      <ref>global.firstname</ref>
      <ref>global.lastname</ref>
      <s>.</s>
    </concat>
</Rule>
```

Identity Manager のフォームとワークフローは、別の規則の名前を動的に計算して呼び出す規則をサポートします。コード例 2-5 は、部門コードを計算する規則を呼び出すフォームフィールドを示しています。

コード例 2-5 部門コードを計算する規則の呼び出し

```
<Field name='DepartmentCode'>
   <Display class='Text'>
     <Property name='title' value='DepartmentCode'/>
   </Display>
     <Expansion>
        <rule>
          <cond>
            <eq>
               <ref>var1</ref>
               <s>Admin</s>
            </eq>
            <s>AdminRule</s>
            <s>DefaultRule</s>
          </cond>
        </rule>
     </Expansion>
</Field>
```

ワークフローアクティビティーには、サブプロセス名を動的に計算する規則を含む、サブプロセスを含めることもできます。

コード例 2-6 サブプロセス名の動的な計算

```
<Activity id='0' name='activity1'>
   <Variable name='ValueSetByRule'>
     <rule>
       <cond>
          <eq><ref>var2</ref><s>specialCase</s></eq>
          <s>Rule2</s>
          <s>Rule1</s>
       </cond>
       <argument name='arg1'>
          <ref>variable</ref>
       </argument>
     </rule>
   </Variable>
</Activity>
```

ワークフロー内での規則の使用

ワークフロー内では、規則を次の目的で使用できます。

- 承認者を算定する
- 遷移に条件を追加する
- アクションを実装する
- 承認のエスカレーションタイムアウトを計算する

コード例 2-7 は、承認リクエストを管理者に送信するために使用される手動アクションを示しています。このアクションにはタイムアウト値を指定できます。管理者が指定された時間内に応答しない場合、アクションは終了し、ワークフローは承認を別の管理者にエスカレーションします。

次の例では、タイムアウトが 86,400 秒、つまり 24 時間の固定値で指定されています。

コード例 2-7 固定値でタイムアウトを指定する

```
<Rule name='Approval Timeout'>
<i>86400</i>
</Rule>
```

コード例 2-8 は、`timeout` 規則を呼び出す手動アクションを示しています。

コード例 2-8 timeout 規則の呼び出し

```
<ManualAction>
   <Owner name='$(approver)'/>
   <Timeout>
      <rule name='Approval Timeout'/>
   </Timeout>
   <FormRule>
      <ref>approvalForm</ref>
   </FormRule>
</ManualAction>
```

ライブラリとは何か

規則ライブラリは、Identity Manager リポジトリに格納される XML 設定オブジェクトです。この設定オブジェクトには、1 つ以上の規則オブジェクトを含む、ライブラリオブジェクトが含まれます。

規則ライブラリ の作成は、密接に関連する規則を単一オブジェクトに編成するために便利な方法です。ライブラリを使用すると、リポジトリ内のオブジェクト数が削減され、フォームやワークフローの設計者は有用な規則を簡単に特定して呼び出せるようになるため、規則の保守が容易になります。

たとえば、コード例 2-9 は、2 つの異なるアカウント ID 生成規則を含むライブラリを示しています。

コード例 2-9 2 つのアカウント ID 生成規則を含む規則ライブラリの使用

```
<Configuration name='Account ID Rules'>
   <Extension>
      <Library>
         <Rule name='First Initial Last'>
            <expression>
               <concat>
                  <substr>
                     <ref>firstname</ref>
                     <i>0</i>
                     <i>1</i>
                  </substr>
                  <ref>lastname</ref>
               </concat>
            </expression>
         </Rule>
         <Rule name='First Dot Last'>
            <expression>
               <concat>
                  <ref>firstname</ref>
                  <s>.</s>
                  <ref>lastname</ref>
               </concat>
            </expression>
         </Rule>
      </Library>
   </Extension>
</Configuration>
```

Identity Manager IDE を使用して、デフォルトの規則ライブラリを表示および編集したり、既存のライブラリオブジェクトに新しい規則を追加したりできます。詳細については、42 ページの「リポジトリオブジェクトの操作」を参照してください。

デフォルト規則および規則ライブラリのカスタマイズ

Identity Manager IDE を使用してデフォルトの Identity Manager 規則を編集し、カスタムのステップセットに従うようにできます。Identity Manager には、デフォルト規則と規則ライブラリのライブラリが付属しており、次の内容が含まれます。

- Active Sync 規則
- 英数字規則ライブラリ
- 監査規則
- リソースアカウント除外規則サブタイプ
- 命名規則ライブラリ
- RegionalConstants ライブラリ

Active Sync 規則

Active Sync アダプタはリソース上のアカウントに加えられた変更を検出すると、着信属性を Identity Manager ユーザーにマップするか、または Identity Manager ユーザーアカウントを作成します。

注	Active Sync 規則には必ず `display.session` ではなく、`context` を使用します。

表 2-1 は、定義済みの Active Sync 規則を一覧で示しています。

表 2-1　定義済み Active Sync 規則

規則名	説明
ActiveSync has isDeleted set	「削除を更新として処理」が false に設定されているリソースからの移行で使用されます
No Correlation Rule	相関規則が不要な場合に使用するデフォルト規則

表 2-1　定義済み Active Sync 規則 (続き)

規則名	説明
No Confirmation Rule	相関規則が不要な場合に使用するデフォルト規則

英数字規則ライブラリ

英数字規則のデフォルトライブラリを使用することにより、Identity Manager のフォームとワークフロー内で、数字と文字を順序付けまたは表示する方法を制御できます。Identity Manager IDE では、このライブラリは `Alpha Numeric Rules` ライブラリオブジェクトとして表示されます。

表 2-2 は、このライブラリの規則を一覧で示しています。

表 2-2　デフォルトの英数字規則

規則名	説明
AlphaCapital	英大文字のリスト
AlphaLower	英小文字のリスト
Numeric	数字のリスト
WhiteSpace	空白文字のリスト
SpecialCharacters	共通特殊文字のリスト
IllegalNTCharacters	不正な NT 文字のリスト
legalEmailCharacters	`str` がすべて数字であるかどうかをテストして確認します。
isNumeric	`str` が数字だけかどうかをテストして確認します。
isAlpha	`str` が英字だけかどうかをテストして確認します。
hasSpecialChar	`str` に特殊文字が含まれているかどうかをテストして確認します。
hasWhiteSpace	`str` に空白文字が含まれているかどうかをテストして確認します。
isLegalEmail	`str` に含まれている文字が電子メールアドレスに適しているかどうかをテストして確認します。
hasIllegalNTChar	`str` がすべて数字であるかどうかをテストして確認します。
stringToChars	指定された文字列 (`testStr` 引数として渡される) をコンポーネント文字のリストに変換します。
StripNonAlphaNumeric	英数字以外の文字を `testStr` から削除します。

監査規則

複雑さを最小限に抑えながら高レベルの設定を可能にするために、Identity Auditor は、監査ポリシーやアクセススキャンオブジェクト設定内で規則をうまく利用します。

表 2-3 は、監査ポリシー是正の動作方法やアクセススキャンの操作方法をカスタマイズするために使用できる規則の概要を示しています。

表 2-3　監査規則の種類のクイック参照

規則の種類	規則の例	subTypes および authTypes	目的
アテスター	Default Attestor	SubType: ATTESTORS_RULE AuthType: AccessScanRule	手動エンタイトルメントに対してデフォルトアテスターを指定することで、アテステーションプロセスを自動化します。
アテスターエスカレーション	Default EscalationAttestor	SubType: AttestorEscalationRule AuthType: AccessScanRule	手動アテステーションに対してデフォルトのエスカレーションユーザーを指定することで、アテステーションプロセスを自動化します。
監査ポリシー	Compare Accounts to Roles	SubType: SUBTYPE_AUDIT_POLICY_RULE SubType: SUBTYPE_AUDIT_POLICY_SOD_RULE AuthType: AuditPolicyRule	ユーザーアカウントを現在のロールによって指定されたアカウントと比較します。
	Compare Roles to Actual Resource Values	SubType: SUBTYPE_AUDIT_POLICY_RULE **SubType:** SUBTYPE_AUDIT_POLICY_SOD_RULE AuthType: AuditPolicyRule	現在のリソース属性を、現在のロールによって指定されたリソース属性と比較します。
是正ユーザーフォーム		SubType: USER_FORM_RULE AuthType: 指定しない	特定のポリシー違反に応答するときにどの部分のユーザービューを表示するのかを、監査ポリシーの作者が制約できるようにすることで、アテステーションプロセスを自動化します。

表 2-3　監査規則の種類のクイック参照 (続き)

規則の種類	規則の例	subTypes および authTypes	目的
是正者	Default Remediator	SubType: REMEDIATORS_RULE AuthType: AccessScanRule	是正状態で作成されるすべてのエンタイトルメントに対して是正者を指定することで、是正プロセスを自動化します。
レビュー決定	Reject Changed Users	SubType: REVIEW_REQUIRED_RULE AuthType: AccessScanRule	ユーザーエンタイトルメントレコードを自動的に却下することによって、アテステーションプロセスを自動化します。
	Review Changed Users	SubType: REVIEW_REQUIRED_RULE AuthType: AccessScanRule	ユーザーエンタイトルメントレコードを自動的に承認することによって、アテステーションプロセスを自動化します。
	Review Everyone	SubType: REVIEW_REQUIRED_RULE AuthType: AccessScanRule	一部のユーザーエンタイトルメントレコードに手動アテステーションを必要とすることで、アテステーションプロセスを自動化します。
ユーザー範囲	All Administrators	SubType: USER_SCOPE_RULE AuthType: AccessScanRule	アクセススキャンによってスキャンされるユーザーのリストを柔軟に選択できるようにします。
	All Non-Administrators	SubType: USER_SCOPE_RULE AuthType: AccessScanRule	アクセススキャンによってスキャンされるユーザーのリストを柔軟に選択できるようにします。
	Users Without a Manager	SubType: USER_SCOPE_RULE AuthType: AccessScanRule	アクセススキャンによってスキャンされるユーザーのリストを柔軟に選択できるようにします。
ViolationPriority	ViolationPriority	SubType: 指定しない AuthType: EndUserAuditorRule	カスタマイズ。配備において、有効な違反の優先度と対応する表示文字列を指定できるようにします。
ViolationSeverity	ViolationSeverity	SubType: 指定しない AuthType: EndUserAuditorRule	カスタマイズ。配備において、有効な違反の重要度と対応する表示文字列を指定できるようにします。

次の節では、次の Identity Auditor 規則について説明し、それぞれをカスタマイズする場合の方法とその理由について説明します。

アテスター規則

保留状態で作成されたユーザーエンタイトルメントは、だれかがアテストする必要があります。Identity Auditor は、アクセスレビュー時に各ユーザービューをアテスター規則に渡し、最初のアテステーションリクエストを取得する担当者を決定します。

`WSUser` オブジェクトの `idmManager` 属性には、ユーザーのマネージャーの Identity Manager アカウント名と ID が含まれています。

- `idmManager` の値を定義したら、アテスター規則は、エンタイトルメントレコードが表すユーザーのアテスターとして `idmManager` を返します。
- `idmManager` の値が NULL の場合、アテスター規則は `Configurator` をアテスターとして返します。

代替の実装を使用して、`IdmManager` と任意のリソース所有者の両方を (ビューに含まれるリソースの) アテスターとして指定できます。この規則では、現在のユーザービューと `LighthouseContext` オブジェクトを入力値として使用するので、Identity Manager によって認識されているすべてのデータを使用できます。

入力値 : 次の入力変数を受け入れます。

- `lhcontext`: LighthouseContext
- `userEntitlement`: 現在のユーザービュー

カスタムアテスト規則には、次を指定する必要があります。

SubType: `ATTESTORS_RULE`

AuthType: `AccessScanRule`

呼び出し: アクセススキャン時で、すべての監査ポリシーの評価後だが、ユーザーエンタイトルメントをディスパッチする前

返される値: ゼロ以上の Identity Manager アテスター名 (特定のユーザーエンタイトルメントのアテストを担当するユーザー) または NamedValue のペアのリスト。

- 結果が文字列の場合は、Identity Manager アカウント ID に解決する必要があります。アクセススキャンの委任が有効である場合、アクセススキャンはコードによって返される Identity Manager ユーザーの委任設定を使用します。
- 結果が NamedValue である場合、これはバインドされた委任ペア [委任者 , 被委任者] であると想定され、アクセススキャンはこれ以上の解決を行いません。

注	NamedValue ペアに対しては何もチェックが行われません。規則が NamedValue ペア要素を返した場合、これらは検証されずに渡されます。

- 結果が有効な Identity Manager ユーザー名でない場合、規則はスキャンタスク結果にエラーを追加しますが、スキャンスレッドは続行されます。
- 結果が長さゼロ (0) のリストの場合、だれもアテステーションリクエストを処理しないので、リクエストは保留状態のままになります。
- 結果が文字列でも NamedValue でもない場合は、例外が発生し、スキャンスレッドが中止されます。

定義済み規則: Default Attestor

場所:「コンプライアンス」>「ポリシーの管理」>「アクセススキャン」>「アテスター規則」

アテスターエスカレーション規則

指定された時間内にアテスターが何もアクションを起こさなかったためにアテステーションが時間切れになった場合、ワークフローはアテスターエスカレーション規則を呼び出します。この規則はサイクルカウントに基づいて、エスカレーションチェーン内の次の人を返します。

入力値: 次の入力変数を受け入れます。

- wfcontext: WorkflowContext
- userEntitlement: ユーザーエンタイトルメントのビュー (ユーザービューを含む)
- cycle: エスカレーションレベル。初めてのエスカレーションの場合、サイクルは 1 です。
- attestor: アテステーションリクエストが時間切れになる前にアテストに失敗したアテスターの名前

カスタムアテスターエスカレーション規則には、次を指定する必要があります。

SubType: `AttestorEscalationRule`

AuthType: `AccessScanRule`

呼び出し : アテステーションワークフローにおいて、作業項目が時間切れした場合 (デフォルトのタイムアウトは *0* に設定されており、絶対に時間切れしないようになっている)。

返される値 : 単一のアテスター名か、複数のアテスター名のリスト。これらは有効な Identity Manager アカウント名である必要があります。

- アテスターにマネージャーが存在していない場合、アテスターエスカレーション規則は `Configurator` を返します。
- 結果が無効なアカウント名や NULL の場合、アテステーションの作業項目はエスカレーションされません。

定義済み規則 : Default EscalationAttestor

場所 :「コンプライアンス」>「ポリシーの管理」>「アクセススキャン」>「アテスターエスカレーション規則」

監査ポリシー規則

監査ポリシーには、監査対象のオブジェクトを表すデータに適用される規則のセットが含まれています。各規則はブール値 (およびいくつかのオプション情報) を返すことができます。

ポリシーに違反しているかどうかを決定するために、監査ポリシーは各規則の結果の論理演算を評価します。監査ポリシーに違反している場合、コンプライアンス違反オブジェクトが、通常はポリシー、規則、または監査対象となったものごとに 1 つずつ発生します。たとえば、5 つの規則を含む監査ポリシーの場合は、5 つの違反が発生します。

入力値 : なし

カスタム監査ポリシー規則には、次を指定する必要があります。

SubType:

- `SUBTYPE_AUDIT_POLICY_RULE` (監査ポリシー規則の場合)
- `SUBTYPE_AUDIT_POLICY_SOD_RULE` (監査ポリシー SOD 規則の場合)

  SOD (*separation* または *segregation of duties*) 規則は、規則の出力内でリスト要素を作成することが想定されている点で、通常の規則と異なっています。リスト要素は必須ではありませんが、これが存在しない場合、何らかの関連する違反が発生します。こうした違反は SOD レポートでは無視されます。

AuthType: `AuditPolicyRule`

注 監査ポリシーウィザードを使用して監査ポリシー規則を作成する場合、このウィザードはデフォルトで `AuditPolicyRule` authType を使用します。

Identity Manager IDE または Identity Manager ビジネスプロセスエディタ (BPE) を使用して監査ポリシー規則を作成する場合は、必ず `AuditPolicyRule` authType を指定するようにしてください。

呼び出し: 監査ポリシーの評価時

返される値: 監査ポリシー規則は整数値を返す必要がありますが、この値は次のいずれかのように表すことができます。

- 純粋な整数:

```
<i>1</i>
```

- 追加データのマップ内にある整数:

```
<map>
  <s>result</s>
  <i>1</i>
  ...
</map>
```

監査ポリシーがマップを返す場合、その他の要素が結果のコンプライアンス違反に影響することもあります。これらの要素には次のものがあります。

- **resources 要素**: コンプライアンス違反が 2 つのリソース (`resource one` および `resource two`) を参照するようになります。コンプライアンス違反には (名前を ID に解決できるように) 実際のオブジェクト参照が含まれているので、これらの値は実際のリソース名でなくてはなりません (デフォルトは *no resource*)。

```
<s>resources</s>
<list>
    <s>resource one</s>
    <s>resource two</s>
</list>
```

- **severity 要素** : コンプライアンス違反が指定した重要度になります (デフォルトは *1*)。

```
<s>severity</s>
<i>3</i>
```

- **priority 要素** : コンプライアンス違反が指定した優先度になります (デフォルトは *1*)。

```
<s>priority</s>
<i>2</i>
```

- **violation 要素** : 監査ポリシーが `true` と評価した場合でも、監査スキャナが規則違反を作成しないようにします。

  デフォルトでは、監査ポリシーが `true` と評価した場合、ゼロ以外の値を返すすべての規則に対してコンプライアンス違反が作成されます。この要素をゼロ (0) に設定することで、規則が `true` を返しても違反が作成されないようにすることができます。

```
<s>violation</s>
<i>0</i>
```

注 監査ポリシーウィザードでは、単一のリソースを参照し、整数値 (マップではない) を返す規則のみが作成されます。

以前のマップ関連の機能を使用するには、ユーザー自身で規則を作成する必要があります。`sample/auditordemo.xml` には、非常に精巧な監査ポリシー規則の例がいくつか提示されています。

定義済み規則 :

- Compare Accounts to Roles: ユーザーアカウントを、ロールによって指定されたアカウントと比較します。ロールによって参照されないアカウントはすべてエラーと見なされます。

- Compare Roles to Actual Resource Values: 現在のリソース属性を、現在のロールによって指定されたリソース属性と比較します。異なるものはすべてエラーと見なされ、ロールによって指定されていないリソースやリソース属性はすべて無視されます。

場所 : なし

是正ユーザーフォーム規則

是正ユーザーフォーム規則は、特定のポリシー違反に応答するときにどの部分のユーザービューを表示するのかを、監査ポリシーの作者が制約できるようにします。

エンタイトルメント是正の処理中に是正者がユーザーを編集すると、JSP (`approval/remModifyUser.jsp`) が是正ユーザーフォーム規則を呼び出します。この規則によって、ユーザー編集のための適切なフォームをアクセススキャンが指定できるようになります。是正者がすでにユーザーフォームを指定済みである場合、アクセススキャンはそちらのフォームを使用します。

入力値 : `item` 変数 (是正作業項目) を受け入れます。

カスタム是正ユーザーフォーム規則には、次を指定する必要があります。

Subtype: `USER_FORM_RULE`

AuthType: 指定しない

呼び出し : JSP フォームの処理中に、是正者が是正フォーム上で「ユーザーの編集」をクリックしたあとに呼び出される

返される値 : ユーザーフォームの名前または NULL

定義済み規則 : なし

場所 :

- 「コンプライアンス」>「ポリシーの管理」>「アクセススキャン」>「是正ユーザーフォーム規則」
- 「コンプライアンス」>「ポリシーの管理」>「監査ポリシー」>「是正ユーザーフォーム規則」

是正者規則

アクセスレビュー時にはすべてのユーザービューが是正者規則に渡され、最初の是正リクエストを取得する担当者が決定されます。この規則はアテスター規則と類似していますが、是正者規則は作業項目が是正状態で作成されたときに呼び出される点が異なります。

入力値 : 次の入力変数を受け入れます。

- lhcontext: LighthouseContext
- userEntitlement: 現在のユーザービュー

カスタム是正者規則には、次を指定する必要があります。

SubType: REMEDIATORS_RULE

AuthType: AccessScanRule

呼び出し: アクセススキャン時で、すべての監査ポリシーの評価後で、ユーザーエンタイトルメントをディスパッチする前

返される値: ゼロ以上の Identity Manager 是正者名のリストまたは NamedValue のペア

- 結果が文字列の場合は Identity Manager ユーザーに解決され、アクセススキャンの委任が有効である場合は、そのユーザーの委任データが使用されます。
- 結果が NamedValue の場合は、バインドされた委任ペア [**委任者** , 被委任者] であると想定されます。
- 結果が 1 つ以上の無効な Identity Manager ユーザー名の場合、スキャンタスク結果に問題点を示すエラーが追加されますが、スキャンスレッドは続行されます。
- 結果が文字列でも NamedValue でもない場合は、例外が発生し、スキャンスレッドが中止されます。
- 結果が長さゼロ (0) のリストの場合、だれも是正リクエストを処理しないので、リクエストは保留状態のままになります。

注	NamedValue ペアに対しては何もチェックが行われません。規則が NamedValue ペア要素を返した場合、これらは検証されずに渡されます。

定義済み規則: Default Remediator

場所:「コンプライアンス」>「ポリシーの管理」>「アクセススキャン」>「是正者規則」

レビュー決定規則

アクセスレビュー時にはすべてのユーザービューがレビュー決定規則に渡され、対応するユーザーエンタイトルメントレコードの自動承認または自動却下が可能か、是正状態への自動配置が可能か、あるいは手動でアテストする必要があるかのどうかが判別されます。

レビュー決定規則を使用すると、アクセスレビューの効率を大幅に向上させることができます。

ユーザーを自動承認または自動却下できるようにする制度的な知識をすべてカプセル化し、その知識をこの規則内で表すことができたら、必要とされる手動アテステーションの数が減少し、レビュー全体のパフォーマンスが向上します。

さらに改善を重ねれば、この規則によって、アテスターに対して「ヒント」として表示される情報を返すこともできます。たとえば、ユーザーがリソースに対して優先的にアクセスできることを規則によって通知する場合は、126 ページの例 1 にある `attestorHint` を参照してください。

また、レビュー決定規則はユーザービュー (コンプライアンス違反を含む) にアクセスして、ユーザーの以前のユーザーエンタイトルメントを比較することができるので、これまでに承認されているユーザーエンタイトルメントと同じ (または異なる) すべてのユーザーエンタイトルメントをこの規則によって承認または却下することもできます。

入力値 : 次の入力変数を受け入れます。

- `context`: LighthouseContext
- `review.scanId`: 現在のアクセススキャン ID
- `review.username`: スキャンされているユーザーの Identity Manager アカウント名
- `review.userId`: スキャンされているユーザーの Identity Manager ID
- `attestors`: アテスターの Identity Manager アカウント名
- `userView`: 現在のユーザービュー

カスタムレビュー決定規則には、次を指定する必要があります。

SubType: `REVIEW_REQUIRED_RULE`

AuthType: `AccessScanRule`

呼び出し : アクセススキャン時で、すべての監査ポリシーの評価後で、ユーザーエンタイトルメントをディスパッチする前

返される値 : 整数またはマップ

- 規則が整数を返す場合、その値は次のように解釈されます。
  - **-1**: アテステーションの必要なし
  - **0**: アテステーションを自動却下
  - **1**: 手動アテステーション
  - **2**: アテステーションを自動承認
  - **3**: アテステーションを自動是正

アテステーションが自動是正モードに設定されている場合、Identity Manager は AccessReviewRemediation 作業項目を作成し、その作業項目の経路をアクセススキャンに関連付けられた是正者規則に設定します。

- 規則がマップを返す場合、その出力は次の例のいずれかと同じになります。

  例 1: ユーザーエンタイトルメントを手動でアテストします。規則は手動アテスターへのヒントを提示しています。

```
<map>
   <result>
   <i>1</i>
   <s>reason</s>
   <s><reason that the attestation was auto-approved/rejected></s>
   <s>attestorHint</s>
   <s><hint to attestor></s>
</map>
```

注 出力マップの attestorHint 値は、文字列または文字列のリストでなくてはなりません。

例 2: ユーザーエンタイトルメントを自動却下します。却下のコメントには、グループメンバーシップが許可されていないことが示されています。

```
<map>
  <s>result</s>
  <i>0</i>
  <s>reason</s>
  <s>User belongs to NT group Domain Administrators</s>
</map>
```

注 attestorHint の値は、ユーザーインタフェースを通してアテスターに表示されます。reason の値は、アテステーション履歴に記録されます。

定義済み規則 :

- Reject Changed Users: 最後の承認状態以後に変更されたユーザーエンタイトルメントを自動的に却下し、変更されていないユーザーエンタイトルメントを自動的に承認します。すべての不明な UserViews は手動アテステーションに転送されます。ユーザービューの `accounts` セクションのみが比較されます。
- Review Changed Users: 最後の承認状態以後にアカウントデータが変更されていないユーザーを自動的に承認します。アカウントデータの変更されたユーザーや、承認データのないユーザーは、手動でアテストします。ユーザービューの `accounts` セクションのみが比較されます。
- Review Everyone: すべてのユーザーエンタイトルメントレコードを手動アテステーションに転送します。

場所 :「コンプライアンス」>「アクセススキャンの管理」>「アクセススキャン」>「レビュー決定規則」

ユーザー範囲規則

アクセススキャンのユーザー範囲が規則によって制限されている場合、ユーザー範囲規則は、スキャンするユーザーのリストを決定するための評価を行います。

入力値 : `lhcontext` 変数 (LighthouseContext) を受け入れます。

カスタムユーザー範囲規則には、次を指定する必要があります。

SubType: `USER_SCOPE_RULE`

AuthType: `AccessScanRule`

呼び出し : アクセススキャンの開始時

返される値 : Identity Manager ユーザー名または Identity Manager ユーザー名のリスト。どの名前も有効な Identity Manager ユーザー名である必要があります。

- 有効な Identity Manager ユーザー名に解決できない名前が結果に含まれている場合、規則はエラーを返します。
- 結果に重複するユーザー名が含まれている場合、規則はエラーを返します。

注

- 同じユーザーを複数回スキャンするアクセススキャンは、同じユーザーの後続インスタンスのためにアテステーションワークフローを作成しようとして失敗する場合があります。そのため、ユーザー範囲規則をカスタマイズして実装する場合には、出力でユーザーの重複を避けるためのチェックを含めるようにします。
- この規則は、スキャンを実行している管理者が使用できないアカウントを返すことがあります。この場合、スキャンはそのアカウントのユーザービューを取得しようとして失敗し、スキャンタスクにエラーが発生します。

定義済み規則 :

- All Administrators: 管理機能が割り当てられているすべてのユーザーを返します。
- All Non-Administrators: 管理機能が割り当てられていないすべてのユーザーを返します。
- Users Without Manager: 管理者 (`idmManager`) が割り当てられていないすべてのユーザーアカウントを返します。

場所 :「コンプライアンス」>「アクセススキャンの管理」>「アクセススキャン」>「ユーザー範囲規則」

ViolationPriority 規則

ViolationPriority 規則を使用すると、配備において、有効な違反の優先度と対応する表示文字列を指定できるようになります。

入力値 : なし

カスタム ViolationPriority 規則には、次を指定する必要があります。

SubType: 指定しない

AuthType: `EndUserAuditorRule`

呼び出し : 違反リストを表示するときと、違反の優先度を変更するとき

返される値 : 優先度の整数値と対応する文字列を示す `key/value` ペアのリスト。この規則はマップではなくリストを返すので、整数値は連続している必要があります。

注

この規則をカスタマイズして、任意の優先度設定の表示値を変更することができます。

コンプライアンス違反が作成されたら、是正作業項目リストのビューア内で優先度の値を変更できます。1 つ以上の是正作業項目を選択して「優先度の設定」を選択すると、優先度の値を変更できるようになります。

これらの値を是正作業項目リストビューに表示するには、`includeCV` オプションを **`true`** (デフォルトは `false`) に設定することで、`approval/remediate.jsp` ページを変更する必要があります。ただし、詳細なビューを有効にするとパフォーマンスに影響が出るので、是正の多い配備では受け入れられないこともあります。

カスタム値は ViolationPriority 規則をマップではなく配列であると想定します。したがって、整数値として 100 を使用する場合、規則には (整数と文字列が交互になった) 200 の要素が必要です。リストにはこの整数の文字列マッピングが表示されると同時に、フォーム内の変更を行なった場所に選択内容が入力されます。

定義済み規則 : ViolationPriority

場所 : 是正リストフォームから呼び出される

ViolationSeverity 規則

ViolationSeverity 規則を使用すると、配備において、有効な違反の重要度と対応する表示文字列を指定できるようになります。

入力値 : なし

カスタム ViolationSeverity 規則には、次を指定する必要があります。

SubType: 指定しない

AuthType: `EndUserAuditorRule`

呼び出し : 違反リストを表示するときと、違反の重要度を変更するとき

返される値 : 重要度の整数値と対応する文字列を示す `key/value` ペアのリスト。この規則はマップではなくリストを返すので、整数値は連続している必要があります。

注

この規則をカスタマイズして、任意の重要度設定の表示値を変更することができます。

コンプライアンス違反が作成されたら、是正作業項目リストのビューア内で重要度の値を変更できます。1 つ以上の是正作業項目を選択して「優先度」を選択すると、重要度の値を変更できるようになります。

これらの値を是正作業項目リストビューに表示するには、includeCV オプションを **true** (デフォルトは false) に設定することで、approval/remediate.jsp ページを変更する必要があります。ただし、詳細なビューを有効にするとパフォーマンスに影響が出るので、是正の多い配備では受け入れられないこともあります。

カスタム値は ViolationSeverity 規則をマップではなく配列であると想定します。したがって、整数値として 100 を使用する場合、規則には (整数と文字列が交互になった) 200 の要素が必要です。リストにはこの整数の文字列マッピングが表示されると同時に、フォーム内の変更を行なった場所に選択内容が入力されます。

定義済み規則 : ViolationSeverity

場所 : 是正リストフォームから呼び出される

監査規則の複数のアカウントタイプのサンプル

監査規則の複数のアカウントタイプのサンプル規則を使用すると、リソースごとに複数のユーザーアカウントを動的にテストすることができます。次に例を示します。

1. 複数のアカウントタイプを持つリソースを設定します (コード例 2-10 を参照)。

コード例 2-10 監査規則の複数のアカウントタイプのサンプル規則

```
<?xml version='1.0' encoding='UTF-8'?>
<!DOCTYPE Waveset PUBLIC 'waveset.dtd' 'waveset.dtd'>
<Waveset>
<Rule subtype='IdentityRule' name='Administrator Identity'>
  <concat>
     <s>adm</s>
     <ref>attributes.accountId</ref>
  </concat>
</Rule>
</Waveset>
```

2. 2 つのアカウントを持つユーザーをリソースに追加して、新しいリソース属性を別々に直接割り当てることができるようにユーザーフォームを設定します。

```
account[Simulated Resource].department
account[Simulated Resource|admin].department
```

3. それぞれのアカウントに別々の値を割り当て、ポリシー規則をテストします。

場所: `sample/rules/SampleAuditorRuleMultipleAccountTypes.xml`

コンプライアンス違反規則

コンプライアンス違反は、優先度や重要度によって違反を見分けられるようにする重要度と優先度の数値属性をサポートしています。これらの属性は、監査規則の出力に基づいて、違反に割り当てることができます。

たとえば、監査規則によって次のような出力が提示された場合、コンプライアンス違反に割り当てられる重要度は 3 で、優先度は 4 になります。

```
<map>
  <s>result</s>
  <i>1</i>
  <s>severity</s>
  <i>3</i>
  <s>priority</s>
  <i>4</i>
</map>
```

次の規則は、コンプライアンス違反の数値と表示文字列をマッピングします。

- ViolationSeverity: 違反の重要度を示します。
- ViolationPriority: コンプライアンス違反の対処されるべき順序を示します。

Identity Auditor では、重要度設定または優先度設定の表示値を変更することで、これらの規則をカスタマイズすることができます。

コンプライアンス違反の作成後、是正作業項目リストビューアで 1 つ以上の是正作業項目を選択して「優先度」をクリックすると、重要度や優先度の値を表示および変更できるようになります。

注 重要度や優先度の値を是正作業項目リストビューアに表示するには、approval/remediate.jsp ページを変更して、includeCV オプションを true (デフォルトは false) に設定する必要があります。

ただし、詳細なビューを有効にするとパフォーマンスに影響が出るので、是正の多い配備では受け入れられないこともあります。

DateLibrary

DateLibrary は、配備における日付と時間の表示方法を制御する規則のデフォルトライブラリです。Identity Manager IDE では、このライブラリは Date Library ライブラリオブジェクトとして表示されます。

次の規則を使用して、日付や時間の表示方法をカスタマイズします。

- Date Validation: 有効な日付の文字列を決定します。

  この規則は、mm/dd/yy の形式で 1 つの引数を取ります (月または日の値が 1 桁で入力された場合は、その値を mm/dd/yy の形式に概算する)。

  返される値 : 入力された文字列に有効な日付コンポーネントが含まれる場合は true

- Validate Day Month Year: 有効な年、月、日の文字列を決定します。

  この規則は、month、day、year という 3 つの引数を取ります。月または日の値が 1 桁で入力された場合は、その値を mm/dd/yy の形式に概算します。

  返される値 : 入力された文字列が有効な日付である場合は true

- Validate Time: 有効な時間の文字列を決定します。

  この規則は、HH:mm:ss の形式で 1 つの引数を取ります。時間の文字列がこの形式でなかったり、コンポーネントが範囲外 (たとえば、時間がゼロより小さかったり、23 より大きい場合) である場合、規則は false を返します。

  返される値 : 入力された文字列が有効な時間である場合は true

リソースアカウント除外規則サブタイプ

ExcludedAccountRule サブタイプは、リソース操作からのリソースアカウントの除外をサポートします。

入力値 : 次の入力変数を受け入れます。

- accountID: テストされる文字列アカウント ID。

  accountID パラメータを、Identity Manager から除外するようにする 1 つ以上のリソースアカウントと比較できます。

- operation: 実行されるリソース操作。

  この規則では、operation パラメータを使用して、operation パラメータで指定されたアクションから免除するリソースアカウントをより細かく制御できます。operation パラメータを規則内で使用しない場合、規則によって識別されるすべてのアカウントは、リストされているすべての操作から除外されます。

  operation パラメータには次の値を含めることができます。

  - create
  - update
  - delete
  - rename (検出された変更が、新規アカウント ID である場合にのみ使用)
  - rename_with_update
  - list
  - iapi_create (Active Sync 内でのみ使用)
  - iapi_update (Active Sync 内でのみ使用)
  - iapi_delete (Active Sync 内でのみ使用)

authType: ExcludedAccountsRule

コード例 2-11 は、サブタイプを使用する例を示していますが、この例では UNIX アダプタの指定されたリソースアカウントが除外されます。

コード例 2-11 サブタイプの使用例

```
<Rule name='Excluded Resource Accounts' authType='ExcludedAccountsRule'>
   <RuleArgument name='accountID'/>
   <defvar name 'excludedList'>
      <List>
         <String>root</String>
         <String>daemon</String>
         <String>bin</String>
         <String>sys</String>
         <String>adm</String>
         <String>uucp</String>
         <String>nuucp</String>
         <String>listen</String>
         <String>lp</String>
      </List>
   </defvar>
   <cond>
      <eq>
         <contains>
            <ref>excludedList</ref>
            <ref>accountID</ref>
         </contains>
         <i>1</i>
      </eq>
      <Boolean>true</Boolean>
      <Boolean>false</Boolean>
   </cond>
</Rule>
```

コード例 2-12 は、operation パラメータの使用例を示しています。
このパラメータを使用することにより、Active Sync が "Test User" リソースに対して実行中である場合にも、Identity Manager に影響を与えることなく、このリソースアカウントを操作できます。

コード例 2-12 operation パラメータの使用

```
<Rule name='Example Excluded Resource Accounts' authType='ExcludedAccountsRule'>
```

コード例 2-12 operation パラメータの使用 (続き)

```
<!--
「Administrator」アカウント上のすべての操作を除外する
「Test User」アカウント上の activeSync イベントを除外する
-->

   <RuleArgument name='accountID'/>
   <RuleArgument name='operation'/>

<!-- IAPI 操作のリスト -->
   <defvar name='iapiOperations'>
      <List>
         <String>iapi_create</String>
         <String>iapi_update</String>
         <String>iapi_delete</String>
      </List>
   </defvar>
   <or>

   <!-- 管理者アカウントは常に無視する。-->
      <cond>
         <eq>
            <s>Administrator</s>
            <ref>accountID</ref>
         </eq>
         <Boolean>true</Boolean>
         <Boolean>false</Boolean>
      </cond>

<!-- 「Test User」アカウントの IAPI イベントは無視する -->
      <and>
         <cond>
            <eq>
               <contains>
                  <ref>iapiOperations</ref>
                  <ref>operation</ref>
               </contains>
               <i>1</i>
```

コード例 2-12　operation パラメータの使用 (続き)

```
            </eq>
            <Boolean>true</Boolean>
            <Boolean>false</Boolean>
         </cond>
         <cond>
            <eq>
               <ref>accountID</ref>
               <s>Test User</s>
            </eq>
            <Boolean>true</Boolean>
            <Boolean>false</Boolean>
         </cond>
      </and>
   </or>
</Rule>
```

命名規則ライブラリ

命名規則ライブラリは命名規則のデフォルトライブラリで、これを使用することにより規則の処理後に名前が表示される方法を制御できます。Identity Manager IDE では、このライブラリは `NamingRules` ライブラリオブジェクトとして表示されます。

表 2-4 は、デフォルトの命名規則を一覧で示しています。

表 2-4　デフォルトの命名規則

規則名	説明 / 出力
AccountName - First dot Last	Marcus.Aurelius
AccountName - First initial Last	MAurelius
AccountName - First underscore Last	Marcus_Aurelius
Email	marcus.aurelius@example.com
Fullname - First space Last	Marcus Aurelius
Fullname - First space MI space Last	Marcus A Aurelius
Fullname - Last comma First	Aurelius, Marcus

RegionalConstants ライブラリ

RegionalConstants ライブラリは地域定数規則のデフォルトライブラリで、これを使用することにより、州、日、月、国、およびカナダの州の表示方法を制御できます。Identity Manager IDE では、このライブラリは `RegionalConstants` 規則ライブラリオブジェクトとして表示されます。

表 2-5 は、デフォルトの命名規則を一覧で示しています。

表 2-5 デフォルト地域定数規則

規則名	説明
US States	米国の州のフルネームのリスト。
US State Abbreviations	米国の州の標準的な省略名のリスト。
Days of the Week	曜日のフルネームのリスト。
Work Days	週の 5 労働日のリスト (米国)。
Months of the Year	年間の月のフルネームのリスト。
Month Abbreviations	選択した月の標準的な省略名のリスト。
Numeric Months of the Year	12 か月のリストを返します。
Days of the Month	31 日 (一か月) のリストを返します。
Smart Days of the Month	数字で月と 4 桁の年に基づいたリストを返します。
Countries	世界の国々の名前を英語でリストします。
Canadian Provinces	カナダの州の名前を英語でリストします。

新しい規則と規則ライブラリの作成

ここでは、配備用に規則を作成する方法を説明します。また、次の内容についても説明します。

- 規則構文について
- JavaScript での規則の記述

注

- ロールへの規則の適用については、110 ページの「ロール内での規則の使用」および『Sun Java™ System Identity Manager 管理ガイド』を参照してください。
- 既存の規則ライブラリへの規則の追加については、114 ページの「デフォルト規則および規則ライブラリのカスタマイズ」を参照してください。
- XPRESS を使用した式の作成については、『Sun Java™ System Identity Manager ワークフロー、フォーム、およびビュー』の XPRESS 言語に関する章を参照してください。

ヒント

規則を設計する際には、経験の少ないユーザーが Identity Manager IDE を使用してさらにカスタマイズできるようにするために、規則ができるだけ簡単になるように心がけてください。

複雑な規則でも、適切な規則引数を使用すれば、XPRESS や JavaScript をユーザーに公開することなく、デフォルト値を変更することで大幅なカスタマイズが可能になります。

規則構文について

通常、規則は XML で作成され、<Rule> 要素でカプセル化されます。

ここでは、次のトピックを扱います。

- <Rule> 要素の使用
- 固定値を返す
- 変数の参照
- 引数を使用した規則の宣言
- 副作用を伴う規則

<Rule> 要素の使用

コード例 2-13 は、<Rule> 要素を使用して基本の規則式を定義する例を示したものです。name プロパティーは規則の名前を識別します。この規則は XPRESS で作成されています。

コード例 2-13 <Rule> 要素を使用した基本の規則式の定義

```
<Rule name='getApprover'>
   <cond><eq><ref>department></ref><s>sales</s></eq>
      <s>Sales Manager</s>
      <s>HR Manager</s>
   </cond>
</Rule>
```

固定値を返す

固定値を返す規則の場合は、XML オブジェクト構文を使用して作成できます。コード例 2-14 は文字列のリストを返します。

コード例 2-14 文字列のリストを返す

```
<Rule name='UnixHostList'>
   <List>
      <String>aas</String>
      <String>ablox</String>
      <String>aboupdt</String>
   </List>
</Rule>
```

注	XML オブジェクト構文の詳細については、『Sun Java™ System Identity Manager ワークフロー、フォーム、およびビュー』の XML オブジェクト言語に関する章を参照してください。

変数の参照

規則では <ref> 式を使用して外部変数の値を参照できます。使用可能な変数の名前は、規則が使用されるコンテキストによって決定されます。フォーム内で使用する場合は、任意のフォームフィールド、表示属性、または <defvar> で定義されている変数を参照できます。ワークフロー内で使用する場合は、ワークフロープロセス内で定義されている任意の変数を参照できます。

コード例 2-15 では、フォームは規則を使用して電子メールアドレスを計算します。フォームはフィールド global.firstname と global.lastname を定義し、規則がそれらのフィールドを参照します。電子メールアドレスは、global.firstname の最初の文字に global.lastname と文字列 @example.com を連結することによって計算されます。

コード例 2-15 電子メールアドレスの計算

```
<Rule name='Build Email'>
   <concat>
      <substr> <ref>global.firstname</ref> <i>0</i> <i>1</i> </substr>
      <ref>global.lastname</ref>
      <s>@example.com</s>
   </concat>
</Rule>
```

コード例 2-16 では、ワークフローは規則を使用して特定のアクティビティーに遷移すべきかどうかをテストします。ワークフローは、ユーザービューを含む user という名前の変数を定義します。この規則は、このユーザーに NT リソースが割り当てられている場合に true を返し、NT リソースが割り当てられていない場合に NULL を返します。ワークフローエンジンは NULL を false と解釈するため、この場合は遷移しません。

コード例 2-16 遷移のテスト

```
<Rule name='Has NT Resources'>
   <notnull>
      <ref>user.accountInfo.types[NT].accounts</ref>
   </notnull>
</Rule>
```

引数を使用した規則の宣言

規則に引数を宣言することは必須ではありませんが、宣言することが良い方法であると考えられています。引数を宣言することにより、規則の作成者にドキュメンテーションを提供し、Identity Manager IDE 内での参照妥当性検査が可能になり、同一の命名規則を使用していない可能性のあるフォームおよびワークフロー内でも、その規則を使用することが可能になります。

規則引数を宣言するには、<RuleArgument> 要素を使用します。次の location 引数に示されているように、引数名の後に値を指定することにより、引数はデフォルト値を取ることができます。

コード例 2-17 デフォルト値の設定

```
<Rule name='description'>
   <RuleArgument name='UserId'/>
   <RuleArgument name='location' value='Austin'/>
   <concat>
      <ref>UserId</ref>
      <s>@</s>
      <ref>location</ref>
   </concat>
</Rule>
```

注 規則を定義する場合は、<Rule name='rulename'> のように **R** が大文字の <Rule> 要素を使用します。規則を呼び出す場合は、XPRESS の <rule> 要素は、<rulename='rulename'> のように、小文字の **r** となります。

この規則をユーザーフォーム内で使用することはできますが、UserId と location はユーザービューの属性ではありません。予期されている引数を規則に渡すために、<argument> 要素を規則の呼び出しで使用します。location という名前の引数を渡すことにより、RuleArgument 要素で宣言されているデフォルト値が上書きされることに注意してください。

コード例 2-18 RuleArgument で宣言されているデフォルト値の上書き

```
<rule name='description'>
   <argument name='UserId' value='$(waveset.accountId)'/>
   <argument name='location' value='global.location'/>
</rule>
```

規則の呼び出しの詳細については、144 ページの「規則の参照」を参照してください。

引数の型を宣言する正式な方法はありませんが、コメントフィールドでは型を指定できます。<Comment> 要素を使用して、規則にコメントを組み込みます。

コード例 2-19 <Comment> を使用した、規則へのコメントの組み込み

```
<Comments>
記述規則では、2 つの引数が予期されている。従業員の
ID 番号である文字列値の UserId と、その従業員の
ビルの位置を記述する文字列値 location である
</Comments>
```

ヒント 規則の編集に Identity Manager IDE を使用している場合、規則引数のリストを形式的に定義すると便利です。このリストは、規則で使用可能になることが予期されている、変数の名前で構成されます。後で Identity Manager IDE で妥当性検査を実行するときに、これらを使用できます。

副作用を伴う規則

一般に、規則は単一の値を返しますが、規則から複数の値が返されるとよい場合もあります。規則式は単一の値しか返すことができませんが、規則は次の XPRESS 式を使用すると外部変数に複数の値を割り当てることができます。

- <setvar>
- <setlist>

- <putmap>

コード例 2-20 では、規則は department という名前の外部変数の値をテストして、値をほかの 2 つの変数に割り当てます。

コード例 2-20 department 変数のテストおよびほかの変数の割り当て

```
<Rule name='Check Department'>
    <switch>
      <ref>global.department</ref>
      <case>
        <s>Engineering</s>
        <block>
          <setvar name='global.location'>
            <s>Building 1</s>
          </setvar>

          <setvar name='global.mailServer'>
            <s>mailserver.somecompany.com</s>
          </setvar>
        </block>
      </case>
      <case>
        <s>Marketing</s>
        <block>
          <setvar name='global.location'>
            <s>Building 2</s>
          </setvar>          <setvar name='global.mailServer'>
            <s>mailserver2.somecompany.com</s>
          </setvar>
        </block>
      </case>
    </switch>
</Rule>
```

上記の例では、変数 global.location と global.mailServer は、ともに変数 department の値に従って設定されています。この場合、規則の戻り値は無視され、規則はその副作用のためにのみ呼び出されます。

JavaScript での規則の記述

規則が複雑になる場合は、規則を XPRESS ではなく JavaScript で作成するほうが都合がよい場合もあります。JavaScript は <script> 要素でラップすることができます。次に例を示します。

コード例 2-21 <script> 要素での JavaScript のラップ

```
<Rule name='Build Email'>
   <script>
      var firstname = env.get('firstname');
      var lastname = env.get('lastname');
      var email = firstname.substring(0, 1) + lastname + "@example.com";
      email;
   </script>
</Rule>
```

フォームおよびワークフロー変数の値を参照するには、env.get 関数を呼び出して変数名を渡します。env.put 関数を使用して、変数名を割り当てることができます。スクリプト内の最後の文の値が規則の値になります。前述の例では、規則は email 変数に値を戻します。

env.call 関数では、ほかの規則を呼び出すことができます。

規則の参照

ライブラリ内の規則は、XPRESS の <rule> 式を使用して参照します。name 属性の値は、ライブラリを含む設定オブジェクトの名前と、ライブラリ内部での規則の名前をコロンで連結した形式です。したがって、ライブラリ内のすべての規則名は必ず一意になります。

たとえば次の式は、Account ID Rules という名前のライブラリに含まれる、First Dot Last という名前の規則を呼び出します。

```
<rule name='Account ID Rules:First Dot Last'/>
```

ここでは、規則の参照について説明します。説明する内容は次のとおりです。

- 基本的な規則呼び出し構文
- 規則引数の解決

基本的な規則呼び出し構文

規則は XPRESS が許可されている場所であればどこからでも、フォーム、ワークフロー、または別の規則からでも呼び出せます。

規則を呼び出すには、次のような XPRESS の <rule> 式を使用します。たとえば、次のようにします。

```
<rule name='Build Email'/>
```

XPRESS インタプリタは、この式を評価すると、name 属性の値がリポジトリ内の規則オブジェクトの名前であるものと判断します。リポジトリから規則が自動的に読み込まれ、評価されます。規則によって返される値が <rule> 式の結果になります。

上記の例では、規則に明示的に渡される引数はありません。次の例は、argument 要素を使用して規則に渡される引数を示しています。

```
<rule name='getEmployeeId'>
   <argument name='accountId' value='jsmith'/>
</rule>
```

上記の例では、引数の値は静的文字列 jsmith として指定されています。式を使用して引数の値を計算することもできます。

```
<rule name='getEmployeeId'>
   <argument name='accountId'>
      <ref>user.waveset.accountId</ref>
   </argument>
</rule>
```

上記の例では、表示属性 user.waveset.accountId の値を返す、単純な ref 式を評価することによって、引数値が計算されます。

属性を参照することによって引数値を計算することが非常に一般的なため、代わりの構文も用意されています。

```
<rule name='getEmployeeId'>
   <argument name='accountId' value='$(user.waveset.accountId)'/>
</rule>
```

上記 2 つの例の動作は同じです。両方が、表示属性 user.waveset.account の値を引数の値として渡します。

規則引数の解決

ほとんどの規則には、変数の値を取得するための XPRESS <ref> 式または JavaScript env.get 呼び出しが含まれています。これらの変数の値を取得する方法を制御するために、いくつかのオプションが使用可能です。

最も単純なケースでは、規則を呼び出すアプリケーションがすべての参照を解決しようとします。ワークフローから呼び出される規則の場合、ワークフロープロセッサはすべての参照先がワークフロー変数であると想定します。フォームから呼び出される規則の場合、フォームプロセッサはすべての参照先がビュー内の属性であると想定します。呼び出し側の規則名を動的に解決することにより、規則を別の規則から呼び出すこともできます。

オプションの <RuleArgument> 要素も使用することができます。この要素については、141 ページの「引数を使用した規則の宣言」で説明されています。

ここでは、次の内容を説明します。

- 範囲または明示的な引数の呼び出し
- ローカル範囲オプション
- 規則引数宣言
- ロックされた引数

範囲または明示的な引数の呼び出し

ここでは、規則引数解決の例を示します。

コード例 2-22 は、ユーザービューで使用できるフォームに規則を追加する例を示しています (ユーザービューに属性名があるため)。

コード例 2-22 フォームに規則を追加

```
<Rule name='generateEmail'>
   <concat>
      <ref>global.firstname</ref>
      <s>.</s>
      <ref>global.lastname</ref>
      <s>@example.com</s>
   </concat>
</Rule>
```

この規則は 2 つの変数を参照します。

- global.firstname
- global.lastname.

コード例 2-23 に示されているように、Field 内でこの規則を呼び出すことができます。

コード例 2-23 フィールド内での規則の呼び出し

```
<Field name='global.email'>
   <Expansion>
      <rule name='generateEmail'/>
   </Expansion>
</Field>
```

これは、プログラミング言語のグローバル変数の概念に似ており、ユーザーフォーム内のみで使用される単純な規則を作成するには便利な方法です。しかし、このスタイルの規則設計には 2 つの問題があります。第一に、規則がどの変数を参照するかがフォーム設計者には不明です。第二に、規則はユーザービューの属性を参照するため、規則はユーザーフォームからしか呼び出せません。一般に、ワークフローでは global.firstname および global.lastname という名前の変数を定義しないため、ほとんどのワークフローからは規則を呼び出すことができません。

規則引数を明示的に渡したり、特定のビューに依存しない名前を使用する規則を作成したりすることにより、これらの問題に対処できます。

コード例 2-24 は、変数 firstname と lastname を参照する規則の修正版を示しています。

コード例 2-24 firstname および lastname 変数を参照する規則

```
<Rule name='generateEmail'>
   <concat> ¥
      <ref>firstname</ref>
      <s>.</s>
      <ref>lastname</ref>
      <s>@example.com</s>
   </concat>
</Rule>
```

コード例 2-25 に示されている規則は、ユーザーフォームから呼び出すことを前提にしていないため、より簡潔でより一般化されています。しかしその場合、次のように明示的な引数を指定して規則を呼び出す必要があります。

コード例 2-25 明示的な引数を指定した規則の呼び出し

```
<Field name='global.email'>
   <Expansion>
      <rule name='generateEmail'>
         <argument name='firstname' value='$(global.firstname)'/>
         <argument name='lastname' value='$(global.lastname)'/>
      </rule>
   </Expansion>
</Field>
```

argument 要素の name 属性は、規則内で参照される変数に対応します。これらの引数の値は、ユーザービューのグローバル属性の値に割り当てられます。こうすることで、規則は呼び出し側アプリケーションによって使用される命名規則から孤立した状態に保たれ、規則はほかのコンテキストで使用できるようになります。

ローカル範囲オプション

引数が明示的に規則に渡されていても、明示的な引数として渡されないその他の変数への参照が、システムのデフォルトでは可能になっています。コード例 2-26 は、規則を呼び出しても引数を 1 つしか渡さないワークフローアクションを示しています。

コード例 2-26 規則を呼び出し、単一の引数を渡すワークフローアクション

```
<Action>
   <expression>
      <setvar name='email'>
         <rule name='generateEmail'>
            <argument name='firstname' value='$(employeeFirstname)'/>
         </rule>
      </setvar>
   </expression>
</Action>
```

規則が評価されると、ワークフロープロセッサは変数 lastname の値を指定するよう求められます。この名前のワークフロー変数が存在しても、それがこの規則での使用を意図した変数ではない場合もあります。意図していない変数参照を防ぐために、規則を定義する際に localScope オプションを指定します。

このオプションは、Rule 要素の localScope 属性を **true** に設定することによって有効になります。

コード例 2-27 Rule 要素の localScope 属性を true に設定

```
<Rule name='generateEmail' localScope='true'>
   <concat>
      <ref>firstname</ref>
      <s>.</s>
      <ref>lastname</ref>
      <s>@example.com</s>
   </concat>
</Rule>
```

このオプションを設定することにより、規則は呼び出し内で引数として明示的に渡された値のみを参照することが許可されます。前述のワークフローアクションの例から呼び出した場合、lastname 変数の参照は NULL を返すことになります。

いろいろなコンテキストでの一般的な使用を意図した規則には、常にローカル範囲オプションを使用するようにします。

規則引数宣言

必須ではありませんが、規則によって参照される可能性のあるすべての引数の明示的な宣言を、規則定義内に含めるのが良い方法であると考えられています。引数宣言には数々の利点があり、次のようなことが可能です。

- 規則の呼び出し側にとってドキュメンテーションの役割を果たす
- デフォルト値を定義できる
- Identity Manager IDE で使用して、規則内にスペルミスの参照がないかどうかをチェックできる
- Identity Manager IDE で使用して、規則呼び出しの設定を単純化できる

コード例 2-28 に示されているような generateEmail 規則を再作成することも可能です。

コード例 2-28　　generateEmail 規則の再作成

```
<Rule name='generateEmail' localScope='true'>
   <RuleArgument name='firstname'>
      <Comments>The first name of a user</Comments>
   </RuleArgument>
   <RuleArgument name='lastname'>
      <Comments>The last name of a user</Comments>
   </RuleArgument>
   <RuleArgument name='domain' value='example.com'>
      <Comments>The corporate domain name</Comments>
   </RuleArgument>
   <concat>
      <ref>firstname</ref>
      <s>.</s>
      <ref>lastname</ref>
      <s>@</s>
      <ref>domain</ref>
   </concat>
</Rule>
```

Comments 要素には、規則を調べるユーザーに役立つと思われる、任意の量のテキストを含めることができます。

規則は修正され、domain という名前の別の引数が定義されています。この引数にはデフォルト値 example.com が指定されています。呼び出し側が domain という名前の明示的な引数を渡さない場合に、規則によってデフォルト値が使用されます。

コード例 2-29 の呼び出しは、文字列 john.smith@example.com を生成します。

コード例 2-29　　john.smith@example.com 文字列の生成

```
<rule name='generateEmail'>
   <argument name='firstname' value='john'/>
   <argument name='lastname' value='smith'/>
</rule>
```

コード例 2-30 の呼び出しは、文字列 `john.smith@yourcompany.com` を生成します。

コード例 2-30 john.smith@yourcompany.com 文字列の生成

```
<rule name='generateEmail'>
   <argument name='firstname' value='john'/>
   <argument name='lastname' value='smith'/>
   <argument name='domain' value='yourcompany.com'/>
</rule>
```

コード例 2-31 の呼び出しは、文字列 `john.smith@` を生成します。

コード例 2-31 john.smith@ 文字列の生成

```
<rule name='generateEmail'>
   <argument name='firstname' value='john'/>
   <argument name='lastname' value='smith'/>
   <argument name='domain'/>
</rule>
```

注 上記の例では、domain 引数に NULL 値が渡されますが、デフォルト値は使用されません。呼び出しに明示的な引数を指定すると、引数が NULL であってもデフォルト値が使用されます。

ロックされた引数

引数をデフォルト値で宣言する方法は、規則の開発とカスタマイズをより簡単にするには便利な方法です。規則内に場合によって変化する定数値がある場合は、その値を規則式の中深くに組み込むのではなく、引数で定義すると、値の特定と変更がより容易になります。

Identity Manager IDE には、引数のデフォルト値を変更することによって規則を設定するための、簡素化された GUI が備わっており、この方法は規則式全体を編集するよりもはるかに簡単です。

しかし、一度引数を宣言したなら、規則の呼び出し側が明示的な引数を渡してデフォルト値を上書きすることも可能です。引数値の制御を呼び出し側に渡したくない場合があるかもしれません。そうならないようにするために、引数をロックすることができます。`RuleArgument` 要素に `locked` 属性を組み込んで `true` の値を指定することにより、引数はロックされます (コード例 2-32 を参照)。

コード例 2-32　　引数のロック

```
<Rule name='generateEmail' localScope='true'>
   <RuleArgument name='firstname'>
      <Comments>The first name of a user</Comments>
   </RuleArgument>
   <RuleArgument name='lastname'>
      <Comments>The last name of a user</Comments>
   </RuleArgument>
   <RuleArgument name='domain' value='example.com' locked='true'>
      <Comments>The corporate domain name</Comments>
   </RuleArgument>
   <concat>
      <ref>firstname</ref>
      <s>.</s>
      <ref>lastname</ref>
      <s>@</s>
      <ref>domain</ref>
   </concat>
</Rule>
```

上記の例では、引数 domain がロックされています。これは、呼び出し側がこの引数に値を渡そうとしても、その値は常に example.com であるという意味です。ドメイン名が example.com ではないサイトでこの規則を使用する場合、管理者がしなければならないことはこの規則を編集して引数の値を変更することだけです。この規則式を理解するまたは変更する必要はありません。

規則のセキュリティー保護

規則に資格などの機密情報が含まれている場合や危険な副作用がありえる Java ユーティリティーに規則が呼び出される場合は、規則が意図しない方法で使用されないようにするために、規則をセキュリティー保護すべきです。

規則のセキュリティー保護がとりわけ重要になるのは、フォームから呼び出される場合です。フォームの規則はセッション上で実行されるので、セッションを作成できるユーザーに API または SOAP リクエストのいずれかを通じて規則が露呈することもありえます。

ここでは、次の内容を説明します。

規則のセキュリティー保護

規則をセキュリティー保護するには、次の手順に従います。

- **適切な組織に規則を配置する**。だれでも規則にアクセスできるように、`All` という名前の組織に規則が配置されることがよくあります。計算を実行し副作用のない単純な規則の場合には、この方法が便利です。
- **機密情報を含む規則は All に配置しない**。代わりに、Top または適正な上位レベルの組織に規則を配置して、上位レベルの管理者のみが規則を直接実行できるようにします。

よりセキュアな規則を参照する規則の作成

ある規則を呼び出すと、まずその規則に対して承認されます。承認されたなら、その規則はそれ以上承認をチェックすることなくほかの規則を呼び出すことができます。このようにして、ユーザーはセキュアな規則に間接的にアクセスできます。ユーザーはセキュアな規則の内容を表示したり変更したりすることはできません。アクセスしている規則を通してセキュアな規則を呼び出すことだけが可能です。

よりセキュアな規則を参照する規則を作成するには、規則を作成するユーザーがそれぞれの規則が含まれる両方の組織を制御する必要があります。一般に、セキュアな規則は `Top` などの上位レベルの組織にあります。一方、セキュリティーが保証されない規則は、より多くのユーザーがアクセスする下位レベルの組織に配置されます。

第 3 章

変数名前空間の操作

この章では、一般的な Identity Manager のタスクやプロセスの概要、その一般的な使用法、および実行環境の名前空間について説明します。

説明する内容は次のとおりです。

- Active Sync
- 対話式編集
- 読み込み操作
- 調整規則
- SPML
- X.509 統合
- その他の変数コンテキスト

Active Sync

次の表に、Active Sync カテゴリに関連する一般的な Identity Manager プロセスまたはタスクに関する情報を示します。

表 3-1　Active Sync のプロセス / タスク

実行するプロセスまたはタスク	用途	名前空間
ActiveSync `IAPIUser`	• 特定のリソース上でユーザーに関係する変更を処理します。 • 指定されたワークフロープロセスを起動する前に、フルユーザービュー上で直接アクションを実行します。	ActiveSync イベントの属性をユーザービューにマージします。 「入力フォーム」の典型的な属性には、次のものがあります。 • `accounts[*].*` • `waveset.*` • `accountInfo.*` • `activeSync.`*<LHS Attr Name>* • `activeSync.`*resourceName* • `activeSync.resourceId` • `activeSync.resource` • `display.session` (Proxy Admin のセッション) • `global.`*<LHS Attr Name>* (`set globals` フラグがリソース上に設定されている場合)
ActiveSync `IAPIProcess`	• プロセスビューを作成することにより、リソース上で汎用イベントを処理します。 • プロセスビューの最上位フィールドは、タスクへの任意入力です。 • タスクの起動に関係する属性を `global` 属性に収集します。 • 最上位属性としてではなく、`global` の下から入力を取得するワークフローを作成します。	最上位レベルのワークフロー `global` 属性にダンプされている ActiveSync ポーリング属性で指定したタスクを起動します。 ワークフロー属性としては、次の形式が想定されています。 `global.`*<LHS Attr Name>*

対話式編集

次の表では、対話式編集カテゴリに関連する一般的な Identity Manager プロセスまたはタスクに関する情報を示しています。

表 3-2 対話式編集のプロセス / タスク

実行するプロセスまたはタスク	用途	名前空間
管理者インタフェースフォーム	要求を開始するための管理者インタフェース JSP を通してのビュー / フォーム対話 (ワークフローはまだ起動していない) 承認ページには適用されません	ビューが直接編集されるため、フォームの典型的な属性名には次のものがあります。 • `accounts[*].*` • `waveset.*` • `accountInfo.*` • `:display.session` (*admin* 用セッション)
WorkItems	`<ManualAction>` 指示を使用して起動します。カスタムタスクと管理者承認の両方に適用されます。 指定されたワークフローに関連付けられているフォームは、ベースコンテキストを `variables.user` に設定できます。これにより、変数名に `user.variables` を付ける必要がなくなります。	WorkItem は名前空間のため、フォームの典型的な属性名には次のものがあります。 • `complete` (WorkItem 属性) • `variables.*` (タスク変数) • `variables.<view>.accounts[*].*` • `variables.<view>.waveset.*` • `variables.<view>.accountInfo.*` • `:display.session` (所有者用セッション)
ロール定義によって割り当てられたリソース属性値規則	ビューが更新されてリソースアカウント属性に値が割り当てられると、規則はロール定義に付加されて評価されます。	呼び出し側のコンテキストに関係なく、規則はビューに直接適用されます。したがって、フォームの典型的なビュー属性名は次のようになります。 • `accounts[*].*` • `waveset.*` • `accountInfo.*`

読み込み操作

次の表に、読み込み操作カテゴリに関連する一般的な Identity Manager プロセスまたはタスクに関する情報を示します。

表 3-3 読み込み操作のプロセス / タスク

プロセス / タスク	用途	名前空間
ファイルから読み込み	管理者インタフェースを通して呼び出した CVS または XML ファイルからアカウント情報を取得します。 Identity Manager はファイルから `WSUser` オブジェクトを読み取ってユーザービューに変換し、フォームを適用します。属性は、Identity Manager ユーザーの拡張属性であるかのように処理されます。属性は `accounts[Lighthouse]` に配置され、フォームが各属性にグローバルフィールドを定義している場合にのみ `global` 属性の下に配置されます。	ファイル内の各行のすべての属性値は、グローバル名前空間に読み込まれます。 `global.`<*attr name*> **注** : `create` 操作のみに適用されます。
リソースから読み込み	特定のリソースからアカウント情報を取得します (管理者インタフェースを通して呼び出され、アダプタを使用してアカウントをリストおよび取得する)。	リソース上の各アカウントのすべての属性値が、グローバル名前空間に読み込まれます。 `global.`<*LHS Attr Name*> **注** : `create` 操作のみに適用されます。
一括操作	(管理者インタフェースを通して呼び出された) CVS ファイルからコマンドおよびユーザービューを取得します。 ユーザービュー名前空間には、任意の属性を指定できます。属性名はビューパス構文を使用して指定されます。ユーザービュー名前空間とビューパス構文の詳細については、『Identity Manager の配備に関する技術概要』の「Understanding the User View」を参照してください。	ファイルから取得した属性値は、グローバル名前空間に読み込まれます。 • `accounts[*].*` • `waveset.*` • `accountInfo.*` • `global.*` **注** : 利用可能な承認済みセッションはありません。

注

「ファイルから読み込み」および「リソースから読み込み」をアイデンティティー属性に対して使用可能なアプリケーションのリストに追加した場合に、「ファイルから読み込み」および「リソースから読み込み」操作中に「アイデンティティー属性」を適用できるようになります。

「ファイルから読み込み」および「リソースから読み込み」で使用可能にした場合、これらのページに「ユーザーフォーム」、「属性の更新」、または「属性値のマージ」を選択するオプションは表示されません。「アカウントの更新」を選択すると、Identity Manager はすべてのアイデンティティー属性を処理し、アカウントを再プロビジョンします。それ以外の場合は、読み込み中のリソースから取得され (そしてアイデンティティーユーザーに伝達される) 属性のみが処理されます。

調整時に、Identity Manager はアイデンティティー属性を次の調整応答にのみ適用します。

- リソースアカウントに基づいたユーザーの作成
- ユーザーのリソースアカウントの作成

調整規則

次の表に、調整規則カテゴリに関連する一般的な Identity Manager プロセスまたはタスクに関する情報を示します。

表 3-4 調整規則のプロセス / タスク

実行するプロセスまたはタスク	用途	名前空間
相関規則	調整中に呼び出されて、リソースアカウントを Identity Manager ユーザー (複数可) に関連付けます	スキーマに定義されているリソースアカウントの、すべての属性値がフォームに提供されます。 `account.<LHS Attr Name>` 返される値 : • 一致する Identity Manager ユーザー名 • 一致する Identity Manager ユーザーの検索に使用される AttributeConditions または WSAttributes のリスト
確認規則	調整中に、相関規則の結果が複数の一致になった場合に呼び出されます。リソースアカウントは、相関のある各 Identity Manager ユーザーと比較されます。	リソースアカウントのすべての属性値およびユーザービューのすべての属性がフォームに提供されます。 • `account.<LHS Attr Name>` • `user.accounts[*].*` • `user.waveset.*` • `user.accountInfo.*` **返される値** : 一致があるかどうかに応じた論理値 true または false (`1` または `0`)

注

アイデンティティー属性に使用可能なアプリケーションのリストに「調整」を追加した場合、調整操作中に Identity Attributes が適用できるようになります。

「調整」で使用可能にした場合、これらのページに「ユーザーフォーム」、「属性の更新」、または「属性値のマージ」を選択するオプションは表示されません。「アカウントの更新」を選択すると、Identity Manager はすべてのアイデンティティー属性を処理し、アカウントを再プロビジョンします。それ以外の場合は、読み込み中のリソースから取得され (そしてアイデンティティーユーザーに伝達される) 属性のみが処理されます。

調整時に、Identity Manager はアイデンティティー属性を次の調整応答にのみ適用します。

- リソースアカウントに基づいたユーザーの作成
- ユーザーのリソースアカウントの作成

SPML

次の表は、SPML カテゴリに関連する一般的な Identity Manager プロセスまたはタスクに関する情報を示しています。

表 3-5 SPML のプロセス / タスク

実行するプロセスまたはタスク	用途	名前空間
Person オブジェクトクラス	SPML インタフェースの汎用実装。SPML 設定オブジェクト内にある SPMLPerson Form は、SPML スキーマのフラットな名前空間からビュー属性へのマッピングを指定します。	フォームで提供されるマッピングフィールドのペア。次の式を含むフィールド • `<Derivation>` 式は、応答スキーマ属性をビュー属性に設定します。Derivation を含むフィールドはフラットな名前ですが、Derivation 式のビューパスを参照します。 • `<Expansion>` 式は、リクエストスキーマ属性をビュー属性に転送します。Expansion を含むフィールドはパス名ですが、Expansion 式のフラットな名前を参照します。 ビュー属性の名前空間は、`accounts`、`waveset`、`accountInfo` 名前空間属性で構成されます。SPML スキーマ属性の名前空間は、フラットな名前空間で構成されます。
フォームパラメータが **view** に設定されている、すべてのリクエスト	フォーム処理なし	ビュー属性は直接設定されます。 • `accounts[*].*` • `waveset.*` • `accountInfo.*`

X.509 統合

次の表では、X.509 統合カテゴリに関連する一般的な Identity Manager プロセスまたはタスクに関する情報を示しています。

表 3-6 X.509 統合のプロセス / タスク

実行するプロセスまたはタスク	用途	名前空間
ログイン相関規則	Identity Manager ユーザーエントリの衝突を解決するためのメカニズムを提供します (規則は標準的な X.509 証明書を組み込む)。	標準的な認証フィールド、および重要な拡張プロパティーと重要ではない拡張プロパティーを提供します。認証プロパティーはフォーム `cert.<field name>.<subfield name>` を前提にしています。 • `cert.subjectDN` • `cert.issuerDN` **注** : 利用可能な承認済みセッションはありません。 **返される値** : • AttributeCondition • AttributeCondition のリスト
新規ユーザー命名規則	ログイン相関規則を使用した相関関係のあるユーザーがない場合、新規 Identity Manager ユーザーの名前を認証情報から設定するメカニズムを提供します。	ログイン相関規則を参照 **返される値** : Identity Manager ユーザーに使用する `name` (または `accountId`)

その他の変数コンテキスト

次の表では、その他の変数コンテキストカテゴリに関連する一般的な Identity Manager タスクまたはプロセスに関する情報を示しています。

表 3-7 その他の変数コンテキストのプロセス / タスク

実行するプロセスまたはタスク	用途	名前空間
フォームの起動	Executor を初期化するために TaskDefinition に組み込まれています	指定されたすべてのフィールド要素はタスクコンテキストに直接同化され、ワークフロータスクを起動する場合は最上位の変数として使用可能です。
ユーザーメンバー規則	特に、メンバーユーザーのリストを動的に返さなければならない組織向けに定義されています。 この規則は、リポジトリに対して取得を実行できません。その代わりに、この規則は指定したディレクトリ OU 内のすべてのエントリを検索するなどの `FormUtil.getResourceObjects` 呼び出しに制限されます。	承認された管理者のユーザービュー (リソースアカウント属性は取得されない) および管理者のセッション : • `accounts[Lighthouse].*` • `waveset.*` • `accountInfo.*` • `context` (承認された管理者のセッション)

第 4 章

アダプタの開発

この章では、企業または顧客用に調整された、独自のカスタム Sun Java™ System Identity Manager リソースアダプタを作成する方法について説明します。具体的には、この章では次の内容について説明します。

- Identity Manager と外部リソースの間の接続を、確立および維持するために使用できるメカニズム
- リソースアダプタ Java ファイルの構造の概要
- リソースアダプタの作成、テスト、および読み込みを行うための方法

この情報は、次のように構成されています。

独自のカスタムアダプタを作成するための出発点として、製品に付属のサンプルアダプタ (スケルトンアダプタ、と呼ばれる) を使用することをお勧めします。これらの重要なスターターファイルをより深く理解できるよう、この章では、これらのファイルを頻繁に使用してリソースアダプタファイルの特定の特性を例で示します。

概要

ここでは、Identity Manager でのアダプタの目的と機能に関連した、基本的な問題の概要について説明します。この節は次のトピックで構成されています。

この章の対象読者

この章では、リソースアダプタの全体的な設計および操作の背景について説明します。

- これらの情報は、次のユーザーに役立つ可能性があります。
- 独自のカスタムリソースアダプタを作成するために作業している開発者 Identity Manager システムの動作の仕組みや、リソースアダプタでの問題のトラブルシューティング方法を学習している Identity Manager 管理者

この章では、読者が組み込み型の Identity Manager リソースの作成と使用に精通していること、および『Sun Java™ System Identity Manager 管理ガイド』の「Resources」の章で説明されているトピックに精通していることを前提にしています。

リソースアダプタについて

Identity Manager の通信または管理の対象となる各リソースは、Identity Manager でリソースオブジェクトを使用して定義する必要があります。Identity Manager には、管理が必要なリソースごとに、リソースオブジェクトが必要です。リソースアダプタクラスには、そのリソースオブジェクトを Identity Manager リポジトリに登録するため、および外部リソースを管理するために必要なメソッドが含まれています。

アダプタの主な目的は、このリソースタイプの基本的な特性を定義することです。この情報は、リソースオブジェクトとして Identity Manager リポジトリに保存されます。

リソースオブジェクトによって、Identity Manager で管理しているリソースの機能と設定が定義されます。これには、表 4-1 に示す情報が含まれます。

表 4-1 リソースオブジェクトで定義される情報

情報の種類	属性の例
接続情報	• ホスト名 • 管理アカウント名 • 管理アカウントパスワード
ユーザー属性	• 名 • 姓 • 電話番号
Identity Manager 属性	• 承認者のリスト • リソースのパスワードポリシー • リソースに接続するときの繰り返し試行回数

カスタマイズされたアダプタによって、これらの基本的な特性を定義するための手段だけでなく、Identity Manager からの要求を、リソースに対して実行する操作に変換するメソッドも提供されます。

注 リソースオブジェクトは、Identity Manager 管理者インタフェースの「デバッグ」ページから表示できます。

リソースアダプタは、Identity Manager と、アプリケーションやデータベースなどの外部リソースの間のプロキシとして機能します。リソースアダプタクラスは、Identity Manager からの情報をリソースにプッシュしたり、オプションでリソースからの情報を Identity Manager にプルしたりするために必要なメソッドを実装します。このオプションのプル機能を Active Sync と呼びます。Active Sync 機能を持つリソースアダプタは、Active Sync 対応と呼ばれます。

Identity Manager は、Active Directory などのいくつかの一般的なリソースには Active Sync 対応アダプタを提供し、Web サーバー、Web アプリケーション、データベース、さらには旧バージョンのアプリケーションやオペレーティングシステムを含む、その他の多くのリソースタイプには、標準リソースアダプタを提供します。Identity Manager で提供されるリソースアダプタは、Identity Manager によってサポートされている、各種のリソースタイプに対して汎用的なインタフェースを提供します。サポートされているこれらのいずれかのリソースタイプのインスタンスが指定されると、そのリソースアダプタは実世界のリソースと通信し、そのリソースを管理することができます。

Identity Manager のオープンアーキテクチャーによって、提供されたリソースアダプタではまだサポートされていない外部リソースを管理できるようにする、カスタムリソースアダプタが可能になります。新しいカスタムリソースアダプタタイプを作成するためのメカニズムについては、この章のあとの方で説明します。

Active Sync 対応アダプタと標準アダプタの相違点

リソースアダプタは、標準アダプタまたは Active Sync 対応アダプタ、あるいはその両方です。Java の用語では、リソースアダプタは `ResourceAdapterBase` クラスを拡張することによって標準リソースアダプタになり、Active Sync インタフェースを実装することによって Active Sync 対応になります。

Active Sync 対応アダプタは、ソースから情報を取得できます。一般には、イベントによって駆動されるか、または情報を取得するためにリソースにポーリングするかのいずれかです。標準リソースアダプタは、アカウント情報への変更を、Identity Manager によって管理されている外部リソースにプッシュします。

標準リソースアダプタは、Identity Manager からの変更を、自身が管理しているリソースに伝播します。リソースアダプタが実行する、代表的な管理アクティビティーには次のものがあります。

- ユーザーの作成、削除、または変更
- ユーザーの有効化、無効化、および取得
- グループメンバーシップやディレクトリ組織構造などのオブジェクトの管理
- ユーザーの認証
- リソースへの接続、およびリソースからの切断

Active Sync 対応アダプタは、データ変更をリソースから直接収集して Identity Manager でのアクティビティーを開始できます。これらのアクティビティーには次のものがあります。

- 変更のイベント通知のポーリングまたは受信
- リソースアカウントを作成、更新、または削除する操作の発行
- カスタムフォームを持つユーザーの編集または作成
- リソースへの変更の保存
- 進捗に関する情報や、発生するすべてのエラーのログ記録

Active Sync 対応アダプタによって監視されるリソースのタイプ

Active Sync 対応アダプタは、次のリソースタイプのサポートに特に適しています。

- **監査または通知インタフェースを備えたアプリケーション**。たとえば、Microsoft Active Directory や PeopleSoft には、指定された変更が発生した場合に、別のアプリケーションに通知するか、または監査ログにイベントを追加するように設定できる、外部インタフェースが用意されています。

  たとえば、Active Directory サーバー上でユーザーアカウントがネイティブに変更されると、そのトランザクションが監査ログに記録されます。このログを 30 分ごとに確認するように Identity Manager の Active Directory リソースを設定でき、それにより、任意の変更が検出されると Identity Manager でイベントがトリガーされます。

  このリソースに、API を通してほかの Active Sync 対応アダプタを登録できます。それにより、項目への変更が発生すると、このアダプタにイベントメッセージが通知されます。これらのメッセージには、変更された項目、更新された情報、および一般にはその変更を行なったユーザーへの参照が含まれます。たとえば、LDAP リソースは、Active Sync 対応アダプタの通知実装を使用しています。

- **更新情報が入力されたデータベース**。データベースリソースは、デルタのテーブルを使用して管理することもできます。このテーブルは、いくつかの異なる方法を使用して生成できます。たとえば、データベースのスナップショットを現在の値と比較し、その差分を含む新しいテーブルを作成できます。アダプタは次に、デルタのテーブルの行をプルして処理し、完了したらそれらの行にマークを付けることができます。

- **変更タイムスタンプを含むデータベース**。3 番目のタイプの Active Sync 対応アダプタは、特定の時刻のあとに変更されたデータベースエントリがないかどうかを問い合わせ、更新を実行したあと、新しいクエリーをポーリングできます。最後に正常に処理された行を格納することによって、Identity Manager は「"starts with" (で始まる)」クエリーを実行して、ポーリングの影響を最小限に抑えることができます。最後のものは処理のために返されているため、リソースに対して行われた変更のみを対象とします。

標準リソースアダプタが機能する仕組み

ここでは、標準アダプタの処理の基本的な手順の概要について説明します。

Identity Manager から、Identity Manager によって管理されているリソースに情報をプッシュする場合、すべての標準リソースアダプタが次の一般的な手順に従います。

1. Identity Manager サーバーがリソースマネージャを初期化します。

   すべての使用可能なリソースタイプが、リソースアダプタインタフェースを通して登録されます。登録プロセスの一部として、リソースアダプタは XML 定義のプロトタイプを提供します。

2. ユーザーが新しいリソースを作成するためのプロセスを開始します。

   Identity Manager 管理者が新しいリソースを作成する場合は、リソースタイプのプロトタイプ定義を表示するフォームを作成しているタスクに、リソース属性フィールドのクエリーが行われます。Identity Manager は、これらの属性を使用してブラウザ上にフォームを表示します。新しいリソースを作成しているユーザーは、この情報を入力して「保存」をクリックします。

3. Identity Manager は、入力された情報をほかのリソースフィールドとともに、新しいリソースオブジェクトの名前でリソースオブジェクトリポジトリに保存します。

   リソース作成中にユーザーが「保存」をクリックすると、作成タスクは入力されたデータを収集して必要な検証をすべて実行し、XML を通してデータを直列化したあと、その直列化されたオブジェクトをオブジェクトリポジトリに書き込みます。

4. Identity Manager は、Identity Manager ユーザーが作成されるか、または変更されると、使用可能なリソースのリストを複数選択ボックスに表示します。

   リソースを選択すると、Identity Manager はそのリソースオブジェクトに、使用可能なアカウント属性フィールドを問い合わせます。Identity Manager は、これらのフィールドの説明を使用して、ユーザーが適切なデータを入力できる属性フィールドが含まれたフォームを表示します。

5. このフォームが保存されると、リソースオブジェクトに接続情報の問い合わせが行われ、そのリソースを使用した接続が確立されます。

6. アダプタは、この接続を通して、そのリソース上のアカウントに対して目的の操作を実行するためのコマンドを送信します。

7. これが作成要求である場合は、Identity Manager ユーザーオブジェクトが、そのリソースアカウント情報を使用して更新されます。

   ユーザーアカウント情報が表示されると、Identity Manager はそのユーザーがアカウントを保持しているリソースのリストを、保存されたアカウントオブジェクトに要求します。リソースごとに、Identity Manager はリソースオブジェクトに問い合わせを行い、その接続情報を使用してリソースへの接続を確立します。

アダプタは、この接続を通してユーザーのアカウント情報を取得するためのコマンドを送信し、取得された情報を使用して、そのリソースオブジェクト内で定義されている属性フィールドに入力します。システムによって、これらの値を表示するためのフォームが作成されます。

Active Sync 対応アダプタが機能する仕組み

この項では、次の内容を説明します。

- 基本的な Active Sync 対応アダプタの処理
- Identity Manager ユーザーの特定

基本的な Active Sync 対応アダプタの処理

すべての Active Sync 対応アダプタは、Identity Manager で管理されているリソースの変更を知るために、次の基本的な手順でリスニングまたはポーリングを実行します。変更されたリソースを検出すると、Active Sync 対応アダプタは次の手順を実行します。

1. リソースから変更された情報を抽出します。
2. どの Identity Manager オブジェクトに関係があるか判断します。
3. `IAPIFactory.getIAPI` メソッドに渡すユーザー属性のマップを、アダプタの参照および任意の追加オプションのマップとともに生成します。これにより、Identity Application Programming Interface (`IAPI`) オブジェクトが作成されます。
4. IAPI イベントに関するロガーをアダプタの Active Sync ロガーに設定します。
5. `IAPI` オブジェクトを Active Sync マネージャーに送信します。
6. Active Sync マネージャーは、このオブジェクトを処理し、処理が成功したかどうかを Active Sync 対応アダプタに通知する `WavesetResult` オブジェクトをアダプタに返します。このオブジェクトには、ID の更新のために Identity Manager システムが使用するさまざまな手順の結果を多く含めることができます。一般に、ワークフローは Identity Manager 内のエラーにも対応し、多くの場合、担当管理者の承認にあとを任せます。

例外は、`ActiveSyncUtil.logResourceException` メソッドを使用して、Active Sync および Identity Manager トレースログに記録されます。

Identity Manager ユーザーの特定

Active Sync 対応アダプタは、リソース上でのアカウントの変更を検出すると、受け取った属性を Identity Manager ユーザーにマップします。または、一致するアカウントがない場合は Identity Manager ユーザーアカウントを作成します。

変更が検出されたときの動作は、次のパラメータによって決定されます。

表 4-2 Active Sync 対応アダプタのパラメータ

パラメータ	説明
処理規則	TaskDefinition の名前、またはフィード内のすべてのレコードに対して実行される TaskDefinition の名前を返す規則のいずれかです。この処理規則は、Active Sync 名前空間内のリソースアカウント属性を、リソース ID およびリソース名とともに取得します。 処理規則は、システムがリソース上の変更を検出したときに実行されるすべての機能を制御します。アカウント処理を完全に制御する必要がある場合に使用します。この結果、処理規則はほかのすべての規則より優先されます。 処理規則が指定されると、このアダプタ上にほかのどんな設定があっても、すべての行に対してその処理が実行されます。 処理規則は、少なくとも次の機能を実行する必要があります。 • 一致するユーザービューに対するクエリー。 • ユーザーが存在する場合は、ビューのチェックアウト。ユーザーが存在しない場合は、ユーザーの作成。 • ビューの更新またはビューへの設定。 • ユーザービューのチェックイン。 ユーザー以外のオブジェクト (LDAP ロールなど) を同期することもできます。
確認規則	相関規則によって返されるすべてのユーザーを対象にして評価される規則です。ユーザーごとに、Identity Manager の ID と (「account.」名前空間にある) リソースアカウント情報の相関を示す完全なユーザービューが確認規則に渡されます。確認規則は、ブール値のように表すことができる値を返すことが期待されます。たとえば、「true」または「1」または「yes」と、「false」または「0」または NULL です。 データベーステーブル、フラットファイル、および PeopleSoft コンポーネントの Active Sync アダプタの場合は、デフォルトの確認規則はリソース上の調整ポリシーから継承されます。 調整と Active Sync で同じ確認規則を使用できます。

表 4-2　Active Sync 対応アダプタのパラメータ (続き)

パラメータ	説明
相関規則	リソースアカウントを所有する Identity Manager ユーザーのリソース情報が特定されない場合は、相関規則が呼び出され、(アカウントの名前空間内の) リソースアカウント属性に基づいて、ユーザーの照合に使用する、一致する可能性のあるユーザーまたはアカウント ID の候補のリスト、あるいは属性条件が特定されます。 エントリを既存の Identity Manager アカウントに関連付けるために使用できる次のいずれかの種類の情報を返します。 • Identity Manager ユーザー名 • `WSAttribute` オブジェクト (属性ベースの検索に使用) • `AttributeCondition` 型または `WSAttribute` 型の項目のリスト (`AND` 結合による属性ベースの検索) • `String` 型の項目のリスト (各項目は Identity Manager アカウントの Identity Manager ID またはユーザー名) 相関規則によって複数の Identity Manager アカウントが識別された場合は、複数の一致を処理するために確認規則またはプロセス解決規則が必要です。 データベーステーブル、フラットファイル、および PeopleSoft コンポーネントの Active Sync アダプタの場合は、デフォルトの相関規則はリソース上の調整ポリシーから継承されます。 調整と Active Sync で同じ相関規則を使用できます。
削除規則	フラットファイル内のエントリまたは行からプルされた `activeSync.` または `account.` という形式のキーを持つ、すべての値のマップを期待できる規則です。プロキシ管理者のセッションに基づく LighthouseContext オブジェクト (`display.session`) は、この規則のコンテキストで利用できます。この規則は、ブール値のように表すことができる値を返すことが期待されます。たとえば、「true」または「1」または「yes」と、「false」または「0」または NULL です。 あるエントリに関してこの規則によって true が返された場合、アダプタの設定方法に応じて、フォームとワークフローを介してアカウント削除要求が処理されます。
プロセス解決規則	TaskDefinition の名前、またはフィード内のあるレコードに対して複数の一致がある場合に実行される TaskDefinition の名前を返す規則のいずれかです。プロセス解決規則は、リソースアカウント属性をリソース ID およびリソース名とともに取得します。 この規則は、一致がなく、「一致しないアカウントの作成」が選択されていない場合にも必要です。 このワークフローは、管理者による手動操作を求める処理にすることもできます。
一致しないアカウントの作成	`true` に設定すると、一致する Identity Manager ユーザーが見つからない場合に、リソース上にアカウントが作成されます。`false` に設定すると、処理規則が設定され、その規則が識別するワークフローによって新しいアカウントが保証されていることが確認されないかぎり、アカウントは作成されません。デフォルトは `true` です。

表 4-2　Active Sync 対応アダプタのパラメータ (続き)

パラメータ	説明
グローバルで利用	true に設定すると、ActiveSync 名前空間に加えてグローバル名前空間にも値が入力されます。デフォルト値は、false です。

処理規則の存在によって、IAPIProcess を使用するか、または IAPIUser を試みるかどうかが決定されます。ほかのパラメータ設定が指定されたイベントに対する Identity Manager ユーザーが相関規則または確認規則で一意に特定されなかったために Identity Manager が IAPIUser を使用できない場合、プロセス解決規則が設定されていれば、その規則を使用して IAPIProcess イベントが作成されます。そうでない場合は、エラー条件が報告されます。

IAPIUser はビューをチェックアウトし、このビューをユーザーフォームに対して使用可能にします。作成と更新の場合は、ユーザービューがチェックアウトされます。削除の場合は、プロビジョン解除ビューがチェックアウトされます。ただし、IAPIProcess ではビューは使用できません。処理規則が設定されているか、またはプロセス解決規則が呼び出されるかのどちらかです。

カスタムアダプタを作成するための準備

次の節では、カスタムアダプタの開発を開始するためのアプローチについて説明します。

リソースの把握

この節では、次の項目を含め、アダプタファイルを編集する準備ができるようにします。

リソースを把握する作業の開始

次の質問を使用して、情報の収集や前提条件の定義に役立てください。

デフォルトの Identity Manager リソースアダプタのどのタイプが、計画しているアダプタにもっともよく似ていますか。

Identity Manager の標準構成に付属のアダプタファイルの簡単な説明については、179 ページの表 4-4 を参照してください。接続先のリソースタイプにもっともよく一致する Identity Manager アダプタファイルを選択してください。

そのリソース上のユーザーアカウントはどのような特性を持っていますか。そのリソースに関する基本的なタスクをどのように実行しますか。

リモートリソースに関する次のタスクの実行方法を見つけてください。

- リモートリソースへのアクセスを認証する
- ユーザーを更新する
- ユーザーへの変更を検索する (Active Sync 対応のみ)
- 変更されたユーザーに関する詳細を取得する
- システム上のすべてのユーザーをリストする
- 変更されたユーザーのみを検索する方法を特定する (Active Sync 対応のみ)
- `listAllObjects` メソッドで使用されている、グループなどのほかのシステムオブジェクトをリストする

また、サポートされているすべての属性を把握するとともに、操作を実行するために必要な最小限の属性を特定することも必要です。

そのリソースへの接続をサポートしている適切なツールがありますか。

多くのリソースには、外部のアプリケーションをリソースに統合する場合に使用できる API のセットまたは完全なツールキットが付属しています。公開された API のセットがリソースに付属しているかどうか、または Identity Manager との統合を高速化するためのマニュアルやツールがツールキットに用意されているかどうかを調べてください。たとえば、データベースの場合は、JDBC を介してそのデータベースに接続する必要があります。

ログインして、そのリソース上のユーザーを検索できるのはだれですか。それはどのように行われますか。

ほとんどのリソースアダプタには、ユーザーの検索や各ユーザーの属性の取得などのタスクを実行するのに使用できる管理アカウントが必要です。これは一般には、特権レベルの高い (またはスーパーユーザー) アカウントですが、読み取り専用アクセスを許可され、委任された管理アカウントである場合もあります。

そのリソースは属性を追加するためのカスタマイズが可能ですか。

ほとんどのリソースは、組み込み型の属性を拡張できます。たとえば、Active Directory と LDAP では、いずれもカスタム属性を作成できます。

Identity Manager でこのリソースが維持する属性の種類、それらの属性のこのリソース上での名前、および Identity Manager で付ける名前を検討してください。これらの属性名はスキーママップに追加され、そのタイプのリソースを作成するときに使用されるフォームへの入力になります。

ソース上のどの属性 / 操作でイベントが作成されますか (Active Sync 対応)。

LDAP などのように、そのリソースで変更が発生した場合の通知メッセージへの登録がサポートされている場合は、どの属性変更で通知をトリガーするか、およびメッセージ内にどの属性を含めるかを特定してください。

ソース上のイベントが検出された場合、Identity Manager はどの操作を実行しますか (Active Sync 対応)。

これらの操作には次のものがあります。

- ユーザーの作成、更新、または削除
- アカウントの無効化または有効化
- ユーザーの認証に使用される答えの更新
- 電話番号の更新

Active Sync 対応アダプタを外部リソース内のイベントによって駆動されるようにしますか。あるいは、ポーリング間隔で設定されるようにしますか。

決定を行う前に、通常の Identity Manager インストール内でポーリングが機能する仕組みを考慮してください。外部イベントを実装しているか、または外部イベントによって駆動されるインストールもありますが、Identity Manager が配備されているほとんどの環境は、ハイブリッドな方法を使用しています。

次のいずれかのアプローチを選択できます。

- **ポーリング間隔の設定**。ポーリングインタフェースは、設定可能な間隔または指定したスケジュールで `ActiveSyncManager` スレッドから呼び出すことができます。作業が受信された場合のポーリングの高速化、アダプタごとのスレッドまたは共通のスレッド、並行操作の量に対する制限などの設定を含む、ポーリングパラメータを設定できます。

- **イベント駆動の環境の設定**。アダプタを純粋にイベント駆動にすることもできます。このモデルの場合、アダプタはリスニング接続 (LDAP リスナーなど) を設定し、遠隔システムからのメッセージを待機します。この場合は、`poll` メソッドを何もしないように実装し、ポーリング間隔を任意の値 (たとえば、週に 1 回) に設定することができます。更新がイベント駆動である場合は、保証された配信メカニズム (MQ Series など) を用意する必要があります。そうしないと、同期が失われます。
- **ハイブリッドソリューションの実装**。このシナリオの場合、外部イベントはスマートポーリングをトリガーすることができ、通常のポーリングルーチンを消失したメッセージから回復できます (スマートポーリングは、変更が頻繁に発生しないかぎり、ポーリングレートを変更レートに適応させてポーリングの頻度を下げる。それにより、頻繁なポーリングによるパフォーマンスへの影響と、頻度の低いポーリングによる更新の遅延のバランスを取る)。

  このモデルでは、受信メッセージをキューに入れ、単一のオブジェクトに対する複数の更新を 1 つの更新にまとめることによって、効率を向上させることができます。たとえば、1 つのディレクトリで複数の属性を更新することができ、各属性によりメッセージがトリガーされます。ポーリングルーチンはメッセージキューを調べ、すべての重複を削除してから、完全なオブジェクトを取得できます。これにより、最新のデータが同期され、更新の効率的な処理が保証されます。

追加の準備

前の質問を使用してリソースのプロファイルを確認したら、次の手順を実行することによって準備作業を続けます。

1. 記述する必要のあるメソッドのリストを特定します。

   アダプタファイルの作成でもっとも時間がかかる部分は、Identity Manager からリソースに情報をプッシュしたり、リソースから Identity Manager へのフィードを作成したりするための独自のメソッドの記述です。

   Java ファイルに含めるメソッドは、ある程度、どのようなタイプのリソースへの接続を作成しているかに依存します。

2. Identity Manager の JavaDoc を確認します。

   関連する JavaDoc は製品 CD に収録されており、そのまま使用できるか、または拡張が必要な基底クラスとメソッドを特定しています。

Resource Extension Facility (REF) キットの内容

Sun Resource Extension Facility キット (製品 CD またはインストールイメージの /REF ディレクトリで提供される) は、カスタムリソースを作成するためのガイドになります。この REF キットは、カスタムリソースや Active Sync 対応アダプタを作成するために必要なコード例とツールを提供します。

表 4-3 は、REF キットで提供されるファイルとディレクトリのカテゴリを示しています。

表 4-3 REF キットのファイルとディレクトリ

コンポーネント	場所	説明
waveset.properties	REF/config	カスタムアダプタをテストするときに必要な、プロパティーファイルのコピーが含まれています。
Javadoc	REF/javadoc	カスタムアダプタを記述するときに使用するべきクラスについて説明した、生成された JavaDoc が含まれています。これらの Javadoc の表示を開始するには、ブラウザで次の場所を参照します。 REF/javadoc/index.html
ソースファイル	REF	独自のアダプタを作成するときにガイドとして使用する、いくつかのサンプルのリソースアダプタソースファイルを提供します。
テストソースファイル	REF/test	カスタムアダプタの基礎として使用する、サンプルのリソースアダプタテストソースファイルが含まれています。
lib	REF/lib	カスタムアダプタのコンパイルやテストに必要な JAR ファイルが含まれています。
Before You Begin.README	REF	アダプタをカスタマイズする方法について説明した 1 ページの概要を提供します。
Design-for-Resource-Adapters.htm		リソースアダプタの基本的なアーキテクチャーを紹介しています。

リソースアダプタのサンプル

REF キットは、独自のカスタムアダプタの作成プロセスをすばやく開始するために使用できる、サンプルのアダプタファイルとツールを提供します。これらのリソースソースファイルでは、一般に開発されている各種タイプのリソースの例が提供されています。REF キットはまた、アダプタ開発の基礎として使用されることを目的とした、スケルトンファイルのセットも提供しています。

表 4-4 は、REF キットで提供されるサンプルファイルを示しています。

表 4-4 リソースアダプタのサンプルファイル

サンプルファイルの種類	ファイル名
データベースアカウント	`MySQLResourceAdapter.java`
データベーステーブル	`ExampleTableResourceAdapter.java*`
ファイルベースアカウント	`XMLResourceAdapter.java`
LDAP アカウント	カスタム LDAP リソースアダプタを開発するための例として、次の 2 つがあります。 • 単純なメソッドの場合は、`LDAPResourceAdapter.java` に基づいてアダプタを作成します。 • 複雑な変更の場合は、`LDAPResourceAdapterBase.java` に基づいてアダプタを作成します。
UNIX アカウント	`AIXResourceAdapter.java`
スケルトンファイル	• 標準リソースの場合 : `SkeletonStandardResourceAdapter.java` • 標準および Active Sync 対応リソースの場合 : `SkeletonStandardAndActiveSyncResourceAdapter.java` • Active Sync のみのリソースの場合 : `SkeletonActiveSyncResourceAdapter.java` カスタムアダプタのユニットテストを作成するには、`test.SkeletonResourceTest.java` ファイルを使用します。

注 テーブルベースのリソースの場合は、カスタムアダプタを記述する代わりに、リソースアダプタウィザードを使用してアダプタを作成します。

このウィザードを使用して、Sun から出荷されたアダプタをカスタマイズする方法の詳細については、『Identity Manager 管理ガイド』の「設定」の章を参照してください。

改訂のためのコードの特定

REF キットで出荷されたスケルトンファイルのどちらにも、アダプタに固有の変更が必要なコード内の場所を特定する、`@todo` と `change-value-here` のテキスト文字列が含まれています。これらのファイルをカスタマイズする場合は、これらの文字列を検索して、変更の必要な場所を特定してください。

表 4-5 は、カスタマイズが必要なコードを特定する場合に、検索できるテキストを示しています。

表 4-5　テキスト文字列の検索

コメント	説明
@todo	サポートしようとしている特定のシナリオを処理するために、書き換えが必要なメソッドを特定します。
change-value-here	実行が必要な置換を特定します。

ビルド環境の設定

ビルド環境を、次の手順で設定します。

前提条件： Identity Manager のバージョンに必要な JDK バージョンをインストールする必要があります。

JDK をインストールしたあと、システムに REF ディレクトリ全体をコピーすることによって REF キットをインストールします。

1. 新しいディレクトリに移動し、ws.bat (Microsoft Windows オペレーティングシステムを実行しているシステムで作業している場合) または ws.sh (UNIX システムで作業している場合) という名前のファイルを作成します。
2. このファイルに次の行を追加したあと、保存して終了します。

:

```
ws.bat ファイルを使用している場合は、次の行を追加します。
set WSHOME=<REF がインストールされているパス >
set JAVA_HOME=<REF がインストールされているパス >
set PATH=%PATH%;%JAVA_HOME%¥bin

ws.sh ファイルを使用している場合は、次の行を追加します。
WSHOME=<REF がインストールされているパス >
JAVA_HOME=<jdk がインストールされているパス >
PATH=$JAVA_HOME/bin:$PATH

export WSHOME JAVA_HOME PATH
```

ここで、WSHOME は REF キットがインストールされているパスであり、JAVA_HOME は Java がインストールされているパスを識別します。

はじめに

次の手順を使用して、カスタムアダプタの作成を開始します。

1. 適切なスケルトンファイルを、選択した名前にコピーします。
   - 標準リソースの場合 : `SkeletonStandardResourceAdapter.java`
   - 標準および Active Sync 対応リソースの場合 : `SkeletonStandardAndActiveSyncResourceAdapter.java`
   - Active Sync のみのリソースの場合 : `SkeletonActiveSyncResourceAdapter.java`
2. アダプタソースファイルを編集します。`Skeleton` のグローバル検索を実行し、それをアダプタの名前に置き換えます。
3. 先に作成した `ws.bat` または `ws.sh` ファイルに基づいて、環境を設定します。
4. 次のコマンドを使用して、ソースファイルをコンパイルします。

   ```
   javac -d . -classpath %CLASSPATH% yourfile.java
   ```

リソースアダプタの把握

ここでは、ほとんどのアダプタに共通する、リソースアダプタのソースコードコンポーネントの概要について説明したあと、Resource Extension Facility の一部であるスケルトンアダプタの例を示します。

アダプタコンポーネント

表 4-6 は、主なリソースアダプタコンポーネントを示しています。

表 4-6　アダプタコンポーネント

ファイルコンポーネント	説明
標準の Java ヘッダー情報	作成している新しいアダプタクラスファイルの親クラスの識別情報、コンストラクタ、およびインポートされたファイルが含まれています。
prototypeXML 文字列	Identity Manager インタフェースに、リソースに関するどの情報が表示されるかを定義します。各リソース属性を Identity Manager ユーザー属性にマップします。
リソースから Identity Manager へのフィードを作成するメソッド	Active Sync 対応アダプタでは、リソースを検索して変更がないかどうかを調べたり、更新を受信したりするメソッドを提供します。これらのメソッドを記述するには、そのリソースとの通信方法だけでなく、そのリソース上の変更を登録または検索する方法も理解していることが必要です。
Identity Manager リポジトリ内の更新操作を実行するメソッド	Active Sync 対応アダプタでは、リソースから Identity Manager へのフィードを実行します。
Identity Manager から外部リソースに情報を書き込むメソッド	これらのカスタマイズされたメソッドは、そのリソースをよく知っている開発者が記述する必要があります。

注　スケルトンアダプタファイルに対する 1 つの拡張が、`@todo` と `@change-attribute-here` のコメントの追加です。これらのコメントは、アダプタ固有のテキストの、置換するべき場所にマークを付けています。

リソースアダプタで定義されるリソース情報

Identity Manager 内のリソースを定義する情報には、次の 3 種類があり、これらの情報はアダプタファイル内でリソースタイプに対して定義されます。

- **リソース属性**。アダプタ Java ファイルの `prototypeXML` セクションで定義されます。これらの値は、このリソースタイプの特定のインスタンスを作成するときに Identity Manager インタフェースから修正できます。
- **アカウントの DN またはアイデンティティーテンプレート**。ユーザーに対するアカウント名の構文が含まれています。アカウント名の構文は、階層構造の名前空間で特に重要です。
- **Identity Manager アカウント属性**。リソースのスキーママップで定義されます。Identity Manager アカウント属性によって、リソースオブジェクトの Identity Manager 属性が定義されます。

標準の Java ヘッダー情報

ヘッダー情報は、標準の Java ファイル (public クラス宣言とクラスコンストラクタを含む) を表しています。コンストラクタと public クラスをリストしているファイルのセクション、および必要に応じてインポートされたファイルを編集する必要があります。

prototypeXML セクション

アダプタ Java ファイル (`prototypeXML`) のこのセクションは、リソースの XML 定義です。このセクションには、Identity Manager インタフェースに表示される、リソースのリソース名とすべての属性が含まれている必要があります。

表 4-7 は、prototypeXML のコンポーネントを説明しています。一部のコンポーネントは、Active Sync 対応アダプタよりも、リソースアダプタとの関連性が高くなっています。Active Sync 対応アダプタに関連する各コンポーネントは、この表のあとの節でより詳細に説明されています。

表 4-7　prototypeXML のコンポーネント

コンポーネント	説明
リソース	リソースのトップレベルの特性を定義します。この要素のキーワードには、次のものがあります。 • **syncSource**: true の場合、アダプタは Active Sync 対応である必要があります。 • **facets**: このリソースに対して有効になっているモードを指定します。有効な値は次のとおりです。 • **provision**: プロビジョニングが有効になっています。syncSource が true の場合は、両方のモードが使用できます。 • **activesync**: Active Sync のみ
リソース属性	リソースを設定するために Identity Manager によって使用されます。リソース属性は、<ResourceAttribute> 要素を使用して定義される XML 要素です。この要素のキーワードには、次のものがあります。 • type: サポートされているデータ型 (現在は string のみ) を特定します。 • multi: 属性として複数の値を受け付けることができるかどうかを指定します。true に設定されている場合は、複数行ボックスが表示されます。 • facets: このリソース属性の使用法を指定します。 • provision: 標準の処理で使用されます。これがデフォルト値です。 • activesync: Active Sync が設定され、有効になっている場合は、Active Sync 処理で使用されます。
アカウント属性	基本的なユーザー属性に対するデフォルトスキーママップを定義します。アカウント属性は、<AccountAttribute> 要素を使用して定義されます。カスタム属性にマップする場合とは異なり、標準の Identity Manager アカウント属性タイプにマップするアカウント属性を定義します。 アカウント属性のリソース属性へのマッピングの詳細については、「リソース属性を Identity Manager アカウント属性にマップする方法」を参照してください。
アイデンティティーテンプレート	<Template> タグで定義されるこのテンプレートは、ユーザーのアカウント名の作成方法を定義します。アカウント名には一般に、2 つの形式があります。1 つの形式は accountId であり、一般には、NT や Oracle などのフラットな名前空間を持つリソースに使用されます。 もう一つは、cn=accountId,ou=sub-org,ou=org,o=company という形式をした、ユーザーの完全な識別名です。この後者の形式は、ディレクトリなどの階層構造の名前空間に使用されます。

表 4-7 prototypeXML のコンポーネント (続き)

コンポーネント	説明
ログイン設定	(標準リソースアダプタのみ) <LoginConfigEntry> 要素で定義されるこの値は、このリソースのパススルー認証サポートのための値を定義します。パススルー認証の詳細については、『Sun Java™ System Identity Manager リソースリファレンス』を参照してください。
フォーム	(Active Sync 対応アダプタのみ) Active Sync 対応アダプタからのデータが Identity Manager に統合される前に、そのデータを処理するフォームオブジェクトを指定します。フォームはオプションですが、ほとんどの場合は、将来の柔軟な変更を可能にします。フォームを使用すると、受信したデータを変換したり、そのデータをほかのリソースアカウント上のほかのユーザー属性にマップしたり、Identity Manager でほかの操作を発生させたりすることができます。
リソースユーザーフォーム	(Active Sync 対応アダプタのみ) <ResourceUserForm> 要素で定義されます。

リソースメソッド

リソースメソッドは、タスクごとに分類できます。独自のカスタムアダプタを開発する場合は、最初に、開発の目標を満たすにはどのカテゴリが必要かを判断します。たとえば、アダプタを Active Sync 対応アダプタにしますか。また、配備の最初の段階では、パスワードリセットのみをサポートしますか。これらの質問への答えによって、どのメソッドを完成する必要があるかが決定されます。各機能カテゴリに関する追加の情報については、この章のあとの方で説明します。

これらのリソースメソッドのカテゴリを、表 4-8 に示します。

表 4-8 リソースメソッドのカテゴリ

カテゴリ	説明
基本	リソースに接続し、単純な操作を実行するための基本的なメソッドを提供します。
一括操作	リソースからすべてのユーザーを取得するための一括操作を提供します。
Active Sync	アダプタをスケジュールするためのメソッドを提供します。
オブジェクト管理	リソース上のグループと組織を管理するためのメソッドを提供します。

prototypeXML セクションの定義

リソースのソースコード内で、prototypeXML は、Identity Manager リポジトリに格納されるリソースオブジェクトを定義します。

表 4-9 は、Identity Manager 内のリソースを定義するための 5 種類の情報を示しています。最初の 3 つの種類は、管理者がそのリソースを定義しているときに表示されます。残りの 2 つの種類はリソースの定義に役立ちますが、Identity Manager 管理者が簡単には変更できません。

表 4-9 Identity Manager リソースを定義するための情報の種類

情報の種類	説明
リソース属性	管理対象のリソース上の接続情報を定義します。リソース属性には一般に、リソースのホスト名、リソースの管理者名とパスワード、およびディレクトリベースのリソースのコンテナ情報が含まれます。また、リソース承認者のリストや、リソース上の操作を再試行する回数などの Identity Manager 属性もリソース属性と見なされます。
アカウント属性	Identity Manager 属性名とリソース属性名の間の関係をマップします。これらの属性はリソーススキーママップ上で定義され、prototypeXML に表示されます。たとえば、LDAP 上のリソース属性 `sn` は、`lastname` という Identity Manager 内の、より一般的な属性にマップされます。
アイデンティティーテンプレート アカウントの DN	ユーザーのためのデフォルトのアカウント名を定義します。
ログイン設定	このリソースをパススルー認証に使用する場合に、使用されるパラメータを定義します。一般に、これらのパラメータは `username` と `password` です。ただし、SecurId の例には、ユーザー名とパスコードが含まれています。
オブジェクト管理	

リソース属性。

リソース属性は、次のものを定義します。

- 管理対象のリソース、およびその他の接続やリソースの特性。

  Identity Manager 管理者インタフェースを使用している管理者から見ると、これらの属性は、Identity Manager インタフェースに表示され、ユーザーに値を入力するよう求めるフィールド名を定義します。

Windows NT リソースの場合は、属性にソース名、ホスト名、ポート番号、ユーザー、パスワード、およびドメインを含めることができます。たとえば、リソースタイプの「リソースの作成」/「リソースの編集」ページには、リソースを作成している管理者がそのリソースが存在するホストを特定するためのホストフィールドが必要です。このフィールド (フィールドの内容ではない) は、このアダプタファイルで定義されます。

- このリソース上にユーザーを作成する権限を持つ、承認されたアカウント。Windows NT リソースの場合は、ユーザーとパスワードのフィールドが含まれます。
- フォーム、アダプタの実行に使用される Identity Manager 管理者、スケジューリングやログの情報、Active Sync メソッドでのみ使用される追加の属性を含むソース属性。

リソース属性の定義方法: リソース属性は、次に示すように、<ResourceAttribute> 要素を使用して Java ファイルで定義されます。

```
<ResourceAttribute name='"+RA_HOST+"' type='string' multi='false'¥n"+
description='&lt;b&gt;host&lt;/b&gt;&lt;br&gt;Enter the resource host name.'>¥n"+
```

description フィールドは、RA_HOST フィールドに対する項目レベルのヘルプを特定します。< の文字を含めることはできません。前の例では、< の文字が < と ' に置き換えられています。

表 4-10 は、<ResourceAttribute> 要素で使用できるキーワードを示しています。

表 4-10 <ResourceAttribute> 要素のキーワード

キーワード	説明
name	属性の名前を特定します。必須属性のリストについては、「必須リソース属性」を参照してください。
type	使用されるデータ型を特定します。現在は、string 型だけがサポートされています。
multi	属性として複数の値を受け付けることができるかどうかを指定します。true の場合は、複数行ボックスが表示されます。
description	RA_HOST フィールドに対する項目レベルのヘルプを特定します。Identity Manager は、説明されている項目 (この場合は host) を含むヘルプをボールドテキストで表示します。これを行うために必要な HTML の山括弧 (< および >) が XML の解析と干渉するため、これらの文字は < と > に置き換えられます。バイナリが変換されると、description の値は次のように表示されます。 Description='host Enter the resource host name.'

表 4-10 <ResourceAttribute> 要素のキーワード (続き)

キーワード	説明
facets	このリソース属性の使用法を指定します。有効な値は次のとおりです。 • provision: 標準の処理で使用されます。これがデフォルト値です。 • activesync: Active Sync 対応アダプタのために Active Sync 処理で使用されます。

リソース属性の上書き: リソースアダプタやアダプタのパラメータを操作する場合は、次の 2 つの方法のいずれかを選択できます。

- アダプタの「属性」ページを使用して、リソース属性値をすべてのユーザーに対して 1 回設定します。
- アダプタでデフォルトの属性値を設定したあと、必要に応じて、ユーザーフォーム内でその値を上書きします。

コード例 4-1 の場合、ユーザーフォームは、各ユーザーの作成中に template のリソース属性値を上書きする必要があります。本稼働環境で同様のコードを実装する場合はおそらく、この template 値を計算するための、より詳細なロジックをユーザーフォームに含めることになります。

コード例 4-1 template のリソース属性値の上書き

```
<Field name='template'>
   <Display class='Text'>
      <Property name='title' value='NDS User Template'/>
   </Display
</Field>

<!-- NDS リソースの名前に合わせて NDS を変更する -->
<!-- 単語 Template は、リソース xml に示されるとおり、属性フィールドの名前である。>
<Field name='accounts[NDS].resourceAttributes.Template'>
   <Expansion>
      <ref>template</ref>
   </Expansion>
</Field>
```

必須リソース属性: 表 4-11 は、スケルトンアダプタファイルで提供されるリソース属性を示しています。

表 4-11 スケルトンアダプタファイル内のリソース属性

必須リソース属性	説明
RA_HOST	リソースのホスト名。これは、「リソースパラメータ」ページの「ホスト」フィールドに対応しています。
RA_PORT	リソースとの通信に使用されるポート番号。これは、「リソースパラメータ」ページの「ポート」フィールドに対応しています。
RA_USER	リソースに接続するための権限を持つ、ユーザーアカウントの名前。フィールド名は、「リソースパラメータ」ページによって異なります。
RA_PASSWORD	RA_USER で指定されたアカウントのパスワード。これは、「リソースパラメータ」ページの「ホスト」フィールドに対応しています。

表 4-12 は、ActiveSync クラスの ACTIVE_SYNC_STD_RES_ATTRS_XML 文字列で定義されている Active Sync 固有の属性を示しています。

表 4-12 ACTIVE_SYNC_STD_RES_ATTRS_XML で定義されている Active Sync 固有の属性

必須リソース属性	説明
RA_PROXY_ADMINISTRATOR	承認とログ記録のための Identity Manager 管理者。これは、Identity Manager 画面内の「プロキシ管理者」フィールドに対応しています。この値は、アダプタ Java ファイル内では定義しません。代わりに、このリソースタイプの特定のインスタンスを定義するときに、管理者がこの情報を入力します。
RA_FORM	受信した属性を処理し、それをビュー属性にマップするフォーム。これは、「入力フォーム」フィールドに対応しています。
RA_MAX_ARCHIVES	保持するログファイルの数を指定します。 • 0 (ゼロ) を指定した場合は、1 つのログファイルが繰り返し利用されます。 • -1 を指定した場合、ログファイルは破棄されません。
RA_MAX_AGE_LENGTH	ログファイルがアーカイブされるまでの最大時間を指定します。期間が 0 (ゼロ) の場合、期間ベースのアーカイブは行われません。RA_MAX_ARCHIVES の値が 0 (ゼロ) の場合、この期間が経過してもアーカイブは行われず、アクティブログは切り捨てられ、再利用されます。
RA_MAX_AGE_UNIT	seconds、minutes、hours、days、weeks、months のいずれかです。この値は、RA_MAX_AGE_LENGTH とともに使用されます。
RA_LOG_LEVEL	ログレベル (0: 無効、4: 非常に詳細)。これは、Identity Manager 画面内の「ログレベル」フィールドに対応しています。

表 4-12　ACTIVE_SYNC_STD_RES_ATTRS_XML で定義されている Active Sync 固有の属性 (続き)

必須リソース属性	説明
RA_LOG_PATH	ログファイルの絶対または相対パス。これは、Identity Manager 画面内の「ログファイルパス」フィールドに対応しています。
RA_LOG_SIZE	ログファイルの最大サイズ。これは、Identity Manager 画面内の「ログファイルの最大サイズ」フィールドに対応しています。
RA_SCHEDULE_INTERVAL	サポートされているスケジューリング間隔 (秒、分、時間、日、週、月) のポップアップメニュー。
RA_SCHEDULE_INTERVAL_COUNT	スケジュールされた期間の間隔の数 (たとえば、10 分は 10 の間隔数と分の間隔で構成される)。Active Sync 対応アダプタには必要ありません。
RA_SCHEDULE_START_TIME	実行する 1 日の中の時刻。この値を 13:00 に設定し、間隔を週に設定すると、アダプタは週に 1 回午後 1 時に実行されるように設定されます。Active Sync 対応アダプタには必要ありません。
RA_SCHEDULE_START_DATE	スケジューリングを開始する日付。日付を 20020601 に、間隔を月に、時刻を 13:00 に設定すると、アダプタは 6 月 1 日に実行を開始し、月に 1 回午後 1 時に実行されます。

表 4-13 は、デフォルトクラス内の ACTIVE_SYNC_EVENT_RES_ATTRS_XML 文字列で定義されている Active Sync 固有の属性を示しています。これらの各属性の詳細については、「Identity Manager ユーザーの特定」を参照してください。

表 4-13　ACTIVE_SYNC_EVENT_RES_ATTRS_XML で定義されている Active Sync 固有の属性

必須リソース属性	説明
RA_PROCESS_RULE	TaskDefinition の名前、またはフィード内のすべてのレコードに対して実行される TaskDefinition の名前を返す規則です。このパラメータは、ほかのすべてのパラメータより優先されます。
RA_CORRELATION_RULE	アカウントの名前空間内のリソースアカウント属性に基づいて、一致する可能性のあるユーザーまたはアカウント ID の文字列のリストを返す規則。
RA_CONFIRMATION_RULE	ユーザーが一致するかどうかを確認する規則。
RA_DELETE_RULE	リソース上で検出された削除が、IAPI 削除イベントまたは IAPI 更新イベントのどちらとして処理されるかを判定する規則。
RA_CREATE_UNMATCHED	true に設定されている場合は、一致しないアカウントを作成します。false に設定すると、処理規則が設定され、その規則が識別するワークフローによって作成が保証されていることが確認されないかぎり、アカウントを作成しません。デフォルトは true です。
RA_RESOLVE_PROCESS_RULE	相関規則の結果に対する確認規則を使用して、複数の一致が存在するときに実行するワークフローを判定する規則。

表 4-13　ACTIVE_SYNC_EVENT_RES_ATTRS_XML で定義されている Active Sync 固有の属性 (続き)

必須リソース属性	説明
RA_POPULATE_GLOBAL	activeSync 名前空間に加えてグローバル名前空間にも値を入力するかどうかを示します。デフォルトは false です。

アカウント属性

標準の Identity Manager アカウント属性 (AttributeDefinition オブジェクトで定義される) には、次の属性が含まれています。

- accountId
- email
- firstname
- fullname
- lastname
- password

注　これらの属性は、「ビュー」インタフェースで定義されます。これらの属性の詳細については、『Sun Java™ System Identity Manager ワークフロー、フォーム、およびビュー』を参照してください。

Identity Manager アカウント属性によって、リソースオブジェクトの Identity Manager 属性が定義されます。アダプタファイルの prototypeXML セクションは、受信したリソース属性を Identity Manager 内のアカウント属性にマップします。これらのアカウント属性は、「スキーマの編集」ページにアクセスしたときに表示されます。「スキーマの編集」ページにアクセスするには、「リソースの作成」/「リソースの編集」ページの最下部にある「スキーマの編集」をクリックします。

リソーススキーママップで指定された属性マッピングによって、ユーザーの作成時にどのアカウント属性を要求できるかが決定されます。ユーザー用に選択されたロールに基づいて、入力を求められる一連のアカウント属性は、選択されたロール内のすべてのリソースの属性の和集合になります。Active Sync 対応アダプタの場合は、これらの属性を使用して Identity Manager ユーザーアカウントを更新できます。Active Sync 対応アダプタはこれらの属性を収集し、入力フォーム用のグローバル領域内に格納します。

管理者は、リソースのインスタンスを作成したあと、スキーママップを使用して次の操作を行うことができます。

- リソース属性を、企業に必須のもののみに制限する。

- Identity Manager 属性をリソース属性にマップする。
- 複数のリソースで使用する一般的な Identity Manager 属性名を作成する。
- 必須のユーザー属性と属性タイプを識別する。

スキーママップはオプションです。管理者が、Active Sync 対応アダプタへの入力を編集できるようにするためのユーティリティーとして提供され、この入力は多くの場合、データベースの列名やディレクトリの属性名などになります。スキーママップとフォームを使用すると、リソースタイプを処理する Java コードを実装して、マップやフォーム内にリソース設定の詳細を定義できます。

リソースの作成またはリソーススキーママップの編集については、『Sun Java™ System Identity Manager 管理ガイド』を参照してください。

スキーママップと Active Sync 対応アダプタ

Identity Manager は、Active Sync リソースのスキーママップを、標準的なスキーママップの場合と同じ方法で使用します。つまり、リソースやそのローカル名からどの属性を取得するかを指定します。スキーママップにリストされているすべての属性名 (つまり、そのリソース上に存在するすべての属性) が、Active Sync フォームと、`activeSync.name` 属性を持つユーザーフォームに使用可能になります。Active Sync リソースがフォームを使用していない場合は、すべての属性がすべてのリソース上の同じ名前を持つ属性に自動的に伝播されることを保証するために、すべての属性がグローバルと見なされます。グローバル名前空間ではなく、フォームを使用してください。

ヒント グローバル名前空間内に `accountId` 属性を含めないでください。これは `waveset.account.global` を特定するために使用される、特殊な属性です。また、リソースアカウントがはじめて作成された場合は、`accountId` 属性が直接そのリソースの `accountId` にもなり、アイデンティティーテンプレートがバイパスされます。

たとえば、新しい Identity Manager ユーザーが Active Sync 対応アダプタを通して作成され、そのユーザーに LDAP アカウントが割り当てられている場合、LDAP の `accountID` は、DN テンプレートの正しい DN ではなく `global.accountId` に一致します。

リソース属性を Identity Manager アカウント属性にマップする方法

アカウント属性は、標準の Identity Manager アカウント属性またはカスタム属性 (拡張スキーマ属性と呼ばれる) のどちらかにマップできます。

コード例 4-2 では、受信されたリソース属性が、標準の Identity Manager アカウント属性にマップされています。<AccountAttributesTypes> 要素は、リソース属性を Identity Manager アカウント属性にマップする prototypeXML の部分を囲みます。各アカウント属性は、<AccountAttributeType> 要素を使用して定義されます。

コード例 4-2 <AccountAttributesType> を使用したアカウント属性の定義

```
"<AccountAttributeTypes>¥n"+
   <AccountAttributeType name='accountId' mapName='change-value-here'
mapType='string' required='true'>¥n"+
      "<AttributeDefinitionRef>¥nt"+
      <ObjectRef type='AttributeDefinition' name='accountId'/>¥n"+
      " </AttributeDefinitionRef>¥n"+
   "</AccountAttributeType>¥n"+
"</AccountAttributeTypes>¥n"+
```

リソース属性のアカウント属性へのマッピングの詳細については、「アダプタのオプションと属性の設定」を参照してください。

prototypeXML での標準リソースアダプタ固有の問題

リソースアダプタ内のアカウント属性のみに固有の問題として、次のものがあります。

- ユーザーアイデンティティーテンプレート
- 複数のユーザー属性からのアイデンティティーテンプレートの作成
- ログイン設定とパススルー認証

ユーザーアイデンティティーテンプレート

ユーザーアイデンティティーテンプレートは、リソース上でのアカウントの作成時に使用されるアカウント名を確立します。Identity Manager ユーザーアカウントに関する情報を、外部リソース上のアカウント情報に変換するために使用されます。

アカウント名は現在、次の 2 つの形式のどちらかです。

- フラットな名前空間
- 階層構造の名前空間

アイデンティティーテンプレートでは、任意のスキーママップ属性 (スキーママップの左側にリストされている属性) を使用できます。

ユーザーアイデンティティーテンプレートは、ユーザーフォームから上書きできます。この操作は、組織名を置換するために一般に実行されます。

Identity Manager 内のユーザー名について

ユーザーは、自分の各アカウントに対する ID を持っています。この ID は、一部またはすべてのアカウントで、同一にすることができます。システムは、アカウントの ID を、そのアカウントがプロビジョニングされるときに設定します。Identity Manager ユーザーオブジェクトは、ユーザーの ID と、それらの ID が対応するリソースの間のマッピングを維持します。ユーザーは、表 4-14 に示すように、キーとして使用される Identity Manager 内の主要な accountId だけでなく、`accountId:<resource name>` の形式で表される、そのユーザーがアカウントを持つ各リソースに対する別の accountId も持っています。

表 4-14 アカウント ID

属性	例
`accountId`	`maurelius`
`accountId:NT_Res1`	`marcus_aurelius`
`accountId:LDAP_Res1`	`uid=maurelius,ou=marketing,ou=employees,o=abc_company`
`accountId:AIX_Res1`	`maurelius`

フラットな名前空間

`accountId` 属性は一般に、次のようなフラットな名前空間を持つシステムに使用されます。

- Microsoft Windows NT 4.0
- UNIX システム (Solaris、AIX、または HP-UX)
- Oracle および Sybase リレーショナルデータベース

フラットな名前空間を持つリソースの場合、アイデンティティーテンプレートは、Identity Manager アカウント名を使用すべきであることを、単純に指定できます。

階層構造の名前空間

識別名には、アカウント名、組織単位、および組織を含めることができます。識別名は、階層構造の名前空間を持つシステムに使用されます。

階層構造の名前空間を持つリソースの場合は、フラットな名前空間の場合よりもテンプレートを複雑にすることができます。これにより、完全な階層構造の名前を作成できるようになります。

表 4-15 には、階層構造の名前空間と識別名の表現方法の例が含まれています。

表 4-15 階層構造の名前空間の例

システム	識別名の文字列
LDAP	`cn=$accountId,ou=austin,ou=central,ou=sales,o=comp`
Novell NDS	`cn=$accountId.ou=accounting.o=comp`
Microsoft Windows 2000	`CN=$fullname,CN=Users,DC=mydomain,DC=com`

次に示すのは、LDAP などの、階層構造の名前空間を持つリソースに対するアイデンティティーテンプレートの例です。

```
uid=$accountID,ou=$department,ou=People,cn=waveset,cn=com
```

各表記の意味は次のとおりです。

- `accountID` は Identity Manager のアカウント名
- `department` はそのユーザーの部署名

複数のユーザー属性からのアイデンティティーテンプレートの作成: アイデンティティーテンプレートはまた、複数のユーザー属性の一部からも作成できます。たとえば、テンプレートを、名の先頭文字と姓の 7 文字の組み合わせで構成することができます。このシナリオの場合は、目的のロジックを実行して、そのリソース上で定義されているアイデンティティーテンプレートを上書きするようにユーザーフォームをカスタマイズできます。

ログイン設定とパススルー認証: `<LoginConfigEntry>` 要素は、ログインモジュールの名前とタイプだけでなく、このリソースタイプが正常なユーザー認証を完了するために必要な一連の認証プロパティーも指定します。

アダプタファイルの `<LoginConfig>` および `<SupportedApplications>` セクションは、このリソースを「ログインモジュール」設定ページのオプションリストに含めるかどうかを指定します。このリソースをオプションリストに表示する場合は、ファイルのこのセクションを変更しないでください。

各 `<AuthnProperty>` 要素には、次の属性が含まれています。

表 4-16 <AuthnProperty> 要素の属性

属性	説明
`name`	内部の認証プロパティー名を特定します。
`displayName`	このプロパティーが HTML 項目としてログインフォームに追加されるときに使用する値を指定します。

表 4-16　<AuthnProperty> 要素の属性 (続き)

formFieldType	text または password のどちらかにできるデータ型を指定します。この型を使用して、このプロパティーに関連付けられた HTML フィールドへのデータ入力を表示 (text) または非表示 (password) のどちらにするかを制御します。
isId	このプロパティー値を Identity Manager の accountID にマップすべきかどうかを指定します。たとえば、プロパティー値が X509 証明書である場合は、このプロパティーをマップすべきではありません。
dataSource	このプロパティーの値のソースを指定します。デフォルト値は user です。
doNotMap	LoginConfigEntry にマップするかどうかを指定します。

ほとんどのリソースログインモジュールは、Identity Manager 管理者インタフェースと Identity Manager ユーザーインタフェースの両方をサポートしています。

コード例 4-3 は、SkeletonResourceAdapter.java で <LoginConfigEntry> 要素を実装する方法を示しています。

コード例 4-3　SkeletonResourceAdapter.java での <LoginConfigEntry> の実装

```
<LoginConfigEntry name='"+Constants.WS_RESOURCE_LOGIN_MODULE+"' type='"+RESOURCE_NAME+"'
displayName='"+RESOURCE_LOGIN_MODULE+"'>¥n"+
   "   <AuthnProperties>¥n"+
   "      <AuthnProperty name='"+LOGIN_USER+"' displayName='"+DISPLAY_USER+"' formFieldType='text'
isId='true'/>¥n"+
   "      <AuthnProperty name='"+LOGIN_PASSWORD+"' displayName='"+DISPLAY_PASSWORD+"'
formFieldType='password'/>¥n"+
   "   </AuthnProperties>¥n"+
   "   <SupportedApplications>¥n"+
   "      <SupportedApplication name='"+Constants.ADMINCONSOLE+"'/>¥n"+
   "      <SupportedApplication name='"+Constants.SELFPROVISION+"'/>¥n"+
   "   </SupportedApplications>¥n"+
"</LoginConfigEntry>¥n"+
```

LoginConfig エントリの例

次の LoginConfig エントリの例は、Identity Manager で提供されている LDAP リソースアダプタから引用されています。ここでは、dataSource 値が (指定されていない場合は) ユーザーによって指定される 2 つの認証プロパティーを定義しています。

コード例 4-4 は、サポートされているログインモジュールデータソースオプションを定義しています。

コード例 4-4 サポートされているログインモジュールデータソースオプションの定義

```
public static final String USER_DATA_SOURCE = "user";
public static final String HTTP_REMOTE_USER_DATA_SOURCE = "http remote user";
public static final String HTTP_ATTRIBUTE_DATA_SOURCE = "http attribute";
public static final String HTTP_REQUEST_DATA_SOURCE = "http request";
public static final String HTTP_HEADER_DATA_SOURCE = "http header";
public static final String HTTPS_X509_CERTIFICATE_DATA_SOURCE = "x509 certificate";
" <LoginConfigEntry name='"+WS_RESOURCE_LOGIN_MODULE+"'
type='"+LDAP_RESOURCE_TYPE+"'
displayName='"+Messages.RES_LOGIN_MOD_LDAP+"'>¥n"+
" <AuthnProperties>¥n"+
" <AuthnProperty name='"+LDAP_UID+"' displayName='"+Messages.UI_USERID_LABEL+"'
formFieldType='text' isId='true'/>¥n"+
" <AuthnProperty name='"+LDAP_PASSWORD+"'
displayName='"+Messages.UI_PWD_LABEL+"'
formFieldType='password'/>¥n"+
" </AuthnProperties>¥n"+
" </LoginConfigEntry>¥n"+
```

コード例 4-5 は、認証プロパティーの dataSource 値が、ユーザーによって指定されない場合のログイン設定エントリを示しています。この場合は、HTTP 要求ヘッダーから取得されます。

コード例 4-5 ログイン設定エントリ

```
" <LoginConfigEntry name='"+Constants.WS_RESOURCE_LOGIN_MODULE+"'
|type='"+RESOURCE_NAME+"'
displayName='"+RESOURCE_LOGIN_MODULE+"'>¥n"+
" <AuthnProperties>¥n"+
" <AuthnProperty name='"+LOGIN_USER+"' displayName='"+DISPLAY_USER+"'
formFieldType='text'
isId='true' dataSource='http header'/>¥n"+
" </AuthnProperties>¥n"+|
" </LoginConfigEntry>¥n"+
```

Windows NT オブジェクトのリソース属性の宣言

コード例 4-6 は、prototypeXML で、「リソースの作成」/「リソースの編集」ページに表示されるフィールドを定義する方法を示しています。

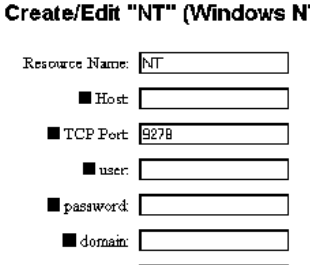
コード例 4-6 prototypeXML での「リソースの作成」/「リソースの編集」ページに表示されるフィールドの定義

```
<ResourceAttributes>
   <ResourceAttribute name='Host' description='The host name running the resource
      agent.' multi='false' value='n'>
   </ResourceAttribute>
   <ResourceAttribute name='TCP Port' description='The TCP/IP port used to communicate
      with the LDAP server.' multi='false' value='9278'>
   </ResourceAttribute>
   <ResourceAttribute name='user' description='The administrator user name with which
      the system should authenticate.' multi='false' value='Administrator'>
   </ResourceAttribute>
   <ResourceAttribute name='password' type='encrypted' description='The password that
      should be used when authenticating.' multi='false' value='VhXrkGkfDKw='>
   </ResourceAttribute>
   <ResourceAttribute name='domain' description='The name of the domain in which
      accounts will be created.' multi='false' value='nt'>
   </ResourceAttribute>
</ResourceAttributes>
```

prototypeXML の Identity Manager レンダリング

Identity Manager 管理者インタフェースには、前の節で指定したように、デフォルトの Windows NT リソースのリソース属性が表示されます。

図 4-1 Windows NT リソースのリソース属性



スケルトンファイルの編集：概要

SkeletonActiveSyncResourceAdapter ファイルは、管理者が特定のインスタンスを作成するために使用できる、新しい Active Sync 対応アダプタタイプの作成の出発点になります。このスケルトンファイルの名前を変更し、アダプタでサポートするデフォルト値を含むように編集したら、そのファイルを Identity Manager に読み込むことができます。

基本的な手順は次のとおりです。

- **リソースアダプタタイプに名前を付けます**。ここで付ける名前は、Identity Manager 管理者インタフェースの「新規リソース」メニューに表示されます。
- **リソース属性を Identity Manager アカウント属性にマップします**。それには、スケルトン prototypeXML 内のデフォルト値を編集して、このアダプタタイプのための独自のデフォルト値を作成する必要があります。たとえば、このファイルに含まれている、独自のアダプタタイプから RA_GROUPS 属性を削除することが必要な場合があります。
- **スケルトンファイルにメソッドを追加または削除します**。特に、このサンプルファイルではサポートされていない joins、leaves、および moves の操作をサポートするための Java コードを追加します。たとえば、Active Sync 対応アダプタが、HR データベースから情報をプルして従業員がまだアクティブかどうかを判定するように記述する必要があります。従業員のアクティブステータスの変更に基づいて更新をトリガーするには、データベース内の有効日列、検索する時間帯、以前の検索の繰り返しですでに実行されている更新のリストなどの、いくつかの項目が必要になります。

アダプタのネーミングと事前情報の編集

アダプタの事前情報を編集するには、次の 3 つの操作が必要です。

- SkeletonActiveSyncResourceAdapter.java ファイルまたはサンプルのアダプタファイルの名前を、独自のアダプタクラスの名前に変更します。
- コードを編集して SkeletonActiveSyncResourceAdapter を新しい名前に置き換えます。

アダプタのオプションと属性の設定

アダプタタイプのオプションを設定するための主な方法として、受信したリソース属性の標準の Identity Manager アカウント属性へのマッピング、または独自の拡張スキーマ属性の作成があります。SkeletonActiveSyncResourceAdapter に含まれている属性は必須です。ファイルをカスタマイズするときに、これらの属性定義を削除しないでください。

リソース属性の標準のアカウント属性へのマッピング

リソース属性を標準の Identity Manager アカウント属性のいずれかにマップする場合は、コード例 4-7 に示す構文を使用します。

コード例 4-7 リソース属性のマッピング

```
"<AttributeDefinitionRef>¥nt"+
   <ObjectRef type='AttributeDefinition' name='accountId'/>¥n"+
" </AttributeDefinitionRef>¥n"+
```

Identity Manager アカウント属性を特定するには、<AttributeDefinitionRef> 要素を使用します。表 4-17 は、<AttributeDefinitionRef> 要素のフィールドを示しています。

表 4-17 <AttributeDefinitionRef> 要素のフィールド

属性フィールド	説明
name	リソース属性がマップされる Identity Manager アカウント属性を特定します (Identity Manager ユーザーインタフェースにあるリソーススキーマページの左の列)。
mapName	受信したリソース属性の名前を特定します。このスケルトンファイルの編集時、change-value-here をこのリソース属性名に置き換えます。
mapType	受信した属性の型 (string、int、または encrypted) を特定します。

リソース属性の拡張スキーマ属性へのマッピング

受信したリソース属性を標準の Identity Manager 属性以外の属性にマップする場合は、拡張スキーマ属性を作成する必要があります。コード例 4-8 は、リソース属性をカスタムアカウント属性にマップする方法を示しています。

コード例 4-8 リソース属性のカスタムアカウント属性へのマッピング

```
<AccountAttributeType name='HomeDirectory' type='string'
   mapName='HomeDirectory' mapType='string'>¥n"+
</AccountAttributeType>¥n"+
```

ObjectRef 型を宣言する必要がないことに注意してください。mapName フィールドは、カスタムアカウント属性 HomeDirectory を特定します。mapType フィールドは、属性を標準のアカウント属性にマップする場合と同様に定義できます。

定義したリソースの有効性 (特に、そのリソースへの接続) をテストするには、アダプタを保存し、Identity Manager に読み込んでから、「リソース」ページの「開始」をクリックします。「開始」ボタンは、そのリソースの起動タイプが「自動」または「手動」の場合にのみ有効になります。アダプタがスケジュールされると、そのアダプタの init() および poll() メソッドが呼び出されます。

識別名の設定

新しいリソースを作成する場合、Identity Manager はリソースアダプタに、そのリソースの XML オブジェクト定義であるプロトタイプリソースを作成するよう依頼します。

この識別名は、次の Identity Manager ユーザーインタフェースページに表示されます。

- リソース
- 識別名テンプレート
- スキーマの編集

リソースがプロトタイプオブジェクトを提供する必要はありません。リソースに必要なのは、リソース属性を定義し、デフォルトを持つことに意味のあるリソース属性の、デフォルト値を設定することだけです。

リソースオブジェクトのクラス、タイプ、機能、および属性の定義

オブジェクトタイプによって、特定のタイプのリソースが一意に定義されます。オブジェクトタイプは、アダプタの prototypeXML セクションで定義されます。

オブジェクトタイプの定義

XML の <ObjectTypes> 要素は、アダプタの prototypeXML 内のコンテナであり、そのリソース上で管理される 1 つ以上のオブジェクトタイプ定義を含んでいます。

XML の <ObjectType> 要素は、<ObjectTypes> 要素内のコンテナであり、リソース固有のオブジェクトを Identity Manager に対して完全に記述しています。これには、次の情報が含まれます。

- そのオブジェクトタイプを構成する、特定のオブジェクトクラス (LDAP 準拠のディレクトリに対してのみ必要)
- サポートされている機能のリスト
- Identity Manager 内で編集や検索に使用可能なオブジェクトタイプ固有の属性のリスト

表 4-18 は、<ObjectType> 要素のサポートされている属性を示しています。

表 4-18 サポートされている <ObjectType> 要素

属性	説明
name	このオブジェクトタイプが、Identity Manager 内で表示または参照されるときに使用される名前を定義します (必須)。
icon	インタフェース内でこのタイプのオブジェクトの左に表示される .gif ファイルの名前を指定します。Identity Manager で使用するには、この .gif ファイルを idm/applet/images にインストールする必要があります。
container	true の場合は、このタイプのリソースオブジェクトに、同一かまたはほかのオブジェクトタイプの、ほかのリソースオブジェクトを含めることができることを示します。

オブジェクトタイプ定義の例

コード例 4-9 には、オブジェクトタイプ定義の例が含まれています。

コード例 4-9 オブジェクトタイプ定義の例

```
static final String prototypeXml ="<Resource name='Skeleton' class=
   'com.waveset.adapter.sample.SkeletonStandardResourceAdapter'
typeString='Skeleton of a resource adapter' typeDisplayString='"+Messages.RESTYPE_SKELETON+"'>¥n"+
   "   <ObjectTypes>¥n"+
   "   <ObjectType name='Group' icon='group'>¥n"+
…other content defined below will go here…
"    </ObjectType>¥n"+
"    <ObjectType name='Role' icon='ldap_role'>¥n"+
... other content defined below will go here ...
"    </ObjectType>¥n"+
"    <ObjectType name='Organization' icon='folder_with_org' container='true'>¥n"+
... other content defined below will go here ...
"    </ObjectType>¥n"+
"  </ObjectTypes>¥n"+
```

オブジェクトクラス

LDAP ベースのリソースオブジェクトの場合、オブジェクトクラスは、ほかのリソースオブジェクトとは別の方法で処理されます。

LDAP ベースのリソースオブジェクト

LDAP ベースのリソースオブジェクトは、複数の LDAP オブジェクトクラスで構成できます。ここで、各オブジェクトクラスはその親オブジェクトクラスの拡張です。ただし、LDAP 内では、これらのオブジェクトクラスの完全なセットが、LDAP 内の単一のオブジェクトタイプとして表示および管理されます。

Identity Manager 内でこのタイプのリソースオブジェクトを管理するには、`<ObjectType>` 定義内に XML 要素 `<ObjectClasses>` を含めます。`<ObjectClasses>` 要素を使用すると、この `<ObjectType>` に関連付けられた一連のオブジェクトクラスや、これらのクラスの相互の関係を定義できます。

LDAP ベース以外のリソースオブジェクト

LDAP ベース以外のリソースオブジェクトの場合は、`<ObjectType>` を使用して、リソースオブジェクトタイプ名以外の情報を表すことができます。

コード例 4-10 で、`primary` 属性は、このタイプのオブジェクトを作成および更新するときに使用されるオブジェクトクラスを定義しています。この場合、`inetorgperson` は、リストされているほかのオブジェクトクラスのサブクラスであるため、第一のオブジェクトクラスとして定義されます。`operator` 属性は、このタイプのオブジェクトをリストまたは取得するときに、オブジェクトクラスのリストを 1 つ (論理 AND) として処理するか、または固有のクラス (論理 OR) として処理するかを指定します。この場合、Identity Manager は、このオブジェクトタイプに対するリストまたは要求取得の前に、これらのオブジェクトクラスに対して AND 操作を実行します。

コード例 4-10 inetorgperson オブジェクトクラスの使用

```
<ObjectClasses primary='inetorgperson' operator='AND'>¥n"+
   <ObjectClass name='person'/>¥n"+
   <ObjectClass name='organizationalPerson'/>¥n"+
   <ObjectClass name='inetorgperson'/>¥n"+
</ObjectClasses>¥n"+
```

コード例 4-11 では、このタイプのリソースオブジェクトの作成または更新に対する要求はすべて、groupOfUniqueNames オブジェクトクラスを使用して実行されます。すべてのリストおよび取得要求で、オブジェクトクラスが groupOfNames または groupOfUniqueNames のどちらかであるすべてのオブジェクトに問い合わせが行われます。

コード例 4-11　　groupOfUniqueNames オブジェクトクラスの使用

```
<ObjectClasses primary='groupOfUniqueNames' operator='OR'>¥n"+    <ObjectClass
name='groupOfNames'/>¥n"+
   <ObjectClass name='groupOfUniqueNames'/>¥n"+
</ObjectClasses>¥n"+
```

コード例 4-12 では 1 つのオブジェクトクラスしか定義されていないため、すべての create、update、list、および get 操作が、オブジェクトクラス organizationalUnit を使用して実行されます。

コード例 4-12　　organizationalUnit オブジェクトクラスの使用

```
<ObjectClasses operator='AND'>¥n"+
   <ObjectClass name='organizationalUnit'/>¥n"+
</ObjectClasses>¥n"+
```

オブジェクトクラスが 1 つしか存在しないため、<ObjectClasses> セクションを取り除くこともできます。取り除いた場合、オブジェクトクラスのデフォルトは、<ObjectType> の name 属性の値になります。ただし、オブジェクトタイプ名をリソースオブジェクトクラス名とは別にする場合は、単一の <ObjectClass> エントリを含む <ObjectClasses> セクションを含める必要があります。

オブジェクト機能

<ObjectFeatures> セクションは、このオブジェクトタイプでサポートされている 1 つ以上の機能のリストを指定します。ここで、各オブジェクト機能は、リソースアダプタ内の関連付けられたオブジェクトタイプメソッドの実装に直接関連付けられています。

各 ObjectFeature 定義には、機能名を指定する name 属性が含まれている必要があります。また、create および update 機能では、form 属性が指定される可能性があります。この属性は、create および update 機能の処理に使用されるリソースフォームを定義します。form 属性が指定されていない場合、Identity Manager はこれらの機能を、指定されたタイプのすべてのリソースで使用されるものと同じフォームを使用して処理します。

表 4-19 は、オブジェクト機能のマッピングを示しています。

表 4-19 オブジェクト機能のマッピング

オブジェクト機能	メソッド	form 属性のサポート
create	createObject	はい
delete	deleteObject	いいえ
find	listObjects	いいえ
リスト	listObjects	いいえ
rename	updateObject	いいえ
saveas	createObject	いいえ
update	updateObject	はい
view	getObject	いいえ

コード例 4-13 の <ObjectFeatures> セクションには、サポートされているすべてのオブジェクト機能が含まれています。リソースアダプタは、すべての機能またはそのサブセットをサポートできます。アダプタでサポートするオブジェクト機能が多くなればなるほど、Identity Manager 内のオブジェクト管理機能は豊富になります。

コード例 4-13 サポートされているすべてのオブジェクト機能を含む <ObjectFeatures> セクション

```
 <ObjectFeatures>¥n"+
   <ObjectFeature name='create' form='My Create Position Form'/>
   <ObjectFeature name='update' form='My Update Position Form'/>
<ObjectFeature name='create'/>¥n"+
   <ObjectFeature name='delete'/>¥n"+
   <ObjectFeature name='rename'/>¥n"+
   <ObjectFeature name='saveas'/>¥n"+
   <ObjectFeature name='find'/>¥n"+
   <ObjectFeature name='list'/>¥n"+
   <ObjectFeature name='view'/>¥n"+
</ObjectFeatures>¥n"+
```

オブジェクト属性

<ObjectAttributes> セクションは、Identity Manager で管理および問い合わせされる属性セットを指定します。各 <ObjectAttribute> 要素の名前は、ネイティブなリソース属性名にすべきです。Identity Manager 内のユーザー属性とは異なり、属性マッピングは指定されません。ネイティブな属性名のみを使用してください。

表 4-20 は、<ObjectAttributes> に必要な属性を示しています。

表 4-20 <ObjectAttributes> の必須属性

属性	説明
idAttr	この属性の値は、リソースのオブジェクト名前空間内で、このオブジェクトを一意に特定するリソースオブジェクト属性名 (たとえば、dn、uid) になります。
displayNameAttr	この属性の値は、リソースオブジェクト属性名であり、その値は、Identity Manager 内でこのタイプのオブジェクトを表示するときに表示される名前 (たとえば、cn、samAccountName) になります。
descriptionAttr	(オプション) この属性の値は、「リソース」ページの「説明」列に値を表示する、リソースオブジェクト属性名になります。

コード例 4-14 は、<ObjectType> 内で定義された <ObjectAttributes> セクションを示しています。

コード例 4-14 <ObjectType> 内で定義された <ObjectAttributes> セクション

```
<ObjectAttributes idAttr='dn' displayNameAttr='cn' descriptionAttr=
       'description'>¥n"+
   <ObjectAttribute name='cn' type='string'/>¥n"+
   <ObjectAttribute name='description' type='string'/>¥n"+
   <ObjectAttribute name='owner' type='distinguishedname' namingAttr=
       'cn'/>¥n"+
   <ObjectAttribute name='uniqueMember' type='dn' namingAttr='cn' />¥n"+
</ObjectAttributes>¥n"+
```

各 <ObjectAttribute> には、表 4-21 に示す属性が含まれています。

表 4-21　<ObjectAttribute> の属性

属性	説明
name	リソースオブジェクトタイプの属性名を特定します (必須)
type	オブジェクトのタイプを特定します。有効なタイプには、string または distinguishedname / 'dn' (デフォルト値は string) があります
namingAttr	オブジェクトタイプが distinguishedname または dn である場合、この値は、Identity Manager 内で dn によって参照される、このオブジェクトタイプのインスタンスの表示に使用される値を持った属性を指定します

注　リソースアダプタのオブジェクトタイプ実装におけるメソッドは、リソース属性名に基づいて、すべての文字列値が適切なタイプになるように強制する役割を果たします。

フォームの割り当て

受信したデータを Identity Manager に格納する前に、処理するためのオプションのフォームを割り当てることができます。このリソースフォームは、受信したデータをスキーママップから変換して、ユーザービューに適用するためのメカニズムです。また、サンプルフォームでは、従業員ステータスなどの、受信したデータの特定の値に基づいた (アカウントの有効化や無効化などの) 操作も実行しています。

リソースフォームの定義

Create 機能をサポートするリソースの各 <ObjectType> に対して、*<resource type>* Create *<object type>* Form という名前のリソースフォームを指定する必要があります。たとえば、*AIX Create Group Form* または *LDAP Create Organizational Unit Form* とします。

また、Update 機能をサポートするリソースの各 <ObjectType> に対して、*<resource type>* Update <object type> Form という名前の ResourceForm を指定する必要があります。たとえば、*AIX Update Group Form* または *LDAP Update Organizational Unit Form* とします。

表 4-22 は、トップレベルの名前空間に含まれている属性を示しています。特に指定されていない限り、すべての値が文字列です。

表 4-22　トップレベルの名前空間の属性

属性	説明
<objectType>.resourceType	Identity Manager のリソースタイプ名 (たとえば、LDAP、Active Directory)
<objectType>.resourceName	Identity Manager のリソース名
<objectType>.resourceId	Identity Manager のリソース ID
<objectType>.objectType	リソース固有のオブジェクトタイプ (たとえば、Group)
<objectType>.objectName	リソースオブジェクトの名前 (たとえば、cn または samAccountName)
<objectType>.objectId	リソースオブジェクトの完全修飾名 (たとえば、dn)
<objectType>.requestor	表示を要求しているユーザーの ID
<objectType>.attributes	リソースオブジェクト属性の名前と値のペア (オブジェクト)
<objectType>.organization	Identity Manager のメンバー組織
<objectType>.attrsToGet	checkoutView または getView を介したオブジェクトの要求時に返すオブジェクトタイプ固有の属性のリスト (リスト)
<objectType>.searchContex	フォーム入力内の非完全修飾名の検索に使用されるコンテキスト
<objectType>.searchAttributes	フォームに入力された名前の、指定された searchContext 内での検索に使用されるリソースオブジェクトタイプ固有の属性名のリスト (リスト)
<objectType>.searchTimeLimit	検索に費やされた最大時間。ここで、<objectType> は、リソース固有のオブジェクトタイプの小文字の名前です。たとえば、group、organizationalunit、organization になります。
<objectType>.attributes <resource attribute name>	指定されたリソース属性の値の取得または設定に使用されます (たとえば、<objectType>.attributes.cn。ここで、cn はリソース属性名)。リソース属性が識別名である場合、値の取得時に返される名前は、<ObjectType> の説明の <ObjectAttribute> セクションで指定された namingAttr の値です。

次の節では、リソースオブジェクトの名前空間にアクセスする方法の例を示す、フォームを参照します。

アダプタメソッドの記述

Identity Manager のアダプタインタフェースは、特定の環境に応じてカスタマイズする必要のある、一般的なメソッドを提供しています。ここでは、これらのメソッドの概要と、次の内容について説明します。

- **標準リソースアダプタ固有のメソッド**。これらのメソッドは、Identity Manager と同期させるように更新しているリソースに固有のものです。
- **Active Sync 固有のメソッド**。これらのメソッドは、信頼性の高いリソースからプルした情報に基づいて Identity Manager を更新するためのメカニズムを提供します。また、アダプタを停止、起動、およびスケジュールするためのメソッドも含まれています。

標準リソースアダプタ固有のメソッド

リソースアダプタの本体は、リソース固有のメソッドで構成されています。そのため、リソースアダプタで提供されるメソッドは、記述しようとしている特定のメソッドのための、汎用のプレースホルダにすぎません。

ここでは、操作を実装するために使用されるメソッドが、どのように分類されるかについて説明します。これらの情報は、次のように構成されています。

- プロトタイプリソースの作成
- リソースへの接続
- 接続と操作の確認
- 機能の定義
- リソース上のアカウントの作成
- リソース上のアカウントの削除
- リソース上のアカウントの更新
- ユーザー情報の取得
- リストメソッド
- 有効化および無効化メソッド

注	カスタムアダプタを記述する場合は、カスタムメソッドによって返される任意の `WSUser` オブジェクト上で `setdisabled()` メソッドを呼び出します。

プロトタイプリソースの作成

表 4-23 は、リソースインスタンスの作成に使用されるメソッドを示しています。

表 4-23　リソースインスタンスの作成に使用されるメソッド

メソッド	説明
`staticCreatePrototypeResource`	リソースアダプタで定義されている、定義済みのプロトタイプ XML 文字列からリソースインスタンスを作成します。static メソッドであるため、リソースアダプタである Java クラスへのパスのみがわかっている場合に呼び出すことができます。
`createPrototypeResource`	リソースアダプタクラスの Java オブジェクトのインスタンスが、すでに存在する場合にのみ実行できるローカルメソッド。通常、`createPrototypeResource()` の実装は、`staticCreatePrototypeResource()` メソッドの呼び出しだけです。

リソースへの接続

次のメソッドは、承認ユーザーとして接続および切断を確立する役割を果たします。すべてのリソースアダプタが、これらのメソッドを実装する必要があります。

- `startConnection`
- `stopConnection`

接続と操作の確認

`ResourceAdapterBase` は、アダプタが実際の操作を試みる前に、操作の有効性 (そのリソースへの接続が機能してるかどうかなど) を確認するために使用できるメソッドを提供します。

表 4-24 に示すメソッドは、通信の確立が可能であり、承認されたアカウントにアクセス権があることを確認します。

表 4-24 通信の確認に使用されるメソッド

メソッド	説明
checkCreateAccount	リソース上でアカウントを作成できるかどうかを確認します。次の機能を確認できます。 • リソースへの基本的な接続を確立できるか。 • このアカウントがすでに存在するか。 • アカウント属性値が、より高いレベルでは未確認のリソース固有の制限またはポリシー (存在する場合) のすべてに従っているか。 このメソッドは、アカウントがすでに存在するかどうかを確認しません。このメソッドには、アカウント名、パスワード、ユーザー名などの、ユーザーアカウントを作成するために必要なアカウント属性情報が含まれています。 アカウントの作成が可能なことを確認したあと、このメソッドはリソースへの接続を閉じます。
checkUpdateAccount	接続を確立し、アカウントの更新が可能かどうかを確認します。 このメソッドは、入力としてユーザーオブジェクトを受け取ります。このメソッドには、アカウント名、パスワード、ユーザー名などの、ユーザーアカウントを作成するために必要なアカウント属性情報が含まれています。 ユーザーオブジェクトは、追加または変更されたアカウント属性を指定します。これらの属性のみが確認されます。
checkDeleteAccount	アカウントが存在し、削除可能かどうかを確認します。次の機能を確認できます。 • リソースへの基本的な接続を確立できるか。 • このアカウントがすでに存在するか。 • アカウント属性値が、より高いレベルでは未確認のリソース固有の制限またはポリシー (存在する場合) のすべてに従っているか。 このメソッドは、アカウントがすでに存在するかどうかを確認しません。このメソッドは、入力としてユーザーオブジェクトを受け取ります。このメソッドには、アカウント名、パスワード、ユーザー名などの、ユーザーアカウントを削除するために必要なアカウント属性情報が含まれています。 アカウントの削除が可能かどうかを確認したあと、このメソッドはリソースへの接続を閉じます。

機能の定義

getFeatures() メソッドは、アダプタでどの機能がサポートされているかを指定します。機能は、次のように分類できます。

- 一般的な機能
- アカウントの機能
- グループの機能
- 組織単位の機能

ResourceAdapterBase クラスは、getFeatures() メソッドの基本実装を定義します。以下の表の「基本で有効」列は、その機能が ResourceAdapterBase 内の基本実装で有効として定義されているかどうかを示します。

表 4-25　一般的な機能

機能名	基本で有効	コメント
ACTIONS	いいえ	前後の操作がサポートされているかどうかを示します。有効にするには、true 値を使用して supportsActions メソッドをオーバーライドします。
RESOURCE_PASSWORD_CHANGE	いいえ	リソースアダプタがパスワード変更をサポートしているかどうかを示します。有効にするには、supportsResourceAccount メソッドをオーバーライドします。

表 4-26　アカウントの機能

機能名	基本で有効	コメント
ACCOUNT_CASE_INSENSITIVE_IDS	はい	ユーザーアカウント名の大文字と小文字が区別されるかどうかを示します。アカウント ID の大文字と小文字が区別されるようにするには、false 値を使用して supportsCaseInsensitiveAccountIds メソッドをオーバーライドします。
ACCOUNT_CREATE	はい	アカウントを作成できるかどうかを示します。この機能を無効にするには、remove 操作を使用します。
ACCOUNT_DELETE	はい	アカウントを削除できるかどうかを示します。この機能を無効にするには、remove 操作を使用します。

表 4-26　アカウントの機能 (続き)

機能名	基本で有効	コメント
ACCOUNT_DISABLE	いいえ	リソース上でアカウントを無効にできるかどうかを示します。この機能を有効にするには、true 値を使用して supportsAccountDisable メソッドをオーバーライドします。
ACCOUNT_EXCLUDE	いいえ	Identity Manager から管理アカウントを除外できるかどうかを判定します。この機能を有効にするには、true 値を使用して supportsExcludedAccounts メソッドをオーバーライドします。
ACCOUNT_ENABLE	いいえ	リソース上でアカウントを有効にできるかどうかを示します。リソース上でアカウントを有効にできる場合は、true 値を使用して supportsAccountDisable メソッドをオーバーライドします。
ACCOUNT_EXPIRE_ PASSWORD	はい	アダプタのスキーママップ内に Identity Manager のユーザー属性 expirePassword が存在する場合は有効になります。この機能を無効にするには、remove 操作を使用します。
ACCOUNT_GUID	いいえ	リソース上に GUID が存在する場合、この機能を有効にするには put 操作を使用します。
ACCOUNT_ITERATOR	はい	アダプタがアカウント反復子を使用するかどうかを示します。この機能を無効にするには、remove 操作を使用します。
ACCOUNT_LIST	はい	アダプタがアカウントをリストできるかどうかを示します。この機能を無効にするには、remove 操作を使用します。
ACCOUNT_LOGIN	はい	ユーザーがアカウントにログインできるかどうかを示します。ログインを無効にできる場合は、remove 操作を使用します。
ACCOUNT_PASSWORD	はい	アカウントにパスワードが必要かどうかを示します。パスワードを無効にできる場合は、remove 操作を使用します。
ACCOUNT_RENAME	いいえ	アカウントの名前を変更できるかどうかを示します。この機能を有効にするには、put 操作を使用します。
ACCOUNT_REPORTS_DISABLED	いいえ	アカウントが無効になっているかどうかを、リソースが報告するかどうかを示します。この機能を有効にするには、put 操作を使用します。

表 4-26　アカウントの機能 (続き)

機能名	基本で有効	コメント
ACCOUNT_UNLOCK	いいえ	アカウントのロックを解除できるかどうかを示します。アカウントのロックを解除できる場合は、put 操作を使用します。
ACCOUNT_UPDATE	はい	アカウントを変更できるかどうかを示します。アカウントを更新できない場合は、remove 操作を使用します。
ACCOUNT_USER_PASSWORD_ON_CHANGE	いいえ	パスワードの変更時にユーザーの現在のパスワードを指定する必要があるかどうかを示します。ユーザーの現在のパスワードが必要な場合は、put 操作を使用します。

表 4-27　グループの機能

機能名	基本で有効	コメント
GROUP_CREATE GROUP_DELETE GROUP_UPDATE	いいえ	グループを作成、削除、または更新できるかどうかを示します。リソース上でこれらの機能がサポートされている場合は、put 操作を使用します。

表 4-28　組織単位の機能

機能名	基本で有効	コメント
ORGUNIT_CREATE ORGUNIT_DELETE ORGUNIT_UPDATE	いいえ	組織単位を作成、削除、または更新できるかどうかを示します。リソース上でこれらの機能がサポートされている場合は、put 操作を使用します。

カスタムアダプタで getFeatures メソッドの ResourceAdapterBase 実装をオーバーライドする場合は、次のようなコードを追加します。

```
public GenericObject getFeatures() {
GenericObject genObj = super.getFeatures();
genObj.put(Features.ACCOUNT_RENAME, Features.ACCOUNT_RENAME);
genObj.remove(Features.ACCOUNT_UPDATE, Features.ACCOUNT_UPDATE);
.. other features supported by this Resource Adapter…
return genObj;
}
```

別のメソッド (supportsActions など) をオーバーライドすることによって機能を有効にするには、次のようなコードを追加します。

```
public boolean supportsActions() {
   return true;
}
```

次の表は、リソース上のアカウントを作成、削除、および更新するために使用されるメソッドを示しています。

表 4-29 リソース上のアカウントの作成

メソッド	説明
realCreate()	リソース上のアカウントを作成します。 入力としてユーザーオブジェクトを受け取ります。このメソッドには、ユーザーアカウントを作成するために必要なアカウント属性情報 (アカウント名、パスワード、ユーザー名など) が含まれています。

表 4-30 リソース上のアカウントの削除

メソッド	説明
realDelete()	リソース上のアカウントを削除します。 入力として、ユーザーオブジェクトまたはユーザーオブジェクトのリストを受け取ります。デフォルトでは、このメソッドはリスト内のユーザーオブジェクトごとに接続を作成し、realDelete を呼び出し、接続を閉じます。

表 4-31　リソース上のアカウントの更新

メソッド	説明
`realUpdate()`	アカウント属性のサブセットを更新します。 デフォルトでは、このメソッドはリスト内のユーザーオブジェクトごとに接続を作成し、`realUpdate` を呼び出し、接続を閉じます。

注　リソースからのユーザーアカウント属性は、Identity Manager からの任意の新しい変更とマージされます。

表 4-32　ユーザー情報の取得

メソッド	説明
`getUser()`	リソースからユーザー属性に関する情報を取得します。 入力としてユーザーオブジェクト (通常は、1 つのアカウントアイデンティティーセットだけを含む) を受け取り、リソーススキーママップで定義されている、任意の属性に対して設定された値を含む新規ユーザーオブジェクトを返します。

リストメソッドを使用すると、アダプタがリソースからユーザー情報を取得するために使用するプロセスを確立できます。

表 4-33　リストメソッド

メソッド	説明
`getAccountIterator()`	リソースからすべてのユーザーを検出またはインポートするために使用されます。 リソースのすべてのユーザーに対して処理を繰り返すために、アカウント反復子のインタフェースを実装します。

表 4-33　リストメソッド (続き)

メソッド	説明
listAllObjects ()	リソースオブジェクトタイプ (accountID やグループなど) が指定されると、リソースからそのタイプのリストを返します。 リソースグループや配布リストのリストなどの、リソースで使用されているリストを生成するには、このメソッドを実装します。 このメソッドは、プロビジョニングエンジンからではなく、ユーザーフォームから呼び出されます。

コード例 4-15 には、リソース上の情報を取得し、それを Identity Manager が操作できる情報に変換するためのコードが含まれています。

コード例 4-15　リソースアダプタ : リソース上の情報の取得

```
public WSUser getUser(WSUser user)
throws WavesetException
String identity = getIdentity(user);
WSUser newUser = null;
try {
startConnection();
Map attributes = fetchUser(user);
if (attributes != null) {
newUser = makeWavesetUser(attributes);
}
} finally {
stopConnection();
}
return newUser;
}
```

表 4-34　有効化および無効化メソッド

メソッド	説明
supportsAccountDisable()	リソースがネイティブアカウントの無効化をサポートしているかどうかに応じて、true または false を返します。
realEnable()	リソース上のユーザーアカウントを有効にするために必要な、ネイティブな呼び出しを実装します。
realDisable()	リソース上のユーザーアカウントを無効にするために必要な、ネイティブな呼び出しを実装します。

ユーザーアカウントの無効化

リソースでサポートされている無効化ユーティリティー、または Identity Manager で提供されるアカウント無効化ユーティリティーを使用することにより、アカウントを無効にすることができます。

注　可能な場合は常に、ネイティブな無効化ユーティリティーを使用してください。

- **アカウント無効化のネイティブサポート**: 特定のリソースでは、設定されるとユーザーをログインできなくする、個別のフラグが提供されています。このユーティリティーの例には、NT 用のユーザーマネージャー、Active Directory 用の Active Directory ユーザーとコンピュータ、NDS/Netware 用の ConsoleOne または Netware Administrator などがあります。アカウントが有効になると、ユーザーの元のパスワードが引き続き有効になります。supportsAccountDisable メソッドを実装することによって、リソース上でアカウント無効化のネイティブサポートが使用可能かどうかを判定できます。
- **Identity Manager の無効化ユーティリティー**: リソースがアカウントの無効化をサポートしていない場合、またはユーザーのパスワードのリセットによって無効化をサポートしている場合は、Identity Manager プロビジョニングエンジンがアカウントを無効にします。ランダムに生成され、表示も保持もされないパスワードをユーザーアカウントに設定することによって、無効化を実行できます。アカウントが有効になると、システムによって新規パスワードがランダムに生成され、それが Identity Manager 管理者インタフェースに表示されるか、または電子メールでユーザーに送信されます。

リソースタイプのパススルー認証の有効化

リソースタイプのパススルー認証を有効にするには、次の一般的な手順を使用します。

1. アダプタの getFeatures() メソッドが、サポートされている機能として ResourceAdapter.ACCOUNT_LOGIN を返すことを確認してください。

   - カスタムアダプタで ResourceAdapterBase 実装をオーバーライドする場合は、次のコードを追加します。

```
public GenericObject getFeatures() {
GenericObject genObj = super.getFeatures();
genObj.put(Features.ACCOUNT_RENAME, Features.ACCOUNT_RENAME);
.. other features supported by this Resource Adapter…
return genObj;
}
```

   - カスタムアダプタで ResourceAdapterBase クラス内の getFeatures() 実装をオーバーライドしない場合は、ACCOUNT_LOGIN に対してデフォルトでエクスポートされた getFeatures() 実装が継承されます。

2. アダプタの prototypeXML に <LoginConfigEntry> 要素を追加します。

3. アダプタの authenticate() メソッドを実装します。

   authenticate() メソッドは、loginInfo マップで提供された認証プロパティーの名前と値のペアを使用して、リソースに対してユーザーを認証します。認証が成功した場合は、次のように結果を追加することにより、認証された一意の ID が WavesetResult で返されるようにしてください。

```
result.addResult(Constants.AUTHENTICATED_IDENTITY, accountID);
```

   認証は成功したが、ユーザーのパスワードの期限が切れた場合は、上で追加した ID に加えて、返される結果にパスワード期限切れインジケータも追加します。これにより、ユーザーが Identity Manager への次回のログイン時に、少なくともリソース上のパスワードの変更を強制されるようになります。

```
result.addResult(Constants.RESOURCE_PASSWORD_EXPIRED, new
Boolean(true));
```

   (ユーザー名またはパスワードが無効であるために) 認証が失敗した場合は、次のようにします。

```
throw new WavesetException("Authentication failed for " + uid + ".");
```

Active Sync 固有のメソッドの記述

アダプタのこのセクションでは、Identity Manager の更新という、アダプタの主要なタスクを実行するためのメソッドを提供する必要があります。記述するメソッドは、スケルトンアダプタファイルで提供されている汎用のメソッドに基づきます。

タスクごとに分類された、これらのメソッドのいくつかを編集する必要があります。これらのタスクを処理するための一般的なガイドラインについて、以下の節で説明します。

- アダプタの初期化とスケジューリング
- リソースのポーリング
- アダプタ属性の格納と取得
- Identity Manager リポジトリの更新
- アダプタの停止

アダプタの初期化とスケジューリング

アダプタの初期化とスケジューリングは、`init()` および `poll()` メソッドを実装することによって行います。

`init()` メソッドは、アダプタマネージャーがアダプタを読み込んだときに呼び出されます。アダプタを読み込む方法には、次の 2 つがあります。

- アダプタの起動タイプが「自動」の場合、管理者は、システムの起動時にアダプタを読み込むことができます。
- アダプタの起動タイプが「手動」の場合、管理者は、「リソース」ページの**「開始」**をクリックすることによってアダプタを読み込みます。

初期化プロセスで、アダプタは独自の初期化を実行できます。一般に、この処理には、ログの初期化 (`ActiveSyncUtil` クラスを使用) や、更新イベントを受信するためのリソースへの登録などの、アダプタ固有の任意の初期化が含まれます。

例外がスローされた場合、アダプタは停止され、読み込み解除されます。

リソースのポーリング

アダプタの機能はすべて、`poll()` メソッドによって実行されます。アダプタをスケジュールするには、`poll()` メソッドを、リソース上の変更された情報を検索して取得するように設定する必要があります。

このメソッドは、Active Sync 対応アダプタのメインのメソッドです。アダプタマネージャーは、リモートリソースの変更をポーリングするために poll() メソッドを呼び出します。次に、この呼び出しによって変更が IAPI 呼び出しに変換され、サーバーに戻されます。このメソッドは独自のスレッド上で呼び出され、必要な期間だけブロックできます。

このメソッドは、自身の ActiveSyncUtil インスタンスの isStopRequested メソッドを呼び出し、true の場合は戻るはずです。変更をループ処理する場合は、ループ条件の一部として isStopRequested をチェックしてください。

ポーリングのデフォルト値を設定するために、アダプタファイル内のポーリング関連のリソース属性を設定できます。これらのポーリング関連の属性を設定すると、あとで Identity Manager インタフェースを使用して、ポーリング間隔の開始時刻や日付、間隔の長さなどを設定するための手段が、管理者に提供されます。

スケジューリングパラメータ

Active Sync 対応アダプタでは、次のスケジューリングパラメータが使用されます。

- RA_SCHEDULE_INTERVAL
- RA_SCHEDULE_INTERVAL_COUNT
- RA_SCHEDULE_START_TIME
- RA_SCHEDULE_START_DATE

これらのパラメータについては、表 4-12 で説明します。

prototypeXML 内のスケジューリングパラメータ

スケジューリングパラメータは、ActiveSync の文字列定数 ACTIVE_SYNC_STD_RES_ATTRS_XML や、ほかのすべての一般的な Active Sync 関連のリソース属性に存在します。

サンプルのポーリングシナリオ

次の表は、いくつかのサンプルのポーリングシナリオを示しています。

表 4-35 サンプルのポーリングシナリオ

ポーリングシナリオ	パラメータ
毎日午前 2 時	Interval = day, count =1, start_time=0200
毎日 4 回	Interval=hour, count=6.
隔週の木曜日午後 5 時にポーリング	Interval = week, count=2, start date = 20020705 (木曜日), time = 17:00.

アダプタ属性の格納と取得

ほとんどの Active Sync 対応アダプタは、標準アダプタでもあります。ここでは、1 つの Java クラスが、ResourceAdapterBase (または AgentResourceAdapter) の拡張と、ActiveSync インタフェースの実装の両方を行います。

したがって、コード例 4-16 に示すように、属性の取得と更新は基底クラスに渡すようにしてください。

コード例 4-16 属性の取得と更新

```
public Object getAttributeValue(String name) throws WavesetException {
      return getResource().getResourceAttributeVal(name);
}
public void setAttributeValue(String name, Object value) throws WavesetException {
      getResource().setResourceAttributeVal(name,value);
```

Identity Manager リポジトリの更新

更新を受信すると、アダプタは IAPI クラス、特に IAPIFactory を使用して次の処理を行います。

- 変更された属性を収集する
- その変更を一意の Identity Manager オブジェクトにマップする
- 変更された情報を使用してそのオブジェクトを更新する

変更の Identity Manager オブジェクトへのマッピング

リソースに対する Active Sync のイベントパラメータ設定プログラムを使用して、IAPIFactory.getIAPI は、変更された属性のマップから IAPI オブジェクト (IAPIUser または IAPIProcess のどちらか) を作成します。リソースに対して除外規則 (iapi_create、iapi_delete、または iapi_update) が設定されている場合、IAPIFactory は、そのアカウントが除外されるかどうかを確認します。null 以外のオブジェクトが作成され、Factory によって返された場合、アダプタはその IAPI オブジェクトを変更 (たとえば、ロガーを追加) して送信することができます。

オブジェクトが送信されると、リソースに関連付けられたフォームは、オブジェクトビューがチェックインされる前にそのビューを使用して展開されます。フォームとビューの詳細については、『Identity Manager ワークフロー、フォーム、およびビュー』を参照してください。

SkeletonActiveSyncResourceAdapter では、このプロセスは buildEvent および processUpdates メソッドで処理されています。

アダプタの停止

アダプタの停止に関連したシステム要件はありません。これはシステムクリーンアップでの作業です。

カスタムアダプタのインストール

カスタマイズしたリソースアダプタをインストールするには、次の手順に従います。

1. `NewResourceAdapter.class` ファイルを、Identity Manager インストールディレクトリの次の場所に読み込みます。

   `idm/WEB-INF/classes/com/waveset/adapter/sample`

   このディレクトリを作成することが必要な場合もあります。

2. `.gif` ファイルを `idm/applet/images` にコピーします。

   この `.gif` ファイルは、「リソースのリスト」ページでリソース名の横に表示されるイメージで、このファイルにはサイズが 18x18 ピクセル (72 DPI) のリソースのイメージを含めるようにしてください。

3. このクラスを `config/waveset.properties` 内の `resource.adapter` プロパティーに追加します。

4. アプリケーションサーバーを停止して再起動します (アプリケーションサーバーの操作については、『Identity Manager インストール』を参照)。

5. リソースの HTML ヘルプファイルを作成します。例については、`com/waveset/msgcat/help/resources` ディレクトリにある `idm.jar` を参照してください。また、アプリケーションにオンラインヘルプを組み込む方法の手順については、『Identity Manager ワークフロー、フォーム、およびビュー』を参照してください。

6. アダプタとそれに関連したヘルプファイルを Identity Manager にインストールします。

7. アダプタを使用して、Identity Manager 内にリソースを作成します。

8. リソースを起動し、接続を確認します。

カスタムアダプタの保守

Identity Manager サービスパックをインストールした場合は、新しい idmcommon.jar および idmformui.jar ファイルを使用してカスタムリソースをテストする必要があります。新しいリリースまたはサービスパックで加えられた変更に適応するために、カスタムアダプタの変更や拡張が必要になる場合があります。あるいは、インストール内でリソースアダプタを再ビルドまたは更新するだけで済む場合もあります。

アダプタの保守の詳細については、『Identity Manager 管理ガイド』を参照してください。

アダプタのテスト

アダプタを記述したら、それをテストして Identity Manager に読み込む必要があります。テストには、Identity Manager からのテストと、独自のマシン上でのユニットテストの実行の両方が含まれます。

定義したリソースの有効性 (特に、そのリソースへの接続) をテストするには、アダプタを保存し、Identity Manager に読み込んでから、「リソースのリスト」ページの「開始」をクリックします。「開始」ボタンは、そのリソースの起動タイプが「自動」または「手動」の場合にのみ有効になります。

この節は、次のように構成されています。

カスタムアダプタのテスト

アダプタをデバッグするための 1 つのツールとして、すべてのアダプタが生成するログファイルがあります。このログファイルは $WSHOME/config ディレクトリ内に生成され、WSTrace1.log という名前が付けられます。

注	ログが生成されるには、トレースが有効になっていて、トレースの要求対象のメソッドが特定されている必要があります。この操作は、Identity Manager の「デバッグ」ページ、またはコマンド行ユーティリティーを使用して実行できます。また、カスタムアダプタにも、新しいメソッドのログエントリを作成するための、呼び出しが含まれている必要があります。

アダプタは、すべてのリソース設定をログファイルに書き込みます。これを使用すると、アダプタが起動されたこと、および設定変更が保存されたことの両方を検証できます。

ActiveSyncUtil インスタンスへのログ呼び出しを行う Active Sync 対応アダプタは、「ログファイルパス」リソース属性で指定されたディレクトリ内に 1 つの (または一連の) ログファイルを作成します。これらのログファイルを確認して、Active Sync 関連のログエントリがほかにないかどうかを調べてください。

一般的な手順

アダプタをデバッグする場合は、次の一般的な手順に従います。

1. アダプタのためのテストプログラムを作成します。

   この Java ファイルでは、次の基本的な機能を実行するようにしてください。

   - 新しいリソースを作成する
   - ユーザーを作成する
   - ユーザーを取得する
   - ユーザーを更新する
   - ユーザーを削除する
   - 複数のユーザーに対して create、get、update、および remove 操作を実行する

   インストール CD の /REF には、サンプルのテストファイル (SkeletonResourceTests.java) が収録されています。

2. デバッグのレベルに応じて、ログレベルを設定します。最初のデバッグでは、ログレベルを 4 に増やし、ログファイルのパスとサイズを設定します。その後アダプタを起動すると、そのアダプタがすべてのリソース設定をログファイルに書き込みます。これを使用すると、アダプタが起動されたこと、および設定変更が保存されたことの両方を検証できます。

3. アダプタをコンパイルしてテストします。テストプログラムをコンパイルするには、javac -d . test/filename.java を使用します。このコマンドによって、適切な com/waveset/adapter/test ディレクトリ内にクラスファイルが作成されます。このファイルを使用して新しいアダプタをテストするには、コンパイルされたアダプタが com/waveset/adapter ディレクトリに存在することを確認してから、次のコマンドを使用して実行します。

```
java ー D waveset.home=<path> com.waveset.adapter.test.MyResourceAdapter
```

4. リソースの HTML ヘルプファイルを作成します。例については、com/waveset/msgcat/help/resources ディレクトリにある idm.jar を参照してください。アプリケーションにオンラインヘルプを組み込む方法の手順については、『Identity Manager ワークフロー、フォーム、およびビュー』を参照してください。
5. (Active Sync 対応アダプタのみ) 最後のリソースに対する同期をリセットするには、XmlData SYNC_*resourceName* オブジェクトを削除します。
6. エラーログを読み取り、アダプタを変更します。
7. デバッグレベルを 2 に設定します。レベル 2 のデバッグでは、アダプタ設定と任意のエラーに関する情報が生成されますが、詳細ログの量は管理しやすいレベルに制限されます。
8. Identity Manager を起動する前に、$WSHOME/config/waveset.properties ファイル内で新しいアダプタを特定する必要があります。このためには、アダプタの名前を resource.adapters エントリの下に配置する必要があります。そうしないと、Identity Manager がそのアダプタを認識しません。
9. アダプタとそれに関連したヘルプファイルを Identity Manager にインストールします。

注 新しいアダプタのインスタンスを表示で認識できるようにするには、そのタイプの新しいリソースを作成する必要があります。この操作は、「リソースのリスト」ページで実行できます。このページから、「新規」>「新しいアダプタ」を選択します。リソースウィザードを使用して新しいアダプタを作成します。

---

10. Identity Manager を使用して、リソースと、そのリソース上のユーザーを作成します。

ヒント

Active Sync 対応アダプタをデバッグしているとき、`XmlData SYNC_`*resourceName* オブジェクトを編集して、「デバッグ」ページから ActiveSync 同期プロセスの `MapEntry` を削除すると、アダプタは最初に検出された変更から起動し直します。

IAPI イベントを使用した場合は、`Property()` メソッドを設定して、そのリソースの同期状態 (`last change processed` 値など) を格納するようにします。このメソッドの設定は、アダプタのデバッグに非常に有効です。動作させて過去の変更を無視するように、アダプタを設定できます。その後、アダプタを変更し、その変更の結果をアダプタログファイルで確認できます。

リソースが Active Sync リソースである場合は、そのリソースの編集ページでログをオンに設定することによって、追加情報を取得できます。ログレベル (0 ～ 4) と、ログファイルを書き込むファイルパス (たとえば、*resource_name*`.log`) を設定します。

11. (Active Sync 対応アダプタのみ) 最後のリソースに対する同期を再起動します。

テストアダプタ内のメソッドのトレース

テストアダプタ内のメソッドをトレースするには、次の手順に従います。

1. 「デバッグ」ページ (`debug/Show_Trace.jsp`) から、トレースをオンに設定します。
2. 次の形式を使用して、アダプタを追加します。

   ```
   com.waveset.adapter.sample.MyResourceAdapter
   ```

3. トレースレベルを 4 に設定して最大の出力にするか、または 2 に設定して十分な出力にします。
4. トレース情報を標準出力に出力するか、またはファイルに送信するかを指定します。
5. 同期プロセスをさらにデバッグするには、テストリソースに対する同期ログを設定します。その手順は、Identity Manager のオンラインヘルプで提供されています。

Identity Manager でのリソースオブジェクトのテスト

Identity Manager 管理者インタフェースの「リソースの検索」および「リソースのリスト」ページを通して実装をテストできます。

- 「リソース」>「リソースのリスト」を選択して、次のパフォーマンス特性を確認します。

表 4-36 リソースのリストのパフォーマンス特性

インタフェースでの期待される動作	異なっていた場合の処置
Identity Manager の新しいリソースのドロップダウンリストに、作成したリソースタイプが含まれている。	作成したリソースタイプを、Waveset.properties ファイル内の resource.adapters 属性に追加したことを確認してください。
リソースフォルダを開いたとき、その内容に、リソースアダプタの <ObjectTypes> セクションで定義されているすべての <ObjectType> 要素が反映されている。	アダプタの prototypeXML 内の <ObjectType> 要素を確認してください。
リソースオブジェクトタイプの 1 つを右クリックしたとき、リソースアダプタの <ObjectType> ごとの <ObjectFeatures> セクションで指定された、サポートされているすべての機能がメニューから選択できる。	「デバッグ」ページに移動し、問題のリソースを表示または編集して、問題の <ObjectType> に対する <ObjectFeatures> のリストが正しいことを確認してください。
新しいリソースを作成したり、既存のリソースオブジェクトを更新したりできる。	リソースアダプタコードが Web-INF/classes/com/waveset/adapter/sample に含まれていることを確認してください。
操作のタイプごとに正しい ResourceForms が読み込まれている。	• 必要なすべてのリソースフォームをチェックインしたことを確認してください。 • システム設定オブジェクト内の各フォームのセクションで、フォームが (大文字と小文字の区別も含め) 正しく参照されていることを確認してください。

- 「リソース」>「リソースの検索」を選択して、次のパフォーマンス特性を確認します。

表 4-37 リソースの検索のパフォーマンス特性

インタフェースでの期待される動作	異なっていた場合の処置
「リソース」>「リソースの検索」ページから、期待されるすべての属性を設定できる	すべての <ObjectType> 要素と、それに関連付けられた <ObjectAttribute> 要素を確認してください。

表 4-37　リソースの検索のパフォーマンス特性 (続き)

インタフェースでの期待される動作	異なっていた場合の処置
リソース検索要求が適切なリソースオブジェクトを返す	クエリーの引数を二重にチェックして、適切な一連のリソースオブジェクトが、そのクエリーに一致することを確認してください。それでも機能しない場合は、別の LDAP ブラウザから同じクエリーを試行して、それがクエリーによる問題ではないことを確認してください。
検索要求から返されたオブジェクトを編集または削除できる。	問題の <ObjectType> の <ObjectFeatures> セクションに、編集を有効にする Update 機能、または削除を有効にする Delete 機能が含まれていることを確認してください。

リソースオブジェクトの表示

「デバッグ」ページからリソースオブジェクトを表示できます。「デバッグ」ページを開くには、「http://*build_name*/idm/debug」と入力します。

リソースアダプタクラスと Active Sync 対応アダプタクラスはすべて、既存の Identity Manager リソースクラスに基づいています。

リソースオブジェクトを表示するには、次の手順に従います。

1. 「List Objects」の横にあるオプションメニューからリソースを選択します。
2. 「List Objects」をクリックします。この操作によって、すべてのリソースアダプタと Active Sync 対応アダプタのリストが表示されます。
3. 表示するリソースオブジェクトの横にある「View」をクリックするか、または「Edit」をクリックしてそのリソースオブジェクトを編集します。

一般的なエラー

一般的なエラーには、次のものがあります。

- フォーム関連のエラー
- 認証プロパティーが存在しない
- 一致するリソースアカウントを持つ Identity Manager ユーザーが見つからない

フォーム関連のエラー

Active Sync 対応アダプタでの一般的なエラーは、フォーム関連のエラーです。これらのエラーは通常、パスワードや電子メールなどの必須フィールドが設定されていないために発生します。

フォームの検証エラーは、表示の最後の xml のあとに出力されます。一般的なエラーは、次のように表示されます。

```
20030414 17:23:57.469: result from submit (blank means no errors):
20030414 17:23:57.509: Validation error: missing required field password
```

すべてメッセージも同時に出力され、アカウントの作成および更新時刻、アダプタエラー、スキーママップデータの概要などが指定されます。

リソースアダプタは、処理した最後の変更に関する情報を *SYNC.resourceName XMLData* オブジェクトに格納します。

認証プロパティーが存在しない

必要な認証プロパティー値が存在しない場合は、そのプロパティー名が、指定されたデータソースタイプのトレースにダンプされている、一連の名前に含まれていることを確認します。

一致するリソースアカウントを持つ Identity Manager ユーザーが見つからない

リソースアダプタの認証は成功しますが、一致したリソースアカウント ID を持つ Identity Manager ユーザーが見つからないことを示す例外がスローされます。そのユーザーに関連付けられたリソース `accountId` が、リソースアダプタの authenticate メソッドが返す ID と同じであることを確認します。

Identity Manager ユーザーのリソース `accountId` は、「デバッグ」ページで確認できます。一致しない場合は、`authenticate` メソッドが返す名前の内容を変更するか、または認証が返す ID に一致するリソース `accountId` が間違いなく生成されるようにリソースの ID テンプレートを変更するか、のどちらかを行う必要があります。

LoginConfig の変更のデバッグ

アダプタへの LoginConfig 関連の変更をデバッグするには、次の操作を行う必要があります。

1. 選択ファイルのトレースを有効にします。
2. Telnet を介したシングルサインオンパススルー認証ログインをテストします。

Identity Manager のトレースの有効化

次のクラスに対する Identity Manager のレベル 1 のトレースを有効にします。

- `com.waveset.security.authn.WSResourceLoginModule`
- `com.waveset.session.LocalSession`
- `com.waveset.session.SessionFactory`
- `com.waveset.ui.LoginHelper`
- `com.waveset.ui.web.common.ContinueLoginForm`
- `com.waveset.ui.web.common.LoginForm`

シングルサインオン (SSO) パススルー認証のテスト

SSO ログインモジュールを正しく設定したら、http ポートに直接 telnet 接続し、http 要求を `login.jsp` に送信できます。

次の要求を telnet セッションにペーストできます。

```
HEAD /idm/login.jsp HTTP/1.0
Accept: text/plain,text/html,*/*
Accept-Language: en-us
Accept-Encoding: gzip, deflate
User-Agent: Mozilla/4.0
Host: LOCALHOST
sm_user: Configurator
```

この要求には、HTTP ヘッダー `sm_user` を検索する SSO ログインモジュールが含まれています。この情報を telnet 画面にペーストしたら、ユーザーが正しくログインしたことを示すトレースを確認するようにしてください。

たとえば、コード例 4-17 を参照してください。

コード例 4-17 サンプル出力

```
2003.07.08 14:14:16.837 Thread-7 WSResourceLoginModule#checkForAuthenticatedResourceInfo()
Found authenticated resource accountId, 'Configurator@Netegrity SiteMinder' on Identity Manager user
'Configurator'. null null 2003.07.08 14:14:16.837 Thread-7
WSResourceLoginModule#checkForAuthenticatedResourceInfo()
Exit null null 2003.07.08 14:14:16.837 Thread-7 WSResourceLoginModule#login()
Exit, return code = true null null 2003.07.08 14:14:16.847
Thread-7 LocalSession#login() Login succeeded via Netegrity SiteMinder null null
2003.07.08 14:14:16.847 Thread-7 LocalSession#login() Overall authentication
succeeded null null 2003.07.08 14:14:16.897 Thread-7 LocalSession#checkIfUserDisabled() Entry null
null 2003.07.08 14:14:16.897 Thread-7 LocalSession#checkIfUserDisabled() Exit null null 2003.07.08
14:14:16.927 Thread-7 LocalSession#login() Exit null null
```

第 5 章

ファイアウォールまたはプロキシサーバーの操作

この章では、Identity Manager での URL (Uniform Resource Locator) の使用方法と、ファイアウォールまたはプロキシサーバーが設置されている場合に、正確な URL データを取得するように Identity Manager を設定する方法について説明します。

Servlet API

Web ベースの Identity Manager ユーザーインタフェースは、URL (Uniform Resource Locator) に大きく依存しながら、Web クライアントで取得されるページの場所を指定します。

Identity Manager は、生成される HTML および HTTP 応答内に有効な URL を配置できるように、アプリケーションサーバー (Apache Tomcat、IBM WebSphere、BEA WebLogic など) で提供される Servlet API に依存して、現在の HTTP 要求内の完全修飾 URL を決定します。

構成によっては、Web クライアントが HTTP 要求のために使用する URL を、アプリケーションサーバーが決定できない場合があります。この例には次のものがあります。

- Web クライアントと Web サーバーの間、または Web サーバーとアプリケーションサーバーの間に配置されたポート転送またはネットワークアドレス変換 (NAT) ファイアウォール
- Web クライアントと Web サーバーの間、または Web サーバーとアプリケーションサーバーの間に配置されたプロキシサーバー (Tivoli Policy Director WebSEAL など)

Servlet API によって HTTP 要求から正確な URL データが提供されない場合は、`Waveset.properties` ファイル (Identity Manager インストールの `config` ディレクトリに格納されている) で正しいデータを設定できます。

次の属性は、Identity Manager Web のドキュメントルートと、Identity Manager が HTML BASE HREF タグを使用するかどうかを制御します。

- `ui.web.useBaseHref` (デフォルト値 : `true`) ー この属性を次のいずれかの値に設定します。
  - `true` ー Identity Manager は、HTML BASE HREF タグを使用して、すべての相対 URL パスのルートを示します。
  - `false` ー HTML に配置されるすべての URL に絶対パス (スキーマ、ホスト、およびポートを含む) が含まれます。
- `ui.web.baseHrefURL` ー この属性に空以外の値を設定して、生成された HTML で使用される BASE HREF を定義します。この値によって、Servlet API を使用して計算された値が上書きされます。

  この計算された値の上書きは、これらの API から返される値が必ずしも正確でない場合に有効です。この状況は、次の場合に発生します。
  - アプリケーションサーバーが、ポート転送または NAT を使用するファイアウォールの背後に位置している
  - アプリケーションサーバーと Web サーバーの間のコネクタから正確な情報が提供されない
  - アプリケーションサーバーのフロントエンドにプロキシサーバーが配置されている

第 6 章

辞書サポートの設定

この章では、単純な辞書攻撃からパスワードを保護するのに役立つ、辞書ポリシーの設定方法について説明します。説明する内容は次のとおりです。

辞書ポリシーについて

辞書ポリシーを使用すると、Identity Manager は単語データベースと照合してパスワードをチェックすることができ、単純な辞書攻撃から保護されることが保証されます。このポリシーをほかのポリシー設定と組み合わせて使用して、パスワードの長さと構成を強制することにより、Identity Manager では、システム内で生成または変更されたパスワードを、他人が辞書を使用して推測することが困難になります。

この辞書ポリシーは、「ポリシーの編集」ページ (「セキュリティー」>「ポリシー」>「Password Policy」) にある「使用禁止単語」機能を使用して、指定されたパスワード除外リストを拡張します。

辞書ポリシーの設定

辞書ポリシーを設定するには、辞書サーバーサポートを設定してから、辞書を読み込む必要があります。それには、次の手順に従います。

1. Identity Manager 管理者ユーザーインタフェースから、「セキュリティー」>「ポリシー」を選択し、「辞書の設定」ボタンをクリックします。
2. 「辞書の設定」ページが表示されたら、次のデータベース情報を指定します。
   - **「データベースタイプ」** – 辞書の保存に使用するデータベースタイプ (Oracle、DB2、SQLServer、または MySQL) を選択します。
   - **「ホスト」** – データベースが実行されているホストの名前を入力します。
   - **「ユーザー」** – データベースに接続するときに使用するユーザー名を入力します。
   - **「パスワード」** – データベースに接続するときに使用するパスワードを入力します。
   - **「ポート」** – データベースがリスニング中のポートを入力します。
   - **「接続 URL」** – 接続のときに使用する URL を入力します。

     次のテンプレート変数を使用することができます。
     - %h – ホスト
     - %p – ポート
     - %d – データベース名
   - **「ドライバクラス」** – データベースを操作する際に使用する JDBC ドライバクラスを入力します。
   - **「データベース名」** – 辞書の読み込み先のデータベースの名前を入力します。
   - **「テーブル命名用コンテキスト」** – データベース内の辞書テーブルの命名に使用されるプレフィックスを入力します。
   - **「辞書ファイル名」** – 辞書を読み込むときに使用するファイルの名前を入力します。
3. データベース接続をテストするには、「テスト」をクリックします。
4. 接続テストが成功したら、「単語の読み込み」をクリックして、辞書を読み込みます。

   注 読み込み作業が完了するまでに、数分かかる場合があります。

5. その辞書が正しく読み込まれたかどうかを確認するには、「テスト」をクリックします。

6. 「保存」をクリックして変更を保存します。

辞書ポリシーの実装

辞書ポリシーを実装するには、次の手順に従います。

1. 「ポリシー」ページで、「パスワードポリシー」リンクをクリックしてパスワードポリシーを編集します。
2. 「ポリシーの編集」ページで、「辞書の単語でパスワードをチェックする」オプションを有効にします。
3. 「保存」をクリックして変更を保存します。

実装すると、変更および生成されたパスワードはすべて、Identity Manager によって辞書と照合してチェックされます。

第 7 章

Identity Manager Web サービスでの SPML 1.0 の使用

この章では、Identity Manager および Identity Manager Service Provider Edition でサポートされる SPML 1.0 について説明しています。これには、サポートされる機能とその理由、SPML 1.0 サポートの設定方法、フィールドでのサポートの拡張方法が含まれます。

説明する内容は次のとおりです。

- Identity Manager Web サービスインタフェースの操作
- SPML の設定
- 要求の処理方法について
- SPML ブラウザの起動
- Identity Manager サーバーへの接続
- SPML 設定のテストとトラブルシューティング
- SPML アプリケーションの開発
- 例

注

この章では、SPML 1.0 のみを扱います。
特に明記されていないかぎり、この章での SPML への参照はすべて version *1.0* を示しています。

ここで説明されている概念は、第 8 章「Identity Manager Web サービスでの SPML 2.0 の使用」で、SPML 2.0 についての説明を参照するときにも役立ちます。

この章の対象読者

アプリケーション開発者および Identity Manager の統合を担当する開発者は、この章で説明されている SPML 1.0 クラスを使用して、サービスプロビジョニング要求メッセージをフォーマットしたり、応答メッセージを解析したりすることができます。

Identity Manager Web サービスインタフェースの操作

Identity Manager Web サービスは、HTTP 用の SOAP メッセージを使用してアクセスされます。Identity Manager は、プロビジョニングシステムとの通信のための OASIS 標準である Service Provisioning Markup Language (SPML) の両方のバージョン、つまり、version 1.0 および 2.0 をサポートしています。

注 SPML 1.0 を使用して Identity Manager Service Provider Edition (SPE) の機能にアクセスすることもできます (これらの機能は SPML version 2.0 では使用できない)。

SPE SPML インタフェースは通常の Identity Manager SPML インタフェースと非常によく似ています。設定および操作上の相違点は、この章の該当箇所に注記してあります。

SPML 1.0 は、サービスプロビジョニングアクティビティーにオープンインタフェースを提供するための OASIS 標準です。SPML 1.0 は、独立系ソフトウェアベンダーによって支持されています。

注 Identity Manager Web インタフェースの操作で最高のパフォーマンスを得るには、Identity Manager にバンドルされている OpenSPML ツールキットを使用してください。http://www.openspml.org/ Web サイトにある `openspml.jar` ファイルを使用すると、メモリーリークが発生する可能性があります。

注 SPE REF キットには、SPE SPML インタフェースの使用方法を実演する `SpmlUsage.java` ファイルが含まれています。

SPML の設定

SPML インタフェースを公開するには、Identity Manager サーバーを正しく設定する必要があります。特定のリポジトリオブジェクトをインストールして変更し、`waveset.properties` ファイルを編集する必要があります。

リポジトリオブジェクトのインストールと変更

表 7-1 は、Identity Manager の SPML を設定するためにインストールして変更する必要のあるリポジトリオブジェクトを示しています。

表 7-1 SPML の設定に使用されるリポジトリオブジェクト

オブジェクト	説明
`Configuration:SPML`	サーバーでサポートされている SPML スキーマの定義、および SPML スキーマと内部のビューモデルの間の変換のための規則が含まれています。各 SPML スキーマには一般に、関連付けられたフォームがあります。
SPML フォーム	SPML スキーマで定義された外部のモデルと、Identity Manager ビューで定義された内部のモデルの間で、変換の規則をカプセル化する 1 つ以上のフォームオブジェクトが含まれています。一般に、SPML スキーマで定義されたオブジェクトクラスごとに、1 つの SPML フォームが存在します。
`Configuration:User Extended Attributes`	SPML フィルタを介したアクセスのために Identity Manager リポジトリ内に格納できるユーザー属性を定義します。
`Configuration:UserUIConfig`	Identity Manager ユーザーオブジェクトのための追加のクエリー可能な属性や、概要の属性が含まれています。クエリー可能な属性は、SPML フィルタで使用するすべての属性に対して定義する必要があります。概要の属性は、最適化された検索で返す、すべての属性に対して定義する必要があります。
`TaskDefinition:SPMLRequest`	非同期 SPML 要求を処理するために使用されるシステムタスク。このオブジェクトをカスタマイズする必要はないはずです。

Identity Manager は、`sample/spml.xml` ファイルで、SPML 設定オブジェクトのサンプルのセットを提供しています。このファイルは、リポジトリの初期化時にデフォルトではインポートされないので、手動でインポートする必要があります。

このサンプル設定では、SPML ワーキンググループによって定義され、作成中の標準スキーマ person という名前のクラスを定義しています。このワーキンググループは、まだ標準のユーザースキーマを公開していませんが、それはほぼ間違いなく、一般的な LDAP inetorgperson スキーマに基づくものになります。

注 person クラスのカスタマイズは避けるようにしてください。代わりに、標準スキーマとの一貫性を維持してください。

注 SPE SPML インタフェースを設定するには、Configuration:SPE SPML 設定オブジェクトをインストールして変更する必要があります。

- person クラス (デフォルトで定義される唯一のオブジェクトクラス) を設定して、SPE 固有のビューハンドラ (IDMXUser) を使用します。
- form 属性を使用して、SPML 要求 / 応答とビューの間の変換を行うユーザーフォームを定義します。

  form 属性は、(view): という特別な値を取ることができます。ここでは、ビューに対してフォーム処理が何も適用されません (たとえば、view はクライアントと Identity Manager の間で直接渡される)。

SPE SPML インタフェースにアクセスするには、次の (デフォルト) パスを使用します。

/servlet/spespml

たとえば、localhost およびポート 8080 上の /idm コンテキスト内に Identity Manager を配備する場合、次の URL でインタフェースにアクセスできます。

http://localhost:8080/idm/servlet/spespml

waveset.properties ファイルの編集

表 7-2 は、SPML 要求の承認方法を制御するために使用できる、waveset.properties ファイル内の 3 つのオプションのエントリを示しています。

表 7-2 waveset.properties 内のオプションのエントリ

エントリ名	説明
soap.username	SPML 要求を実行するための実効ユーザーとして使用される Identity Manager ユーザーの名前

表 7-2　waveset.properties 内のオプションのエントリ (続き)

エントリ名	説明
soap.password	soap.username で指定されたユーザーのクリアテキストのパスワード。
soap.epassword	soap.username で指定されたユーザーの暗号化パスワードの base 64 表現。

soap.epassword および soap.password プロパティーの編集

soap.username で指定されたユーザーは、プロキシユーザーと呼ばれます。プロキシユーザーを定義する場合は soap.username を設定する必要がありますが、設定する必要があるのは 2 つのパスワードエントリのいずれかのみです。soap.password の使用がもっとも簡単ですが、この方法では、プロパティーファイル内にクリアテキストパスワードが公開されます。soap.epassword の使用の方が安全ですが、暗号化パスワードの生成に追加の手順が必要になります。

プロキシユーザーには Web サービスを使用するための認証が必要ないため、プロキシユーザーの確立はクライアントにとって便利です。これは、Identity Manager サーバーが、ユーザーの認証を自身で処理する別のアプリケーションからのみアクセスされるポータル環境では、一般的な設定です。

警告　サーバーが応答している HTTP ポートが一般にアクセス可能な場合、これは危険な設定になります。Identity Manager サーバーの URL を知っていて、SPML 要求を作成する方法を理解しているユーザーの場合、プロキシユーザーが実行できる任意の Identity Manager 操作を実行できます。

SPML 標準では、認証や承認を実行する方法が指定されていません。認証に関連する Web 標準規格はいくつか存在しますが、これらの標準が広範囲に使用されることは当面ありません。おそらく、認証に対するもっとも一般的な当面のアプローチは、アプリケーションとサーバーの間での SSL の使用に依存することです。SSL の設定方法を Identity Manager で要求することはできません。

プロキシユーザーも SSL も使用できない場合、Identity Manager では、クライアントがログインしたあとも以降の要求の認証に使用されるセッショントークンを維持できるようにする、SPML に対するベンダー固有の拡張がサポートされています。これを行うためのもっとも簡単な方法は、資格の指定、ログイン要求の実行、およびすべての SPML 要求内のセッショントークンの引き渡しをサポートする、SpmlClient クラスの拡張である LighthouseClient クラスを使用することです。

注 SPE SPML インタフェースは認証や承認をサポートしていません。SPE では、Identity Manager にアクセスしているクライアントはすでにアクセス管理アプリケーションによって認証および承認済みであることを前提にしています。SPE SPML インタフェースを使用するときには、クライアントはすべての可能な権限を持っています。

クライアントと Identity Manager の間で機密データが露呈されることを防ぐために、SSL を使用して SPE SPML インタフェースにアクセスすることを検討してみてください。

暗号化パスワードの取得

暗号化パスワードを取得するための 1 つの方法は、Identity Manager コンソールでの `encrypt` コマンドの使用です。別の方法は、「デバッグ」ページまたはコンソールから、XML のプロキシユーザーを表示することです。`password` 属性の値に対する `WSUser` 要素を調べます。この値を `soap.epassword` プロパティーの値として使用できます。

設定オブジェクトの編集

アプリケーションには、SPML メッセージを送信したり、SPML 応答を受信したりするためのメカニズムが必要です。

Identity Manager で SPML を設定するには、次の設定オブジェクトを操作する必要があります。

- `Configuration:SPML`
- `Configuration:User Extended Attributes`
- `Configuration:UserUIConfig`
- `TaskDefinition:SPMLRequest`
- `SPML Forms`

注 SPE SPML インタフェースには 1 つだけ設定オブジェクト (`Configuration:SPE SPML`) があります。このオブジェクトは `Configuration:SPML` オブジェクトに構造が似ています。

設定の編集 : SPML

SPML オブジェクトには、公開する SPML スキーマの定義と、これらの SPML スキーマが Identity Manager ビューにマップされる方法に関する情報が含まれています。この情報は、設定オブジェクトの拡張として格納されている `GenericObject` を使用して表されます。

この `GenericObject` で定義される属性には、`schemas` と `classes` の 2 つがあります。

- schemas: 各文字列に 1 つの SPML `<schema>` 要素のエスケープされた XML が含まれている、文字列のリスト。SPML 要素は `waveset.dtd` では定義されていないため、Identity Manager XML ドキュメントに直接含めることはできません。代わりに、エスケープされたテキストとして含める必要があります。
- Classes: サポートされている SPML クラスと、これらのクラスがビューにマップされる方法に関する情報を含むオブジェクトのリスト。このリストには、SPML スキーマの `schemas` リストで定義されているクラスごとに 1 つのオブジェクトが存在すべきです。

最初は、この 2 つのリストの区別がわかりにくいかもしれません。**schemas** リストに関する情報は、Identity Manager が SPML SchemaRequest メッセージに応答して何を返すかを定義します。クライアントがこの情報を使用すると、`AddRequest` などの、ほかのメッセージに含まれている可能性のある属性を理解できます。Identity Manager は、schemas リストの内容には関知しません。このリストは、単純にそのままクライアントに返されます。

SPML スキーマの定義は必須ではありません。Identity Manager は、スキーマがなくても機能します。SPML スキーマが定義されていない場合、Identity Manager は、スキーマ要求メッセージを受信すると空の応答を返します。スキーマがない場合、クライアントは、サポートされているクラスや属性についての既存の知識に依存する必要があります。これが一般的な状況ですが、それでもなお、要求の作成には OpenSPML Browser などの汎用のツールを使用できるように、SPML スキーマを記述することが良い方法であると考えられています。

デフォルトの SPML 設定

コード例 7-1 は、デフォルトの SPML 設定を示しています。簡潔にするために、SPML スキーマ定義のテキストは省略しています。

コード例 7-1 デフォルトの SPML 設定

```
<Configuration name='SPML'>
   <Extension>
      <Object>
         <Attribute name='classes'>
            <List>
               <Object name='person'>
```

コード例 7-1 デフォルトの SPML 設定 (続き)

```
                  <Attribute name='type' value='User'/>
                  <Attribute name='form' value='SPMLPerson'/>
                  <Attribute name='default' value='true'/>
                  <Attribute name='identifier' value='uid'/>
               </Object>
               <Object name='request'>
                  <Attribute name='type' value='TaskInstance'/>
                  <Attribute name='filter'>
                     <AttributeCondition attrName='defName' operator='equals'
                      operand='SPMLRequest'/>
                  </Attribute>
               </Object>
            </List>
         </Attribute>
         <Attribute name='schemas'>
            <List>
               <String>
                  <![CDATA[
                  <schema xmlns="urn:oasis:names:tc:SPML:1:0"
                  ...SPML standard schema...
                  </schema>
                  ]]>
               </String>
               <String>
                  <![CDATA[
                  <schema xmlns="urn:oasis:names:tc:SPML:1:0"
                  ...Waveset custom schema...
                  </schema>
                  ]]>
               </String>
            </List>
         </Attribute>
      </Object>
   </Extension>
</Configuration>
```

この例では、標準の `person` と、`request` という名前の Identity Manager 拡張の 2 つのクラスが定義されています。クラス定義では、次の属性がサポートされています。

- name: クラスの名前を特定します。この値は、SPML スキーマ内の `<ObjectClassDefinition>` 要素に対応している場合がありますが、これは必須ではありません。この名前は、追加要求または検索要求での `objectclass` 属性の値として使用されます。
- type: このクラスのインスタンスの管理に使用される Identity Manager のビュータイプを定義します。通常は、`User` ですが、ビューを介してアクセスできる任意のリポジトリタイプにすることができます。ビューについては、『Sun Java™ System Identity Manager ワークフロー、フォーム、およびビュー』を参照してください。

- form: フォームを含む設定オブジェクトの名前を特定します。このフォームには、このクラスで定義される外部の属性と内部のビュー属性の間での、変換のための規則が含まれています。
- default: `true` に設定されている場合は、これがこのタイプのみのデフォルトクラスであることを示します。同じタイプに対して複数の SPML クラスが実装されている場合は、その 1 つをデフォルトとして指定するようにしてください。
- identifier: 各クラスは一般に、そのオブジェクトのアイデンティティーと見なされる 1 つの属性を定義します。可能な場合は、この属性の値が、そのインスタンスを表すために作成する対応したリポジトリオブジェクトの名前として使用されます。クラス定義内の *identifier* 属性は、どの属性がアイデンティティーを表すかを指定します。
- filter: SPML 検索要求がクラスに対して評価される場合は、一般に、そのクラスに関連付けられたすべてのリポジトリオブジェクトをその検索に含めます。これはユーザーオブジェクトについては問題ありませんが、一部のクラスは、`TaskDefinition` や `Configuration` などの、必ずしも SPML クラスのインスタンスとは見なされない、汎用的なタイプを使用して実装される可能性があります。

  検索に不要なオブジェクトが含まれることを避けるために、filter 属性を指定できます。この値は、`<AttributeCondition>` 要素、または `<AttributeCondition>` 要素の `<List>` であると想定されます。カスタムクラスは、ほぼ常にユーザータイプのために作成されるため、フィルタを使用することは一般的ではありません。デフォルト設定では、カスタムクラスを使用して、非同期 SPML 要求の処理のために作成されたことが知られている `TaskInstance` オブジェクトのサブセットを公開します。

デフォルトのスキーマ

`schemas` 属性には、SPML `<schema>` 要素のエスケープされた XML を含む文字列のリストが含まれています。`spml.xml` ファイルを調べてみると、schema 要素が、CDATA でマークされたセクションで囲まれていることに気付きます。XML の長い文字列のエスケープには、この方法のほうが便利です。これを正規化すると、`&lt;` 文字エンティティーを含む文字列に変換されます。

デフォルト設定には、次の 2 つのスキーマが含まれます。

- SPML ワーキンググループで定義される標準スキーマ
- Identity Manager で定義されるカスタムスキーマ。これらのスキーマをカスタマイズしないでください。Identity Manager のスキーマには、**request** のためのクラス定義のほか、一般的なアカウント管理操作のための多数の拡張要求が含まれています。

設定の編集 : SPMLPerson オブジェクト

Configuration:SPML で定義されている各クラスには一般に、そのクラスで定義された外部の属性モデルと、関連付けられたビューで定義された内部のモデルの間での、変換の規則を含むフォームオブジェクトが関連付けられています。

標準の person クラスは、次のフォーム (コード例 7-2) を参照します。

コード例 7-2 標準の Person クラスでのフォームの参照

```
<Configuration name='SPMLPerson'>
   <Extension>
      <Form>

         <Field name='cn'>
            <Derivation><ref>global.fullname</ref></Derivation>
         </Field>
         <Field name='global.fullname'>
            <Expansion><ref>cn</ref></Expansion>
         </Field>

         <Field name='email'>
            <Derivation><ref>global.email</ref></Derivation>
         </Field>
         <Field name='global.email'>
            <Expansion><ref>email</ref></Expansion>
         </Field>

         <Field name='description'>
            <Derivation>
               <ref>accounts[Lighthouse].description</ref>
            </Derivation>
         </Field>
         <Field name='accounts[Lighthouse].description'>
            <Expansion><ref>description</ref></Expansion>
         </Field>

         <Field name='password'>
            <Derivation><ref>password.password</ref></Derivation>
         </Field>
         <Field name='password.password'>
            <Expansion><ref>password</ref></Expansion>
         </Field>

         <Field name='sn'>
            <Derivation><ref>global.lastname</ref></Derivation>
         </Field>
         <Field name='global.lastname'>
            <Expansion><ref>sn</ref></Expansion>
         </Field>
```

コード例 7-2 標準の Person クラスでのフォームの参照 (続き)

```
          <Field name='gn'>
             <Derivation><ref>global.firstname</ref></Derivation>
          </Field>
          <Field name='global.firstname'>
             <Expansion><ref>gn</ref></Expansion>
          </Field>
          <Field name='telephone'>
             <Derivation>
                <ref>accounts[Lighthouse].telephone</ref>
             </Derivation>
          </Field>
          <Field name='accounts[Lighthouse].telephone'>
             <Expansion><ref>telephone</ref></Expansion>
          </Field>
        </Form>
   </Extension>
</Configuration>
```

注 SPML クラスのフォームに <Display> 要素は含まれていません。これらのフォームは、対話型編集のためではなく、データ変換のためにのみ定義されます。

クラス定義内の属性ごとに、1 対のフィールド定義が存在します。1 つのフィールドは <Derivation> 式を使用して、内部のビュー属性 name を外部名に変換します。1 つのフィールドは <Expansion> 式を使用して、外部名を内部名に変換します。

このフォームは、属性がクライアントに返されるときは、<Derivation> 式の結果のみが含まれるという方法で処理されます。属性がクライアントからサーバーに送信されると、<Expansion> 式の結果のみがビューに反映されます。この効果は、リソース定義のスキーママップに似ています。

設定の編集 : ユーザー拡張属性オブジェクト

SPML 検索フィルタで使用する任意の属性を、Identity Manager ユーザーの拡張属性として定義する必要があります。これにより、その属性の値は、同時にリソースアカウント属性として格納される場合でも、Identity Manager リポジトリ内に格納されるようになります。拡張属性によって、リポジトリのサイズが増加するだけでなく、Identity Manager 内に格納された属性とリソース上に格納された属性の実際の値の間で、一貫性の問題が発生する可能性も増加するため、一般には、拡張属性の数は最小限に抑えるようにしてください。ただし、Identity Manager クエリーで使用される属性の場合は、リポジトリのクエリーインデックスが作成されたときにその値が常にアクセス可能になるように、拡張属性として宣言する必要があります。

ユーザーのための概要の属性の設定に含める任意の属性を、「拡張属性」として定義する必要があります。概要の属性を使用すると、オブジェクト XML のデシリアライズを回避することによって検索を最適化し、代わりにユーザーのもっとも重要な属性のいくつかだけを返すことができます。Identity Manager SPML の実装では、返される属性のリストを検索要求で明示的に指定しない場合は常に、概要の属性が返されます。

デフォルトの SPML 設定では、標準の `person` スキーマの属性 `telephone` と `description` が拡張属性として宣言されます。

コード例 7-3 拡張属性として宣言された telephone と description

```
<Configuration id='#ID#Configuration:UserExtendedAttributes' name='User Extended Attributes'>
   <Extension>
      <List>
         <!-- これは標準のセットである -->
         <String>firstname</String>
         <String>lastname</String>
         <String>fullname</String>
         <!-- これらは、SPML の拡張である -->
         <String>description</String>
         <String>telephone</String>
      </List>
   </Extension>
</Configuration>
```

属性のリストは、サイトのニーズに応じてカスタマイズできます。

拡張属性として選択する名前は、クラスフォームで実行されるマッピングによって異なります。デフォルトの SPMLPerson フォームによって `sn` が `lastname` にマップされるため、拡張属性を `lastname` として宣言する必要があります。このフォームでは `telephone` または `description` の名前は変換されないため、拡張属性の名前は SPML スキーマから直接採用されます。

拡張属性を宣言するだけでなく、Configuration:UserUIConfig オブジェクトも変更して、どの属性をクエリー可能 (つまり、SPML フィルタで使用可能) にし、どの属性を (最適化された検索の結果によって返される) 概要の属性にするかを宣言する必要があります。

設定の編集 : UserUIConfig オブジェクト

次の属性を宣言する必要があります。

- UserUIConfig オブジェクトの <QueryableAttrNames> セクション内の SPML 検索フィルタで使用する、任意の属性。
- <SummaryAttrNames> セクション内の最適化された SPML 検索の結果で返す、任意の属性。

コード例 7-4 は、拡張属性 telephone をクエリー可能な概要の属性として定義しています。また、firstname と lastname の宣言も含まれていますが、これらは通常、すでに宣言されています。これらのリストは、サイトのニーズに応じてカスタマイズできます。

コード例 7-4 クエリー可能な概要の属性として定義された telephone

```
<Object>
   <Attribute name='add'>
      <Object>
         <Attribute name='SummaryAttrNames'>
            <List>
               <!-- これらは通常ここに存在するが、確認すること -->
               <String>firstname</String>
               <String>lastname</String>
               <!-- これは SPML の追加である -->
               <String>telephone</String>
            </List>
         </Attribute>
         <Attribute name='QueryableAttrNames'>
            <List>
               <!-- これらは通常ここに存在するが、確認すること -->
               <String>firstname</String>
               <String>lastname</String>
               <!-- これは SPML の追加である -->
               <String>telephone</String>
            </List>
         </Attribute>
      </Object>
   </Attribute>
</Object>
```

TaskDefinition の編集 : SPMLRequest オブジェクト

spml.xml ファイルにもまた、SpmlRequest という名前の新しいシステムタスクの簡単な定義が含まれています。このタスクは、非同期 SPML 要求を実装するために使用されます。サーバーが非同期要求を受信すると、このタスクの新しいインスタンスが起動され、その SPML メッセージがタスクへの入力変数として渡されます。このタスクインスタンスのリポジトリ ID は、あとの状態要求に対する SPML 応答で返されます。

```
<TaskDefinition name='SPMLRequest'
   executor='com.waveset.rpc.SpmlExecutor'
   execMode='asyncImmediate'
   resultLimit='86400'>
</TaskDefinition>
```

定義の名前、executor の名前、および実行モードは変更しないでください。ただし、resultLimit の値は変更することができます。非同期要求が完了すると、クライアントが結果を取得するための SPML 状態要求を発行できるように、その結果は通常、一定期間保持されます。結果が保持される期間は、サイト固有の値です。

値が負でない場合は、resultLimit によって、タスクが完了したあとにシステムに結果が保持される時間 (秒単位) が指定されます。SPMLRequests のデフォルト値は通常、3600 秒 (約 1 時間) です。ほかのタスクは、そのタスクが別の値に変更されないかぎり、デフォルトで 0 秒になります。

負の場合、その要求インスタンスは自動的には削除されません。

ヒント リポジトリが乱雑にならないように、resultLimit の値はできるだけ短時間に設定してください。

注 SPE SPML インタフェースは非同期要求をサポートしていません。

配備記述子

Identity Manager の配備記述子 (通常、ファイル Web-INF/web.xml に含まれている) には、SOAP メッセージを受信するサーブレットの宣言が含まれている必要があります。

SPML Web サービスへの接続で問題が発生している場合は、web.xml ファイル内にある、コード例 7-5 に示すようなサーブレット宣言を探してください。

コード例 7-5 サーブレット宣言

```
<servlet>
   <servlet-name>rpcrouter2</servlet-name>
   <display-name>OpenSPML SOAP Router</display-name>
   <description>no description</description>
   <servlet-class>
      org.openspml.server.SOAPRouter
   </servlet-class>
   <init-param>
      <param-name>handlers</param-name>
      <param-value>com.waveset.rpc.SimpleRpcHandler</param-value>
   </init-param>
   <init-param>
      <param-name>spmlHandler</param-name>
      <param-value>com.waveset.rpc.SpmlHandler</param-value>
   </init-param>
   <init-param>
      <param-name>rpcHandler</param-name>
      <param-value>com.waveset.rpc.RemoteSessionHandler</param-value>
   </init-param>
</servlet>
```

この宣言を使用すると、次の URL を介して `addRequest`、`modifyRequest`、および `searchRequest` Web サービスにアクセスできます。

`http://<host>:<port>/idm/servlet/rpcrouter2`

<servlet-mapping> を定義する必要はありません (ただし、定義することは可能)。このサーブレット宣言の内容を変更しないでください。

要求の処理方法について

ここでは、Identity Manager での SPML 要求の処理方法の一般的な概要について説明します。

追加要求の処理方法

次の手順は、追加要求の処理方法について説明しています。

1. SPML <addRequest> メッセージが受信されます。この要求には、objectclass 属性の値が含まれている必要があります。
2. サーバーは、Configuration:SPML オブジェクトを検査してクラスの定義を見つけます。このクラス定義から、サーバーは、関連付けられたビュータイプとフォーム名を取得します。
3. サーバーは、Session.createView メソッドを呼び出して、そのタイプの新しいビューを作成します。
4. その要求に含まれている属性がクラスフォームによって処理されます。<Expansion> 式の結果がビューに反映されます。
5. このビューがチェックインされます。

変更要求の処理方法

次の手順は、変更要求の処理方法について説明しています。

1. SPML <modifyRequest> メッセージが受信されます。この要求には、オプションの *objectclass* 属性が含まれている場合があります。この要求には、既存のオブジェクトの識別子が含まれている必要があります。この識別子には、リポジトリタイプとオブジェクト名の両方が含まれている必要があります。
2. サーバーは、既存のオブジェクトの Session.checkoutView を呼び出します。
3. サーバーは、Configuration:SPML オブジェクトを検査してクラスの定義を見つけます。要求で objectclass 属性が渡された場合は、その値によってクラスが決定されます。それ以外の場合は、リポジトリタイプのデフォルトクラスとしてマークされたクラスが使用されます。
4. その要求に含まれている属性が、クラス定義で指定されたフォームによって処理されます。<Expansion> 式の結果がビューに反映されます。
5. このビューがチェックインされます。

検索要求の処理方法

次の手順は、検索要求の処理方法について説明しています。

1. SPML <searchRequest> メッセージが受信されます。この要求には、オプションの *objectclass* 属性が含まれている場合があります。
2. サーバーは、Configuration:SPML オブジェクトを検査してクラスの定義を見つけます。要求で objectclass 属性が渡された場合は、その値によってクラスが決定されます。それ以外の場合は、ユーザータイプのデフォルトクラスとしてマークされたクラスが使用されます。
3. この要求にフィルタが含まれている場合、そのフィルタは AttributeCondition オブジェクトのリストに変換されます。フィルタの用語は外部名を使用して記述されているため、これらの名前を、リポジトリタイプに対してクエリー可能な属性の名前に変換するためにクラスフォームが参照されます。
4. サーバーは、リポジトリタイプとオプションの条件を使用して Session.listObjects メソッドを呼び出します。
5. サーバーは、次の手順を適用しながら、listObjects 呼び出しの各行に対して処理を繰り返すことによって、検索応答を作成します。
6. 返される属性のリストが検索で指定されていない場合は、リポジトリタイプに対して定義された概要の属性のみが返されます。概要の属性の内部名を外部名に変換するために、クラスフォームが使用されます。
7. 返される属性のリストが指定されていて、これらがすべて概要の属性に対応している場合は、概要の属性の値が返されます。ここでも、内部名を外部名に変換するために、フォームが使用されます。
8. 概要の属性ではない、返される属性が指定されている場合、サーバーは、このオブジェクトに対して Session.getView を呼び出してビューを生成します。このビューがクラスフォームを使用して処理されたあと、<Derivation> 式の結果が取得され、その行の結果として返されます。

Identity Manager は、最初、タイプに対して定義された概要の属性のみを使用して、検索要求を満たそうとします。その結果、特にフィルタが指定されておらず、結果に多数のオブジェクトが含まれる場合、検索がはるかに高速になります。検索を実行するためにビューをインスタンス化する必要がある場合は、検索の速度が大幅に低下するため、結果を少数のオブジェクトに制限するためのフィルタを指定する場合にのみ、検索を実行するようにしてください。

オブジェクトのアイデンティティーがわかっていて、フィルタ式を記述することなくその属性を取得したい場合は、検索要求でそのアイデンティティーを *baseContext* の値として使用します。これによって、Identity Manager がクエリーを回避してビューを作成するだけで済むため、検索が高速になります。

返される属性を指定しない場合は、概要の属性のみが返されます。そのため、すべての属性が必要な場合は、返される属性のリストにそれらの属性を明示的に含める必要があります。使用可能なすべての属性を要求することが一般的な操作であるため、これは多少不便です。属性の完全なリストを指定する代わりの方法として、Identity Manager は、結果を生成するためにオブジェクトビューを完全にインスタンス化し、クラスフォームを使用して処理すべきであることを示す、`view` という名前の 1 つの返される属性を認識します。

また、属性のレベル (`accounts[Lighthouse]` または `accountinfo` など) を指定することもできます。レベルを指定した場合、Identity Manager は、完全にインスタンス化されたビューの、そのレベルの適用範囲内にあるすべての属性値を返します。

SPML ブラウザの起動

OpenSPML Browser アプリケーションを使用して Identity Manager SPML 設定をテストできます。

コマンド行からブラウザを起動するには、次のコマンドを入力します。

```
lh spml
```

注 `lh spml` コマンドの使用の詳細については、『Sun Java™ System Identity Manager 管理ガイド』を参照してください。

Identity Manager サーバーへの接続

Identity Manager サーバーに接続するには、**「接続」**ページを開き、Identity Manager サーバーの URL を入力します。たとえば、サーバーがローカルマシンのポート 8080 上で実行されている場合、URL は次のようになります。

```
http://localhost:8080/idm/servlet/rpcrouter2
```

ここで、`localhost` は Identity Manager を実行しているマシンです。

SPML 設定のテストとトラブルシューティング

SPML 設定をテストするには、次の手順に従います。

1. **「接続」**ページを開き、**「テスト」**をクリックします。

   接続が成功したことを示すダイアログが表示されます。

2. **「スキーマ」**ページを開き、**「送信」**をクリックします。

   Identity Manager サーバーでサポートされているスキーマのツリー表示が表示されます。

正常な接続を確立できない場合は、次の操作を行います。

- 入力した URL を二重にチェックします。
- 受信したエラーに「応答なし」や「接続が拒否されました」などの語句が含まれている場合、問題としてもっとも可能性が高いのは、接続 URL で使用されているホストまたはポートです。
- エラーによって、接続は確立されたが、Web アプリケーションまたはサーブレットが見つからなかったことが示されている場合、問題としてもっとも可能性の高いのは、`Web-INF/web.xml` ファイルです。詳細については、252 ページの「配備記述子」を参照してください。

SPML アプリケーションの開発

サーバーを設定したら、アプリケーションには、SPML メッセージを送信したり、SPML 応答を受信したりするメカニズムが必要になります。Java アプリケーションの場合、これを行うためのもっとも簡単な方法は OpenSPML ツールキットの使用です。このツールキットは `www.openspml.org` から入手でき、Identity Manager にもバンドルされています。

このツールキットでは、次のコンポーネントが提供されます。

- SPML メッセージのための Java クラスモデル
- クライアントでメッセージを送受信するためのクラス
- サーバーで要求を受信し、処理するためのクラス

表 7-3 は、このツールキットで提供される、もっとも重要なクラスの概要を示しています。要求の種類ごとに、対応するクラスが存在します。詳細については、ツールキットとともに配布されている JavaDoc を参照してください。

表 7-3　OpenSPML ツールキットで提供されるクラス

クラス	説明
`AddRequest`	新しいオブジェクトの作成を要求するメッセージを作成します。 作成するオブジェクトのタイプは、`objectclass` という名前の属性を渡すことによって定義されます。渡されるほかの属性は、このオブジェクトクラスに関連付けられたスキーマに従っているべきです。SPML では、標準スキーマがまだ定義されていません。必要な任意のスキーマをサポートするように Identity Manager を設定できます。
`ModifyRequest`	既存のオブジェクトの変更を要求するメッセージを作成します。この要求には、変更する属性のみを含める必要があります。要求に含まれていない属性は、現在の値を維持します。
`DeleteRequest`	オブジェクトの削除を要求するメッセージを作成します。
`SearchRequest`	特定の条件に一致する、オブジェクトの属性を要求するメッセージを作成します。
`BatchRequest`	複数の SPML 要求を含むことができるメッセージを作成します。
`CancelRequest`	以前の非同期に実行された要求を取り消すメッセージを作成します。
`SchemaRequest`	サーバーでサポートされている、SPML オブジェクトクラスに関する情報を要求するメッセージを作成します。
`StatusRequest`	以前の非同期に実行された要求のステータスを要求するメッセージを作成します。
`SpmlResponse`	サーバーから送り返された応答メッセージを表す、オブジェクトの基底クラス。各要求クラスには、対応する応答クラス (たとえば、`AddResponse` や `ModifyResponse`) があります。
`SpmlClient`	SPML メッセージを送受信するための単純なインタフェースを提供します。

注

SPE REF キットには、SPE SPML インタフェースの使用方法を実演する `SpmlUsage.java` ファイルが含まれています。この REF キットには `SpmlUsage` クラスをコンパイルする `ant` スクリプトも含まれています。

使用方法:

```
java [ -Dtrace=true ] com.sun.idm.idmx.example.SpmlUsage [ URL ]
```

ここで `URL` は SPE SPML インタフェースをポイントしており、デフォルトでは次のようになります。

```
http://localhost:8080/idm/spespml
```

SPE のトレースを有効にすると、すべての SPE SPML メッセージが標準出力に出力されます。

ExtendedRequest の例

表 7-4 は、クライアントでメッセージを送受信するために使用される ExtendedRequest クラスを示しています。

表 7-4 メッセージを送受信するための ExtendedRequest クラス

ExtendedRequest	説明
`deleteUser`	ユーザーの削除を要求するメッセージを作成します。
`disableUser`	ユーザーの無効化を要求するメッセージを作成します。
`enableUser`	ユーザーの有効化を要求するメッセージを作成します。
`resetUserPassword`	ユーザーパスワードのリセットを要求するメッセージを作成します。
`changeUserPassword`	ユーザーパスワードの変更を要求するメッセージを作成します。
`launchProcess`	プロセスの起動を要求するメッセージを作成します。
`listResourceobjects`	Identity Manager リポジトリ内のリソースオブジェクトの名前と、そのリソースでサポートされているオブジェクトのタイプを要求するメッセージを作成します。この要求では、名前のリストが返されます。
`runForm`	Identity Manager Session API を呼び出すことによって取得される情報を返す、カスタム SPML 要求を作成できるようにします。

サーバーコードは、これらの拡張要求をビュー操作に変換します。

サンプルの拡張要求

拡張要求は通常、コード例 7-6 に示す形式を取ります。

コード例 7-6　　拡張要求の形式

```
ExtendedRequest req = new ExtendedRequest();
req.setOperationIdentifier("changeUserPassword");
req.setAttribute("accountId", "jlarson");
req.setAttribute("password", "xyzzy");
req.setAttribute("accounts","Lighthouse,LDAP,RACF");
ExtendedResponse res = (ExtendedResponse) client.send(req);
```

ほとんどの SPML 拡張要求は、次の引数を取ります。

- `accountId` ― Identity Manager ユーザー名を特定します。
- `accounts` ― リソース名をコンマ区切りのリストにします。

`accounts` 属性を渡さない場合は、この操作によって、そのユーザーにリンクされたすべてのリソースアカウント (そのユーザー自身を含む) が更新されます。`accounts` を渡す場合は、この操作によって、指定したリソースのみが更新されます。特定のリソースアカウントに加えて Identity Manager ユーザーを更新する場合は、null 以外のアカウントリストに `Lighthouse` を含める必要があります。

deleteUser

コード例 7-7 は、`disableUser` 要求の標準的な形式を示しています。
(ビュー ― 「プロビジョン解除」ビュー)。

コード例 7-7　　deleteUser 要求

```
ExtendedRequest req = new ExtendedRequest();
req.setOperationIdentifier("deleteUser");
req.setAttribute("accountId", "jlarson");
req.setAttribute("accounts", "xyzzy");
req.setAttribute("accounts","Lighthouse,LDAP,RACF");
ExtendedResponse res = (ExtendedResponse) client.send(req);
```

disableUser

コード例 7-8 は、disableUser 要求の標準的な形式を示しています。(ビュー ー 「無効化」 ビュー)。

コード例 7-8　　disableUser 要求

```
ExtendedRequest req = new ExtendedRequest();
req.setOperationIdentifier("disableUser");
req.setAttribute("accountId", "jlarson");
req.setAttribute("accounts", "xyzzy");
req.setAttribute("accounts","Lighthouse,LDAP,RACF");
ExtendedResponse res = (ExtendedResponse) client.send(req);
```

enableUser

コード例 7-9 は、enableUser 要求の標準的な形式を示しています。(ビュー ー 「有効化」 ビュー)。

コード例 7-9　　enableUser 要求

```
ExtendedRequest req = new ExtendedRequest();
req.setOperationIdentifier("enableUser");
req.setAttribute("accountId", "jlarson");
req.setAttribute("accounts", "xyzzy");
req.setAttribute("accounts","Lighthouse,LDAP,RACF");
ExtendedResponse res = (ExtendedResponse) client.send(req);
```

resetUserPassword

コード例 7-10 は、resetUser 要求の標準的な形式を示しています。(ビュー ー 「ユーザーパスワードのリセット」 ビュー)。

コード例 7-10　　resetUser 要求

```
ExtendedRequest req = new ExtendedRequest();
req.setOperationIdentifier("resetUserPassword");
req.setAttribute("accountId", "jlarson");
req.setAttribute("accounts", "xyzzy");
req.setAttribute("accounts","Lighthouse,LDAP,RACF");
ExtendedResponse res = (ExtendedResponse) client.send(req);
```

changeUserPassword

コード例 7-11 は、changeUserPassword 要求の標準的な形式を示しています。(ビュー ー 「ユーザーパスワードの変更」 ビュー)。

コード例 7-11　　changeUserPassword 要求

```
ExtendedRequest req = new ExtendedRequest();
req.setOperationIdentifier("changeUserPassword");
req.setAttribute("accountId", "jlarson");
req.setAttribute("password", "xyzzy");
req.setAttribute("accounts","Lighthouse,LDAP,RACF");
ExtendedResponse res = (ExtendedResponse) client.send(req);
```

launchProcess

コード例 7-12 は、launchProcess 要求の標準的な形式を示しています。(ビュー ー 「プロセス」 ビュー)。

コード例 7-12　　launchProcess 要求

```
ExtendedRequest req = new ExtendedRequest();
req.setOperationIdentifier("launchProcess");
req.setAttribute("process", "my custom process");
req.setAttribute("taskName", "my task instance");
ExtendedResponse res = (ExtendedResponse) client.send(req);
```

各表記の意味は次のとおりです。

- Process ー 起動するワークフローの名前
- taskName ー 任意の TaskName

process 属性は、実行対象の、Identity Manager リポジトリ内のタスク定義オブジェクトを指定します。taskName 属性は、プロセス実行時の状態の保持のために作成される、タスクインスタンスオブジェクトの指定に使用されます。残りの属性は任意であり、タスクに渡されます。launchProcess 要求を使用すると、任意のカスタムプロセスを起動できます。

listResourceObjects

コード例 7-13 は、listResourceObjects 要求の標準的な形式を示しています。

コード例 7-13　　listResourceObjects 要求

```
ExtendedRequest req = new ExtendedRequest();
req.setOperationIdentifier("listResourceObjects");
req.setAttribute("resource", "LDAP");
req.setAttribute("type", "group");
ExtendedResponse res = (ExtendedResponse) client.send(req);
```

各表記の意味は次のとおりです。

- resource: Identity Manager リポジトリ内のリソースオブジェクトの名前を指定します
- type: そのリソースでサポートされているオブジェクトのタイプを指定します

runForm

コード例 7-14 は、runForm 要求の一般的な形式を示しています。

コード例 7-14　　runForm 要求

```
ExtendedRequest req = new ExtendedRequest();
req.setOperationIdentifier("runForm");
req.setAttribute("form", "SPML Get Object Names");
ExtendedResponse res = (ExtendedResponse) client.send(req);
```

ここで、form は、フォームを含む設定オブジェクトの名前です。

フォームの例

コード例 7-15 に示すフォームは、クエリーを実行し、現在のユーザーにアクセス可能なロール、リソース、および組織名のリストを返します。

コード例 7-15　　クエリーフォーム

```
<Configuration name='SPML Get Object Names'>
  <Extension>
    <Form>
      <Field name='roles'>
        <Derivation>
          <invoke name='com.waveset.ui.FormUtil'>
```

コード例 7-15 クエリーフォーム (続き)

```
            <ref>display.session</ref>
            <s>Role</s>
          </invoke>
        </Derivation>
      </Field>
      <Field name='resources'>
        <Derivation>
          <invoke name='com.waveset.ui.FormUtil'>
            <ref>display.session</ref>
            <s>Resource</s>
          </invoke>
        </Derivation>
      </Field>
      <Field name='organizations'>
        <Derivation>
          <invoke name='com.waveset.ui.FormUtil'>
            <ref>display.session</ref>
            <s>ObjectGroup</s>
          </invoke>
        </Derivation>
      </Field>
    </Form>
  </Extension>
</Configuration>
```

runForm 要求を使用すると、Identity Manager Session API を呼び出すことによって、取得される情報を返すカスタム SPML 要求を作成できます。たとえば、ユーザーを編集するためのユーザーインタフェースを設定する場合は、ユーザーに割り当てることのできる組織、ロールリソース、およびポリシーの名前を表示するセレクタの提供が必要になることがあります。これらのオブジェクトを SPML オブジェクトクラスとして公開するように SPML インタフェースを設定したあと、searchRequest を使用してそれらの名前をクエリーできます。ただし、それには、情報を収集するために 4 つの searchRequests が必要になります。代わりに、これらのクエリーをフォーム内にコード化したあと、単一の runForm 要求を使用してクエリーを実行し、結合された結果を返すことによって SPML 要求の数を減らすことができます。

SPML でのトレースの使用

Identity Manager の SPML トラフィックをロギングし、問題の診断に役立てることができるように、SPML では、次のようなトレース出力を有効にするためのオプションが提供されています。

メソッド 1

`SpmlClient` および `LighthouseClient` クラスは、ブール型の引数を取る `setTrace` メソッドを提供しています。この `setTrace` メソッドを有効にすると、クライアントによって送信された要求の XML や、サーバーから受信された応答の XML は、それらの送受信時にクライアントのコンソールに出力されます。たとえば、次のようにします。

```
SpmlClient client = new SpmlClient();
client.setURL("http://www.company.com/idm/spml");
client.setTrace(true);
```

メソッド 2

個別の SPML RPC 要求に対するトレースを有効にするために、`trace` というオペレーショナル属性をサーバー側の RPC 要求に渡すことができます。

トレースはサーブレットの初期化中に行われ、SPMLv2 要求を処理するサーブレットの RPC トラフィックに関する情報の出力方法を制御します。たとえば、トレースは、そのサーブレットについてどのような `System.out` (アプリケーションコンテナの関数) が指定されていても、そこでやり取りされる生の XML を出力します。たとえば、次のようにします。

```
AddRequest ar = new AddRequest();
ar.setOperationalAttribute("trace", "true");
```

`trace` 属性の使用方法がサーバー操作に与える影響はベンダー固有です。現在、Identity Manager は生の要求および応答データをサーバーコンソールに出力しています。これは、クライアントアプリケーションがコンソールウィンドウと関連付けられていない場合に役立ちます。

Identity Manager にはまだコードが実装されていないので、詳細については、OpenSPML ツールキットの製品マニュアルを参照してください。

例

ここでは、SPML を実装するためのいくつかの一般的なメソッドを示すために、次の例について説明します。

- 追加要求
- 変更要求
- 検索要求

追加要求

追加要求の例をコード例 7-16 に示します。

コード例 7-16 追加要求

```
SpmlClient client = new SpmlClient();
   client.setURL("http://www.company.com/idm/spml");

   AddRequest req = new AddRequest();

   req.setObjectClass("person");
   req.setIdentifier("maurelius");
   req.setAttribute("gn", "Marcus");
   req.setAttribute("sn", "Aurelius");
   req.setAttribute("email", "maurelius@example.com");

   SpmlResponse res = client.request(req);

   if (res.getResult() .equals(SpmlResponse.RESULT_SUCCESS))
      System.out.println("Person was successfully created");
```

変更要求

ここでは、認証された SPML 変更要求の 2 つの例を示します。

これらの例の唯一の違いは、コード例 7-18 では `LighthouseClient` クラス、および `client.setUser` と `client.setPassword` への 2 つの追加のメソッド呼び出しを使用している点です。

コード例 7-17　　認証された SPML 要求

```
SpmlClient client = new SpmlClient();
   client.setURL("http://www.company.com/idm/spml");
   ModifyRequest req = new ModifyRequest();
   req.setIdentifier("maurelius");
   req.setModification("email", "marcus.aurelius@example.com");
   SpmlResponse res = client.request(req);
   if (res.getResult() .equals(SpmlResponse.RESULT_SUCCESS))
      System.out.println("Person was successfully modified");
```

コード例 7-18　　LighthouseClient を使用して認証された SPML 要求

```
LighthouseClient client = new LighthouseClient();
   client.setURL("http://www.company.com/idm/spml");
   client.setUser("maurelius");
   client.setPassword("xyzzy");
   ModifyRequest req = new ModifyRequest();
   req.setIdentifier("maurelius");
   req.setModification("email", "marcus.aurelius@example.com");
   SpmlResponse res = client.request(req);
   if (res.getResult() .equals(SpmlResponse.RESULT_SUCCESS))
      System.out.println("Person was successfully modified");
```

検索要求

検索要求の例をコード例 7-19 に示します。

コード例 7-19　　検索要求

```
SpmlClient client = new SpmlClient();
   client.setURL("http://www.company.com/idm/spml");
   SearchRequst req = new SearchRequest();
   // 返す属性を指定する
   req.addAttribute("sn");
   req.addAttribute("email");
   // フィルタを指定する
   FilterTerm ft = new FilterTerm();
   ft.setOperation(FilterTerm.OP_EQUAL);
   ft.setName("gn");
   ft.setValue("Jeff");
   req.addFilter(ft);
```

コード例 7-19　検索要求 (続き)

```
SearchResponse res = (SearchResponse)client.request(req);
// 結果を表示する
List results = res.getResults();
if (results != null)  {
   for (int i = 0 ; i < results.size() ; i++) {
      SearchResult sr = (SearchResult)results.get(i);
      System.out.println("Identifier=" +
                         sr.getIdentifierString() +
                         " sn=" +
                         sr.getAttribute("sn") +
                         " email=" +
                         sr.getAttribute("email"));
      }
}
```

第 8 章

Identity Manager Web サービスでの SPML 2.0 の使用

この章では、Identity Manager 7.1 でサポートされる SPML 2.0 について説明しています。これには、サポートされる機能とその理由、SPML 2.0 サポートの設定方法、フィールドでのサポートの拡張方法が含まれます。

注 この章では、SPML 2.0 のみを扱います。特に明記されていないかぎり、この章での SPML への参照はすべて version *2.0* を示しています。

SPML の使用方法について、役立つ情報が含まれている、第 7 章「Identity Manager Web サービスでの SPML 1.0 の使用」も読まれることをお勧めします。

この情報は、次のように構成されています。

この章の対象読者

アプリケーション開発者および Identity Manager の統合を担当する開発者は、この章で説明されている SPML クラスを使用して、サービスプロビジョニング要求メッセージをフォーマットしたり、応答メッセージを解析したりすることができます。

概要

Identity Manager の Web サービスは SOAP ベースのプログラムインタフェースであり、Identity Manager はこのインタフェースを介して、特定のオペレーティングシステムやプログラミング言語に制限されることなく、ほかの Web ベースアプリケーションと通信できます。

Web サービスインタフェースは Web サーバー上でホストされ、その機能は、次のオープンな標準規格を通してインターネットプロトコルフレームワーク上で公開されます。

- XML: データのタグ付けに使用される
- SOAP: インターネット上でデータを符号化して転送するために使用される
- WSDL: 使用可能なサービスを記述するために使用される
- UDDI: 使用可能なサービスを登録および検索するために使用される

Identity Manager の Web サービスには、SOAP メッセージを使用して HTTP 接続経由でアクセスできます。

SPML 2.0 と SPML 1.0 の比較

Identity Manager の Web サービスは、プロビジョニングシステムとの通信のために、XML を使用したサービスプロビジョニングのためのオープンな標準規格である SPML version 1.0 および version 2.0 の両方のプロトコルをサポートします。

注 Identity Manager での SPML version 1.0 の使用方法の詳細は、第 7 章「Identity Manager Web サービスでの SPML 1.0 の使用」を参照してください。

SPML 2.0 は SPML 1.0 と比較して、次を含む、多くの点が改善されています。

- SPML 1.0 は DSML を多少改良したものと考えられていましたが、SPML 2.0 は、XML Schema プロファイルに加えて DSML プロファイルもサポートする拡張可能なプロトコルを、一連の機能を通じて定義しています。SPML 2.0 は、プロトコル自体と、そのプロトコルによって伝送されるデータを区別しています。
- SPML 2.0 のプロバイダは複数のターゲット (終端) を提供できます。
- SPML 2.0 プロトコルでは、特に 1.0 に存在するコア機能に関して、ベンダー間の相互運用性の向上が実現しています。

SPML 1.0 は `ExtendedRequest` を使用して「拡張」できますが、要求をどのように拡張できるかについてのガイダンスはありません。SPML 2.0 では、十分に定義された方法でサポートを追加できる、「標準機能」のセットが定義されています。

- SPML 2.0 では、プログレッシブサポートを有効にする追加機能が提供されています (表 8-1 を参照)。

表 8-1 SPML の機能

SPML 1.0	SPML 2.0
Add	Add
Modify	Modify
削除	削除
Lookup	Lookup
SchemaRequest	ListTargets
Search	「標準」機能としての Search (このリリースでは未サポート)
ExtendedRequest	「標準」機能で捕捉 : • Async: 要求の非同期処理 • Batch: 要求の一括処理 • BulK: 繰り返し処理を使用したプロセスの変更または削除 • Password: パスワードの変更、設定、リセット、検証、または失効処理 • Reference: ターゲット間での PSO の参照 • Search: PSO の検索 • Suspend: PSO の有効化または無効化 • Update: 更新されたオブジェクトの変更レコードの検索 (「カスタム」機能での捕捉も可能)

SPML 2.0 の概念の Identity Manager へのマッピング

SPML 2.0 では、プロビジョニングシステムによって管理されるオブジェクトの説明で、PSO や psoID などの定義用語をはじめとした、独自の用語が使用されています。この節では、これらの概念が Identity Manager にどのようにマップされるかについて説明します。

ターゲット

ターゲットは、サーバー内の論理終端です。各ターゲットには名前が付けられ、そのターゲットが管理するオブジェクト (次の「PSO」を参照) のスキーマを宣言します。ターゲットはサポートされる機能 (要求のセット) も宣言します。

現時点で、Identity Manager では 1 つのターゲットのみがサポートされており、複数のターゲットを宣言することはできません。このターゲットには任意の名前を付けることができますが、データオブジェクトの形式は DSML プロファイルに適合している必要があります。

サポートされるターゲットは、`spml2.xml` ファイル (`Configuration:SPML2` オブジェクト) で定義されているターゲットです。たとえば、279 ページのコード例 8-6 で、`ListTargetResponse` は 1 つのターゲット `spml2-DSML-Target` を返します。

PSO

前の項で説明したように、ターゲットは PSO を管理します。PSO (プロビジョニングサービスオブジェクト) は Identity Manager のビューに似ていますが、動作を持っていません。つまり、PSO は Identity Manager のビュー (特にユーザービュー) のデータ部分として考えることができます。

注	このリリースの時点で、Identity Manager はユーザーのみを管理し、*`objectclass`* という `UserExtendedAttribute` の定義を要求します。

Identity Manager の目的として、PSO は、フォームを介してユーザービューとの間でマップされる属性のコレクションになります。各オブジェクトは `objectclass` 属性を指定します。この属性は、ターゲットに対して定義されるスキーマ内の `objectclass` 定義に、オブジェクトをマップするために使用されます。この属性はその後、`repoType` の検索や、Identity Manager のビューとの間で属性をマップするフォームの検索に使用されます。

PSOIdentifier

SPML には、*PsoID* と呼ばれるオブジェクト ID が存在します。

仕様では PSOIdentifier (PsoID) を要求元 (クライアント) からは隠すことが推奨されているため、PSO をシステムに追加するとき、Identity Manager はそのリポジトリ ID (repoID) を PsoID として使用します。

repoID は識別用の ID であり、ユーザーに対する提示は想定されていません。要求元がユーザーに PSO を表示するとき、要求元はオブジェクトの ID を提示する目的で、同等の `waveset.accountid` (または、Identity テンプレート内で属性が使用されているもの) を使用するようにします。

(`ModifyRequest` などで) PSO を識別するとき、要求元は `waveset.accountId` ではなく repoID を使用するようにします。要求元は `waveset.accountId` を PSOIdentifier として使用することもできますが、推奨されていません。この属性は将来のリリースで変更される可能性があります。要求元は PsoID の不透明性を、できるかぎり維持することが推奨されます。

PSO では、`objectclass` 属性を使用してオブジェクトタイプを指定します。Identity Manager では、この属性は `UserExtendedAttribute` である必要があります。Identity Manager では、要求が行われたときに、この属性が存在しない場合に使用するための「デフォルト」を指定できます。SPML 2.0 を有効にする以前から存在していたユーザーについては、`objectclass` 属性を見つけられない可能性があります。

オープンコンテンツとオペレーショナル属性

SPML の `.xsd` ファイルでは、仕様の中でオープンコンテンツとして定義されている要素を識別するために、`xsd:any` が頻繁に使用されています。SPML でのオープンコンテンツとは、ほとんどの要素が任意のタイプの要素を含むことができるという意味です。Identity Manager ではこの概念を利用して、処理を制御する *OperationalNVPs* (`NameValuePairs`) および *OperationalAttributes* を提供しています。OperationalNVPs は XML 内の要素として出現する一方で、オペレーショナル属性は属性として出現します。詳細は、OpenSPML 2.0 Toolkit (`http://www.openspml.org`) を参照してください。

OperationalNVPs およびオペレーショナル属性については、「サポートされる SPML 2.0 の機能」の節で詳しく説明します。ただし、`ListTargets` を除くすべての要求およびすべての応答で、使用する NVP は 1 つです。Identity Manager は、`session` という OperationalNVP に `sessionToken` を格納します。これにより、システムはユーザーの代わりに自動的にセッションをキャッシュして、処理効率を改善できます。

サポートされる SPML 2.0 の機能

Identity Manager 7.0 では DSML プロファイルを使用して、SPML 2.0 仕様のすべてのコア機能をサポートします。Identity Manager は、Batch や Async などの一部のオプション標準機能もサポートし、Bulk などの一部の標準機能については部分的にサポートします。

この節では、Identity Manager 7.0 でサポートされている SPML 2.0 の機能、Identity Manager で意図的に加えられた仕様およびプロファイル文書との相違点、および、Identity Manager で必須のオペレーショナル属性について説明します。

サポートされるコア機能

Identity Manager は、次のコア機能をサポートしています。

表 8-2 コア機能

機能	説明	オペレーショナル属性	相違点
AddRequest	指定された PSO をシステムに追加します。	なし	Identity Manager は正式には 1 つのターゲットのみをサポートします。
DeleteRequest	指定された PSO をシステムから削除します。	なし	Identity Manager は正式には 1 つのターゲットのみをサポートします。

表 8-2　コア機能 (続き)

機能	説明	オペレーショナル属性	相違点
ListTargetsRequest	Identity Manager を通して利用可能なターゲットをリストします。	• username: ユーザーの名前を指定します。 • password: セッションを確立するために使用するパスワードを指定します。	• Identity Manager は正式には 1 つのターゲットをサポートします。 • Identity Manager では、対話での最初の呼び出しに listTargets を使用する必要はありません。ただし、この要求に対する operationalAttributes で、サーバーとのセッションを確立するためのユーザー名とパスワードの組を指定することは可能です (Waveset.properties も使用できる)。 一般に、ログインしてセッショントークンを使用するほうが、より効率的です。Identity Manager では、この目的のための SessionAwareSpml2Client というクラスが提供されています。
LookupRequest	名前付き PSO の属性を検索して返します。	なし	なし
ModifyRequest	指定された PSO 属性を変更します。	なし	メインの SPML 2.0 仕様と DSML Profile 仕様の間の相違が原因で、Identity Manager は select (および component など) をサポートしません。その代わりに、Identity Manager は DSML Profile に従って、DSML の変更モードおよび要素を使用します。

注 一般的な相違点には、次のものがあります。

- Identity Manager は DSML Profile のみをサポートします。
- Identity Manager では、対話での最初の呼び出しが `listTargets` である必要はありません。ただし、この要求に対する `operationalAttributes` を使用して、サーバーとのセッションを確立するための `username/password` の組を指定することが可能です (`Waveset.properties` も使用できる)。

`AddRequest` および `ListTargetRequest` の例を次に示します。

AddRequest の例

ここでは、AddRequest の例をいくつか示します。

コード例 8-1 は、Identity Manager の `SessionAwareSpml2Client` クラスを通して `ListTargetsRequest` を呼び出す `.jsp` です。

コード例 8-1 クライアントコードの例

```
<%@page contentType="text/html"%>
<%@page import="org.openspml.v2.client.*,
                com.sun.idm.rpc.spml2.SessionAwareSpml2Client"%
<%@page import="org.openspml.v2.profiles.dsml.*"%>
<%@page import="org.openspml.v2.profiles.*"%>
<%@page import="org.openspml.v2.util.xml.*"%>
<%@page import="org.openspml.v2.msg.*"%>
<%@page import="org.openspml.v2.msg.spml.*"%>
<%@page import="org.openspml.v2.util.*"%>

<%
final String url = "http://localhost:8080/idm/servlet/openspml2";
%>

<html>
<head><title>SPML2 Test</title></head>
<body>
<%

 // クライアントが必要。
 SessionAwareSpml2Client client = new SessionAwareSpml2Client( url );

 // ログイン
 client.login("configurator", "password");

 // AddRequest
 String rid = "rid-spmlv2"; // RequestId は厳密には必須ではない。

 Extensible data = new Extensible();
```

コード例 8-1 クライアントコードの例 (続き)

```
 data.addOpenContentElement(new DSMLAttr("accountId", user));
 data.addOpenContentElement(new DSMLAttr("objectclass", "spml2Person"));
 data.addOpenContentElement(new DSMLAttr("credentials", password));

 AddRequest add = new AddRequest(rid, // String requestId,
                ExecutionMode.SYNCHRONOUS, // ExecutionMode executionMode,
                null, // PSOIdentifier type,
                null, // PSOIdentifier containerID,
                data, // Extensible data,
                null, // CapabilityData[] capabilityData,
                null,  // String targetId,
                null  // ReturnData returnData
            );

    // 要求を送信する
    Response res = client.send( add );
%>
<%= res.toString()%>
</body>
</html>
```

コード例 8-2 は、送信される SOAP メッセージの本体を示します。

コード例 8-2 要求 XML の例

```
<addRequest xmlns='urn:oasis:names:tc:SPML:2:0' requestID='rid-spmlv2'
    executionMode='synchronous'>
  <openspml:operationalNameValuePair xmlns:openspml='urn:org:openspml:v2:util:xml'
     name='session' value='AAALPgAAYD0A...'/>
  <data>
    <dsml:attr xmlns:dsml='urn:oasis:names:tc:DSML:2:0:core' name='accountId'>
      <dsml:value>exampleSpml2Person</dsml:value>
    </dsml:attr>
    <dsml:attr xmlns:dsml='urn:oasis:names:tc:DSML:2:0:core' name='objectclass'>
      <dsml:value>spml2Person</dsml:value>
    </dsml:attr>
    <dsml:attr xmlns:dsml='urn:oasis:names:tc:DSML:2:0:core' name='credentials'>
      <dsml:value>pwdpwd</dsml:value>
    </dsml:attr>
  </data>
</addRequest>
```

コード例 8-3 は、クライアントに返される SOAP メッセージの本体を示します。

コード例 8-3 応答 XML の例

```
<addResponse xmlns='urn:oasis:names:tc:SPML:2:0' status='success' requestID='rid-spmlv2'>
  <openspml:operationalNameValuePair xmlns:openspml='urn:org:openspml:v2:util:xml'
      name='session' value='AAALPgAAYD0A...'/>
  <pso>
    <psoID ID='anSpml2Person'/>
    <data>
      <dsml:attr xmlns:dsml='urn:oasis:names:tc:DSML:2:0:core' name='accountId'>
        <dsml:value>anSpml2Person</dsml:value>
      </dsml:attr>
      <dsml:attr xmlns:dsml='urn:oasis:names:tc:DSML:2:0:core' name='objectclass'>
        <dsml:value>spml2Person</dsml:value>
      </dsml:attr>
      <dsml:attr xmlns:dsml='urn:oasis:names:tc:DSML:2:0:core' name='credentials'>
        <dsml:value>pwdpwd</dsml:value>
      </dsml:attr>
    </data>
  </pso>
</addResponse>
```

ListTargetsRequest の例

次の例は、Identity Manager を介して利用可能な ListsTargetRequest を示します。

コード例 8-4 は、Identity Manager の `SessionAwareSpml2Client` クラスを通して `ListTargetsRequest` を呼び出す `.jsp` を示します。

コード例 8-4 クライアントコードの例

```
<%@page contentType="text/html"%>
<%@page import="org.openspml.v2.client.*,
                com.sun.idm.rpc.spml2.SessionAwareSpml2Client"%
<%@page import="org.openspml.v2.profiles.dsml.*"%>
<%@page import="org.openspml.v2.profiles.*"%>
<%@page import="org.openspml.v2.util.xml.*"%>
<%@page import="org.openspml.v2.msg.*"%>
<%@page import="org.openspml.v2.msg.spml.*"%>
<%@page import="org.openspml.v2.util.*"%>

<%
final String url = "http://localhost:8080/idm/servlet/openspml2";
%>

<html>
<head><title>SPML2 Test</title></head>
```

コード例 8-4 クライアントコードの例 (続き)

```
<body>
<%

 // クライアントが必要。
 SessionAwareSpml2Client client = new SessionAwareSpml2Client( url );

 // ログイン (ListTargetsRequest を送信する )
 Response res = client.login("configurator", "password");

%>
<%= res.toString()%>
</body>
</html>
```

コード例 8-5 は、送信される SOAP メッセージの本体を示します。

コード例 8-5 要求 XML の例

```
<listTargetsRequest xmlns='urn:oasis:names:tc:SPML:2:0' requestID='rid[7013]'
    executionMode='synchronous'>
  <openspml:operationalNameValuePair xmlns:openspml='urn:org:openspml:v2:util:xml'
    name='accountId' value='configurator'/>
  <openspml:operationalNameValuePair xmlns:openspml='urn:org:openspml:v2:util:xml'
    name='password' value='password'/>
</listTargetsRequest>
```

コード例 8-6 は、クライアントが受信する (クライアントに返される) SOAP メッセージの本体を示します。

コード例 8-6 応答 XML の例

```
<listTargetsResponse xmlns='urn:oasis:names:tc:SPML:2:0' status='success' requestID='rid[6843]'>
  <openspml:operationalNameValuePair xmlns:openspml='urn:org:openspml:v2:util:xml'
     name='session' value='AAALPgAAYD0A...'/>
  <target targetID='spml2-DSML-Target' profile='urn:oasis:names:tc:SPML:2:0:DSML'>
    <schema>
      <spmldsml:schema xmlns:spmldsml='urn:oasis:names:tc:SPML:2:0:DSML'>
        <spmldsml:objectClassDefinition name='spml2Person'>
          <spmldsml:memberAttributes>
            <spmldsml:attributeDefinitionReference required='true' name='objectclass'/>
            <spmldsml:attributeDefinitionReference required='true' name='accountId'/>
            <spmldsml:attributeDefinitionReference required='true' name='credentials'/>
            <spmldsml:attributeDefinitionReference name='firstname'/>
```

コード例 8-6　応答 XML の例 (続き)

```
            <spmldsml:attributeDefinitionReference name='lastname'/>
            <spmldsml:attributeDefinitionReference name='emailAddress'/>
          </spmldsml:memberAttributes>
        </spmldsml:objectClassDefinition>
        <spmldsml:attributeDefinition name='objectclass'/>
        <spmldsml:attributeDefinition description='Account Id' name='accountId'/>
        <spmldsml:attributeDefinition description='Credentials, e.g. password'
            name='credentials'/>
        <spmldsml:attributeDefinition description='First Name' name='firstname'/>
        <spmldsml:attributeDefinition description='Last Name' name='lastname'/>
        <spmldsml:attributeDefinition description='Email Address' name='emailAddress'/>
      </spmldsml:schema>
      <supportedSchemaEntity entityName='spml2Person'/>
    </schema>
    <capabilities>
      <capability namespaceURI='urn:oasis:names:tc:SPML:2:0:async'/>
      <capability namespaceURI='urn:oasis:names:tc:SPML:2:0:batch'/>
      <capability namespaceURI='urn:oasis:names:tc:SPML:2:0:bulk'/>
      <capability namespaceURI='urn:oasis:names:tc:SPML:2:0:pass'/>
      <capability namespaceURI='urn:oasis:names:tc:SPML:2:0:suspend'/>
    </capabilities>
  </target>
</listTargetsResponse>
```

Async 機能のサポート

Identity Manager は、表 8-3 で説明した Async 機能をサポートします。

表 8-3　Async 機能

機能	説明	オペレーショナル属性	相違点
CancelRequest	要求 ID を使用して要求をキャンセルします。	なし	
StatusRequest	要求 ID を使用して要求のステータスを返します。	なし	

Batch 機能のサポート

Identity Manager は、表 8-4 で説明した Batch 機能をサポートします。

表 8-4　Batch 機能

機能	説明	オペレーショナル属性	相違点
BatchRequest	要求のバッチを実行します。	なし	

Bulk 機能のサポート

Identity Manager は、表 8-5 で説明した Bulk 機能をサポートします。

表 8-5　Bulk 機能

機能	説明	オペレーショナル属性	相違点
BulkDeleteRequest	PSO の一括削除を実行します。	なし	
BulkModifyRequest	一致する PSO の一括変更を実行します。	なし	

Password 機能のサポート

Identity Manager は、表 8-6 で説明した Password 機能をサポートします。

表 8-6　Password 機能

機能	説明	オペレーショナル属性	相違点
ExpirePasswordRequest	パスワードを失効させます。	なし	• リソースやターゲットは指定できません。指定すると、Identity Manager の User オブジェクトのパスワードが失効します。これが原因でその後、全ユーザーのリソースのパスワードが失効します。 • Identity Manager は `remainingLogins` 属性をサポートしません。 この属性をデフォルト以外の値に設定すると、`OperationNotSupported` エラーが発生します。
ResetPasswordRequest	すべてのアカウントに対してパスワードをリセットし、新しい値を返します。	なし	パスワードは漏洩を防ぐ必要があります。SSL またはその他のセキュリティー保護された伝送手段を使用してください。
SetPasswordRequest	パスワードを設定します。	なし	パスワードは漏洩を防ぐ必要があります。SSL またはその他のセキュリティー保護された伝送手段を使用してください。
ValidatePasswordRequest	指定されたパスワードが有効かどうかを判断します。	なし	パスワードは漏洩を防ぐ必要があります。SSL またはその他のセキュリティー保護された伝送手段を使用してください。

Password 機能の例を次に示します。

ResetPasswordRequest の例

コード例 8-7 は ResetPasswordRequest の例です。

コード例 8-7　ResetPasswordRequest の例

```
ResetPasswordRequest rpr = new ResetPasswordRequest();
...
PSOIdentifier psoId = new PSOIdentifier(accountId, null, null);
rpr.setPsoID(psoId);
...
```

SetPasswordRequest の例

コード例 8-8 は SetPasswordRequest の例です。

コード例 8-8　SetPasswordRequest の例

```
SetPasswordRequest spr = new SetPasswordRequest();
...
PSOIdentifier psoId = new PSOIdentifier(accountId, null, null);
              spr.setPsoID(psoId);
spr.setPassword("newpassword");
spr.setCurrentPassword("oldpassword");
...
```

ValidatePasswordRequest の例

コード例 8-9 は ValidatePasswordRequest の例です。

コード例 8-9　ValidatePasswordRequest の例

```
ValidatePasswordRequest vpr = new ValidatePasswordRequest();
...
PSOIdentifier psoId = new PSOIdentifier(accountId, null, null);
vpr.setPsoID(psoId);
vpr.setPassword("apassword");
...
```

Suspend 機能のサポート

Identity Manager は、表 8-7 で説明した Suspend 機能をサポートします。

表 8-7　Suspend 機能

機能	説明	オペレーショナル属性	相違点
ResumeRequest	PSO ユーザーを再開 (有効化) します。	なし	`EffectiveDate` をサポートしません。 `EffectiveDate` を設定すると、Identity Manager は `OperationNotSupported` エラーを返します。
SuspendRequest	アカウントや PSO を中断 (無効化) します。	なし	`EffectiveDate` をサポートしません。 `EffectiveDate` を設定すると、Identity Manager は `OperationNotSupported` エラーを返します。

サポートされない SPML 2.0 の機能

Identity Manager 7.1 では、Reference 機能、Search 機能、および Updates 機能はサポートされません。

加えて、サポートされているどの機能でも `CapabilityData` クラスは使用されないため、Identity Manager コードベースにはこのクラスをサポートするためのインフラストラクチャーはありません。これは、カスタム機能を実装する場合に重要です。OpenSPML 2.0 Toolkit では、整列化、非整列化などで `CapabilityData` がサポートされません。

Reference 機能がサポートされない理由

Identity Manager が参照をサポートしない理由は 2 つあります。Reference 機能は通常、ターゲット間、または同じターゲット上のオブジェクト間での参照に使用されます。

Identity Manager は正式には 1 つのターゲットしかサポートしないため、ターゲット間の参照は使用できません。また、Identity Manager でサポートされている (`objectclass` を使用して PSO の型にマップされる) `repoType` は User のみであり、これらのオブジェクトはほかのオブジェクトを参照しないため、現時点で Reference をサポートする必要性はありません。

Search 機能がサポートされない理由

現在、Search 機能は SPML 1.0 を通じてのみサポートされます。

Identity Manager で SPML 2.0 の Search 機能をサポートしない理由は、この機能によって実行される検索が現時点ではそれほど効率的でないためです。検索フィルタは完全なユーザービューに対して機能するため、すべての User オブジェクトを完全なビューにインスタンス化する必要があります。これはそれほど効率的ではありませんが、プロビジョニングシステムとしては必要です。この理由から、Identity Manager は検索の結果に対して、繰り返し処理 (または効率的な検索) を適切には提供できません。Search 機能の追加は、将来のリリースで対応される予定です。

Updates 機能がサポートされない理由

Identity Manager で Updates 機能がサポートされない理由は、この機能のために必要な情報である変更レコードを Identity Manager が追跡しないためです。

SPML 2.0 を使用するための Identity Manager の設定

SPML 2.0 を使用するために Identity Manager サーバーを設定するとき、最初に行うことは、ターゲットを通じて管理する属性の決定です。

注　　ターゲットには複数の属性を割り当てることができます。

このインタフェースを使用する Identity Manager インスタンスでユーザーを管理するときに、インタフェースクライアントがどの属性セット (`objectclasses`) を使用するかを決定します。この属性セットが *PSO* です。

フォームを使用して、それらの属性をユーザービューとの間でマップする方法についても理解する必要があります。

この節では、`spml2Person` という DSML オブジェクトクラスに対して、次の属性を含む PSO を使用するシステムの設定方法について説明します。

- `accountId`
- `objectclass`
- `credentials`
- `firstname`
- `lastname`

- emailAddress

これらの属性はユーザービューにマップされます。またこの節では、Identity Manager での SPML 2.0 サポートを使用して、これらの PSO の管理方法を説明する、簡単な例も示します。この設定は、製品に付属する sample/spml2.xml ファイルから派生したものです。このファイルを参照することで、詳しい情報を確認できます。

PSO の形式を決定したあとで、次の節で説明するサービスを有効にします。次の節では、web.xml ファイルと、SPML 2.0 で追加された要素について説明しています。

SPML2 設定オブジェクト

SPML 2.0 サポートの初期状態の設定は、sample/spml2.xml に定義されています。このファイルでは、SPML 2.0 をサポートするために Identity Manager で必要なオブジェクトを定義しています。

この節では、これらの設定型オブジェクトの 1 つである SPML2 について説明します。このオブジェクトを使用して、SPML 2.0 サポートの動作の変更やシステムの拡張を行います (拡張内容については、個別の節で詳しく説明)。

次の例は、オブジェクトの注釈付きエクスポートです。

コード例 8-10 SPML2.XML オブジェクトの注釈付きエクスポート

```
<Configuration name='SPML2' authType='SPML'>
    <Extension>
        <Object>

            <!-- このリスト内の各オブジェクトは、「executorClass」の組と、その組が
                 処理方法を知っている要求のセットを表す。要求のタイプは複数の executor の
                 組に出現する可能性がある。この結果、その型に対して executor の
                 チェーンが発生する。最初の executor がその型の与えられたインスタンスを
                 処理できない場合、要求はチェーン内の次の executor に渡される。
                 重要な点として、この executor のリストは順番に処理される。
                 一般に、このセクションは変更しない。-->

           <Attribute name='executors'>
               <List>
                   <Object name='com.sun.idm.rpc.spml2.core.ListTargetsExecutor'>
                       <Attribute name="requests" value="org.openspml.v2.msg.
                            spml.ListTargetsRequest"/>
                   </Object>
                   <Object name='com.sun.idm.rpc.spml2.pass.PasswordRequestExecutor'>
                       <Attribute name='requests'>
                           <List>
                               <String>org.openspml.v2.msg.pass.SetPasswordRequest</String>
                               <String>org.openspml.v2.msg.pass.ResetPasswordRequest</String>
```

コード例 8-10 SPML2.XML オブジェクトの注釈付きエクスポート (続き)

```
<String>org.openspml.v2.msg.pass.ValidatePasswordRequest</String>
<String>org.openspml.v2.msg.pass.ExpirePasswordRequest</String>
                        </List>
                    </Attribute>
                </Object>
                <Object name='com.sun.idm.rpc.spml2.core.AddRequestExecutor'>
                    <Attribute name="requests"
                         value='org.openspml.v2.msg.spml.AddRequest'/>
                </Object>
                <Object name='com.sun.idm.rpc.spml2.core.DeleteRequestExecutor'>
                    <Attribute name='requests'
                         value='org.openspml.v2.msg.spml.DeleteRequest'/>
                </Object>
                <Object name='com.sun.idm.rpc.spml2.core.LookupRequestExecutor'>
                    <Attribute name='requests'
                         value='org.openspml.v2.msg.spml.LookupRequest'/>
                </Object>
                <Object name='com.sun.idm.rpc.spml2.core.ModifyRequestExecutor'>
                    <Attribute name='requests'
                         value='org.openspml.v2.msg.spml.ModifyRequest'/>
                </Object>
                <Object name='com.sun.idm.rpc.spml2.suspend.SuspendRequestExecutor'>
                    <Attribute name='requests'
                         value='org.openspml.v2.msg.spmlsuspend.SuspendRequest'/>
                </Object>
                <Object name='com.sun.idm.rpc.spml2.suspend.ResumeRequestExecutor'>
                    <Attribute name='requests'
                         value='org.openspml.v2.msg.spmlsuspend.ResumeRequest'/>
                </Object>
                <Object name='com.sun.idm.rpc.spml2.suspend.ActiveRequestExecutor'>
                    <Attribute name='requests'
                         value='org.openspml.v2.msg.spmlsuspend.ActiveRequest'/>
                </Object>
                <Object name='com.sun.idm.rpc.spml2.batch.BatchRequestExecutor'>
                    <Attribute name='requests'
                         value='org.openspml.v2.msg.spmlbatch.BatchRequest'/>
                </Object>
                <Object name='com.sun.idm.rpc.spml2.bulk.BulkDeleteRequestExecutor'>
                    <Attribute name='requests'
                         value='org.openspml.v2.msg.spmlbulk.BulkDeleteRequest'/>
                </Object>
                <Object name='com.sun.idm.rpc.spml2.bulk.BulkModifyRequestExecutor'>
                    <Attribute name='requests'
                         value='org.openspml.v2.msg.spmlbulk.BulkModifyRequest'/>
                </Object>
                <Object name='com.sun.idm.rpc.spml2.async.StatusRequestExecutor'>
                    <Attribute name='requests'
                         value='org.openspml.v2.msg.spmlasync.StatusRequest'/>
                </Object>
                <Object name='com.sun.idm.rpc.spml2.async.CancelRequestExecutor'>
```

コード例 8-10　　SPML2.XML オブジェクトの注釈付きエクスポート (続き)

```
                        <Attribute name='requests'
                             value='org.openspml.v2.msg.spmlasync.CancelRequest'/>
                    </Object>
                </List>
            </Attribute>
            <Attribute name='targets'>
                <List>

                    <!-- これはターゲットの名前である。読みやすく参照しやすいように
                         区切られている。これは、このオブジェクトの名前が「xmlTemplate」内の
                         $OBJECT_NAME$ 変数を置き換えるときに targetID になる。-->

                    <Object name="spml2-DSML-Target">

                        <!-- ターゲットがプロファイルに添付されることを指定できる。
                            「xmlTemplate」内の $PROFILE_ATTRIBUTE$ を
                              「profile="<value>"」に置き換える。
                        -->

                        <Attribute name="profile" value="urn:oasis:names:tc:SPML:2:0:DSML"/>

                        <!-- これはターゲット定義である。最初の 2 行と最後の行は
                             (「targets」リスト内の各オブジェクトに対して ) 常に同じである。-->

                        <Attribute name='xmlTemplate'>
                           <String><![CDATA[
<target targetID="$OBJECT_NAME$" $PROFILE_ATTRIBUTE$>
    <schema>
        <spmldsml:schema xmlns:spmldsml="urn:oasis:names:tc:SPML:2:0:DSML">
            <spmldsml:attributeDefinition name="objectclass"/>
            <spmldsml:attributeDefinition name="accountId" description="Account Id"/>
            <spmldsml:attributeDefinition name="credentials" description="Credentials,
                e.g. password"/>
            <spmldsml:attributeDefinition name="firstname" description="First Name"/>
            <spmldsml:attributeDefinition name="lastname" description="Last Name"/>
            <spmldsml:attributeDefinition name="emailAddress" description="Email Address"/>
            <spmldsml:objectClassDefinition name="spml2Person">
                <spmldsml:memberAttributes>
                    <spmldsml:attributeDefinitionReference name="objectclass"
                        required="true"/>
                    <spmldsml:attributeDefinitionReference name="accountId" required="true"/>
                    <spmldsml:attributeDefinitionReference name="credentials"
                         required="true"/>
                    <spmldsml:attributeDefinitionReference name="firstname"/>
                    <spmldsml:attributeDefinitionReference name="lastname"/>
                    <spmldsml:attributeDefinitionReference name="emailAddress"/>
                </spmldsml:memberAttributes>
            </spmldsml:objectClassDefinition>
        </spmldsml:schema>
        <supportedSchemaEntity entityName="spml2Person"/>
    </schema>
    <capabilities>
```

コード例 8-10 SPML2.XML オブジェクトの注釈付きエクスポート (続き)

```
        <capability namespaceURI="urn:oasis:names:tc:SPML:2:0:async"/>
        <capability namespaceURI="urn:oasis:names:tc:SPML:2:0:batch"/>
        <capability namespaceURI="urn:oasis:names:tc:SPML:2:0:bulk"/>
        <capability namespaceURI="urn:oasis:names:tc:SPML:2:0:pass"/>
        <capability namespaceURI="urn:oasis:names:tc:SPML:2:0:suspend"/>
    </capabilities>
</target>
                            ]]></String>
                        </Attribute>
                    </Object>
                </List>
            </Attribute>

                    <!-- これは spml.xml 設定の「classes」リストと同様である。ターゲットが
                         共通のオブジェクトクラス / 型の名前を持つ場合に、1 つまたは複数の
                         ターゲットにマッピングを適用できるようにした。詳細は SPML 1.0 の
                         ファイルおよびドキュメントを参照。
            -->

            <Attribute name='mappings'>
                <List>
                    <Object name='spml2Person'>
                        <Attribute name='type' value='User'/>
                        <Attribute name='form' value='spml2PersonForm'/>
                        <Attribute name='default' value='true'/>
                        <Attribute name='targets'>
                            <String>"spml2-DSML-Target"</String>
                        </Attribute>
                    </Object>
                    <Object name='request'>
                        <Attribute name='type' value='TaskInstance'/>
                        <Attribute name='filter'>
                            <AttributeCondition attrName='defName' operator='equals'
                                operand='SPML2Request'/
                        </Attribute>
                    </Object>
                </List>
            </Attribute>
        </Object>
    </Extension>
</Configuration>

<!-- オブジェクトクラス内の属性をビューとの間でマップするフォームを定義する必要もある。次の
フォームは spml2Person オブジェクトクラスに対してこのマッピングを行う。-->

<Configuration name='spml2PersonForm' authType='SPML'>
    <Extension>
        < フォーム >
            <Field name='accountId'>
                <Derivation>
                    <ref>waveset.accountId</ref>
                </Derivation>
            </Field>
```

コード例 8-10　SPML2.XML オブジェクトの注釈付きエクスポート (続き)

```
            <Field name='waveset.accountId'>
                <Expansion>
                    <ref>accountId</ref>
                </Expansion>
            </Field>
            <Field name='emailAddress'>
                <Derivation>
                    <ref>waveset.email</ref>
                </Derivation>
            </Field>
            <Field name='global.email'>
                <Expansion>
                    <ref>emailAddress</ref>
                </Expansion>
            </Field>
            <Field name='objectclass'>
                <Derivation>
                    <ref>accounts[Lighthouse].objectclass</ref>
                </Derivation>
            </Field>
            <Field name='accounts[Lighthouse].objectclass'>
                <Expansion>
                    <ref>objectclass</ref>
                </Expansion>
            </Field>
            <Field name='credentials'>
                <Derivation>
                    <ref>password.password</ref>
                </Derivation>
            </Field>
            <Field name='password.password'>
                <Expansion>
                    <ref>credentials</ref>
                </Expansion>
            </Field>
            <Field name='lastname'>
                <Derivation>
                    <ref>accounts[Lighthouse].lastname</ref>
                </Derivation>
            </Field>
            <Field name='global.lastname'>
                <Expansion>
                    <ref>lastname</ref>
                </Expansion>
            </Field>
            <Field name='firstname'>
                <Derivation>
                    <ref>accounts[Lighthouse].firstname</ref>
                </Derivation>
            </Field>
            <Field name='global.firstname'>
                <Expansion>
                    <ref>firstname</ref>
                </Expansion>
```

コード例 8-10 SPML2.XML オブジェクトの注釈付きエクスポート (続き)

```
            </Field>
        </Form>
    </Extension>
</Configuration>
```

web.xml

web.xml の次のセクションでは、SPML 2.0 要求を処理するサーブレットである openspmlRouter サーブレットを設定します。

注 web.xml のこのセクションには、出荷時点でデフォルトのインストールが定義されており、このコンポーネントに対するアクションは必要ありません。

コード例 8-11 openspmlRouter サーブレットの設定

```
<servlet>
    <servlet-name>openspmlRouter</servlet-name>
    <display-name>OpenSPML SOAP Router</display-name>
    <description>A router of RPC traffic - nominally SPML 2.0 over SOAP</description>
    <servlet-class>
        org.openspml.v2.transport.RPCRouterServlet
    </servlet-class>
    <init-param>
        <param-name>dispatchers</param-name>
        <param-value>org.openspml.v2.transport.SPMLViaSoapDispatcher</param-value>
    </init-param>
    <init-param>
        <param-name>trace</param-name>
        <param-value>false</param-value>
    </init-param>
    <init-param>
        <param-name>SpmlViaSoap.spmlMarshallers</param-name>
        <param-value>com.sun.idm.rpc.spml2.UberMarshaller</param-value>
    </init-param>
    <init-param>
        <param-name>SpmlViaSoap.spmlMarshallers.UberMarshaller.trace</param-name>
        <param-value>true</param-value>
    </init-param>
    <init-param>
        <param-name>SpmlViaSoap.spmlExecutors</param-name>
        <param-value>com.sun.idm.rpc.spml2.UberExecutor</param-value>
```

コード例 8-11　openspmlRouter サーブレットの設定 (続き)

```
    </init-param>
</servlet>
```

このファイルには、オプションの init-param が含まれています。このパラメータは、SPML 2.0 メッセージのフローを表示する (Swing の) 監視ウィンドウを開くために追加できます。この監視ウィンドウはデバッグに役立ちます。

追加内容の例を次に示します。

```
<init-param>
    <param-name>monitor</param-name>
 <param-value>org.openspml.v2.util.SwingRPCRouterMonitor</param-value>
</init-param>
```

コード例 8-12 は、コメント付きのセクションで、その他の init-params についての情報を提供しています。

コード例 8-12　コメント付きの例

```
<servlet>
    <servlet-name>openspmlRouter</servlet-name>
    <display-name>OpenSPML SOAP Router</display-name>
    <description>A router of RPC traffic - nominally SPML 2.0 over SOAP</description>
    <servlet-class>
    org.openspml.v2.transport.RPCRouterServlet
    </servlet-class>

    <!--
        Router はディスパッチャーを使用して SOAP メッセージを処理する。これは、ツールキット内の
        SOAP に対応した対応したディスパッチャーである。命名規則を介した独自のパラメータを持つ。
        次を参照。
    -->

    <init-param>
        <param-name>dispatchers</param-name>
        <param-value>org.openspml.v2.transport.SPMLViaSoapDispatcher</param-value>
    </init-param>

    <!--
        トレースを有効にし、サーブレットが情報メッセージをログに書き込むようにする。
    -->
```

コード例 8-12 コメント付きの例 (続き)

```
    <init-param>
        <param-name>trace</param-name>
        <param-value>false</param-value>
    </init-param>

    <!--
          先に定義した SpmlViaSOAPDispatcher は整列化処理 (Marshaller) を使用する。XML および SPML の
          オブジェクト間で移動を行うためのチェーンが存在する可能性がある。この目的のために実装した
          UberMarshaller を使用する。これは実際にはツールキットのクラスを変換したものである。
    -->
    <init-param>
        <param-name>SpmlViaSoap.spmlMarshallers</param-name>
        <param-value>com.sun.idm.rpc.spml2.UberMarshaller</param-value>
    </init-param>

    <!--
          ここで使用する UberMarshaller は独自のトレース設定を持つ。
          このリリースでは、この設定は実際には何も行わない。
    -->

    <init-param>
        <param-name>SpmlViaSoap.spmlMarshallers.UberMarshaller.trace</param-name>
        <param-value>true</param-value>
    </init-param>

    <!--
      最後に、ディスパッチャーは機能を実際に実装する executor のリストを持つ。要求を受け取ると、
      SOAP エンベロープを除去し、XML から本体を抽出して OpenSPML Request クラスに渡し、要求を処理できるか
      どうかを executor のリストに問い合わせる。ここでは UberExecutor を定義した。
      この executor は要求をほかの executor に再振り分けする。
      ほかの executor は spml2.xml (Configuration:SPML2) で指定される。
    -->

    <init-param>
        <param-name>SpmlViaSoap.spmlExecutors</param-name>
        <param-value>com.sun.idm.rpc.spml2.UberExecutor</param-value>
    </init-param>
</servlet>
```

SPML でのトレースの使用

Identity Manager の SPML トラフィックをロギングし、問題の診断に役立てることができるように、SPML では、次のようなトレース出力を有効にするためのオプションが提供されています。

メソッド 1

個別の SPML RPC 要求に対するトレースを有効にするために、`trace` というオペレーショナル属性をサーバー側の RPC 要求に渡すことができます。

```
AddRequest ar = new AddRequest();
ar.setOperationalAttribute("trace", "true");
```

トレースによって、SPMLv2 要求を処理するサーブレットの RPC トラフィックに関する情報の出力内容を制御します。たとえば、次のようにします。

`trace` 属性の使用方法がサーバー操作に与える影響はベンダー固有です。現在、Identity Manager は生の要求および応答データをサーバーコンソールに出力しています。これは、クライアントアプリケーションがコンソールウィンドウと関連付けられていない場合に役立ちます。

詳細については、OpenSPML ツールキットの製品マニュアルを参照してください。

メソッド 2

SOAP `rpcrouter` サーブレットを初期化することでトレースを有効にすることもできます。これによって、SPML 要求を処理するサーブレットの RPC トラフィック情報の出力を制御します。`rpcrouter` サーブレットは、サーバー側の SOAP トレースを有効にする *<init parameter>* を取ります。サーブレットの初期化ロジックが `trace` 設定パラメータのチェックを行い、パラメータ値が `true` であれば、生の要求および応答データをコンソールに出力します。

たとえば、トレースは、そのサーブレットについてどのような `System.out` が指定されていても、そこでやり取りされる生の XML を出力します (この `System.out` は、アプリケーションコンテナの関数)。

注 SOAP `rpcrouter` サーブレットは、SPML 組織によるサードパーティー製のオープンソースの `org.openspml.server.SOAPRouter` クラスです。このサーブレットの詳細については、お使いの OpenSPML ツールキットのマニュアルを参照してください。

次の節では、いくつかの注釈付きで spml2.xml を説明します。

システムの拡張

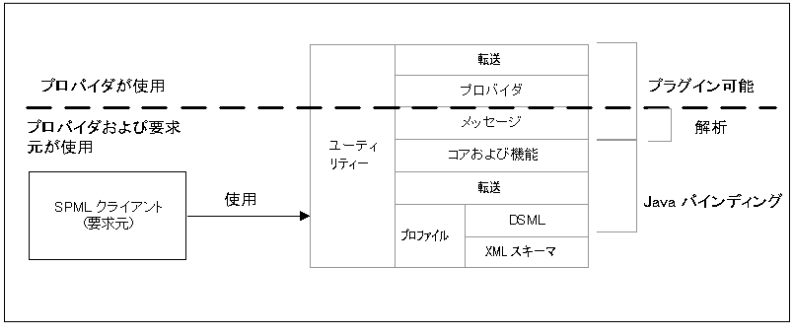
設定オブジェクトを変更することによって、スキーマを拡張します。セクションを変更することにより、要求の executor を追加できます。フォームを使用して、DSML とビューの間でマッピングを行うことができます。

少し難しくなりますが、ディスパッチャー、整列化クラス、および UberExecutor を、カスタマイズしたものと置き換えることもできます。

- SOAP を使用しない場合は、単に最初のケースのディスパッチャーを置き換えます。
- HTTP を使用しない場合は、Router を別の種類のサーブレットに置き換えます。
- 別の XML 解析処理を使用する場合は、Marshaller を独自のものに置き換えます。

SPML 2.0 は広く開かれたプラグイン可能性を提供しており、これは Identity Manager で OpenSPML 2.0 Toolkit を利用することによって実現されています。図 8-1 は、OpenSPML 2.0 Toolkit のアーキテクチャーを示しています。

図 8-1　　OpenSPML 2.0 Toolkit のアーキテクチャー



SPML 2.0 アダプタの例

Identity Manager には、サンプルの SPML 2.0 リソースアダプタが用意されています。このアダプタを変更して使用することにより、複数の Identity Manager 7.0 インストールや、SPML 2.0 コア操作をサポートするサードパーティーのリソースに接続することができます。

注 このサンプルアダプタは、製品 CD またはインストールイメージ上 (/REF に格納されている) の Sun Resource Extension Facility Kit に収録されています。

ビジネスプロセスエディタの使用方法

この付録では、ビジネスプロセスエディタ (BPE) の使用方法を説明します。この章で説明する内容は次のとおりです。

概要

ビジネスプロセスエディタ (BPE) は Swing ベースのスタンドアロン Java アプリケーションで、Sun Java™ System Identity Manager のワークフロー、フォーム、規則、一般オブジェクト、設定オブジェクト、およびビューを、グラフィカルにフォームベースで表示します。

BPE を使用して、環境に合わせて Identity Manager を次のようにカスタマイズします。

- フォーム、ワークフロー、規則、電子メールテンプレート、およびルールライブラリを表示、編集、および作成する
- 設定オブジェクトと一般オブジェクトを表示および編集する
- Identity Manager の公開 API を構成するクラスの JavaDoc を表示する

- フォーム、ワークフロー、および規則をデバッグする
- 特定のリポジトリと関連付けられるワークスペースを作成する

BPE の起動と設定

注	BPE を実行するには、ローカルシステムに Identity Manager がインストールされており、Identity Manager への Configurator レベルのアクセス権を付与されている必要があります。

この節では、BPE の起動および設定方法を説明します。

BPE の起動

コマンド行から BPE を起動するには、次の手順に従います。

1. Identity Manager のインストールディレクトリに移動します。
2. 次のコマンドで環境変数を設定します。

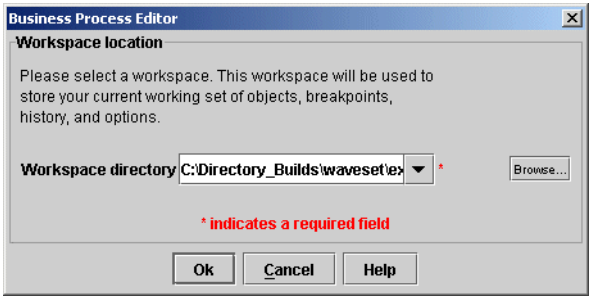
   ```
   set WSHOME=<Path_to_idm_directory>
   set JAVA_HOME=<path_to_jdk>
   ```

   UNIX システムで BPE を起動するには、次のコマンドも入力する必要があります。

   ```
   export WSHOME JAVA_HOME
   ```

3. `idm¥bin` ディレクトリに移動し、「`lh config`」と入力して BPE を起動します。図 A-1 に示すように、「Workspace location」ダイアログが表示されます。

図 A-1　　BPE の「Workspace location」ダイアログ



「Workspace location」ダイアログを使用して、新しいワークスペースを作成するか、既存のワークスペースを選択します。これら両方の処理の手順については、次の節で説明します。

ワークスペースの指定

ワークスペースは、リポジトリ接続情報 (デフォルトのサーバーやパスワードなど)、オプション、BPE デバッガによって設定されたブレークポイント、オープンソース、および自動保存されたファイルを保存するためのメカニズムです。

ワークスペースは特定のリポジトリに固定されます。1 つのリポジトリに複数のワークスペースを関連付けることができますが、ワークスペースごとに作成できるリポジトリは 1 つだけです。

BPE には、Identity Manager リポジトリへの 2 種類の接続があります。

- **エディタ接続** - この接続は、BPE のクラシックエディタ部分によって使用されます。

  エディタは次の方法で接続可能です。

  - **ローカル** : エディタは、`WSHOME` 内の `ServerRepository.xml` を使用して、ディレクトリをリポジトリに接続します。

    アプリケーションサーバーが動作していないときは、ローカル接続を使用してリポジトリ内のオブジェクトを編集できます。

  - SOAP: エディタは SOAP を使用してアプリケーションサーバーに接続します。

- **デバッガ接続** - この接続は、BPE のデバッガ部分によって次の目的で使用されます。

  - アプリケーションサーバーからソースコードを取得する
  - 現在のデバッグ状態 (変数、現在の位置) を受信する

- アプリケーションサーバーの内部で動作するデバッガエージェントに、コマンドを送信する (ブレークポイントの設定、ステップコマンドの送信)

デバッガエージェントへのコマンド送信には、稼働中のアプリケーションサーバーへの接続が必要なため、有効なデバッガ接続用の設定は SOAP のみです。エディタ接続に対して SOAP を選択した場合、デバッガはエディタと同じ接続を使用します。

この節では、次の手順を説明します。

- 新規ワークスペースの作成
- ワークスペースの選択
- 起動のトラブルシューティング

新規ワークスペースの作成

新しいワークスペースを作成するには、次の手順に従います。

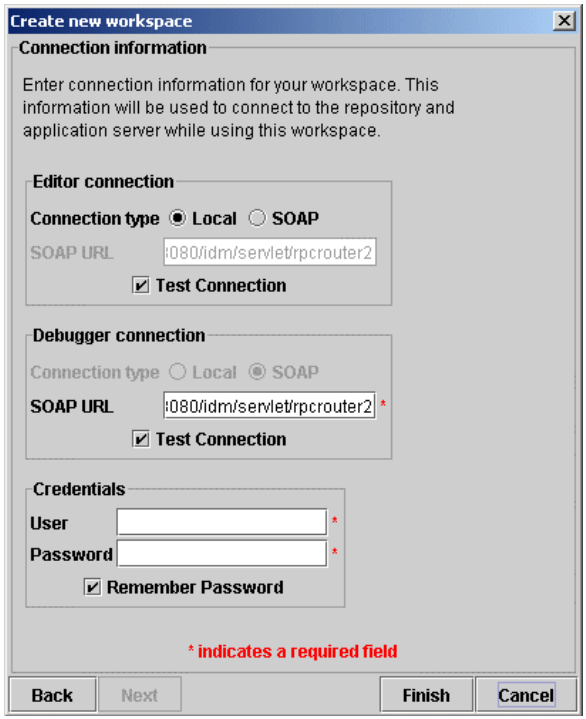
1. 「Workspace location」ダイアログで、新規ワークスペースの一意の名前を「Workspace Directory」フィールドに入力して「OK」をクリックします。

   まだ存在しないワークスペースの名前を指定すると、新規ワークスペースの作成ウィザードが表示され、ワークスペースのディレクトリを指定するように指示されます。

2. 「Workspace Directory」フィールドにディレクトリ名を入力して、「Next」をクリックします。

   「Connection Information」ダイアログが表示され、ワークスペースの接続情報を指定できます。

図 A-2　　BPE の「Connection Information」ダイアログ



3. 「Editor connection」情報を次のように指定します。

   a. 接続タイプを選択します。

      - **ローカル** (デフォルトで選択): ローカルリポジトリ内のオブジェクトに対する、BPE の操作を有効にする場合に選択します。

        ローカル接続を指定すると、BPE は `WSHOME` 内の `ServerRepository.xml` を使用してリポジトリに接続します (「SOAP URL」フィールドは無効になる)。

      - SOAP: 異なるリポジトリ内のオブジェクトに対する、BPE の操作を有効にする場合に選択します。

        SOAP 接続を指定すると、BPE デバッガのデフォルト接続タイプとして、同時に SOAP を指定することになります。

   b. SOAP 接続を使用する場合は、「SOAP URL」フィールドに完全修飾 URL を入力します。たとえば、「http://localhost:8080/<*idm*>/servlet/rpcrouter2」と入力します。< *idm*> は Identity Manager をインストールしたディレクトリです。

c. Identity Manager でリポジトリへのこの接続をテストするには、「Test Connection」を有効にします。

4. BPE デバッガの「Debugger connection」情報を次のように指定します。

すでに述べたように、エディタ接続タイプに対して SOAP を選択した場合は、デフォルトのデバッガ接続タイプをデフォルトで SOAP に設定します。「Debugger connection」領域のすべてのオプションは無効になります。

a. 接続タイプを選択し、SOAP URL を指定します (必要な場合)。

b. Identity Manager でリポジトリへのこの接続をテストするには、「Test Connection」を有効にします。

5. 次の資格情報を指定します。

a. 「User」にログイン名を、「Password」にパスワードを入力します。

b. BPE にログインするたびに、これらの資格情報をデフォルトで使用する場合、「Remember Password」オプションを選択します。

6. 「Finish」をクリックすると、新しいワークスペースが作成され、BPE のメインウィンドウが表示されます。

ワークスペースの選択

「Workspace location」ダイアログから、次のいずれかの方法で既存のワークスペースを選択します。

- 「Workspace Directory」メニューリストからワークスペース名を選択します。
- 「Browse」をクリックし、ワークスペースを探して選択します。

ワークスペースを選択したら、「OK」をクリックします。BPE のメインウィンドウが表示されます。

起動のトラブルシューティング

BPE が配下のサーバーに接続しようとしたとき、次のエラーメッセージが表示される場合があります。

```
HTTP 404 - /idm/servlet/rpcrouter2
Type Status report
message /idm/servlet/rpcrouter2 description
The requested resource (/idm/servlet/rpcrouter2) is not available
```

この接続エラーが発生した場合は、Identity Manager を実行しているブラウザインスタンスの URL フィールドを確認してください。フィールドにリストされている URL の最初の部分 (たとえば、http://localhost:8080/idm) は、デバッガ接続時に入力した URL と同一である必要があります。

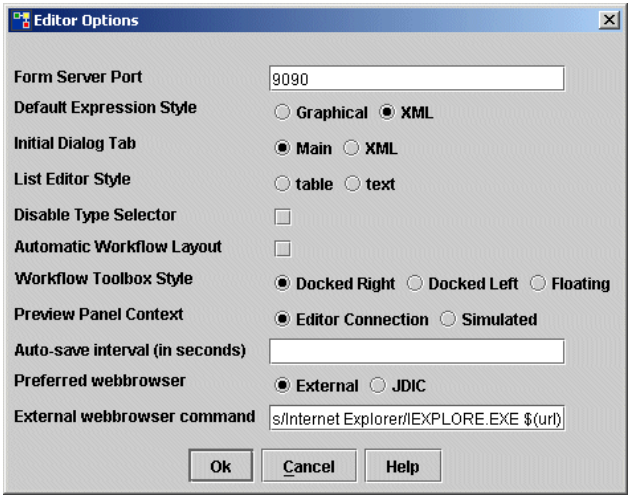
JDIC の有効化

Web ブラウザパネルを「Form Preview」パネルに組み込む場合、優先する Web ブラウザとして JDIC を選択する必要があります。選択しない場合、Web ブラウザパネルは `External webbrowser` コマンドを使用して外部の Web ブラウザを起動します。

JDIC を指定するには、次の手順に従います。

1. 「Tools」>「Options」の順に選択して「Editor Options」ダイアログを開きます。

図 A-3　「Editor Options」ダイアログ



2. 「Preferred webbrowser」で「JDIC」オプションを選択します (このオプションは、アプリケーションが 1.4 よりも前のバージョンの JRE を実行している場合は表示されません)。

注

- Windows 用の JDIC を有効にするには、Internet Explorer をインストールする必要があります (Mozilla は現時点で、Windows 上ではサポートされていない)。
- Linux または Solaris 上で JDIC を有効にするには、Mozilla をインストールする必要があります。現時点でサポートされているデスクトップは GNOME のみです。

  また、`MOZILLA_FIVE_HOME` 環境変数を、Mozilla インストールのルートディレクトリに設定する必要もあります。

x86 版 Solaris 10 以降用の JDIC を設定するには、次の手順に従います。

1. https://jdic.dev.java.net から `jdic-0.9.1-bin-cross-platform.zip` をダウンロードします。
2. zip ファイルを展開します。
3. <*wshome*>`/WEB-INF/lib/jdic.jar` を `jdic-0.9.1-bin-cross-platform/jdic.jar` と置き換えます。
4. `jdic-0.9.1-bin-cross-platform/sunos/x86/*` を <*wshome*>`/bin/solaris/x86` にコピーします。

BPE での SSL の使用

SSL を使用するには、BPE で新規ワークスペースの作成ウィザードを開き、SOAP URL プロトコルを `https` に、ポート番号をアプリケーションの SSL ポートに変更します。

ビジネスプロセスエディタのナビゲーション

Identity Manager のプロセスまたはオブジェクトのカスタマイズを開始する前に、BPE で情報を操作、表示、入力する方法および選択を実行する方法について理解してください。

この情報は次の各節で構成されています。

BPE インタフェースの操作

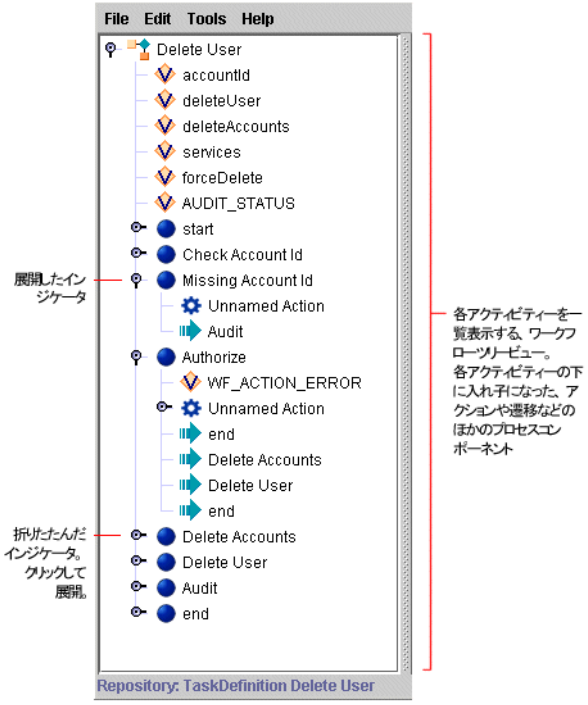
BPE のインタフェースには、メニューバーと、選択のためのダイアログが含まれています。主な表示は 2 つのメイン区画に分かれています。

- ツリービュー
- 次を含む補助表示ビュー
  - ダイアグラムビュー
  - グラフィカルビュー
  - プロパティービュー

ツリービューの操作

左区画のツリービューには、タスク、フォーム、ビュー、または規則が階層的に表示されます。このビューには、個々の変数、アクティビティー、およびサブプロセスが順番に表示されます。アクションおよび遷移は各活動の下に入れ子にされます。図 A-4 は、ワークフローを強調したサンプルのツリービューです。

図 A-4　　BPE のツリービュー



補助表示ビューの操作

BPE には、次の補助表示ビューがあります。

- ダイアグラムビュー
- グラフィカルビュー
- プロパティービュー

これらのビューが使用できるかどうかは、選択したオブジェクトタイプまたはプロセスによって異なります。
たとえば、フォームがブラウザに出現したとき、BPE はフォームのグラフィカル表示になります。このビューは、プロパティービューおよび一意のフォーム要素の XML 表示を補完します。

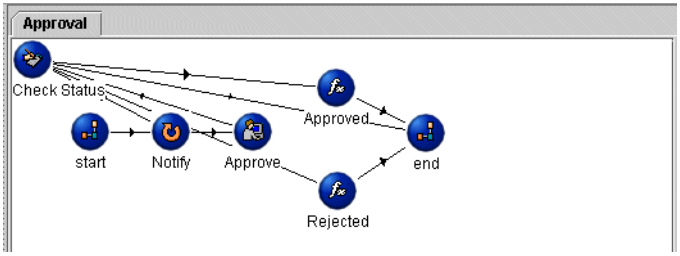
これらのビューについては、次の節で説明します。

注 Identity Manager の各オブジェクトまたはワークフロープロセスに対して使用可能な表示タイプと、これらの追加ビューの操作方法の詳細は、『Sun Java™ System Identity Manager ワークフロー、フォーム、およびビュー』を参照してください。

ダイアグラムビュー

ワークフローについては、インタフェースの右区画にダイアグラムビューが表示され、プロセスのグラフィカル表現を提供します。各アイコンは、特定のプロセスアクティビティーを表します。

図 A-5 ダイアグラムビュー (ワークフロー)



グラフィカルビュー

グラフィカルビューは BPE ウィンドウの右下区画に表示され、現在選択されているフォームをブラウザウィンドウでの表示と同様に表示します。

プロパティービュー

プロパティービューは BPE 表示の右上区画に表示され、現在選択されているフォーム内の要素についての情報を提供します。

図 A-6 プロパティービュー (フォーム)

AIX Create Group Form

Title	Class	Required	Action	No New Row	Hidden	Size	Max Length	Name
Create on:	Label	☐	☐	☐	☐			create on
Name:	Text	☐	☐	☐	☐			group.attributes.groupName
Group ID:	Text	☐	☐	☐	☐			group.attributes.id
Administrative:	Text	☐	☐	☐	☐			group.attributes.admin
Users	MultiSelect	☐	☐	☐	☐			AIXUsers
		☐	☐	☐	☐			group.attributes.users
	Button	☐	☐	☐	☐			
	Button	☐	☐	☐	☐			
		☐	☐	☐	☐			group.objectName
		☐	☐	☐	☐			group.objectId

プロセスまたはオブジェクトの読み込み

Identity Manager のプロセスまたはオブジェクトを読み込むには、次の手順に従います。

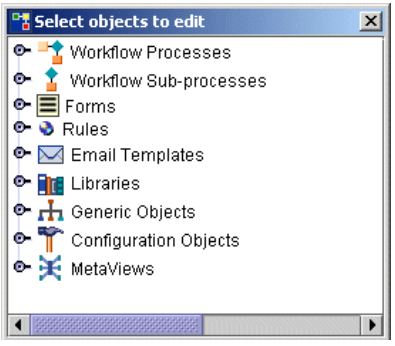
1. メニューバーから「File」>「Open Repository Object」の順に選択します。

ヒント	Ctrl-O のショートカットも使用できます (BPE のショートカットの完全な一覧については、315 ページの「キーボードショートカットの使用」を参照)。

2. 「Login」ダイアログで入力を求められたら、Identity Manager Configurator の名前とパスワードを入力して「Login」をクリックします。

   図 A-7 のような「Select objects to edit」ダイアログが表示されます。

図 A-7 「Select objects to edit」ダイアログ (「Library」オプションを展開)



このダイアログには、次のオブジェクトタイプを含む、オブジェクトの一覧が表示されます。

- ワークフロープロセス
- ライブラリ
- ワークフローサブプロセス
- 汎用オブジェクト
- フォーム
- 設定オブジェクト
- 規則
- メタビュー

- 電子メールテンプレート

表示される項目は、Identity Manager の実装によって異なる場合があります。

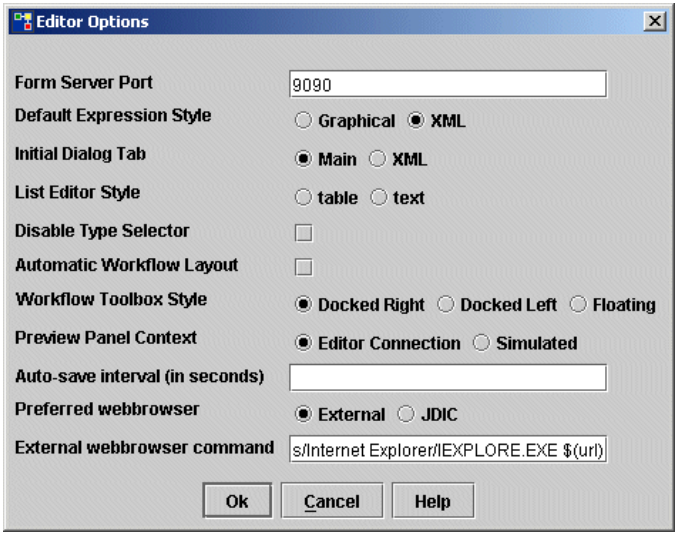
3. オブジェクトタイプをダブルクリックすると、そのタイプに対して表示アクセス権のあるオブジェクトがすべて表示されます。
4. プロセスまたはオブジェクトを選択して「OK」をクリックします。

エディタオプションの設定

BPE を起動するたびに好みの設定が反映されるように、各種のオプションを設定できます。エディタで作業するたびに、これらのオプションを個別に設定することもできます。

エディタオプションを設定するには、「Tools」>「Options」の順に選択して「Editor Options」ダイアログを開きます。

図 A-8　「Editor Options」ダイアログ



このダイアログのオプションを使用して、次の設定を指定できます。

- 「Form Server Port」- HTML プレビューページのデフォルトポートを指定します。このページはフォームの編集時に使用します。
- 「Default Expression Style」- フォーム、規則、およびワークフローでの式の表示オプションを制御します (「Graphical」または「XML」)。

- 「Initial Dialog Tab」- 最前面に表示されるタブを制御します (「Main」または「XML」)。
- 「List Editor Style」- リスト式のデフォルト表示を制御します。リストはテーブル形式またはテキストボックスで表示できます。
- 「Disable Type Selector」- テキストボックスの隣に表示される「Type Selector」オプションを無効にします。タイプを変更するためのオプションは、引き続き「Edit」ダイアログから利用できます。
- 「Automatic Workflow Layout」- 最初に開かれたときに、ワークフローアクティビティーの自動レイアウトを有効にします。
- 「Workflow Toolbox Style」- ワークフローツールボックスの表示位置を、メインの BPE ウィンドウからの相対位置で指定します。次のオプションがあります。
  - 「Docked Right」(デフォルト): BPE ウィンドウの右側にツールボックスを固定します。
  - 「Docked Left」: BPE ウィンドウの左側にツールボックスを固定します。
  - 「Floating」: BPE ウィンドウの周辺でツールボックスを移動できるようにします。
- 「Preview Panel Context」- 「Preview」区画に表示される情報の描画コンテキストを指定します。次のオプションがあります。
  - 「Editor Connection」- BPE が常にリポジトリへの接続を試みるようにします。
  - 「Simulated」- フォーム上でオフライン作業を行います。
- 「Auto-save interval (in seconds)」- BPE がセッションを自動保存する間隔を、秒数で指定します (デフォルトは 30 秒)。
- 「Preferred webbrowser」- Web ブラウザの起動方法を指定します。次のオプションがあります。
  - 「External」(デフォルト): BPE で `External webbrowser` コマンドを使用して外部の Web ブラウザを起動します。
  - 「JDIC」:「Form Preview」パネル内に Web ブラウザパネルを起動します。
- 「External webbrowser command」- 外部 Web ブラウザを起動するための `External webbrowser` コマンドを指定します。

ワークフローリビジョンの検証

カスタマイズプロセスの各段階で、ワークフローのリビジョンを検証できます。

- XML 表示値を操作している場合、変数、アクティビティー、アクション、遷移の追加またはカスタマイズ時に「Validate」をクリックすると、それぞれの変更を検証できます。
- 変更を行なったあとに、ツリービューでオブジェクトまたはプロセスを選択し、「Tools」>「Validate」の順に選択してテストを実行します。

BPE では、プロセスのステータスを示す、検証メッセージが表示されます。

- **警告インジケータ (黄色のドット)** - プロセスの操作は有効だが、構文スタイルが最適ではないことを示します。
- **エラーインジケータ (赤のドット)** - プロセスが正常に実行されないことを示します。プロセスの操作を修正する必要があります。

ワークフローのリビジョンを検証するには、次の手順に従います。

1. インジケータをクリックして、そのプロセスアクションを表示します。
2. 変更を行なったあとに、「Re-validate」をクリックしてプロセスを再テストし、エラーが修正されたことを確認して、別のエラーをチェックします。
3. ワークフローのダイアグラムビューにカーソルをドラッグします。

   アクティビティーがビューに表示されます。

ヒント	最初のアクティビティーよりもあとに作成した、すべてのアクティビティーには番号が付けられます。2 つを超えるアクティビティーを作成する前に、類似のアクティビティーの番号を再設定してください。

変更の保存

プロセスまたはオブジェクトへの変更を保存してリポジトリにチェックインするには、メニューバーから「File」>「Save in Repository」の順に選択します。「Save」を選択すると、最後に保存された場所 (リポジトリまたは最後に保存されたファイルのどちらか) にオブジェクトが保存されます。同じオブジェクトの複数のコピーを、異なる状態で、また複数の異なるファイルまたはリポジトリで開くことができます。

注	「File」>「Save As File」の順に選択して、オブジェクトまたはプロセスを XML テキストファイルに保存することもできます。ファイルへの保存はファイル名 `.xml` の形式で行います。

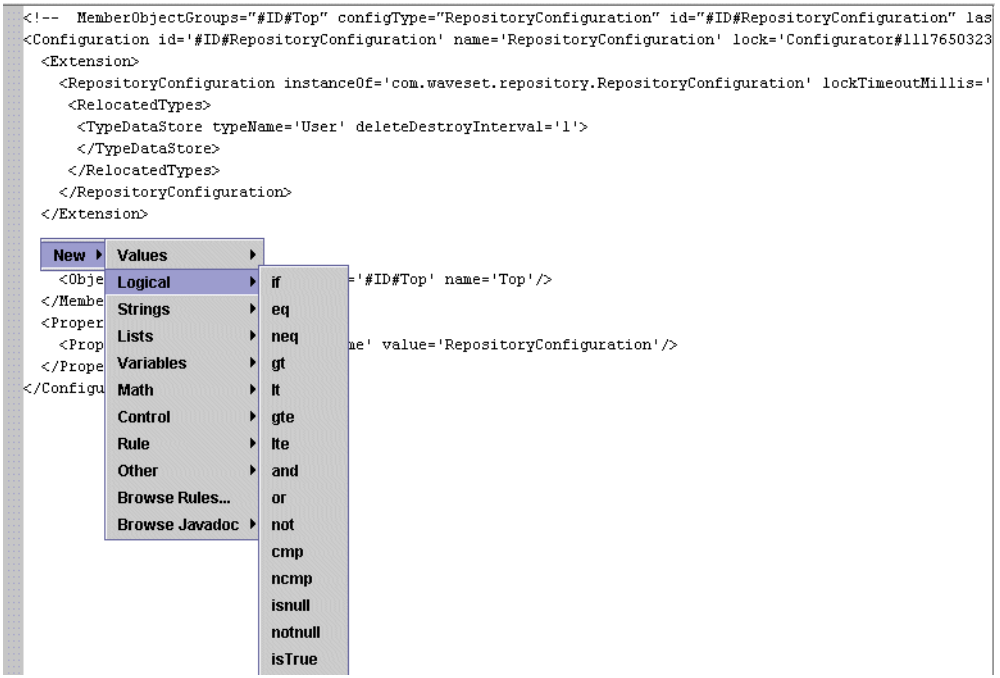
XPRESS の挿入

BPE の「XML」区画で規則、ワークフロー、設定オブジェクト、汎用オブジェクト、またはフォームを編集中に、カーソルが置かれている任意の場所に XPRESS 要素の XML テンプレートをすばやく挿入できます。

1. 新しい XPRESS 文を追加する場所にカーソルを置きます。
2. マウスの右ボタンをクリックして「New」メニューを表示します。
3. XML に追加する XPRESS 文の種類を選択します。

   たとえば、カーソル挿入ポイントに空の cond 文を追加するには、「New」>「Logical」>「cond」の順に選択します。次の図に示すように、内容が空の cond 文が表示されます。

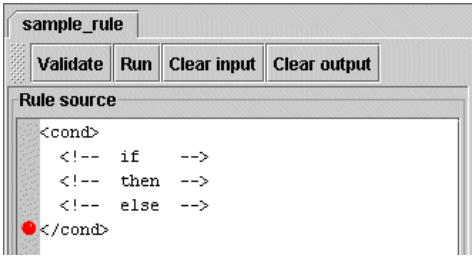
図 A-9　　XML への XPRESS 関数挿入メニュー



4. 必要に応じて文を完成します。

無効な位置に XPRESS 要素を挿入した場合、新しいコード行の左隣に 1 つまたは 2 つの赤のドット (インジケータ) が表示されます。これらは、挿入されたコードの最初の行と最後の行を示します。これらのインジケータの詳細は、「Validating Workflow Revisions」を参照してください。

図 A-10　　XPRESS 関数の挿入



キーボードショートカットの使用

BPE では、タスクを実行するための、次のキーボードショートカットがサポートされています。

表 A-1　　BPE のキーボードショートカット

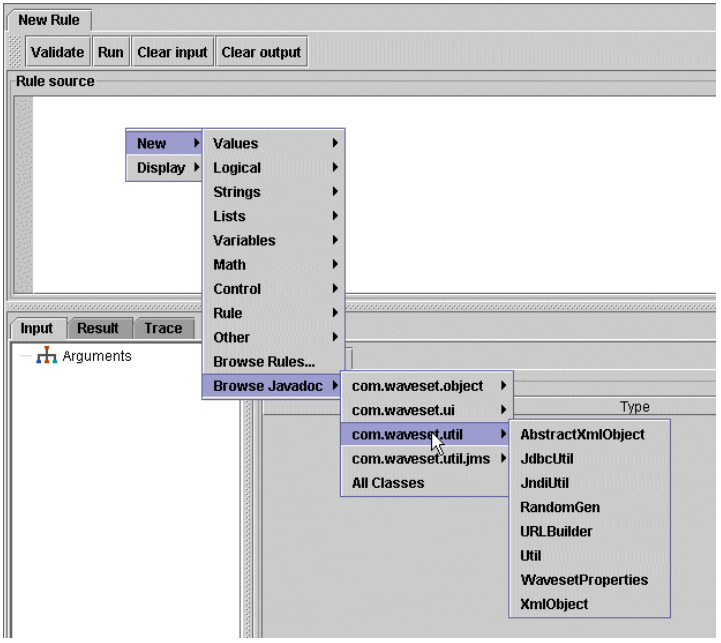
キーボートコマンド / キー	アクション
Ctrl-C	コピー
Ctrl-O	開く (リポジトリオブジェクト)
Ctrl-R	ソースの更新
Ctrl-S	保存 (リポジトリオブジェクト)
Ctrl-V	貼り付け
Ctrl-X	切り取り
Delete	削除
F5	現在の行の選択
F6	ステップアウト
F7	ステップイン
F8	ステップオーバー
F9	続行 (デバッグ)

JavaDoc へのアクセス

XML を表示するすべての BPE ウィンドウからは、次のようにして、すべての公開メソッドクラスの JavaDoc にアクセスできます。

1. XML ウィンドウ内で右クリックして、カスケードメニューを表示します。
2. 「New」>「Browse Javadoc」の順に選択します。

図 A-11　　Javadoc を開く



3. カスケードメニューから、次のいずれかのオプションを選択します。メニューには次のパッケージが含まれており、これらのパッケージはさらにコンポーネントクラスに分かれています。
   - 「com.waveset.object」：この親クラスに従属する、すべてのクラスを表示します。
   - 「com.waveset.ui」：この親クラスに従属する、すべてのクラスを表示します。
   - 「com.waveset.util」：この親クラスに従属する、すべてのクラスを表示します。
   - 「com.waveset.util.jms」：この親クラスに従属する、すべてのクラスを表示します。

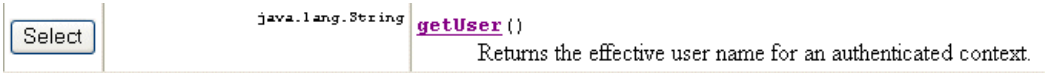
- 「All Classes」: Javadoc クラスのフレームビューを表示します。ブラウザ内で、このビューから各クラスの JavaDoc にジャンプできます。

これらのメニューオプションのいずれかを選択すると、クラスの Javadoc を表示するブラウザウィンドウが開きます。

メソッド参照の挿入

メソッド呼び出しを XML に挿入するには、クラス Javadoc のメソッド要約セクションにアクセスします。メソッドの要約で、メソッド名の前にある選択ボタンをクリックします。

図 A-12　getUser メソッドの選択



カーソル挿入ポイントの位置に、XML からメソッドを呼び出すために必要な <invoke> 要素が BPE によって挿入されます。

ヒント　文の呼び出し構文および XML を事前に確認するには、「**Validate**」をクリックします。

汎用オブジェクトと設定オブジェクトの操作

Identity Manager の基本オブジェクトモデルは、持続オブジェクトモデルです。Identity Manager のほぼすべての操作は、オブジェクトの作成によって実行するため、持続オブジェクト API は Lighthouse をカスタマイズおよび制御するための基本オブジェクトモデルです。

ここでは、持続オブジェクトの操作についての情報を提供します。説明する内容は次のとおりです。

- 共通持続オブジェクトクラス
- オブジェクトの表示と編集
- 新しいオブジェクトの作成
- 新規設定オブジェクトの検証

共通持続オブジェクトクラス

PersistentObject はすべての持続オブジェクトの共通基底クラスであり、Identity Manager をカスタマイズおよび制御するための基本オブジェクトモデルを提供します。PersistentObject は、すべての持続オブジェクトに共通のインフラストラクチャーの一部である Java クラスの集合で構成されます。

これらの共通 PersistentObject クラスには、次のものが含まれます。

- Type: 参照されるオブジェクトの型を示すために、多くのメソッドで使用される定数の集合。
- PersistentObject: すべてのリポジトリオブジェクトの共通基底クラス。最も重要なプロパティーは「*ID*」、「*member object groups*」、および「*property list*」です。
- ObjectRef: オブジェクトが別のオブジェクトを参照するとき、参照はこのオブジェクトに符号化されます。参照にはオブジェクトの型、名前、およびリポジトリ識別子が含まれます。
- Constants: 多数の異なるシステムコンポーネント用の、ランダム定数のコレクション。
- ObjectGroup: Identity Manager のインタフェース内で、組織を表すグループ。すべての持続オブジェクトは、少なくとも 1 つのオブジェクトグループに属する必要があります。特に指定しない場合、オブジェクトは最上位のグループに配置されます。
- Attribute: オブジェクトによってサポートされる共通属性を表す、定数オブジェクトのコレクション。多くの場合、オブジェクトクエリーを構築するときに内部的に使用されます。メソッドが Attribute 引数を取るとき、通常は、属性名を格納した文字列を引数に取る、対応したメソッドが存在します。

オブジェクトの表示と編集

BPE を使用して、最もカスタマイズされることが多い、2 種類の持続オブジェクトを表示および編集できます。

- **設定オブジェクト** : フォームおよびワークフロープロセスを含む、持続オブジェクト。
- **汎用オブジェクト** : <*Object*> 型の <*Extension*> を持つ設定オブジェクト。これは、<*WFProcess*> 型の <*Extension*> を持つ設定オブジェクトであるワークフローと対称をなすオブジェクトです。汎用オブジェクトは、一般的にはビューを表現するために使用され、名前 / 値ペアの単純なコレクションです。これらの属性には、パス式を使用して外部からアクセスできます。

以降の節では、設定オブジェクトタイプおよび汎用オブジェクトタイプの概要を説明します。詳細は、『*Sun Java*™ System Identity Manager ワークフロー、フォーム、およびビュー』を参照してください。

設定オブジェクト

BPE では、フォームおよびワークフローに直接アクセスできます。ただし BPE には、カスタムビューアと関連付けられていない、その他の設定オブジェクトへのアクセス手段も用意されています。これらのその他設定オブジェクトには、「Configuration Object」カテゴリの下にある「BPE」からアクセスできます。

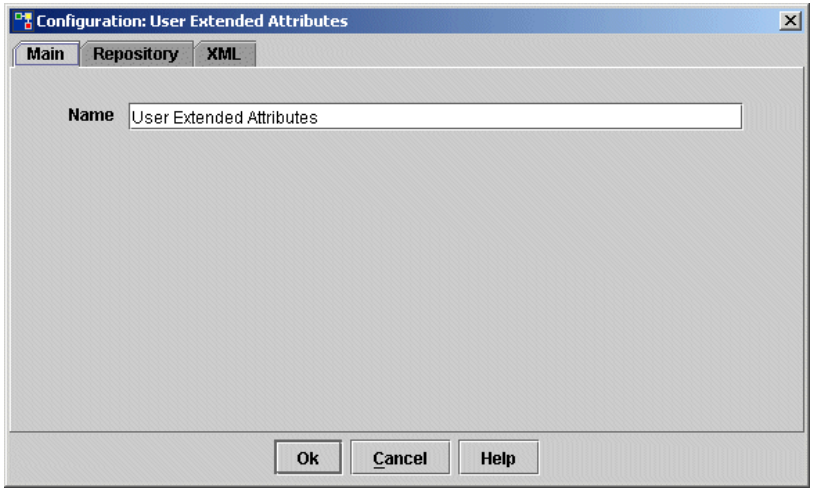
BPE では、次の図に示すように、これらの各種設定オブジェクトのリストが左区画のツリービューに表示されます。

図 A-13　　BPE での設定オブジェクトのツリー表示

ツリービューでオブジェクト名をダブルクリックすると、オブジェクトウィンドウが表示されます。このウィンドウには、「Main」、「Repository」、および「XML」の 3 つのオブジェクトビュー (タブ) があります。

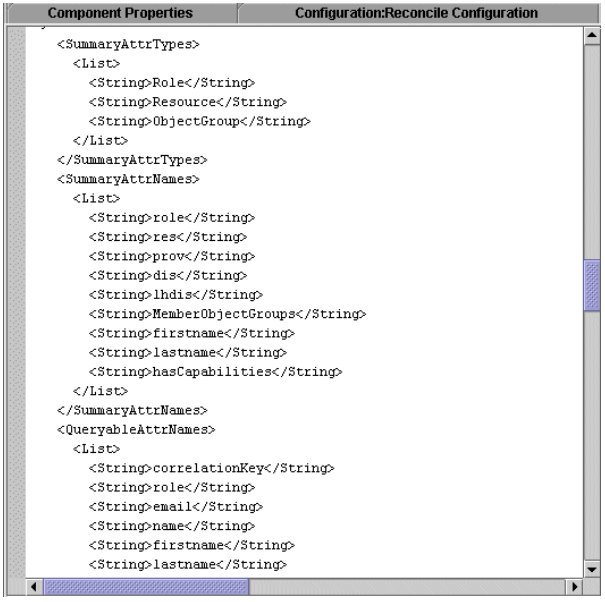
たとえば、ツリービューで「User Extended Attributes」をダブルクリックすると、次のダイアログが表示されます。

図 A-14　　オブジェクトの「User Extended Attributes」ダイアログ



BPE ウィンドウの左区画では、未フィルタの XML 形式でも、設定オブジェクトが表示されます。たとえば、次の図のようになります。

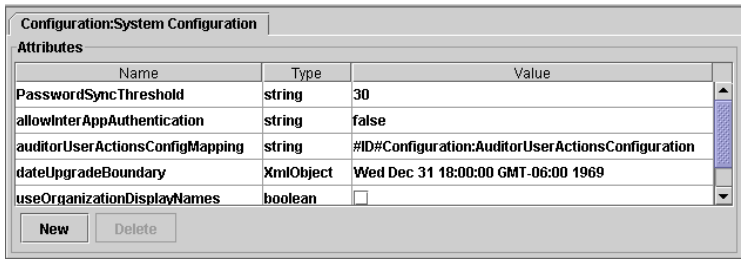
図 A-15　BPE での調整設定オブジェクトの XML 表示



汎用オブジェクト

汎用オブジェクトは名前 / 値ペアの単純なコレクションであり、ビューを表現するために使用できます。BPE では、これらの名前 / 値ペアが属性のデータ型とともに列形式で一覧表示されます。有効なデータ型には Boolean、int、string、xmlobject などがあります。

図 A-16　BPE での汎用オブジェクト (System Configuration) の属性表示



多くのカスタマイズでは、汎用オブジェクトタイプの System Configuration オブジェクトを編集する必要があります。

新しいオブジェクトの作成

新しい設定オブジェクトまたは汎用オブジェクトを作成するには、次の手順に従います。

1. 「File」>「New」の順に選択し、「GenericObject」または「Configuration:New Configuration」を選択します。

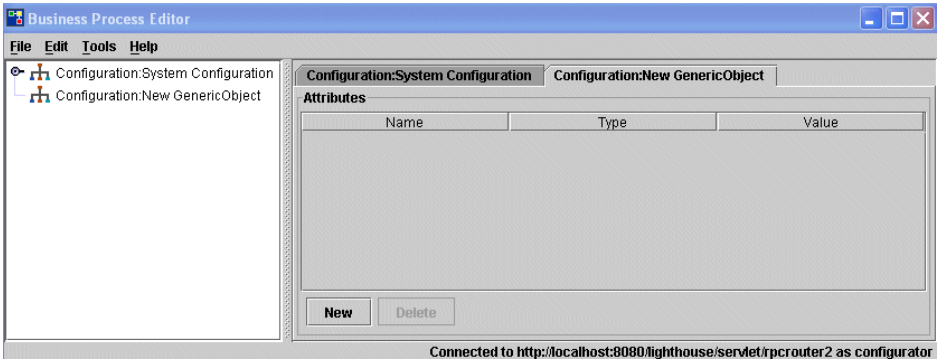
   「Configuration:New GenericObject」または「Configuration:New Configuration」ダイアログが開き、メインパネルが表示されます。

2. 「Name」フィールドに新しいオブジェクト名を入力します。

   BPE のメインウィンドウで、新しいオブジェクト名がツリービューに追加されます。また、次の処理が行われます。

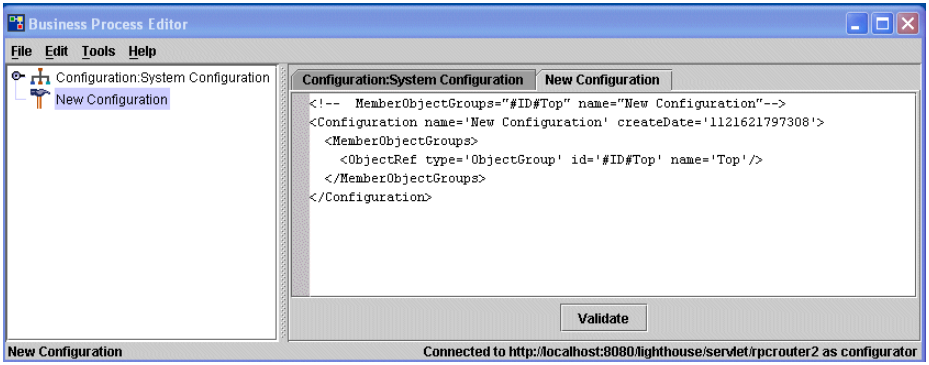
   - 汎用オブジェクトを作成した場合は、空の「Attributes」区画が次のように表示されます。

図 A-17　　BPE での新規汎用オブジェクトの表示



   - 設定オブジェクトを作成した場合は、BPE で次のウィンドウが表示されます。このウィンドウには、新しい XML オブジェクトのテンプレートが表示されます。

図 A-18　BPE での新規設定オブジェクトの表示

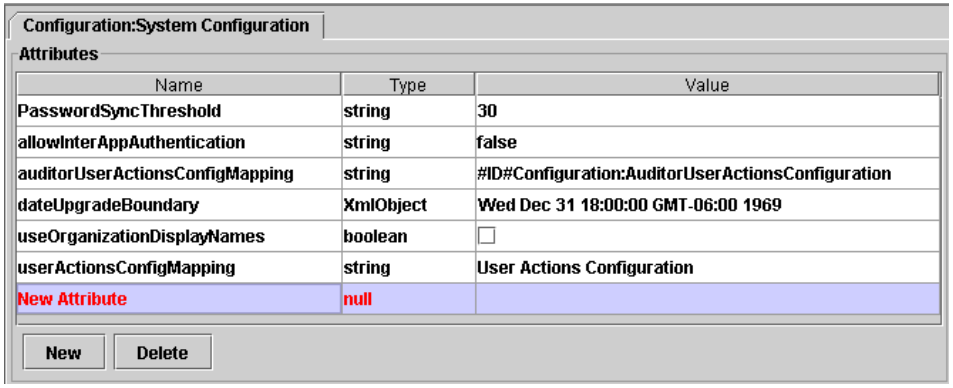


3. 汎用オブジェクトを作成する場合は、次のように属性を追加し、必要に応じて手順を繰り返します。

   a. 「Attributes」区画の下部にある「New」をクリックします。属性リストの一番下に、新しい属性フィールドが表示されます。「New Attribute」を選択し、その属性の名前を入力します。

   b. 「Type」列で「null」をクリックしてデータタイプを割り当てるか、またはドロップダウンメニューからデータタイプを選択します。

図 A-19　BPE の汎用オブジェクト表示の新規属性



注　属性を削除するには、属性名をクリックしてから「Delete」をクリックします。

4. 「File」>「Save in Repository」の順に選択して、新しいオブジェクトをリポジトリに保存します。

新規設定オブジェクトの検証

BPE のメインウィンドウの右区画で「Validate」をクリックすると、新規設定オブジェクトの XML をただちに確認できます。

規則の作成と編集

BPE を使用して、次の操作を行うことができます。

- 規則を表示、作成、編集する
- Lighthouse コンテキストを使用して規則をテストする
- 規則に渡されるデータを定義する
- 規則定義をファイルに保存する
- 選択した規則についてのデータ (属性の型など) を取得する
- 規則をカスタマイズするときに、参照の表示属性を表示する

この節では、BPE を使用して規則を作成および編集するための情報および手順を示します。説明する内容は次のとおりです。

注 BPE アプリケーションの起動手順は、300 ページの「BPE の起動と設定」で説明しています。

BPE インタフェースの使用方法

規則のカスタマイズを開始する前に、BPE インタフェースのナビゲーションおよび使用方法の基本を理解する必要があります。規則を操作するとき、初期状態の BPE インタフェースは、表示区画、メニューバー、操作メニュー、および「Rule」ダイアログで構成されます。

注	BPE のインタフェースは、オブジェクトタイプまたはプロセスの選択に応じて変化します。

この節では、規則の作成と編集に関係するインタフェースについて説明します。説明する内容は次のとおりです。

BPE の表示区画

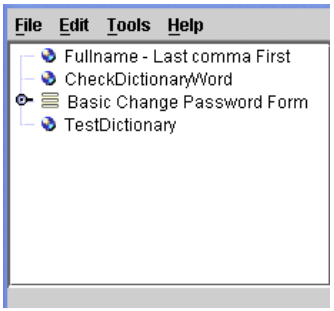
規則を操作するとき、BPE のインタフェースには次の表示区画があります。

- ツリービュー
- Rule source
- 「Input」タブ
- 「Trace」タブ
- 「Result」タブ

ツリービュー

インタフェースの左区画のツリービューには、選択した規則がスタンドアロンのアイコンとして一覧表示されます。

図 A-20　　ツリービューでの規則表示



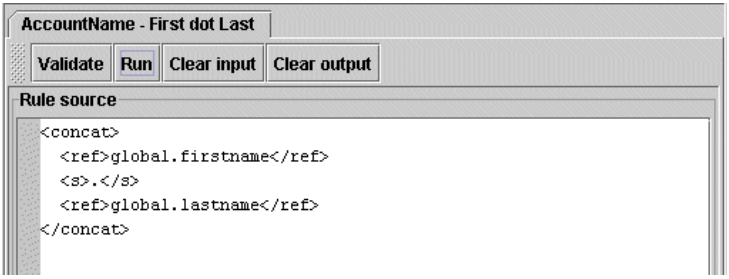
一般に、ツリービューにはタスク、フォーム、またはビューの階層が表示されます。階層では各要素が順番に表示され、親要素の下にサブ要素が入れ子にされます。

ただし、(規則ライブラリオブジェクト、ワークフロー、またはフォームにすでに取り込まれている場合を除いて) 規則は Identity Manager 内部の階層内に存在しないため、ツリービューに表示される規則間には階層関係はありません。その代わりに、ライブラリ、ワークフロー、またはフォームに取り込まれない規則は、単一のアイコンとしてツリービューに表示されます。

Rule source

インタフェース内の右上部分の「Rule source」区画には、規則のソース情報が表示されます。

図 A-21　　「Rule source」区画



この区画では、右クリックしてカスケードメニューを表示し、次のタスクを実行できます。

- 新しい規則を作成する、または選択した規則に新しい値を追加する
- 既存の規則およびライブラリを、参照および選択する
- 既存の Javadoc を参照および表示する

- 規則ソースの表示形式を XML、グラフィカル、プロパティーシート、または設定の間で切り替える

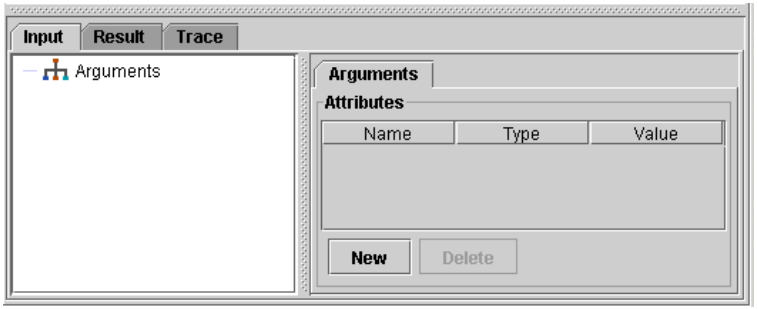
この区画の上にあるボタンを使用して、次の操作を実行することもできます。

- 「Validate」: 現在の引数セットを使用して規則を検証する
- 「Run」: 現在の引数セットを使用して規則を実行する
- 「Clear the input」: 入力引数をデフォルトにリセットする
- 「Clear the output」: 「Result」および「Trace」区画をクリアする

「Input」タブ

「Input」タブ区画は、ウィンドウの右下隅にデフォルトで表示されます。

図 A-22　「Input」タブ区画



このタブを使用して、テストのために規則に渡される引数を制御できます。このタブは基本的には、BPE の汎用オブジェクトエディタ (320 ページの「汎用オブジェクト」を参照) と同じです。

この区画では、次の操作を実行できます。

- 引数名をダブルクリックすると、「Arguments」ダイアログが表示され、引数の検証を実行できます。
- 引数名を右クリックしてカスケードメニューを表示し、ビューまたはファイルからテストデータをインポートできます。特に、次のタスクを実行できます。
  - List、GenericObject、Map、または Test データに引数を挿入する
  - 引数を編集する
  - 引数をコピーする
  - コピーした引数を別の場所に貼り付ける
  - ファイルからテストデータをインポートする

- テストデータをファイルにエクスポートする

- 「New」をクリックし、名前、型、および値を指定して、新しい引数を作成します。
- 「Delete」をクリックして、選択した引数を削除します。

「Result」タブ

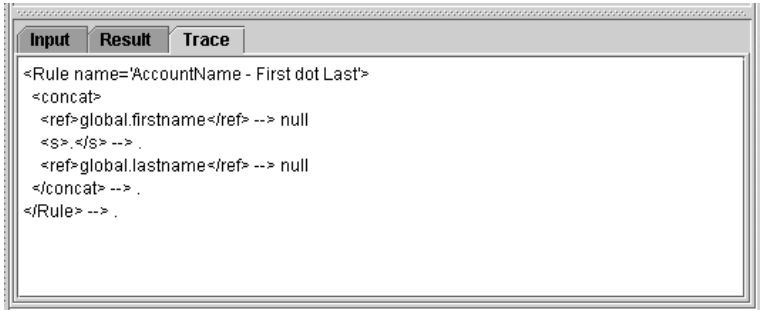
「Result」タブを選択し、「Rule source」区画の上にある「Run」をクリックすると、選択した規則を実行できます。「Result」タブ区画には、規則の戻り値が XML 形式で表示されます。

図 A-23　「Result」タブ区画

「Trace」タブ

「Trace」タブを選択して、規則の実行中に XPRESS トレースをキャプチャーします。

図 A-24　「Trace」タブ区画



メニュー選択

メニューバーまたは操作 (右クリック) メニューを使用して、インタフェース内で作業を実行できます。

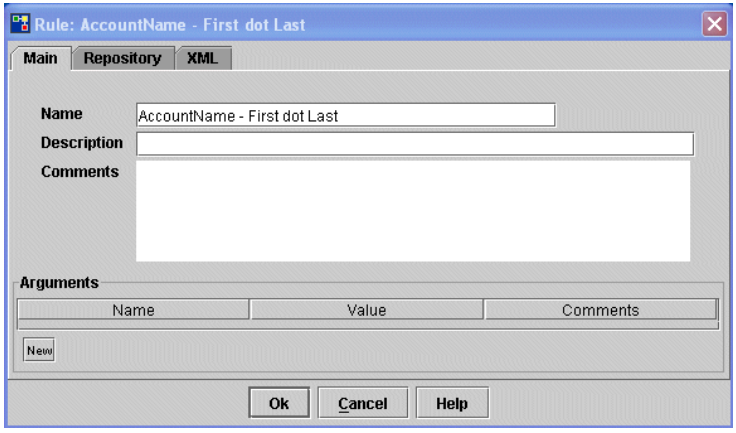
ツリービューまたはダイアグラムビューで項目を選択して右クリックすると、その項目に対して実行できる操作がメニュー項目として表示されます。

「Rule」ダイアログ

個々の規則および規則要素には、要素の型および特性を定義するために使用できるダイアログが関連付けらています。

これらのダイアログにアクセスするには、ツリービューで規則名を右クリックします。選択した規則の「Rule」ダイアログが表示されます。デフォルトでは「Main」タブが前面に表示されます。たとえば、次の図のようになります。

図 A-25　「Rule」ダイアログ (「Main」タブビュー)



このダイアログの次のオプションを使用して、規則を定義します。

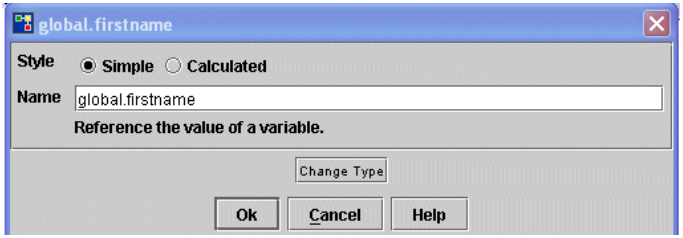
- 「Name」: 選択した規則の名前が自動的に表示されます。これは、Identity Manager のインタフェースに表示される名前です。
- 「Description」(省略可能): 規則の目的を説明するテキストを指定します。
- 「Comments」: `<Comment>` 要素を使用して規則の本体に挿入されるテキストを指定します。
- 「Arguments」: 必要な引数を指定します。

規則要素の編集 (フィールド値の型の変更)

フィールド値の型の選択によっては、ダイアログ内の一部のフィールドの動作が異なる場合があります。

- 値の型が `String` の場合、フィールドにテキストを直接入力できます。
- 値の型が `Expression`、`Rule`、または `Reference` の場合、「Edit」をクリックして値を編集します。

図 A-26　「Rule Argument」ダイアログ



次のいずれかの方法で、値の型を変更できます。

- 「Edit」、「Change Type」の順にクリックします (現在の値が `String` 型の場合)。
- 右クリックして操作メニューを表示し、「Change Type」を選択します (現在の値が `Expression`、`Rule`、または `Reference` 型の場合)。

表示タイプの変更

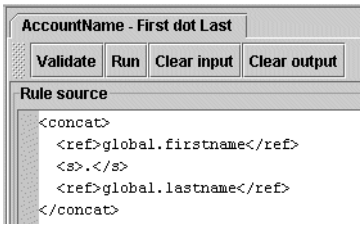
ダイアグラムビューでの情報表示形式を変更するには、次の手順に従います。

1. 右クリックして操作メニューを表示します。
2. 「Display」を選択し、表示タイプを選択します。

   表示タイプには次のものがあります。

   - 「XML」- XPRESS または JavaScript ソースを表示します。XML ソースを直接編集する場合は、この表示タイプを選択します。

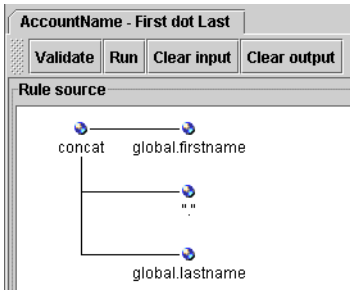
図 A-27　XML 表示



- 「Graphical」- 式ノードのツリーを表示します。この表示タイプでは、構造の概要を確認できます。

  注 スペースの都合のため、図 A-28 には選択した規則の一部のみを示しています。

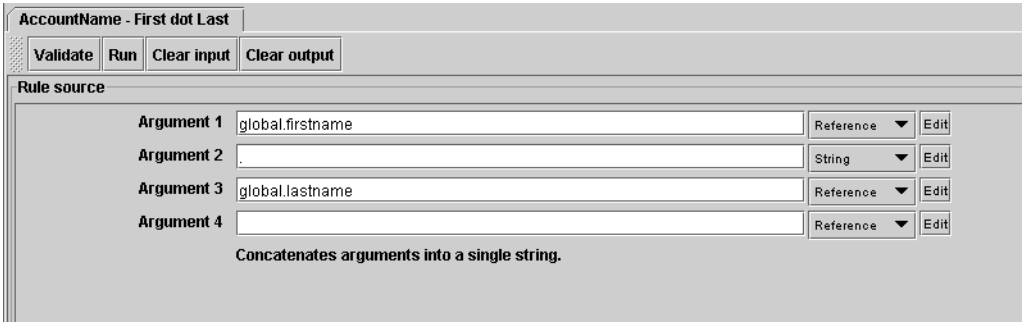
図 A-28 グラフィカル表示



- 「Property Sheet」- プロパティーを一覧表示します。一部のプロパティーは直接編集できます。

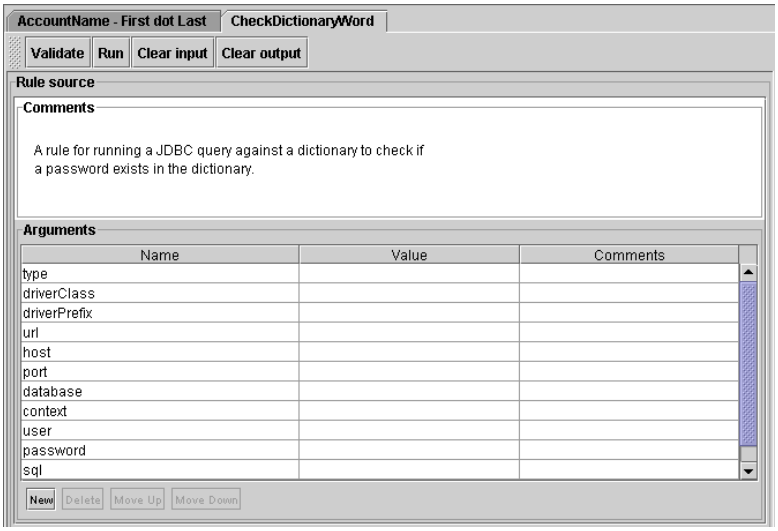
  その他のプロパティーについては、別のダイアログを開くことが必要な場合があります。
  新しい式を作成するときは、「Property Sheet」表示タイプを使用すると効率的に作業できます。このビューでは、グラフィカルビューを使用する場合と比べて、式の引数をすばやく入力できます。

図 A-29 プロパティーシート表示



- 「Configuration」- 引数の情報をプロパティーシート形式で一覧表示します (図 A-30 を参照)。加えて、ルールの作成者が、データベース内でルールを説明するために使用したコメントも表示されます。

図 A-30　　設定表示



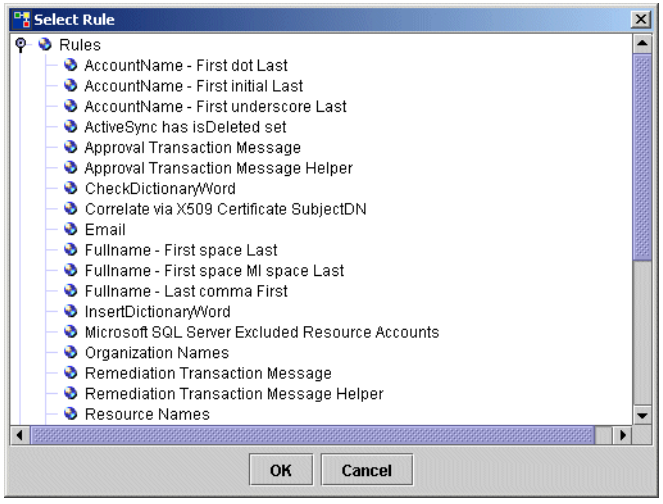
規則の参照

Identity Manager からアクセスできる規則を参照および選択するには、次の方法を使用します。

- メインのメニューバーから「File」>「New Repository Object」の順に選択します。「Select objects to edit」ダイアログが表示されたら、「Rule」ノードを展開し、編集可能な規則を表示します。
- 「Rule source」区画内で右クリックし、操作メニューから「New」>「Browse Rules」の順に選択します。

「Select Rule」ダイアログ (図 A-31) が表示されたら、「Rule」ノードを展開し、編集可能な規則を選択します。

図 A-31　「Select Rule」ダイアログ



規則要約の詳細の検討

ツリー区画で規則名をダブルクリックすると、規則の要素が一覧表示されます。「Rule」ダイアログには、次のタブがあります。

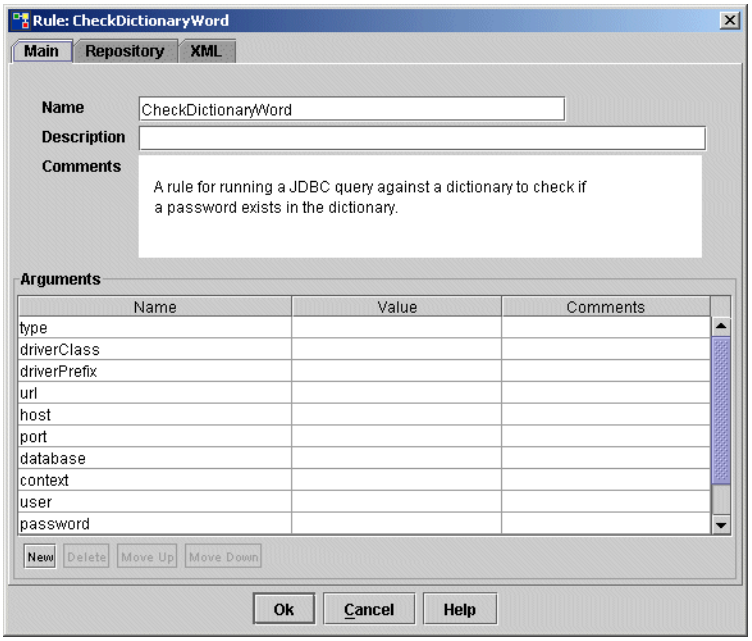
- Main
- Repository
- XML

「Main」タブ

このタブを選択すると、要素の引数プロパティー (各引数の名前や値など) にアクセスできます。見やすくするために、引数の順序を変更することもできます。このリストで順序を変更しても、規則の解釈は変わりません。

「Main」タブでは、規則に関して、「Rule」ダイアログの「Main」ビュー (図 A-32 を参照) と同じ情報が表示されます。

図 A-32　「Main」タブ表示



「Repository」タブ

注	規則ライブラリに含まれていない規則には、「Repository」タブがあります。

「Repository」タブを選択すると、選択した規則についての次の情報を表示できます。

図 A-33　「Repository」タブ表示

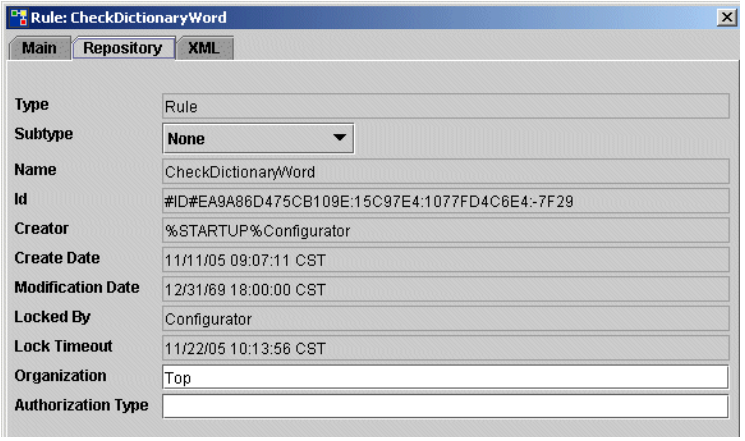


表 A-2　「Repository」タブのフィールド

フィールド	説明
Type	リポジトリオブジェクトのタイプを特定します。この値は常に「Rule」です。
Subtype	サブタイプを識別します (存在する場合)。規則のサブタイプは現在、Reconciliation インタフェースの内部でのみ実装されています。 デフォルトは「None」です。
Name	「Rule」ダイアログの名前フィールドで割り当てられます。
Id	Identity Manager によって割り当てられる識別番号。
Creator	規則を作成したアカウントを一覧表示します。
CreateDate	オブジェクトの作成時に Identity Manager によって割り当てられた日付。
Modification Date	オブジェクトが最後に変更された日付。
Organization	規則が保存される組織を識別します。

表 A-2　「Repository」タブのフィールド (続き)

フィールド	説明
Authorization Type	(省略可能) 管理権限を持たないユーザーに、個別操作のアクセス権を付与します。たとえば、EndUserRule 認証タイプは、Identity Manager ユーザーインタフェース内のフォームから規則を呼び出す権限をユーザーに付与します。

「Repository」タブに含まれるのは、主に読み取り専用の情報ですが、次の値は変更が可能です。

- 「Subtype」: 新しいサブタイプ割り当てをメニューから選択します。

  デフォルトでは、規則にはサブタイプはありません。そのため、規則を作成するとき、「Subtype」の値はデフォルトで「None」です。

  ただし、この規則を Reconciliation インタフェースで表示するには、Reconciliation のグラフィカルユーザーインタフェースで、どの選択リストに規則を表示させるかに応じて、この値を「Account Correlation」または「Account Confirmation」に設定する必要があります。

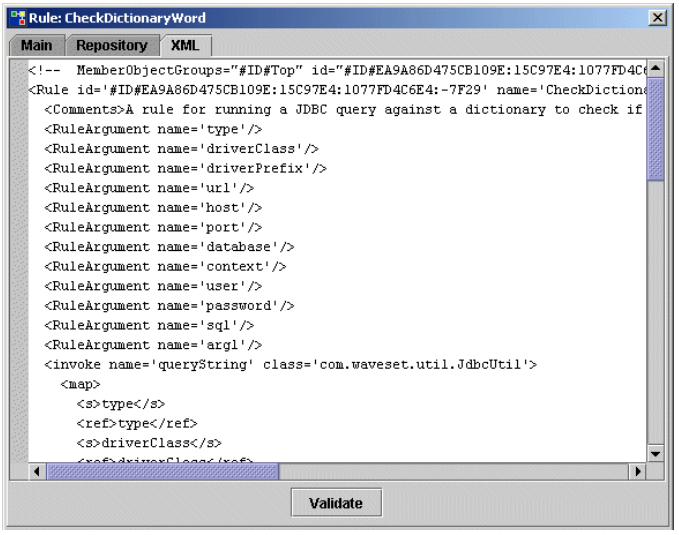
- 「Organization」: 新しい組織割り当てをテキストフィールドに入力します。
- 「Authorization Type」: 新しい認証タイプをテキストフィールドに入力します。

注　規則のサブタイプの例については、133 ページの「リソースアカウント除外規則サブタイプ」を参照してください。

「XML」タブ

「XML」タブを選択すると、選択した XML のコードを表示して直接編集できます。「OK」をクリックして保存する前に、「Validate」をクリックすると、変更内容を検証できます。XML パーサー は、`waveset.dtd` を使用して規則の XML を検証します。

図 A-34　「XML」タブ表示

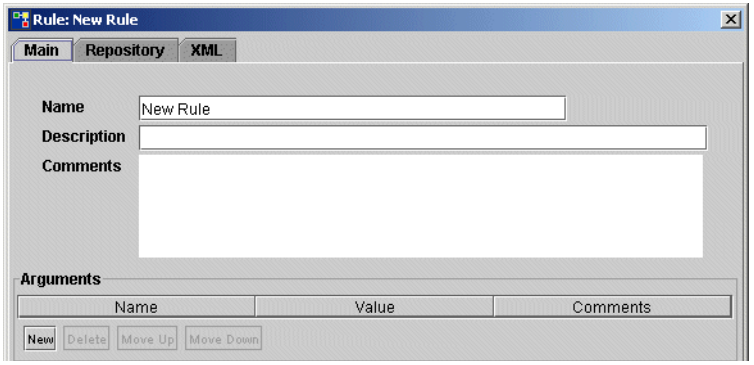


新しい規則の作成

新しい規則を作成するには、次の手順に従います。

1. 「File」>「New」>「Rule」の順に選択します。「Rule: New Rule」ダイアログが表示されます。デフォルトでは「Main」タブが前面に表示されます。

図 A-35　「Rule: New Rule」ダイアログ



2. 新しい規則の次のパラメータを指定します。

- 「Name」- 規則の名前を入力します。この名前は Identity Manager のインタフェースに表示されます。
- 「Description」(省略可能) - 規則の目的を説明するテキストを入力します。
- 「Comments」- <Comment> 要素を使用して、規則の本体に挿入されるテキストを入力します。

3. 新しい規則に引数を追加するには、「New」をクリックします。
4. 「Argument: Null」ダイアログが表示されたら、「Name」、「Value」、および「Comments」の各フィールドにテキストを入力して「OK」をクリックします。

   このテキストは「Arguments」テーブルに表示され、<RuleArgument> 要素として規則に挿入されます。
5. 終了したら、「OK」をクリックして変更を保存します。

注

- 引数を削除するには、「Delete」をクリックします。
- 「Arguments」テーブル内での引数の位置を変更するには、「Move Up」または「Move Down」をクリックします。

規則要素の定義

関数、XPRESS 文、複数のデータ型のうちの 1 つを、規則を構成する XML 要素にすることができます。次に示す BPE の「Rule Element」ダイアログを使用して、規則要素を作成または編集できます。

- **「Arguments」ダイアログ** - 引数の特性を表示または定義します。
- **「Element」ダイアログ** - 選択した要素を表示または定義します。
- **「Object Access」ダイアログ** - オブジェクトの操作や、オブジェクトを操作する Java メソッドの呼び出しを行います。
- **「Diagnostics」ダイアログ** - JavaScript、トレース、出力、ブレークポイントのデバッグまたは検証を行います。

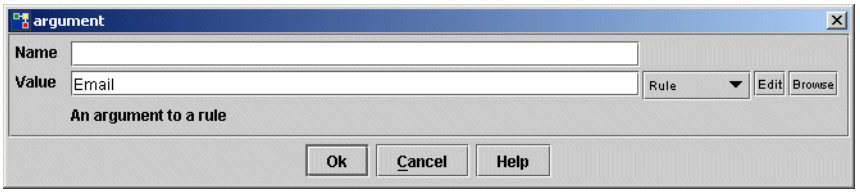
以降の節では、これらの各ダイアログについて詳しい情報を示します。

注 規則の構造の詳細は、139 ページの「規則構文について」を参照してください。

「引数」ダイアログ

「Argument」ダイアログを使用して、規則要素にアクセスしたり、規則要素を定義したりできます。

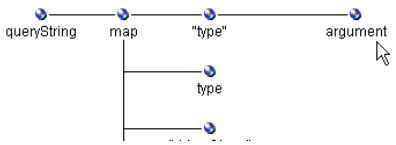
図 A-36　「Argument」ダイアログ



「Argument」ダイアログを開くには、次のいずれかの方法を使用します。

- ツリービュー区画で規則名をダブルクリックして「Rule」ダイアログを開き、「Main」タブで引数名をダブルクリックします。
- 「Rule source」区画 (グラフィカルビューのみ) 内で右クリックし、「New」>「Rule」>「Argument」の順に選択します。
- 「Rule source」区画 (グラフィカルビューのみ) で、引数ノードをダブルクリックします。

図 A-37　引数ノードのダブルクリック



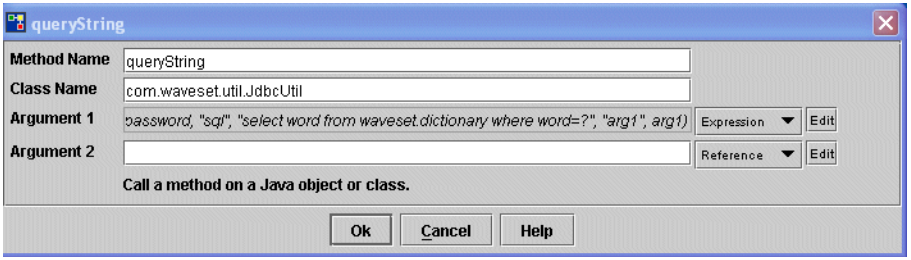
「Argument」ダイアログには、次の基本オプションがあります。

- 「Name」- 引数の名前を指定します。このダイアログで名前を変更できます。
- 「Value」- 選択した引数の名前を指定します。
- 「Comments」- 省略可能なコメントを指定します。

これらのオプションに加えて、表示または編集対象の要素のタイプによっては、その他のフィールドが「Argument」ダイアログに表示される場合があります。

たとえば、メソッド要素を表示している場合、クエリーメソッドに対して表示されるもの類似した、次のような「Argument」ダイアログが表示されます。

図 A-38　「Argument Popup」ダイアログ (メソッド)



引数のデータ型を変更するには、「Change Type」ボタンをクリックして「Select Type」ダイアログを表示します。

図 A-39　「Select Type」ダイアログ

次の表に、有効な引数の型の一覧を示します。

表 A-3　有効な引数の型

データ型	説明
String	単純文字列定数。
Reference	変数への単純参照。
規則	ルールへの単純参照。
List	XML オブジェクトリストなどの静的リスト。この型はワークフローで頻繁に使用されますが、フォームではほとんど使用されません。
Expression	複合式。
Map	XML オブジェクトマップなどの静的マップ。ほとんど使用されません。

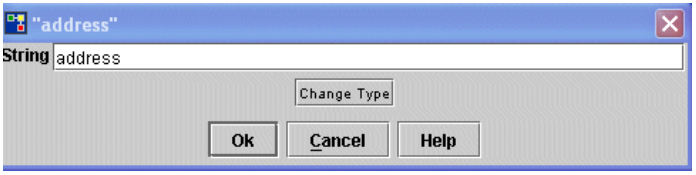
表 A-3　有効な引数の型 (続き)

データ型	説明
Integer	整数型の定数。値のセマンティクスをより明確にするために使用できます。文字列として指定できます。BPE により、文字列が正しい型に強制変換されます。
Boolean	ブール型の定数。値のセマンティクスをより明確にするために使用できます。文字列として指定できます。BPE により、文字列が正しい型に強制変換されます。Boolean 型の値は、文字列 `true` および `false` を使用して指定できます。
XML オブジェクト	複合オブジェクト。XML 表現を使用して、多数の複合オブジェクトの中から任意のものを指定できます。例には EncryptedData、Date、Message、TimePeriod、WavesetResult などがあります。

「Element」ダイアログ

「Element」ダイアログには、引数の名前と値が表示されます。

図 A-40　address 変数の要素ポップアップ



(グラフィカルビューのみ) 「Rule source」区画から「Element」ダイアログを表示するには、次のいずれかの手順に従います。

- 要素アイコンをダブルクリックする
- 要素アイコンを右クリックし、操作メニューから「Edit」を選択する

引数名をクリックしてダイアログを開く場合、引数のデータ型およびスタイル (「simple」または「calculated」) を変更できます。

さまざまなタイプの要素を定義できます (表 A-4 を参照)。要素のデータ型を変更するには、「Change Type」ボタンをクリックします。「Select Type」ポップアップが開き、選択した規則の要素に割り当てることのできるデータ型の一覧が表示されます。

新しい要素を作成するには、グラフィカルビューで右クリックし、メニューから「New」を選択し、要素タイプを選択します。このメニューに表示される要素タイプは、XPRESS 関数のカテゴリを表します。

表 A-4　XPRESS 関数カテゴリを表す要素タイプ

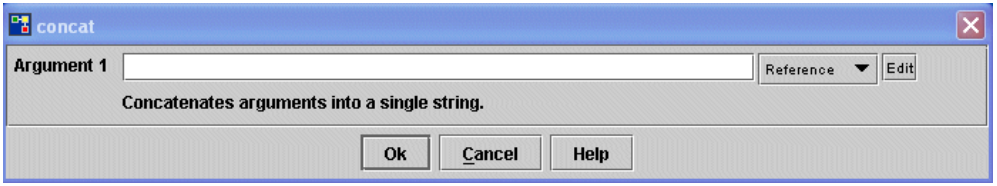
メニューオプション	XPRESS 関数 / 呼び出し可能な追加操作 ...
Values	`string`、`integer`、`list`、`map`、`message`、`null`
Logical	`if`、`eq`、`neq`、`gt`、`lt`、`gte`、`lte`、`and`、`or`、`not`、`cmp`、`ncmp`、`isnull`、`notnull`、`isTrue`、`isFalse`
String	`concat`、`substr`、`upcase`、`downcase`、`indexOf`、`match`、`length`、`split`、`trim`、`ltrim`、`rtrim`、`ztrim`、`pad`
Lists	`list`、`map`、`get`、`set`、`append`、`appendAll`、`contains`、`containsAny`、`containsAll`、`insert`、`remove`、`removeAll`、`filterdup`、`filternull`、`length`、`indexOf`
Variables	• 変数を定義する • 参照を作成する • 変数またはオブジェクトの属性に値を代入する
Math	`add`、`sub`、`mult`、`div`、`mod`
Control	`switch`、`case`、`break`、`while`、`dolist`、`block`
Rule	• 新しい規則を作成する • 引数を作成する
Other	その他のオプションを表示する :
(functions, object access, diagnostics)	• 関数には関数の定義、引数の定義、関数の呼び出しが含まれる • オブジェクトアクセスには `new`、`invoke`、`getobject`、`get`、および `set` の各関数が含まれる • 診断には、JavaScript の作成または呼び出し、トレース、出力、およびブレークポイント関数のためのオプションが含まれる

注　これらの関数の詳細については、『Sun Java™ System Identity Manager ワークフロー、フォーム、およびビュー』を参照してください。

BPE セッションで最近作成された要素タイプには、操作メニューの「Recent」オプションからもアクセスできます。

次の図では、「New」>「Strings」>「concat」の順に選択したときに表示される ウィンドウを示します。

図 A-41　「concat」ダイアログ



「Object Access」ダイアログ

「Object Access」ダイアログを使用して、オブジェクトの操作や、オブジェクトを操作する Java メソッドの呼び出しを実行できます。

「Object Access」ダイアログを開くには、グラフィカル表示内で任意の場所を右クリックし、ポップアップメニューから「New」>「Other」>「Object Access」の順に選択し、操作オプションを選択します。

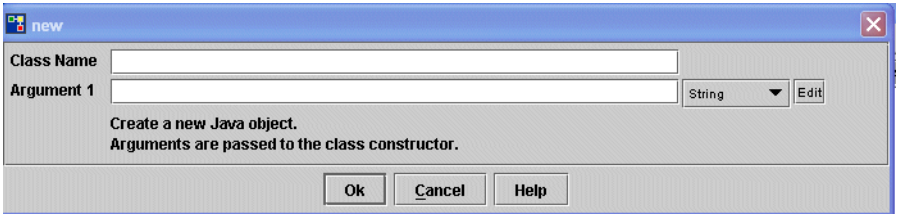
操作オプションとしては、表 A-5 で説明されているオプションのいずれかを選択できます。

表 A-5　オブジェクトアクセスのオプション

オプション	説明
new	新しい Java オブジェクトを作成します。引数はクラスコンストラクタに渡されます。
invoke	「invoke」ダイアログを表示します。Java オブジェクトまたは Java クラスに対して Java メソッドを呼び出すために使用します。
getobj	「getobj」ダイアログを表示します。リポジトリからオブジェクトを取得するために使用します。
get	オブジェクトの内部から値を取得します。 最初の引数は List、GenericObject、または Object である必要があり、2 番目の引数は String または Integer である必要があります。 • 最初の引数が List の場合、2 番目の引数は整数に強制変換され、リストのインデックスとして使用されます。 • 最初の引数が GenericObject の場合、2 番目の引数は文字列に強制変換され、パス式として使用されます。 • 最初の引数がその他の任意のオブジェクトである場合、2 番目の引数には JavaBean プロパティーの名前が想定されます。
set	変数またはオブジェクトの属性に値を代入します。

オブジェクトを作成するには、右クリックして操作メニューを表示し、「New」>「Other」>「Object Access」>「new」の順に選択します。

図 A-42　「new」ダイアログ

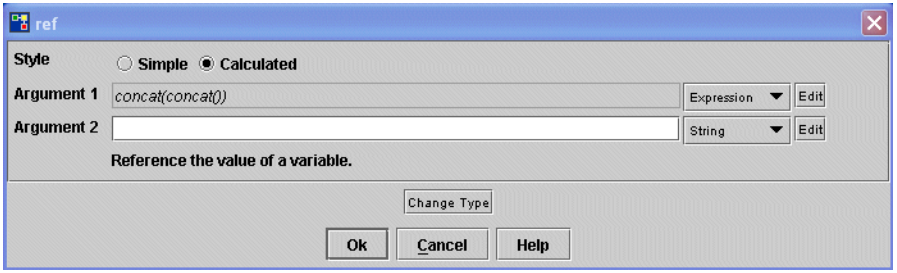


要素詳細の編集

「Argument」ダイアログから、変数の値を定義できます。「Value」フィールドを使用して、単純文字列値を変数の初期値として入力します。代わりに、(Expression や Rule などの) 値の型を選択してから、「Edit」をクリックして値を入力する方法もあります。

図 A-43 は、`ref` 文の変数ウィンドウを示します。

図 A-43　「ref」ダイアログ



引数のタイプ (単純または複合) を指定できます。値が文字列型、ブール型、整数型などであり、テキストフィールドに引数の値を入力できる場合は「simple」を選択します。追加のポップアップが必要なリスト、XML オブジェクト、その他の式などを扱っている場合は、「calculated」を選択します。

「Diagnostics」ダイアログ

「Diagnostics」ダイアログを使用して、次の要素のデバッグまたは検証を実行できます。

- JavaScript

- トレース
- 出力
- ブレークポイント

「Diagnostics」ダイアログにアクセスするには、右区画内で操作メニューから「New」>「Other」>「Diagnostics」>「trace」の順に選択します。表 A-6 で説明されているオプションを選択して、その項目をデバッグします。

表 A-6　トレースオプション

オプション	説明
JavaScript	独自の JavaScript を入力できる「Script」ダイアログを表示します。
trace	XPRESS 関数 `<trace>` を規則に挿入します。この関数は、この規則の評価時に XPRESS トレースを有効または無効にします。トレースを有効にする場合は `true` に、無効にする場合は `false` (または単に null) に設定します。
print	`tan` 引数の名前を入力できる「Print」ダイアログを表示します。この関数は、任意の数の式を含み、最後の式の結果を返す点で、`block` 関数に似ています。 「**Argument**」フィールドに引数名を入力し、フィールドの隣にあるメニューから型を選択します。デフォルトは `String` です。
breakpoint	「Breakpoint」ポップアップを表示します。「**OK**」をクリックしてデバッグブレークポイントを設定します。

規則の編集

規則をカスタマイズする場合、変更を保存および検証して、規則が正確かつ予測どおりに完了することを確認する必要があります。保存したあとで、変更した規則を Identity Manager で使用するためにインポートします。

この節では、次の手順を説明します。

- 規則のロード
- 変更の保存
- ワークフローリビジョンの検証

規則のロード

BPE で規則をロードするには、次の手順に従います。

1. メニューバーから「File」>「Open Repository Object」の順に選択します。

2. 表示された「Login」ダイアログで入力を求められたら、Identity Manager Configurator のユーザー名とパスワードを入力して「Login」をクリックします。

   次の項目が表示されます。

   - ワークフロープロセス
   - ワークフローサブプロセス
   - フォーム
   - 規則
   - 電子メールテンプレート
   - ライブラリ
   - 汎用オブジェクト
   - 設定オブジェクト
   - メタビュー

   注 表示される項目は、Identity Manager の実装によって異なる場合があります。

3. 「Rule」ノードを展開して、すべての既存の規則を表示します。
4. ロードする規則を選択して「OK」をクリックします。

   注 規則をはじめてロードする場合、右区画に表示される規則コンポーネントの表示が正しくない場合があります。右区画内で右クリックして「Layout」を選択すると、図が再表示されます。

変更の保存

規則への変更を保存してリポジトリにチェックインするには、メニューバーから「File」>「Save in Repository」の順に選択します。

注 「File」>「Save As File」の順に選択して、規則を XML テキストファイルとして保存することもできます。ファイルへの保存はファイル名 `.xml` の形式で行います。

変更の検証

カスタマイズプロセスの各段階で、規則への変更を検証できます。

- 「Rule source」区画で「Validate」ボタンをクリックすると、現在の引数のセットを使用して規則を検証できます。
- XML 表示値を操作している場合は、引数の追加またはカスタマイズ時に「Validate」をクリックすると、ルールへの個々の変更を検証できます。
- 変更を行なったあとに、ツリービューで規則を選択し、「Tools」>「Validate」の順に選択してテストを実行します。

BPE では、規則のステータスを示す検証メッセージが表示されます。

- **警告インジケータ (黄色のドット)** - プロセスの操作は有効だが、構文スタイルが最適ではないことを示します。
- **エラーインジケータ (赤のドット)** - プロセスが正常に実行されないことを示します。プロセスの操作を修正する必要があります。

規則ライブラリ

規則ライブラリは、密接に関係する規則を、Identity Manager リポジトリ内の 1 つのオブジェクトに整理するための便利な手段として機能します。ライブラリを使用すると、リポジトリ内のオブジェクト数が削減され、フォームやワークフローの設計者は有用な規則を簡単に特定して呼び出せるようになるため、規則の保守が容易になります。

規則ライブラリは、XML の 設定オブジェクトとして定義されます。設定オブジェクトには、1 つ以上の規則オブジェクトを含む、ライブラリオブジェクトが含まれます。コード例 A-1 は、2 つの異なるアカウント ID 生成規則を含むライブラリを示します。

コード例 A-1 2 つのアカウント ID 生成規則を含むライブラリ

```
<Configuration name='Account ID Rules'>
   <Extension>
      <Library>
         <Rule name='First Initial Last'>
            <expression>
               <concat>
                  <substr>
                     <ref>firstname</ref>
                     <i>0</i>
                     <i>1</i>
                  </substr>
                  <ref>lastname</ref>
               </concat>
            </expression>
```

コード例 A-1　　2 つのアカウント ID 生成規則を含むライブラリ (続き)

```
          </Rule>
          <Rule name='First Dot Last'>
             <expression>
                <concat>
                   <ref>firstname</ref>
                   <s>.</s>
                   <ref>lastname</ref>
                </concat>
             </expression>
          </Rule>
       </Library>
    </Extension>
</Configuration>
```

ライブラリ内の規則は、XPRESS の `<rule>` 式を使用して参照します。name 属性の値は、ライブラリを含む設定オブジェクトの名前と、ライブラリ内部での規則の名前をコロンで連結した形式です。

たとえば次の式は、`Account ID Rules` という名前のライブラリに含まれる、`First Dot Last` という名前の規則を呼び出します。

```
<rule name='Account ID Rules:First Dot Last'/>
```

表示またはカスタマイズするライブラリの選択

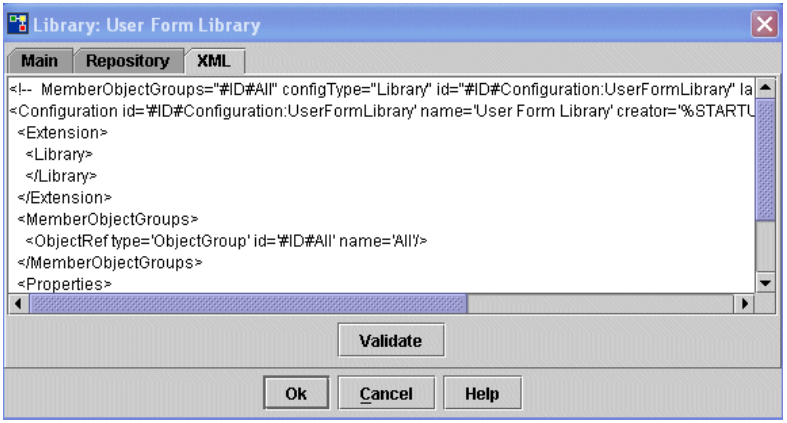
表示または編集する規則ライブラリを選択するには、次の手順に従います。

1. ビジネスプロセスエディタで、「File」>「Open Repository Object」の順に選択します。

   BPE では、規則ライブラリは次のアイコンで表されます。 

2. ツリービューで規則ライブラリオブジェクトを選択して「Edit」を選択します。
3. 右側の編集区画内で右クリックしてから、「XML」タブを選択します。

図 A-44　　規則ライブラリ (XML ビュー)



これで、規則ライブラリの XML を編集できます。

既存のライブラリオブジェクトへの規則の追加

規則ライブラリをチェックアウトしたあとで、<Library> 要素の内部のどこかに <Rule> 要素を挿入することにより、新しい規則を追加できます。ライブラリ内部での規則の位置は重要ではありません。

ワークフロープロセスのカスタマイズ

ここでは、「Email Notification」の例を使用して、ワークフロープロセスをカスタマイズするために実行する手順全体を説明します。具体的には、次のことを行います。

1. カスタムの Identity Manager 電子メールテンプレートを作成します。
2. Identity Manager の「Create User」ワークフロープロセスをカスタマイズして、新しいテンプレートを使用し、会社の新しいユーザーに歓迎の電子メールを送信します。

注

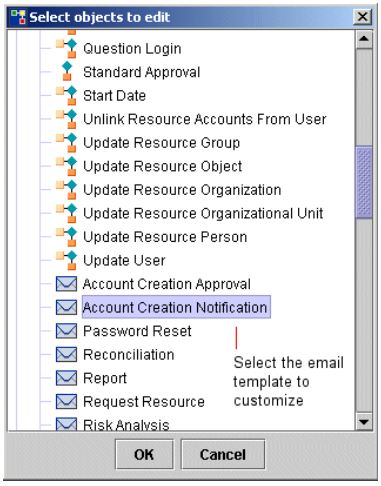
- この例で示す画面例は、プロセスを読み込むときのものとは多少異なる場合があります。結果として、コンポーネントの位置が違っていたり、プロセスの操作の一部が省略されていたりする場合がありますが、この例の目的にとって重要な違いではありません。
- 多くのタスクはツリービューまたはダイアグラムビューから実行できますが、この例では主に BPE のツリービューを使用します。

ステップ 1: カスタム電子メールテンプレートの作成

カスタム電子メールテンプレートを作成するには、次のようにして、既存の Identity Manager 電子メールテンプレートを開いて変更します。

1. BPE のメニューバーから、「File」>「Open Repository Object」>「Email Templates」の順に選択します。
2. 「selection」ダイアログ (図 A-45) が表示されたら、「Account Creation Notification」テンプレートを選択して「OK」をクリックします。

図 A-45　　電子メールテンプレートの選択



3. 選択した電子メールテンプレートが BPE で表示されたら、テンプレート名を右クリックし、ポップアップメニューから「Copy」を選択します。
4. もう一度右クリックして「Paste」を選択します。

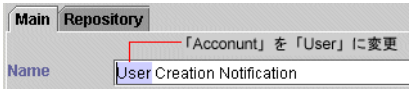
   電子メールテンプレートのコピーがリストビューに表示されます。

ヒント　貼り付けを行うときは、マウスカーソルが項目を覆っていないことと、どの項目も選択されていないことを確認してください。これらの条件が満たされていない場合、貼り付け操作は無視されます。

5. リストビューで新しい電子メールテンプレートをダブルクリックして、テンプレートを開きます。

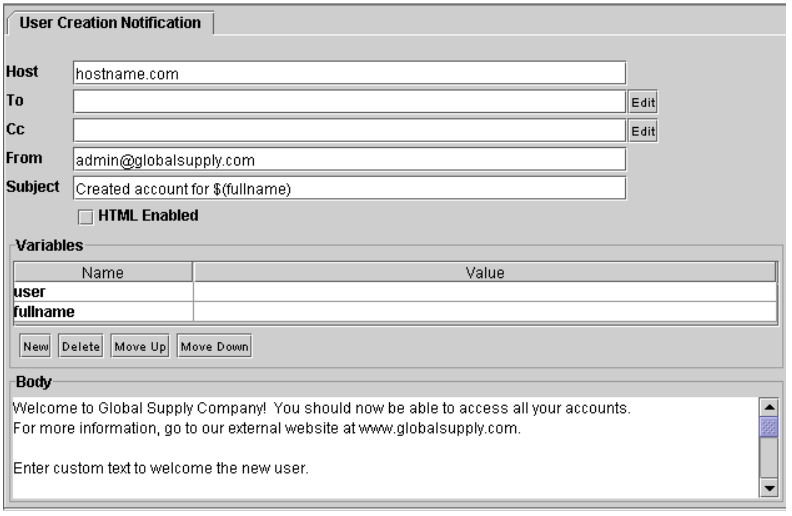
6. 「Name」フィールドに「**User Creation Notification**」と入力して、テンプレート名を変更します。

図 A-46　新しいテンプレートの名前変更



7. 新しく作成した「User Creation Notification」テンプレートで、「Subject」および「Body」フィールドを必要に応じて変更します。

図 A-47　ユーザー作成通知電子メールテンプレートのカスタマイズ



Identity Manager アカウントまたは電子メールアドレスのコンマ区切りのリストを、「Cc」フィールドに追加することもできます。

8. 完了したら、「OK」をクリックします。
9. テンプレートを保存してリポジトリにチェックインするには、メニューバーから「File」>「Save in Repository」の順に選択します。

これで、「Create User」ワークフロープロセスを変更する準備ができました。次の手順に進みます。

ステップ 2: ワークフロープロセスのカスタマイズ

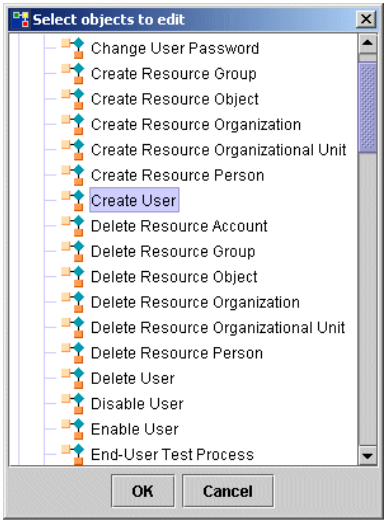
次の手順に従って、新しい電子メールテンプレートを使用するように「Create User」ワークフロープロセスを変更します。

1. BPE で「File」>「Open Repository Object」>「Workflow Processes」の順に選択して、ワークフロープロセスをロードします。

   編集可能な Identity Manager オブジェクトを含むダイアログが表示されます。

2. 「Create User」ワークフロープロセスを選択して「**OK**」をクリックします。

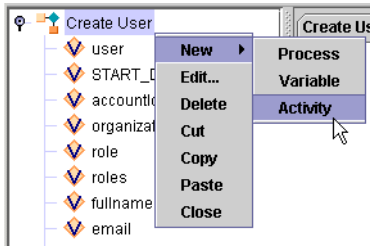
図 A-48　　ワークフロープロセスのロード



「Create User」ワークフローが表示されます。

3. ツリービューで、「Create User」プロセスを右クリックし、ポップアップメニューから「New」>「Activity」の順に選択します。

図 A-49　アクティビティーの作成と命名



ツリービュー内のアクティビティーリストの一番下に、activity1 という名前の新しいアクティビティーが表示されます。

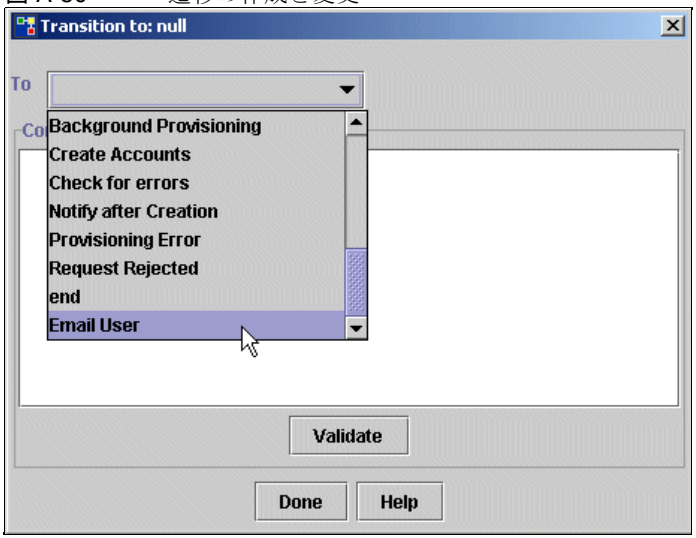
4. activity1 をダブルクリックして「Activity」ダイアログを開きます。
5. 「Name」フィールドに「**Email User**」と入力して、アクティビティー名を変更します。

   デフォルトの「Create User」ワークフローでは、アカウントが作成されたことをアカウント要求者に通知するステップ (Notify) は端まで直接遷移します。

   新しいステップをワークフローに含めるには、この遷移を削除し、新しい遷移 (Notify と Email User の間、および Email User から端へ) を作成し、プロセスが終了する前に新しいユーザーに電子メールを送信する必要があります。
6. 「Notify」を右クリックして「Edit」を選択します。
7. 「Activity」ダイアログの「Transitions」領域で、端を選択して「Delete」をクリックすることにより、その遷移を削除します。
8. 「Transitions」領域で、「New」をクリックして遷移を追加します。

9. 「Transitions」ダイアログが表示されたら、リストから「Email User」を選択して「Done」をクリックします。

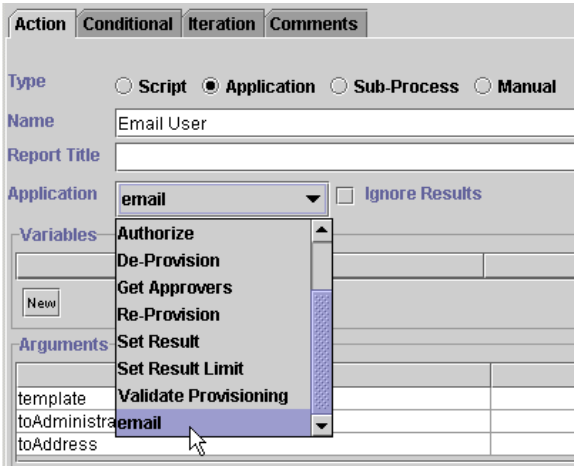
図 A-50　　遷移の作成と変更



10. BPE のツリービューで「Email User」を右クリックし、「New」>「Transitions」の順に選択して遷移を作成し、「Transitions」ダイアログを開きます。
11. 端を選択して「OK」をクリックします。
12. 次に、新しい「Email User」アクティビティーに対して、電子メール操作とその受信者を定義する操作を作成する必要があります。ツリービューで「Email User」を右クリックし、「New」>「Action」を選択して「Action」ダイアログを開きます。
13. 「Type」オプションで「Application」ボタンを選択します。
14. 「Name」フィールドに、新しい操作の名前を入力します。

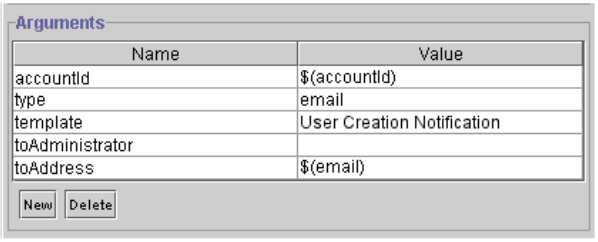
15. 「Application」メニューから「**email**」を選択します。

図 A-51　　操作の作成



16. 「Argument」テーブルに新しい選択が表示されます。次の情報を入力します。
    - template: 新しいテンプレート名「**`User Creation Notification`**」を入力します。
    - toAddress: ユーザーの **`$(user.waveset.email)`** 変数を入力します。
17. 「**New**」をクリックして、引数をテーブルに追加します。引数に **`accountId`** という名前を付け、この引数の値として「**`$(accountId)`**」と入力します。

図 A-52　　操作の作成



18. 完了したら、「OK」をクリックします。
19. BPE のメニューバーから「File」>「Save in Repository」の順に選択して、プロセスを保存し、リポジトリに再びチェックインします。

保存したあとは、Identity Manager を使用してユーザーを作成することにより、新しいプロセスをテストできます。簡潔にするため、ここでは新しいユーザーの承認者またはリソースを選択しません。ユーザーの作成時に新しい歓迎メッセージの到着を確認できるように、自分の電子メールアドレス、または自分が確認できる電子メールアドレスを使用します。

ワークフロー、フォーム、規則のデバッグ

BPE には、ワークフロー、規則、フォーム用のグラフィカルデバッガが含まれています。BPE のデバッガを使用して、ブレークポイントを視覚的に設定したり、ワークフローまたはフォームをブレークポイントまで実行したり、プロセス実行を停止して変数を検証したりできます。

手続き型プログラミング言語のコードデバッガを使用した経験があれば、この節で使用されている用語の理解は難しくありません。

ビュー、ワークフロー、フォームの詳細については、『Sun Java™ System Identity Manager ワークフロー、フォーム、およびビュー』のそれぞれの該当する章を参照してください。

この節では、BPE のデバッガの使用方法を説明します。説明する内容は次のとおりです。

使用にあたっての推奨事項

BPE のデバッガは、次の条件に当てはまる場合にのみ使用してください。

- **開発環境またはテスト環境で使用する**。本稼働環境ではデバッガを使用しないでください。ブレークポイントの設定はグローバル設定であるため、ブレークポイントに到達した時点で着信要求スレッドが中断されます。
- **デバッガ実行権限をユーザーに割り当てる** (この権限は Waveset Administrator 機能の一部として付与される)。デバッガでは、スレッドを中断させることができますが、これによってほかのユーザーがシステムからロックアウトされる可能性があります。また、ほかのユーザーのセッションの変数を表示できますが、この変数に重要なデータが含まれている可能性があります。この権限を悪用すると多大な影響があることを考慮して、権限を割り当てるときには十分に注意してください。
- **ユーザーにはアプリケーションサーバーの非公開コピーを割り当てる**。2 人のユーザーが同じアプリケーションサーバー上で開発を行なっており、一方のユーザーがデバッガをそのサーバーに接続した場合、デバッガのブレークポイントに到達すると、そのサーバーを使用中のもう一方のユーザーがロックアウトされます。

クラスタの使用は、BPE デバッガとの組み合わせではサポートされていません。

テスト環境の外部でのデバッガの実行

デバッグが必要な問題が本稼働環境に見つかった場合は、その問題をテスト環境で再現してデバッグしてください。デバッガでブレークポイントを設定すると、大量のトラフィックが発生している本稼働環境内のアプリケーションサーバーを短時間のうちに停止させる可能性があります。また、ブレークポイントを設定する位置によっては、ユーザーがシステムの利用をブロックされる可能性があります。

独立したテスト環境でデバッグを実行できない場合は、次の手順に従います。

1. クラスタ内のノードのうちの 1 つをオフラインにすることにより、すべての有効なトラフィックをクラスタのサブセットに振り分けます (以後、このタスクの説明では、このノードを server-a とする)。
2. BPE を使用して、システム設定オブジェクトを編集します。`SystemConfiguration serverSettings.server-a.debugger.enabled` プロパティーを `true` に設定します。

   BPE でのシステム設定オブジェクトへのアクセス方法の詳細については、366 ページの「ステップ 2: システム設定オブジェクトの編集」を参照してください。
3. server-a を再起動し、システム設定オブジェクトのプロパティー設定の変更を有効にします。
4. 「Tools」>「Debugger」の順に選択して、デバッガを起動します。

5. 新しいワークスペースを作成します。このワークスペースで、デバッガ接続は次の URL を使用します。

   ```
   server-a:<port>
   ```

   デバッグが完了したら、次の手順に従います。

6. `serverSettings.server-a.debugger.enabled` を `false` に設定し、server-a を再起動して、稼働中の本稼働環境にデバッガが接続しないようにします。
7. server-a をオンラインのクラスタに再統合します。

デバッガの無効化

本稼働環境では、誰かが誤ってデバッガをアプリケーションサーバーに接続することを防ぐために、`serverSettings.server-a.debugger.enabled` プロパティーを常に無効にしてください。

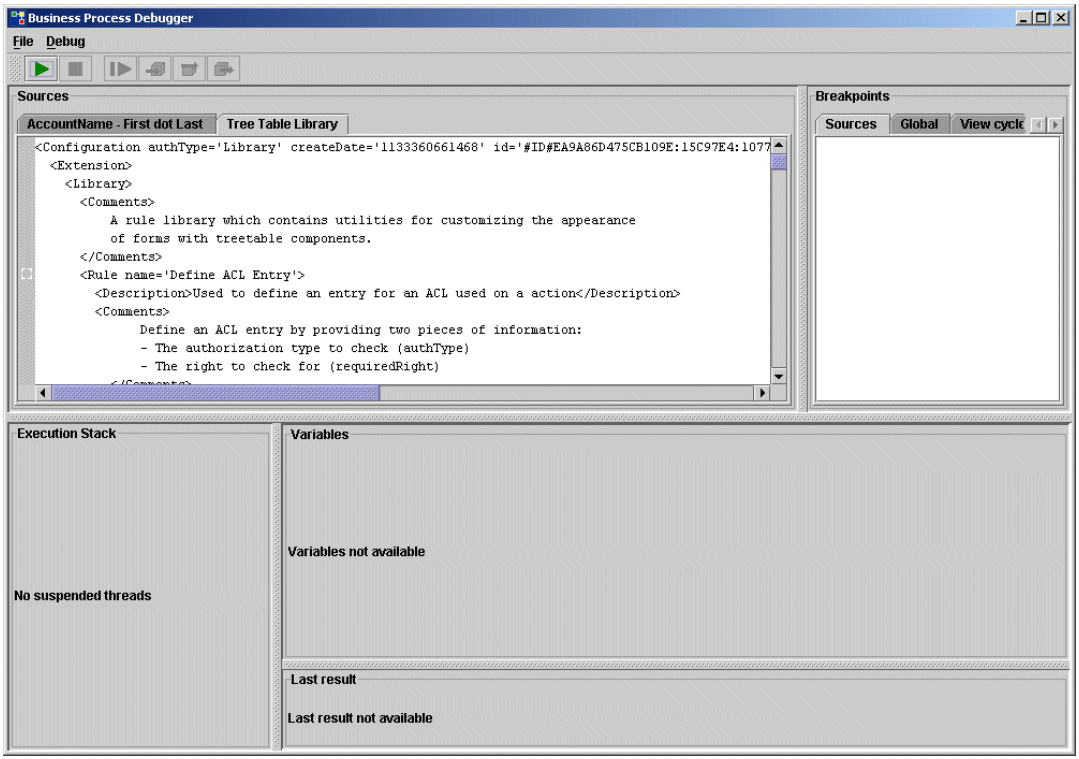
デバッガを無効にするには、システム設定オブジェクトの `serverSettings.<server>.debugger.enabled` プロパティーを `false` に設定します。

デバッガのメインウィンドウの使用

デバッガのメインウィンドウでは、選択したオブジェクトの XML が表示され、そのオブジェクトの実行についての情報が提供されます。このウィンドウから、次の操作を実行できます。

- デバッグプロセスの開始と停止
- プロセス実行のナビゲーション
- プロセス実行内での個別の停止ポイント (ブレークポイント) の設定。ブレークポイントの詳細については、「Setting Breakpoints」を参照してください。

図 A-53　BPE デバッガ : メインウィンドウ



注　BPE のデバッガには、タスクを実行するための多数のキーボードショートカットが用意されています。ショートカットの一覧については、315 ページの「キーボードショートカットの使用」を参照してください。

メインウィンドウには、以降で説明する次の領域が含まれています。

- ソース領域
- 実行スタック
- 「Variables」領域
- 「Variables Not Available」領域
- 「Last Result」領域

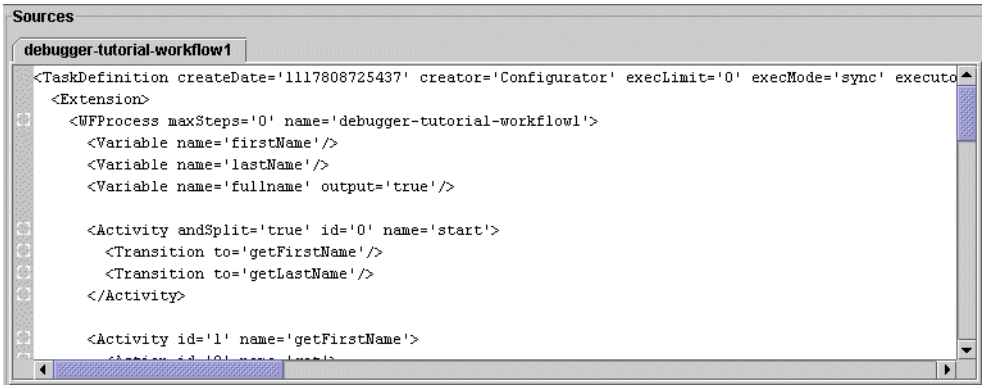
- 「Last Result Not Available」領域
- ブレークポイントの設定

ソース領域

「Sources」領域には、選択したオブジェクトの未フィルタの XML が表示されます。

「XML」パネルの左余白には、ブレークポイントを設定できるコード内のポイントを示す、一連のボックスが表示されます。<WFProcess...> タグのすぐ近くにあるボックスをクリックすると、ワークフローの開始位置にブレークポイントが設定されます。

図 A-54　　BPE デバッガのメインウィンドウの「Source」パネル

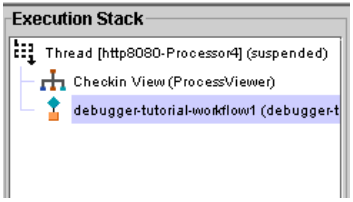


実行スタック

実行スタックは、選択したオブジェクト内のどの関数が実行中であるかを特定します。この領域には、実行中の関数の名前と、その関数を呼び出した関数の名前が一覧表示されます。

追加の関数が呼び出しチェーンに出現する場合、これらの関数は順番に一覧表示されます。このリストは「スタックトレース」とも呼ばれ、プログラムのライフサイクルにおけるこの時点での実行スタックの構造を表示します。

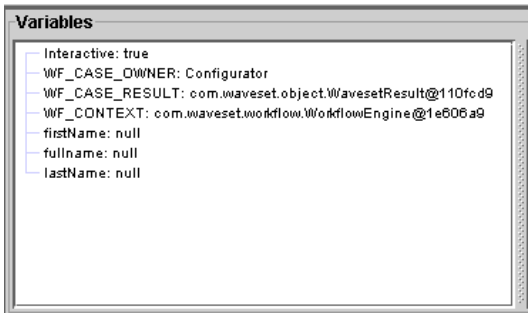
図 A-55　　BPE デバッガのメインウィンドウの「Execution Stack」パネル



「Variables」領域

「Variables」領域には、現在の実行のポイントで、現在スコープ内にあるすべての変数が一覧表示されます。変数オブジェクト名をクリックすると、そのオブジェクトが展開され、各変数の名前が表示されます。

図 A-56　BPE メインウィンドウの「Variables」パネル



「Variables Not Available」領域

「Variables Not Available」領域は、デバッグがアクティブでない場合、または選択したスタックフレームが現在のスタックフレームでない場合に表示されます。

「Last Result」領域

現在の要素が XPRESS の終了タグである場合、「Last Result」領域にはその評価の結果が表示されます。これは、最後の値が意味を持つ、その他のタグにも適用されます。たとえば、`<Argument>` のワークフローへのサブプロセスが評価される過程で、この領域にはその引数の値が表示されます。この領域は、デバッグが現在進行中でない場合は使用できません。

図 A-57　BPE デバッガのメインウィンドウの「Last Result」パネル

「Last Result Not Available」領域

「Last Result Not Available」領域は、デバッグがアクティブでない場合に表示されます。

ブレークポイントの設定

「Breakpoint」は、特定のコード行を実行する前に、オブジェクトの実行を停止するためにデバッガが使用するコマンドです。Identity Manager のデバッガでは、コードのブレークポイントは、フォームまたはワークフローの起動された場所に関係なく適用されます。

ほとんどのデバッガではソース上の位置にしかブレークポイントを設定できませんが、BPE のデバッガでは、「Refresh view」などの概念的な実行ポイントにもブレークポイントを設定できます。この場合、デバッガは「Refresh view」操作が発生した時点で中断します。その後、更新ビューにステップインし、処理が進行中の配下のフォームを確認できます。

ブレークポイントの設定はグローバル設定です。つまり、ブレークポイントを設定すると、指定されたブレークポイントに到達した時点で着信要求スレッドが中断します。これは、どのユーザーが要求を行なっているかに関係なく発生します。

ブレークポイントの設定

ソースの全ブレークポイントの要約を表示するには、「Sources」タブをクリックします。「Breakpoints」区画に、ソースの全ブレークポイントが一覧表示されます。ブレークポイントをクリックすると、特定のブレークポイントに移動します。

ブレークポイントのタイプ

「Breakpoints」領域には、次のタイプのブレークポイント設定があります。

- **グローバルブレークポイント** (「Global」タブ)
- **よく使用するビューに関連付けられたブレークポイント** (「View cycle」タブ)
- **フォーム処理の段階に関連付けられたブレークポイント** (「Form cycle」タブ)

指定されたタブをクリックすることにより、各タイプのブレークポイントにアクセスします。

- 次の方法でコードにブレークポイントを設定するには、「Global」タブを選択します。
  - 「All anonymous breakpoints」: 匿名ソース上にブレークポイントを設定します。
  - 「All named breakpoints」: ステップオーバーおよびステップアウト処理をステップイン処理に変えます。

    ブレークポイントは常に、ステップオーバーおよびステップアウト機能よりも優先されます。その結果、この設定を有効にした場合、実質的にはステップオーバーおよびステップアウト処理をステップイン処理に変えたことになります。一般的な用途では、「All named breakpoints」は、特定のページがど

のフォームまたはワークフローを使用するかが不明な場合に設定します。この設定を有効にする場合、デバッグプロセスでフォームまたはワークフローを特定したあとでただちにこの設定を無効にしてください。そうしないと、すべての実行ポイントのステップスルーを強制されることになります。

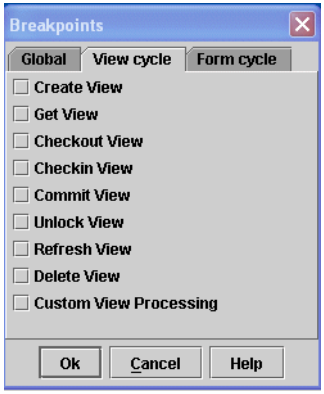
- 両方の設定を有効にすると、デバッガがすべてのブレークポイントをチェックする結果となります。

(省略可能) グローバル設定を選択して「OK」をクリックします。

図 A-58　BPE デバッガの「Breakpoints」パネル : 「Global」タブ

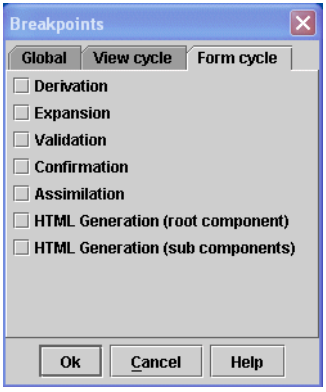
- プロセス実行中に発生するビュー処理に基づいて、コードにブレークポイントを設定するには、「View cycle」タブを選択します。このダイアログには、最も頻繁に呼び出されるビュー操作が一覧表示されます。一覧表示されたそれぞれのビュー操作は、各ビューで使用可能です。

図 A-59　BPE デバッガの「Breakpoints」パネル：「View cycle」タブ



- フォーム処理の指定された段階に基づいて、コードにブレークポイントを設定するには、「Form cycle」タブを選択します。フォーム処理の段階については、『Sun Java™ System Identity Manager ワークフロー、フォーム、およびビュー』を参照してください。

図 A-60　BPE デバッガの「Breakpoints」パネル：「Form cycle」タブ



実行プロセスのステップスルー

ステップスルーとは、実行中のプロセスの関数を逐次、計画的に分析する処理のことです。

用語

ステップイン、ステップオーバー、およびステップアウトは、言語の構造によって実行順が暗黙的に決定される手続き型プログラミング言語のデバッガに由来する用語です。ただし、Identity Manager のフォームおよびワークフローでは、コード内で要素が出現する順序はその実行順に影響しません。

このため、これらの用語はビジネスプロセスエディタで使用するときには、多少異なった意味を持ちます。

- **ステップイン** : 現在のスレッド上の次の実行ポイントに移動することを指します。ステップインは常に、デバッガの XML 表示でプロセス内を進むことのできる最小の単位です。
- **ステップオーバー** : 現在の `begin` タグから現在の `end` タグまで、中間の要素で停止することなく移動することを指します。ステップオーバーでは、`start` タグと `end` タグの間のほとんどすべての要素をスキップできます。ただし、現在の要素の `start` タグと `end` タグの間に次の実行ポイントが出現しない場合、デバッグはそこで停止します。

  たとえば、複数のアクティブな仮想スレッドを含むワークフローで、アクションの `start` タグにステップできますが、実行される次の要素は別のアクションです。この場合、プロセスは別のポイントで実行を停止します。そのため、重要な可能性がある要素を誤ってスキップすることを回避できます。
- **ステップアウト** : 実行スタックが現在よりも 1 少なくなるまで、増分的に移動することを指します。ステップオーバーに類似しています。次の実行ポイントが、異なる親の実行スタックを持つ場合は、代わりにそこで停止します。

全般的なヒント

次に示すのは、実行プロセスのステップスルーを活用するために役立つヒントの一覧です。

- デバッガでのステップインを、デバッグタスクのコンテキストで実現可能な範囲で細かく設定します。これは、デバッグにとって重要な可能性がある要素を空過するのを避けるために役立ちます。
- ステップ実行は、プログラムの実行順を変更しません。プログラムの実行順は、デバッガを接続しない場合と同じです。目に見える実行部分をスキップ可能です (ただし、それでも実行自体は行われる)。
- コード内でステップをできるだけ小さくしたい場合は、「step-into」をクリックします。

- 開始タグと終了タグの間で、内容に関して問題が発生しそうにないと思われるときは、「step-over」をクリックします。デバッガはこの要素をスキップしますが、これらのタグ内のコードは引き続き実行されます。

表 A-7 は、BPE のデバッガによる、次のコードサンプルの処理方法のスナップショットを示します。

```
<A>
   <B/>
</A>
<D/>
(A、B、D は何らかの XML 要素 )
```

表 A-7 デバッグプロセスの例

実行順	結果
<A>、<B/>、</A>、<D/>	「**step-into**」をクリックすると、デバッガはその実行順で行を強調表示します。 「**step-over**」をクリックすると、デバッガは <A>、</A> (B をスキップ)、<D/> を強調表示します。
<A>、<D/>、<B/>、</A>	「**step-over**」をクリックすると、<A>、<D/>、<B/>、</A> の順でコード行が表示されます (この場合、ステップオーバーはステップインと同じ)。

はじめに

BPE には、ワークフロー、フォーム、および規則に対する、デバッガの使用方法についてのチュートリアルが含まれています。デバッガに付属する sample/debugger-tutorial.xml ファイルは、サンプルのワークフロー、規則、およびフォームを含んでいます。この章では、これらのサンプルをチュートリアルに使用します。

ステップ 1: チュートリアルファイルのインポート

次のいずれかの方法で、チュートリアルファイルをインポートします。

- Identity Manager で、「Configure」>「Import Exchange File」の順に選択します。「File to Upload」フィールドに「**sample/debugger-tutorial.xml**」と入力するか、「Browse」をクリックしてこのファイルを選択します。
- コンソールから「**import -v sample/debugger-tutorial.xml**」と入力します。

ファイルが正常にインポートされたら、次のステップに進みます。

ステップ 2: システム設定オブジェクトの編集

システム設定オブジェクトを編集するには、次の手順に従います。

1. BPE で、次の順に選択することにより、編集するシステム設定オブジェクトを開きます。
   「File」>「Open Repository Object」>「Generic Objects」>「System Configuration」
2. ツリービューで、`serverSettings` および `default` 属性を展開し、`debugger` を選択します。
3. 「Attributes」パネルで、「Value」列をクリックしてデバッグを有効にします。
4. 「File」>「Save in Repository」の順に選択して、変更を保存します。
5. アプリケーションサーバーを再起動します。

警告 本稼働環境ではこのプロパティーを有効にしないでください。

ステップ 3: デバッガの起動

アプリケーションサーバーの再起動が完了すると、「Tools」>「Debugger」の順に選択して BPE デバッガを起動できるようになります。

例 : タブ付きユーザーフォームと更新ビューのデバッグ

ここでは、サンプルのデバッグ手順を通じて、フォームまたはワークフローの呼び出し元の場所に関係なく、デバッガのブレークポイントがどのように適用されるかを示します。

このサンプル手順は、次の各ステップで構成されます。

1. ブレークポイントの設定
2. 新規ユーザーの作成
3. 「Before Refresh View」結果の表示
4. 「After Refresh View」結果の表示
5. フォームのステップスルー
6. フォーム処理の完了

ブレークポイントの設定

ブレークポイントを設定するには、次の手順に従います。

1. 「Breakpoints」パネルの「View cycle」タブをクリックします。

2. 「Refresh view」をチェックします。これで、実行中にビューが更新されるたびに、デバッガでブレークポイントが実行されるようになります。

新規ユーザーの作成

新規ユーザーを作成するには、次の手順に従います。

1. Identity Manager で、「Accounts」>「New」...「User」の順に選択します。
2. 名 (例 :「**jean**」) と姓 (例 :「**faux**」) を入力します。
3. 「ID」タブをクリックして、ビューの更新操作をトリガーします。

「Before Refresh View」結果の表示

デバッガフレームに戻ります。このフレームはこの時点で、「Refresh view」に設定したブレークポイントで中断した状態です。「Execution Stack」には「Before Refresh View」が表示されます。これは、更新操作が発生する直前のビューの状態を示します。「Variables」パネルには、更新される直前のビューが表示されます。

図 A-61 例 1:「Before Refresh View」ブレークポイントでの中断のデバッグ

グローバルサブツリーを展開し、フォームで入力した `firstname` および `firstname` の値を探します。`fullname` の値は、この時点では「`null`」です。

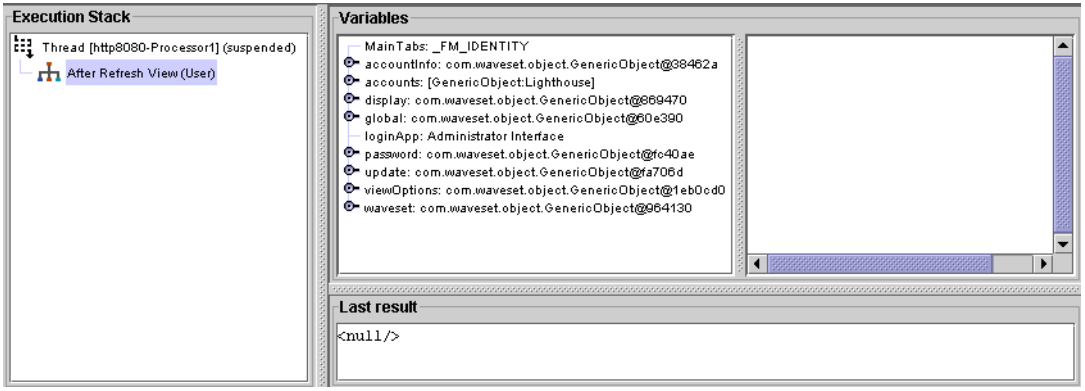
「After Refresh View」結果の表示

「After Refresh View」の結果を表示する手順は、次のとおりです。

1. 「Continue」をクリックします。

   「Execution Stack」には「After Refresh View」が一覧表示されます。ここには、更新操作が発生した直後のビューの状態が表示されます。`fullname` の値はこの時点では「**`jean faux`**」です。

図 A-62　例 1:「After Refresh View」ブレークポイントでの中断のデバッグ



2. 「Continue」をもう一度クリックします。

   フォームの実行が再開されます。ブラウザウィンドウに戻ります。「Name」を「**jean2**」に変更し、「ID」タブをもう一度クリックして、更新をもう一度トリガーします。

3. デバッガフレームに戻ります。

   フォーム処理は「Before Refresh View」の箇所で中断されます。

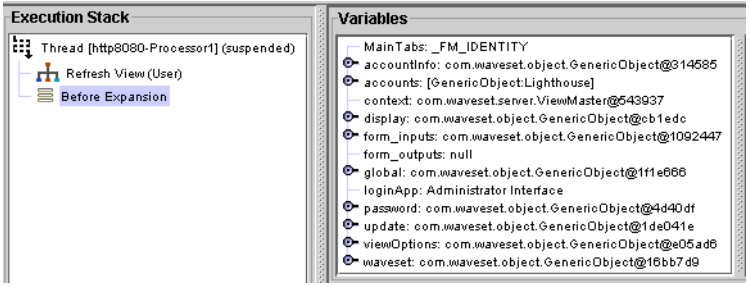
フォームのステップスルー

フォームをステップスルーするには、次の手順に従います。

1. 「step-into」をクリックして、実行内の姓名の展開部分を表示します。

   デバッガに「Before Expansion」が表示されます。これは、フォームの変数が展開されていないことを示します。

図 A-63　例 1:「Before First Expansion」パスでの中断のデバッグ



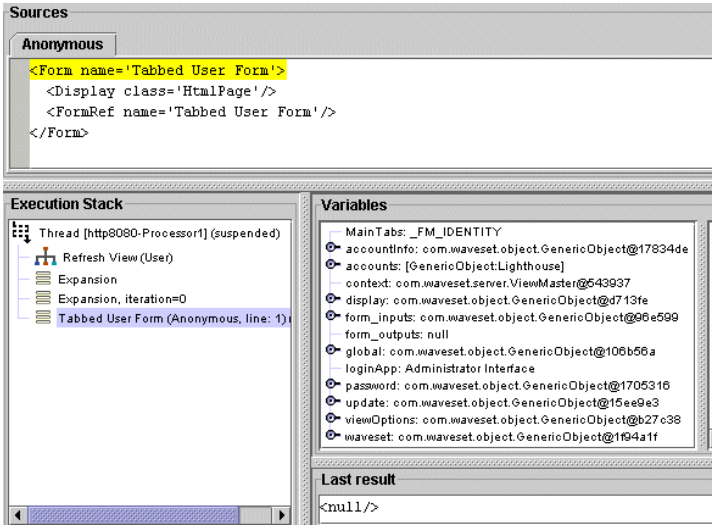
2. 「step-into」をもう一度クリックします。

   デバッガに「`Before Expansion, iteration=0`」と表示されます。これは、最初の「Expansion」パスの前にフォーム変数が出現することを示します。

3. 「step-into」をもう一度クリックします。

   この時点で、デバッガは匿名ソース上にあります。匿名ソースは一時的に作成されるラッパーフォームであり、MissingFields フォームに関連します。

図 A-64 例 1: タブ付きユーザーフォームの開始時点へのステップイン



4. タブ付きユーザーフォームの先頭に達するまで、「step-into」をさらに 2 回クリックします。

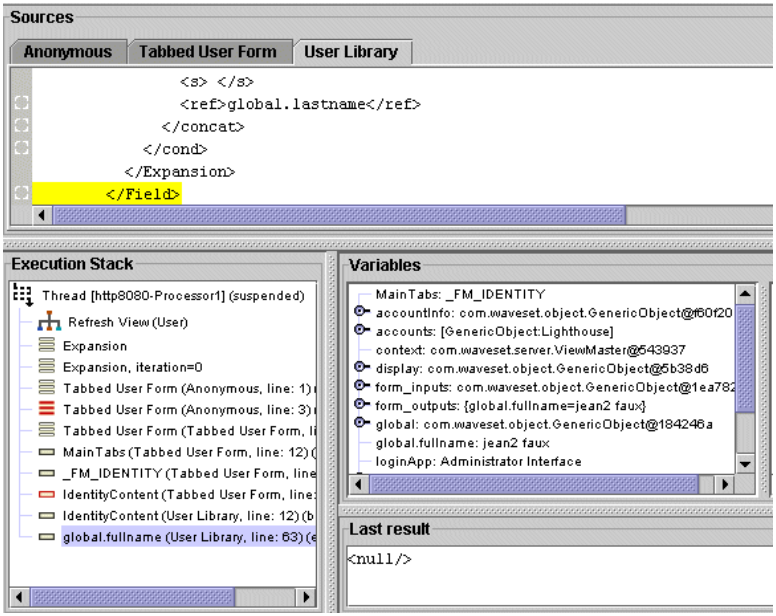
5. 「`<Field name='global.fullName'>`」に達するまで「step-into」をクリックし続けます (約 20 ～ 30 回のステップイン操作)。

6. 15 回または `</Field>` 要素に達するまで「step-into」をクリックします。

   ステップの間、`</concat>` タグの最後の結果は「**`jean2 faux`**」です。

   `form_outputs` の内容は「`global.fullname: jean2 faux`」です。

図 A-65　例 1: タブ付きユーザーフォームのデバッグ完了



フォーム処理の完了

フォーム処理を完了するには、次の手順に従います。

1. 「Step-out」を 7 回クリックします。

   この時点で、スタックが示す内容は次のようになるはずです。

   ```
   Refresh View (User)
   After Expansion
   ```

   「Variables」パネルは、すべての展開が実行されたあとのフォーム変数の状態を反映します。

2. 「Step-out」をもう一度クリックします。

   これで、「After Refresh View」に到達しました。この時点で表示される変数は、ビュー変数です。

3. グローバルサブツリーを展開します。

   この時点で、`fullname` の値は「**`jean2 faux`**」です。

4. 「Continue」をクリックします。

ワークフローのデバッグ

ここでは、ワークフローのデバッグに関する情報を示します。

ワークフロー実行モデル

ワークフローは単一の Java スレッドによって実行され、「Execution Stack」パネルで単一の Java スレッドによって表されます。ただし、ワークフローの内部で、各アクティビティーは個別の仮想スレッドになります。

ワークフロー実行の間、ワークフローエンジンは仮想スレッドのキューを循環的に処理します。各仮想スレッドは、次の表で説明する状態のいずれかになります。

表 A-8 仮想スレッドの状態

ワークフローアクティビティーの状態	定義
準備完了	遷移したばかりのアクティビティーを特定します (この状態はごく一時的であり、アクションは通常、準備完了と指定された直後に実行を開始する)。
実行中	現在実行中であるか、まだ実行されていない 1 つ以上のアクションを含むアクティビティーを特定します。 これは論理状態であり、Java スレッドがその時点でそのアクションを実行していることを意味しません。その時点で実行中のアクションは常に、デバッガで強調表示されているアクションです。
保留中のアウトバウンド	アクティビティー内のすべてのアクションが実行された直後のアクティビティーを特定します。このようなアクティビティーは、保留中のアウトバウンド状態に移行します。この状態のアクションは、アウトバウンド遷移の発生を待機します。OR 分岐の場合、アクションは 1 つの遷移が発生するまでこの状態です。AND 分岐の場合、その条件が true と評価されるすべての遷移が発生するまで、アクションはこの状態です。
非アクティブ	すべての遷移が発生済みのアクティビティーを特定します。
保留中のインバウンド	そのアクティビティーが AND 合流である仮想スレッドを特定します。これは、この仮想スレッドへの 1 回の遷移が発生したが、プロセスはまだほかの遷移を待機していることを意味します。

すべての遷移が完了したあとで、ワークフロープロセスは実行を開始します。

例 1: ワークフローと規則のデバッグ

ここで示す例は、BPE デバッガおよび `debugger-tutorial-workflow1` (Identity Manager に付属) で提供されるワークフローを使用して、サンプルのワークフローと規則をデバッグする方法を示します。この例では、ワークフローのデバッグおよび規則の実行で、ステップインおよびステップスルーする方法を示します。

この例では、次の手順を実行します。

1. プロセスの起動
2. 実行の開始
3. `getFirstName` スレッドのステップスルー
4. `getlastname` スレッドのステップインおよびステップオーバー
5. `computefullname` 処理のステップイン
6. 規則処理のステップスルー
7. ワークフロー処理の完了

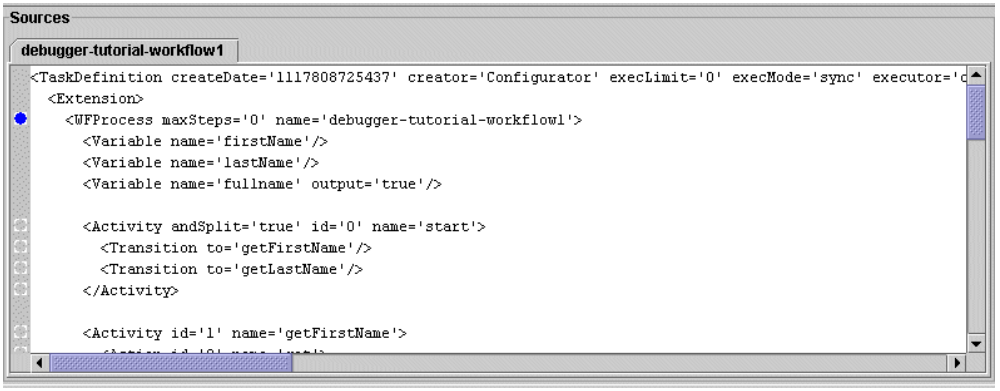
ステップ 1: プロセスの起動

ワークフローのデバッグプロセスを起動するには、次の手順に従います。

1. デバッガのメインウィンドウから、「File」>「Open Repository Object」の順に選択します。
2. 「debugger-tutorial-workflow1」をクリックします。

   XML 表示の左余白に小さなボックスがあります。これらのボックスは、コードに挿入できる潜在的なブレークポイントを示します。

図 A-66　　最初のブレークポイントの設定

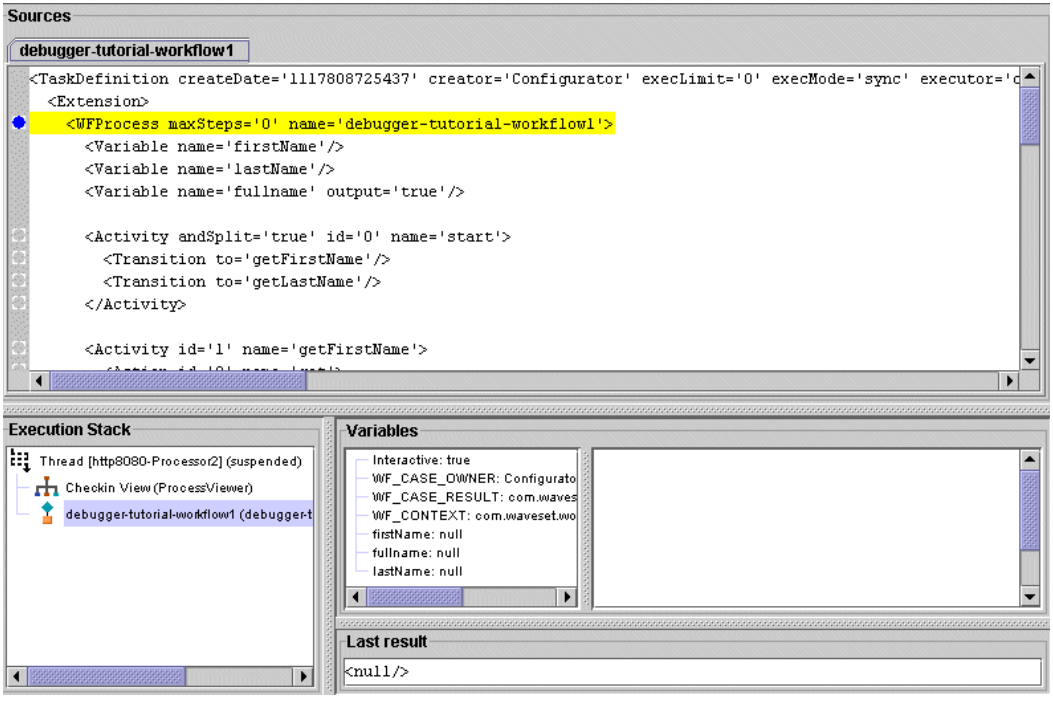


3. `<WFProcess>` タグのすぐ近くにあるボックスをクリックして、ワークフローの開始位置にブレークポイントを設定します。
4. Identity Manager にログインし、「Tasks」>「Run Tasks」の順に選択します。

5. 「debugger-tutorial-workflow1」をクリックします。

   デバッガフレームには、ブレークポイントでデバッグが停止したことが示されます。

図 A-67　　ブレークポイントでのデバッグ停止



次のことに注意してください。

- **「Execution Stack」パネル** - 「Execution Stack」パネルの上部に、「`Thread [thread name] (suspended)`」と表示されます。これは、指定された名前のスレッドによってこのワークフローが現在実行中であり、設定されたブレークポイントの位置で中断されていることを示します。

  「Thread」の下には実行スタックが表示されます。このスタックは逆順のスタックトレースであり、呼び出し元の関数が上に、呼び出される関数が下に表示されます (これは、ほとんどのデバッガでの実行トレースの表示とは逆の順序)。

スタックの一番上のフレームは「Checkin View (ProcessViewer)」という名前であり、これは、ワークフローがその時点で ProcessViewer の `checkinView` メソッドによって呼び出されていることを示します。このスタックフレームの Java ソースコードにはアクセスできないため、このフレームをクリックしても新しい情報は表示されません。ただし、スタックフレームは、ワークフローがどの場所から起動されているかについてのコンテキストを提供します。

スタック内の次のフレームは、ワークフロープロセス (`<WFProcess>`) の開始位置である現在の実行ポイントに対応しているため、強調表示されています。

- **「Variables」パネル** - 現在の実行ポイントの位置で、現在スコープ内にあるすべての変数が一覧表示されます。次の変数が表示されます。
  - **Interactive** - この変数は、ビューによってプロセスへの入力として渡されます。
  - **WF_CASE_OWNER, WF_CASE_RESULT, WF_CONTEXT** - これらの変数は、暗黙的なワークフロー変数です。
  - **firstName, fullname, lastName** - これらの変数は、`<Variable>` 宣言を使用してワークフロー内で宣言されます。

6. 「Debug」>「Current Line (F5)」の順に選択して、現在の実行行をふたたび強調表示します。

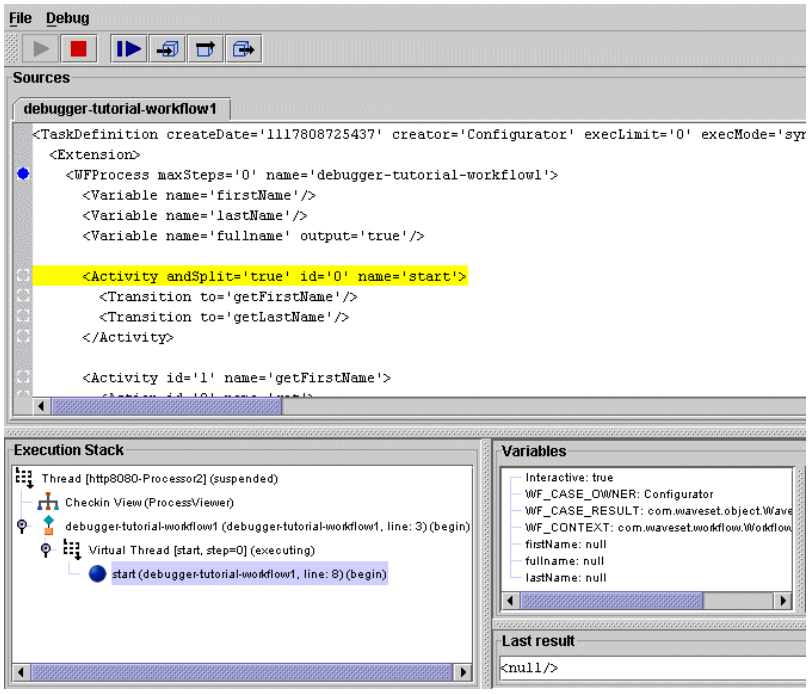
ステップ 2: 実行の開始

実行を開始するには、次の手順に従います。

1. 「step-into」をクリックします。

   この時点で、デバッガは開始アクティビティーに移動します。実行スタックに「Virtual Thread `[start, step=0]` (executing)」が含まれていることを確認してください。これは、現在実行中の状態である開始アクティビティーの仮想スレッドがあることを示します。

図 A-68　　最初の仮想スレッドの実行へのステップイン



2. 「debugger-tutorial-workflow-1」フレームの 2 レベル上をクリックして、「WFProcess」を強調表示します。これにより、呼び出し元の位置が示されます。
3. F5 キーを押して現在の行に戻ります。
4. 「step-into」をクリックします。

   この時点で、デバッガは </Activity> の位置に移動し、開始仮想スレッドは、保留中のアウトバウンドの状態になります。

ステップ 3: getFirstName スレッドのステップスルー

次の手順を使用して、getFirstName スレッドをステップスルーします。

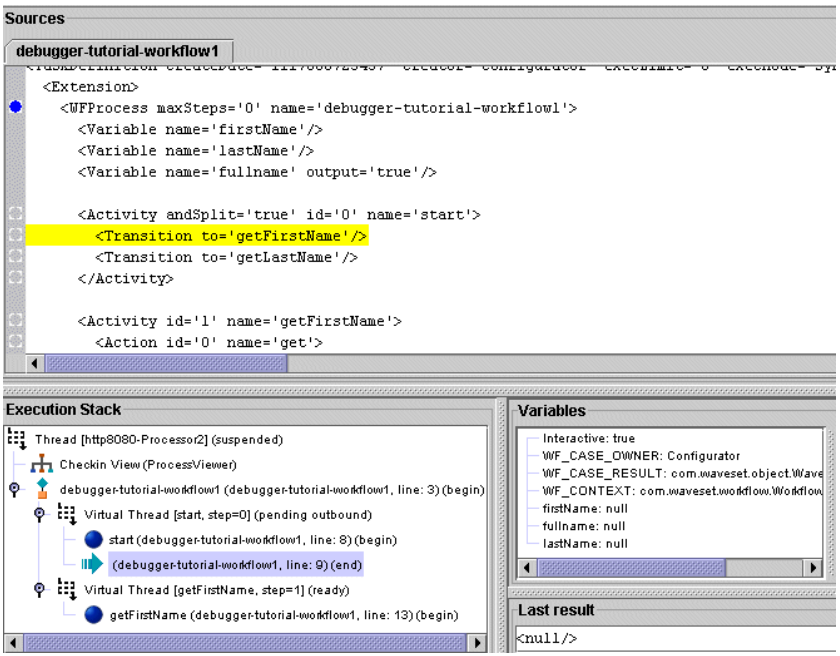
1. 「step-into」をクリックします。

   この時点で、デバッガでは getFirstName への遷移が強調表示されています。

2. 「step-into」をクリックします。

   この遷移の結果として、getFirstName の新しい仮想スレッドが作成されています。この時点で、この仮想スレッドは準備完了の状態です。開始仮想スレッドはまだ、保留中のアウトバウンド状態です (これは AND 分岐操作であるため、すべての可能な遷移が発生する必要がある)。

図 A-69　　例 2: getFirstName の実行へのステップイン



3. 「step-into」をもう一度クリックします。

   デバッガは getFirstName アクティビティーにジャンプします。状態は、準備完了から実行中に変化します。

4. 「step-into」をクリックします。

   デバッガは、get アクションに移動します。

5. 「step-into」をあと 3 回、またはデバッガが </set> タグに達するまでクリックします。

   「Variables」パネルで、</set> の結果として firstName が「myfirstname」に設定されたことが示されます。

ステップ 4: getLastName スレッドのステップインとステップオーバー

次の手順を使用して、getLastName スレッドをステップインおよびステップオーバーします。

1. 「step-into」をあと 3 回、またはデバッガが getFirstName の </Activity> に達するまでクリックします。

   この時点で、getFirstName 仮想スレッドの状態は、保留中のアウトバウンドです。

2. 「step-into」をクリックします。

   デバッガは開始仮想スレッドに戻り、getLastName への遷移を処理する準備をします。

3. 「step-into」をクリックします。

   すべての遷移が処理されたため、開始は非アクティブになります。この遷移により、この時点で getLastName は「ready」状態です。

4. 「step-into」をクリックします。

   開始仮想スレッドは非アクティブであるため、この時点でなくなります。デバッグは、この時点で実行中の状態である getLastName 仮想スレッドに移動します。

5. 「step-over」をクリックして、getLastName の終わりまでスキップします。

   「Variables」パネルで、lastName 変数は「mylastname」に設定されています。getFirstName および getLastName の両方の仮想スレッドは、保留中のアウトバウンド状態です。

6. 「step-into」をクリックします。

   デバッガは getFirstName から computeFullName に遷移します。

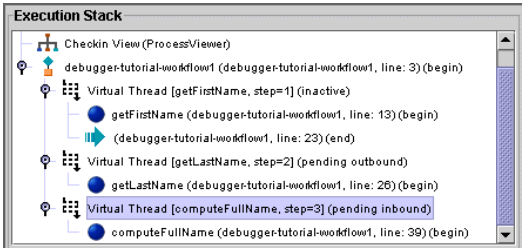
7. 「step-into」をクリックします。

   getFirstName は非アクティブになり、新しい仮想スレッド computeFullName が作成されます。このスレッドは、getLastName からのインバウンド遷移をまだ待機しているため、「pending inbound」状態です (待機が発生するのは、これが and-join 操作であるためです。or-join 操作の場合は、プロセスの状態がただちに「ready」になる)。

8. 「step-into」をクリックします。

   デバッガは getLastName から computeFullName に遷移します。

図 A-70　getFirstName から computeFullName へのデバッガ遷移



ステップ 5: computeFullName 処理へのステップイン

次の手順を使用して、computeFullName 処理にステップインします。

1. 「step-into」をクリックします。

   この遷移により、computeFullName 仮想スレッドの状態が、保留中のインバウンドから準備完了に変化します。

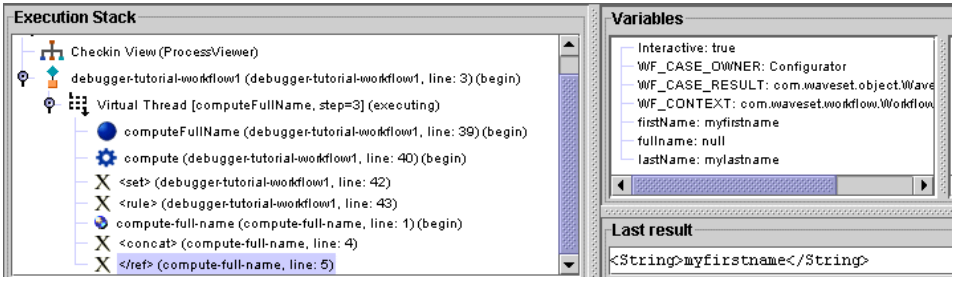
2. 「step-into」をクリックします。

   この時点で、computeFullName の状態は、実行中です。

3. 「step-into」をあと 5 回クリックします。

   この時点で、デバッガは firstName の </argument> タグの位置です。「last result」パネルには「<String>myfirstname</String>」と表示されます。この値は firstName 引数に渡されます。

図 A-71　computeFullName 処理へのステップイン



ステップ 6: 規則処理のステップスルー

規則処理をステップスルーするには、次の手順に従います。

1. 「step-into」をあと 3 回クリックします。

   デバッガが「Compute-Full-Name」規則にステップインします。実行スタックで、フレームをクリックして 1 つ上のフレームに移動します。debugger-tutorial-workflow-1 内の `<rule>` 呼び出しが強調表示され、規則の呼び出し元の場所を示します。F5 キーを押して現在の行を再選択します。

2. 「step-into」をあと 3 回、またはデバッガが `</ref>` タグに達するまでクリックします。

   「last result」パネルには、「`<String>myfirstname</String>`」と表示されます。これは、「`<ref>firstName</ref>`」の結果です。

3. 「step-into」をあと 3 回、またはデバッガが `</concat>` タグに達するまでクリックします。

   「last result」パネルには、`<concat>` 式の結果が表示されます。

   ```
   <String>myfirstname mylastname</String>
   ```

4. 「step-into」をあと 2 回クリックします。デバッグは `</rule>` タグに戻ります。

ステップ 7: ワークフロープロセスの完了

ワークフロープロセスを完了するには、次の手順に従います。

5. `</set>` 要素に達するまで「step-into」をクリックします。

   `fullname` 変数が「`myfirstname mylastname`」に更新されています。

6. 「step-into」をあと 2 回クリックします。

   この時点で、`computeFullName` の状態は、保留中のアウトバウンドです。

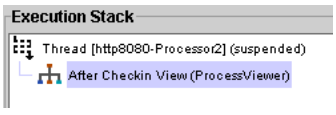
7. 「step-into」をあと 4 回クリックします。`end` の状態が、準備完了、実行中、と順に変化します。

   デバッガは `</WFProcess>` タグに到達し、プロセスが完了したことを示します。

8. 「step-into」をクリックします。

   「Execution Stack」には「After Checkin view」と表示されます。これは、ワークフローを呼び出した、ビューのチェックイン操作が完了したことを示します。

図 A-72　例 2: 「Check-in View」操作の完了



9. 「Continue」をクリックして実行を再開します。

   ブラウザの要求がタイムアウトしていない場合、プロセスダイアグラムを伴う「Task Results」ダイアグラムが表示されます。

例 2: 手動アクションとフォームを含むワークフローのデバッグ

ここで示す例では、手動アクションとフォームを含む、サンプルワークフローのデバッグ方法を説明します。

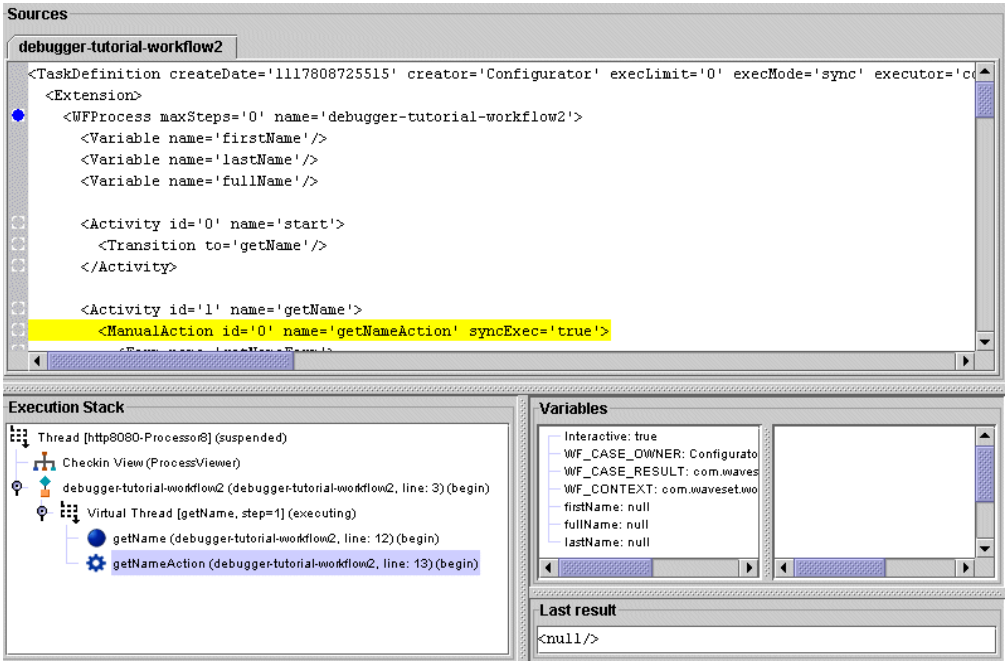
デバッガのチュートリアルファイルにある `workflow2` を使用し、次の手順を実行します。

1. 「File」>「Open Repository Object」の順に選択します。
2. 「Workflow Processes」を展開し、`debugger-tutorial-workflow2` を選択します。
3. `<WFProcess...>` タグにブレークポイントを設定します。
4. Identity Manager にログインし、「Tasks」>「Run Tasks」の順に選択します。
5. `debugger-tutorial-workflow2` をクリックします。

   設定したブレークポイントでデバッガが停止します。

6. 「step-into」を 6 回、つまり、デバッガが「<ManualAction... name='getNameAction'>)」に達するまでクリックします。

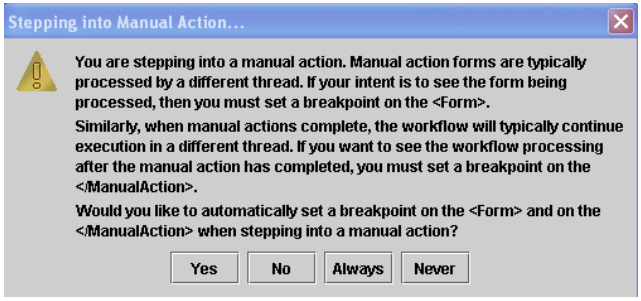
図 A-73　　手動アクションへのステップイン



7. 「step-into」をクリックします。

8. 別のスレッドでフォーム処理が発生するという説明のダイアログが表示されたら、<Form> タグにブレークポイントを設定して、処理の発生を確認します。

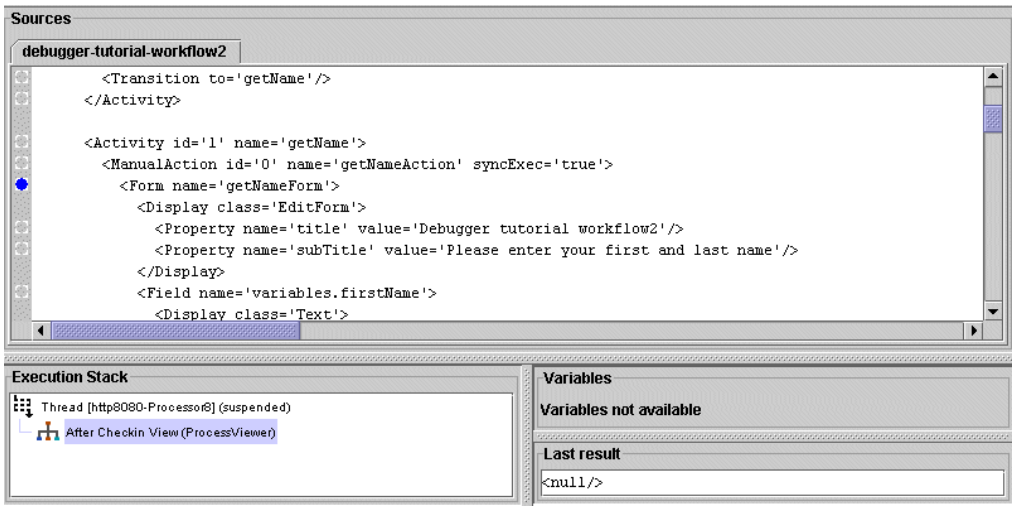
図 A-74　　「Stepping Into Manual Action」ダイアログ



9. 「Yes」または「Always」を選択します。

   フォーム処理が完了したあとで、ワークフローは別のスレッドでの実行を継続します。その結果、</ManualAction> にブレークポイントを設定して、フォームが処理を完了したあとのワークフロー処理を監視する必要があります。

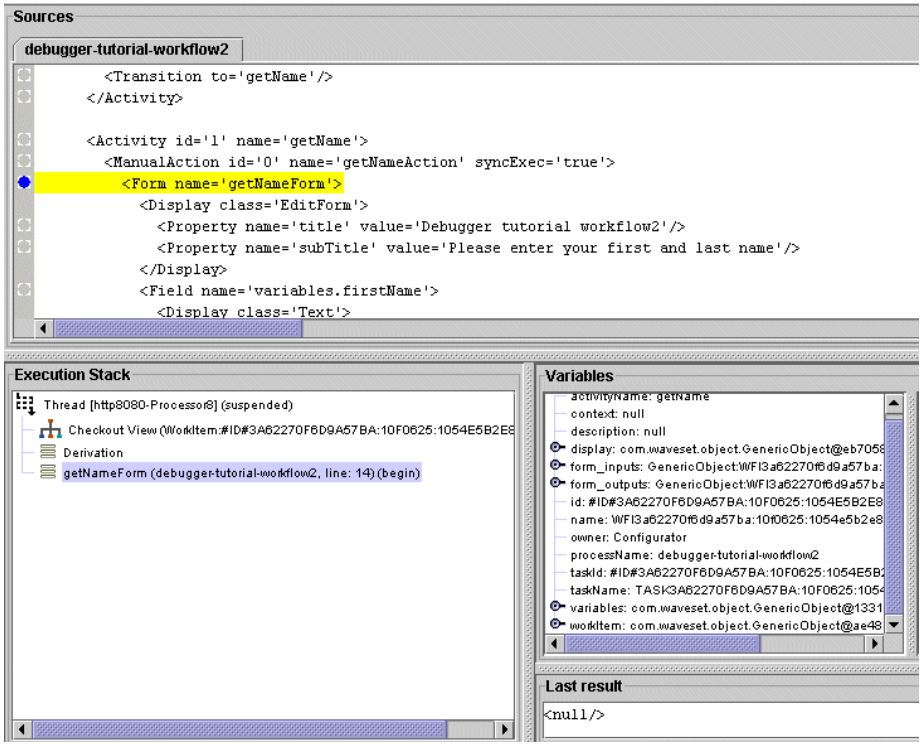
図 A-75　　フォームの開始を示すブレークポイント



   デバッガでは、指示どおりに、<Form> タグおよび </ManualAction> タグにブレークポイントが設定されています。加えて、「Execution Stack」には「After Checkin view」が示されます。ワークフロー処理は可能なかぎり (手動アクションが完了するまで) 進行済みであるため、ワークフロープロセスからのステップアウトが完了します。

10. 「Continue」をクリックします。デバッガは <Form> 要素に設定されたブレークポイントで処理を停止します。

図 A-76　　手動アクション処理を表示するデバッガ



「Execution Stack」領域には次の内容が表示されます。

- 「**Checkout View (WorkItem:...)**」- 特定の作業項目に対し、ビューのチェックアウトのコンテキストで処理が発生していることを示します。
- 「**ManualAction forms**」- 作業項目ビューに対して作用し、変数オブジェクトを通じてワークフロー変数を操作します。変数オブジェクトを展開して、null でないワークフロー変数を表示します。
- 「**Derivation**」- フォーム実行が「Derivation」パス上にあることを示します。

11. このフォームには <Derivation> 式が含まれないため、「Continue」をクリックして次のフェーズまたは処理に進みます。フォーム処理の「HTML Generation (root component)」パスが開始されます。

HTML 生成フェーズ (root コンポーネント)

root コンポーネントの HTML を生成するには、次の手順に従います。

1. 「step-into」を 2 回クリックします。

   デバッガはタイトルの `<Property>` 要素の処理を完了した状態になります。「last result」パネルには、このプロパティーの値が表示されます。

2. 「step-into」をあと 3 回クリックします。

   このパスではページの root 要素の構築のみを扱うため、デバッガはフォーム内のフィールドをスキップし、`</Form>` 要素に直接移動します。

3. 「Continue」をクリックします。

   フォーム処理の「HTML Generation (subcomponents)」パスが開始します。

HTML 生成 (サブコンポーネント)

サブコンポーネントの HTML を生成するには、次の手順に従います。

1. 「step-into」を 13 回、またはデバッガが `</Form>` タグに達するまでクリックします。

   デバッガはこれらの各フィールドを反復処理し、それらの表示プロパティーを評価します。

2. 「Continue」をクリックします。

   実行が再開されたため、デバッガには中断されたスレッドは表示されません。ブラウザウィンドウに制御が戻ります。

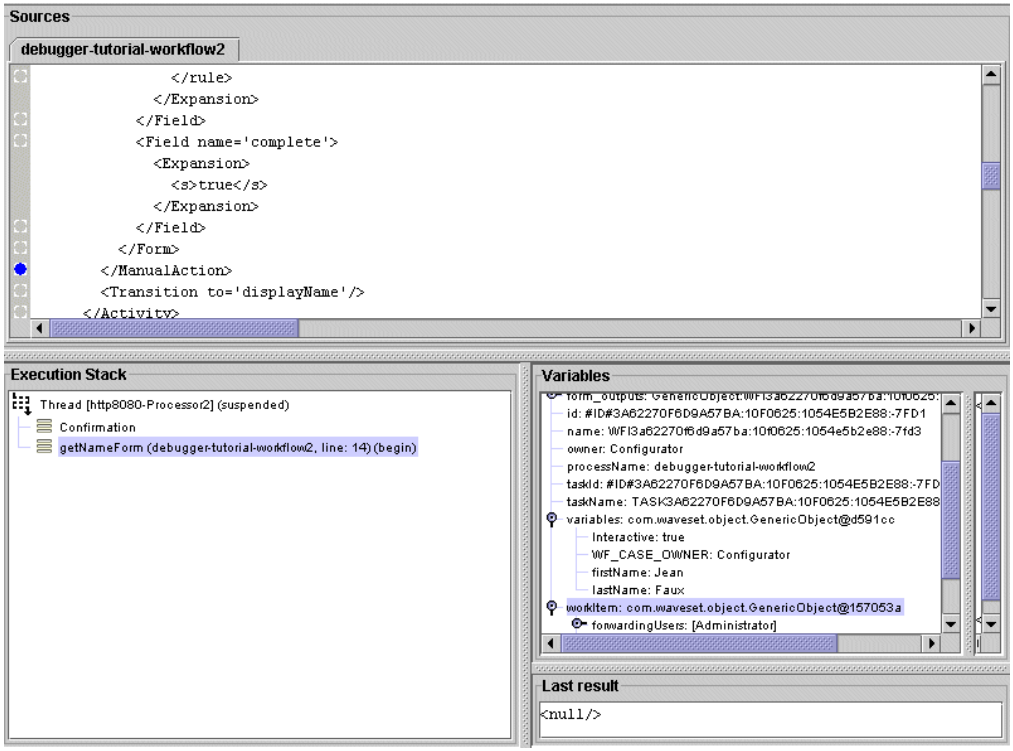
3. ブラウザウィンドウに戻り、入力を求められたら姓と名を入力して「Save」をクリックします。

   デバッガフレームに戻ります。この時点で、デバッガはブレークポイントで中断しています。

4. 「Variables」サブツリーを展開します。

   `firstName` および `lastName` は、入力したばかりの値です。デバッガはこの時点で、フォーム処理の確認フェーズです。

図 A-77　　フォーム処理の確認フェーズ



確認

このフォームには確認フィールドがないため、処理は発生しません。「Continue」をクリックして、フォーム処理の検証フェーズを開始します。

検証と展開

このフォームには検証式が含まれないため、明示的な処理は発生しません。

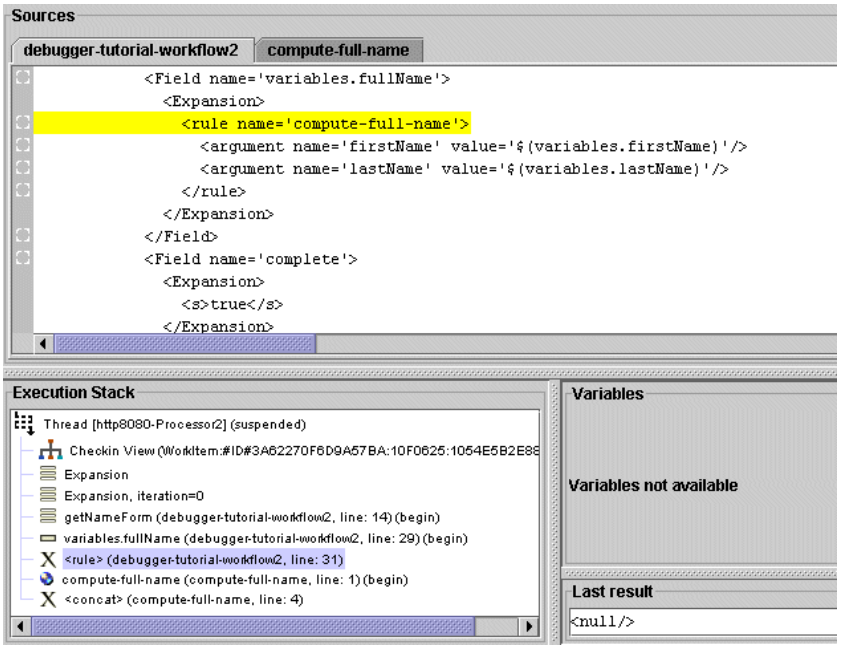
1. 「Continue」をクリックして検証フェーズをスキップします。

   この時点では、フォーム処理の展開フェーズです。

2. 「step-into」を 6 回クリックします。

   この時点で、デバッガは `variables.fullName` フィールドの `<Expansion>` の `<rule>` タグの位置です。

図 A-78　規則処理へのステップイン



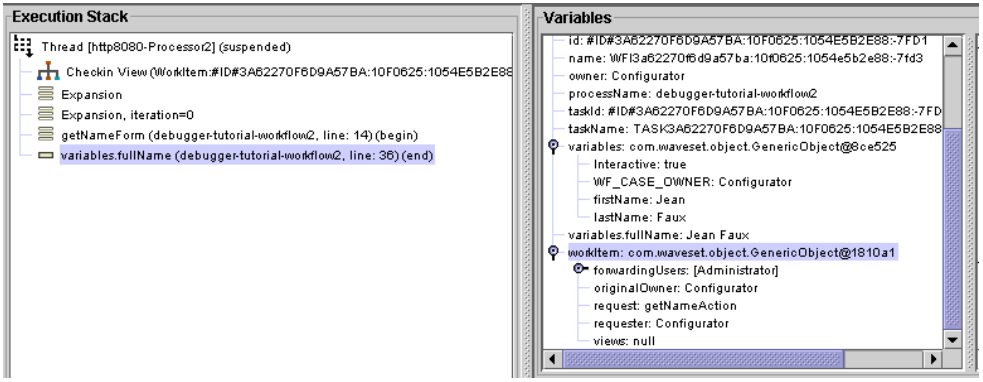
3. 「step-into」を 5 回クリックします。デバッガは <rule> 要素にステップインした状態になります。
4. 「step-into」を 7 回、またはデバッガが </Rule> 要素に達するまでクリックします。

   「last result」に姓名が表示されます。
5. 「step-into」をもう一度クリックすると、フォームでの処理が再開します。
6. 「step-into」をもう一度クリックします。

   トップレベルの variables.fullName には、実行されたばかりの展開式の値が格納されています。これは、variables データ構造の子ではなくトップレベルのエンティティーです。その理由は、フォーム処理の間、フォーム出力は専用の一時的な form_outputs データ構造に、パス式が平坦化されて保持されるためです。

   フォーム処理のあと、フォーム出力は元のビューに同化されます。暗黙的な変数 form_inputs および form_outputs において、form_inputs は未変更の作業項目ビューを示し、form_outputs は、フォーム処理の完了後にビューに同化される出力フィールドを示します。

図 A-79　　デバッガでの variable.fullName の実行完了の表示

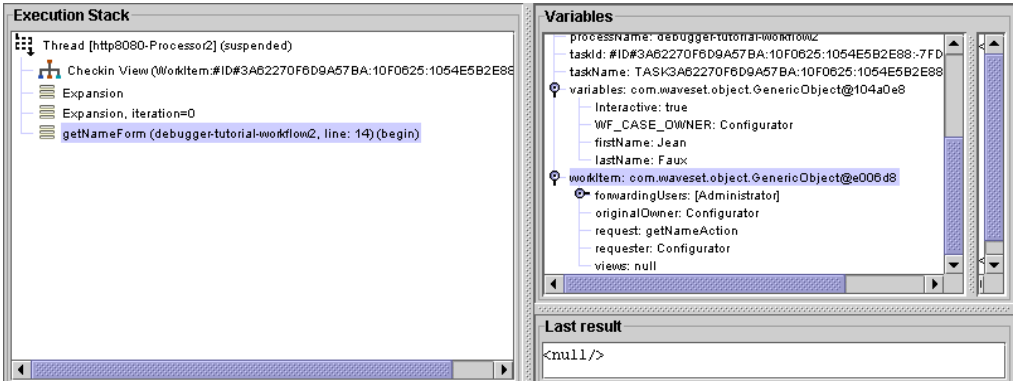


一般に、form_inputs はビューを特定し、form_outputs にはビューに同化されるデータが含まれます。ただし、Active Sync フォームのように、必ずしもすべてのフォームがビューに結び付けられるわけではありません。フォームエンジンは一般的なデータマッピングエンジンであり、フォーム入力からフォーム出力へのマッピングを行います。ビューハンドラは、フォームエンジンにビューを渡す処理と、出力をビューに戻して反映する処理を受け持ちます。

7. 「Continue」をクリックします。

   デバッガは </ManualAction> ブレークポイントに到達します。これは、デバッガが手動アクションにステップインする時点よりも前に設定されたブレークポイントです。変数 firstName および lastName は入力した値です。fullName は、実行されたばかりの展開式の結果です。

図 A-80　　デバッガでの展開式の結果表示



8. 「step-into」を 5 回、<ManualAction... name='displayNameAction'> に達するまでクリックします。
9. 「step-into」をもう一度クリックします (表示が出た場合は、「Yes」または「Always」をクリックする)。
10. 「Continue」をクリックします。

    この時点で、デバッガは displayNameForm の「Derivation」パスの位置です。

取得と HTML 生成 (root コンポーネント)

取得フェーズと HTML 生成フェーズを完了するには、次の手順に従います。

1. 「Continue」をクリックして、displayNameForm の HTML 生成 (root コンポーネント) 処理を開始します。
2. 「step-into」を 8 回、またはデバッガが subTitle の </Property> 要素に達するまでクリックします。
3. 「Continue」を 2 回クリックします。

    デバッガは次のメッセージを表示します。

    ```
    No suspended threads because execution has resumed.Control has now
    returned to the browser window.
    ```

4. ブラウザウィンドウに戻ります。

    表示される情報は、入力したものと同じです。

5. 「Save」をクリックしてデバッガフレームに戻ります。

    この時点で、デバッガは「Confirmation」パスの位置であり、displayNameForm. を処理しています。

検証と展開

検証と展開を開始するには、次の手順に従います。

1. 「Continue」をクリックして検証パスを開始します。
2. 「Continue」をクリックして展開パスを開始します。
3. 「Continue」をもう一度クリックします。

    手動アクションが完了したため、この時点でデバッガは </ManualAction> タグの位置です。この時点で、ワークフロー処理は再開されています。

4. 「step-into」を 5 回、またはデバッガが </WFProcess> タグに達するまでクリックします。このタグは、ワークフローが実行を完了したことを示しています。
5. 「Continue」をクリックします。

    デバッガは次のメッセージを表示します。

```
No suspended threads because execution has resumed. Control has now
returned to the browser window.
```

6. ブラウザウィンドウに戻り、ワークフロープロセスダイアグラムを監視します。

フォームのデバッグ

フォームは一連のパスで処理されます。どのパスが進行中かに応じて、特定の要素が処理され、ほかの要素は無視されます。デバッガのメインウィンドウの実行スタックは、フォーム処理の現在のフェーズを示します。最も外側のフォームに先行する実行スタックフレームには、常にパスの名前があります。

取得

フォーム実行の取得フェーズの間、フォームエンジンは各フィールドを反復処理し、個々の <Disable> 式を処理します。

またフォームエンジンは、その <Disable> 式が false を返すフィールドについて、<Derivation> 式を処理します。

展開

デバッガは各フィールドを反復処理し、個々の <Disable> 式を処理します。その <Disable> 式が false を返すフィールドについて、デバッガは <Expansion> 式を処理します。

「Expansion」処理フェーズは、次のいずれかの条件が満たされるまで実行を継続します。

- それ以上変更が発生しない。
- maxIterations を超過した。maxIterations は、フォームエンジンにおけるフォームの Expansion 要素の処理に渡されるパラメータです。

デフォルトでは、maxIterations は 1 に設定されています。結果として、デバッガは 1 つのパスしか作成しません。このため、展開の実行時に、「Execution Stack」パネルには「Expansion, iteration=0」と表示されます。

検証

デバッガは各フィールドを反復処理し、個々の <Disable> 式を処理します。<Disable> フィールドが false を返すフィールドについて、フォームエンジンは次の要素も処理します。

- required プロパティーを持つ <Display> 式がフィールドに存在する場合、フィールドはそのプロパティー式を評価します。

- フィールドが <Validation> 式を持つ場合、フィールドはその検証式を評価します。

確認

デバッガは各フィールドを反復処理し、個々の <Disable> 式を処理します。<Disable> 式が false を返し、confirm 属性も持つフィールドについて、デバッガは confirm によって参照されるフィールドが、このフィールドと一致することを確認します。フィールドが一致しない場合、デバッガは「Variables」パネルで display.errors にエラーを追加します。

同化

デバッガは各フィールドを反復処理し、個々の <Disable> 式を処理します。その <Disable> 式が false を返すフィールドについて、デバッガはフィールドの <Display> 要素の <Property> オブジェクトを処理します。

このフェーズは通常はスキップされます。このフェーズは、display.mementos を含まない (ログインフォームなどの) 特定のフォームのみに関係します。これらのフォームではデータの同化後に、HTML コンポーネントを再構築するためにこのフェーズが必要です。

HTML 生成 (root コンポーネント)

デバッガは、トップレベルフォームおよびフォームの <FieldDisplay> 要素のみを反復処理します。このパスの目的は、トップレベル HTML コンポーネントを構築することです。直後に「HTML Generation (subcomponents)」パスが続きます。

HTML 生成 (サブコンポーネント)

デバッガは各フィールドを反復処理します。また、個々の <Disable> 式も処理します。その <Disable> 式が false を返すフィールドについて、デバッガはフィールドの <Display> 要素の <Property> 要素を処理します。

カスタムビュー処理

一部のビューでは、フォームに対して追加のパスが必要です。これらのパスの間、デバッガは各フィールドを反復処理し、個々の <Disable> 式を処理します。

匿名ソースの操作

フォームをステップスルーするとき、デバッガは匿名ソースを識別できます。匿名ソースは、一時的に生成されるフォーム (またはフォームの一部) です。結果として、匿名ソースは、Identity Manager リポジトリに格納される持続的フォームとは対応しません。匿名ソースの例には、ログインフォームや MissingFields フォームがあります。

匿名ソースは Identity Manager リポジトリに格納されず、一意の識別子を持たないため、匿名ソースには個別のブレークポイントを設定できません。

ただし、匿名ソースのステップスルーは可能です。

すべての匿名ソースにブレークポイントを設定するには、「Breakpoints」パネルで「Global」タブを選択します。デバッガは以降、匿名ソースの行に達するたびに実行を中断します。たとえば、ログインフォームをデバッグするには、このオプションを選択してログインページに移動します。

索引

記号

A

B

C

D

E

F

G

H

I

J

L

M

N

O

P

R

S

T

U

V

W

X

あ

い

う

え

お

か

き

く

け

こ

さ

し

す

せ

そ

た

ち

つ

て

と

な

ほ

ま

む

め

も

ゆ

よ

ら

り

れ

ろ

わ