Sun™ Identity Manager 8.0
配備ツール

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

Part No: 820-5455

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。この情報は、次のように構成されています。

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「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 2 章「カスタムアダプタの開発」- 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 3 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 4 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 5 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。

注

- 以前のリリースで提供されていた『Identity Manager IDE』という章は、本書では削除されています。Identity Manager Integrated Development Environment (Identity Manager IDE) のインストールと設定の手順については、https://identitymanageride.dev.java.net を参照してください。

  参考として、Identity Manager のプロファイラの使用手順と Identity Manager の FAQ については、『Sun Java™ System Identity Manager 8.0 リリースノート』の「ドキュメントの追加事項と修正事項」という章の「Identity Manager 配備ツール」に収録されています。
- ビジネスプロセスエディタ (BPE) の使用は非推奨となり、次回の Identity Manager リリースでは削除される予定です。代わりに Identity Manager IDE を使用してください。

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。	これらは *class* オプションと呼ばれます。 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。

表 2　記号の表記規則 (続き)

記号	説明	例	意味
+	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。
>	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 Deployment』: Sun Java™ System Identity Manager Service Provider 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『Sun Java™ System Identity Manager 配備ツール』であり、Part No. は 820-5455 です。

ご意見、ご要望の送付先

第 1 章

規則の操作

Identity Manager の規則と規則ライブラリは、頻繁に使用されるプログラミングロジックや静的な変数をカプセル化し、配備環境全体の多くの場所で再利用できるようにするために使用するリポジトリオブジェクトです。規則と規則ライブラリを使用すると、データをより効率的に管理できるようになります。

この章では、Identity Manager の規則および規則ライブラリの操作方法と、Identity Manager によって提供されるデフォルトの規則および規則ライブラリのカスタマイズ方法について説明します。この情報は次の各節で構成されています。

注	配備環境用に規則を作成、編集、およびテストするには、Identity Manager Integrated Development Environment (Identity Manager IDE) を使用します。 Identity Manager IDE のインストールと設定の手順については、https://identitymanageride.dev.java.net を参照してください。

開始する前に

Identity Manager の規則と規則ライブラリを操作する前に、次の節の情報を確認してください。

対象読者

この章は、Identity Manager の配備環境用に規則を作成、編集、およびテストする個人を対象としています。Identity Manager の規則と規則ライブラリを操作するには、次の知識を備えている必要があります。

- プログラミングに関する基本知識
- XPRESS および XML オブジェクト言語に対する理解

  XPRESS の使用方法の詳細については、『Sun™ 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。
- Java および Java スクリプトに対する基本的な理解

関連ドキュメントと Web サイト

Identity Manager の規則と規則ライブラリについては、この章で提供する情報のほかに、次に紹介するマニュアルや Web サイトを参照してください。

推奨ドキュメント

Identity Manager の規則に関連する情報については、『Sun™ Identity Manager ワークフロー、フォーム、およびビュー』の次の章を参照してください。

- 第 4 章「XPRESS 言語」: XPRESS 言語に関する説明
- 第 5 章「XML オブジェクト言語」: XML オブジェクト構文に関する説明

有用な Web サイト

次の表は、Identity Manager の規則と規則ライブラリを操作するときに参考にできる Web サイトをまとめたものです。

表 1-1 有用な Web サイト

Web サイトの URL	説明
https://identitymanageride.dev.java.net	オープンソースの Identity Manager Integrated Development Environment (Identity Manager IDE) プロジェクト。Identity Manager IDE のインストールと設定の手順が含まれています。
http://sunsolve.central.sun.com/	診断ツール、フォーラム、機能と記事、セキュリティー情報、パッチの内容を含む Sun の Web サイト。 **注** : このサイトの情報は、次の 3 つの分野に分かれています。 • 内部 (Internal): Sun の従業員のみ • 契約 (Contract): アクセス契約している顧客のみに公開 • パブリック (Public): すべての人に公開
http://forum.java.sun.com/	フォーラムを参照したり、質問を投稿したりできる SDN (Sun Developer Network) Web サイト。
https://sharespace.sun.com/gm/folder-1.11.60181?	Sun の共有スペース上の Identity Manager のリンク。 **注** : このサイトで提供される情報にアクセスするには、共有スペース ID を作成する必要があります。
http://sharespace.sun.com/gm/document-1.26.2296	Sun の共有スペース上の Identity Manager の FAQ。 **注** : この FAQ にアクセスするには、共有スペース ID を作成する必要があります。

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 Manager 内で、規則は、頻繁に使用されるプログラミングロジックや静的な変数を再利用できるように格納して実行するためのメカニズムを提供します。規則はプログラミングのサブルーチンや関数と意味が似ています。規則は入力パラメータを取り、何らかのロジックを実行して、呼び出し元に値を返すことができます。

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。

規則は主にフォームやワークフロー内で参照されますが、次のような、その他のユーザーデータ関連の領域でも規則を参照することができます。

- **ロール** : ロール割り当て規則を使用して、所有者と承認者を動的にロールに割り当てます。
- Active Sync: プロセス規則または修正規則を使用して、Active Sync 対応のアダプタがリソースアカウントに対する変更を検出したときの動作を制御します。
- **調整** : 調整時に特別な規則のサブタイプ (確認規則や相関規則など) を使用します。これらのサブタイプについては、この章のあとの方で説明します。

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。

JavaScript での規則の記述については、19 ページの「JavaScript での規則の記述」を参照してください。

次の例は、<Rule> 要素を使用して基本の規則式を定義する方法を示したものです。ここで、規則定義名は getApprover、規則引数名は department、引数のデフォルト値は Tampa で、規則本体は文字列値 Sales Manager または HR Manager を返します。

コード例 1-1 XML 規則の例

```
<Rule name='getApprover'>
  <Comments> This rule determines the appropriate approver for a particular department.</Comment>
  <RuleArgument name='department'/>
  <RuleArgument name='location' value='Tampa'/>
  <cond>
    <eq><ref>department</ref><String>sales</String></eq>
    <cond>
        <eq><ref>location</ref><String>Tampa</String></eq>
        <String>Tampa Sales Manager</String>
        <String>Sales Manager</String>
    </cond>
    <String>HR Manager</String>
  </cond>
  <MemberObjectGroups>
    ObjectRef type='ObjectGroup' name=ExampleChoc'/>
  </MemberObjectGroups>
</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、<rule name='rulename'> のように **r** が小文字の XPRESS の <rule> 要素を使用します。

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。特にフォーム、Java コード、ワークフロー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などのデータをカプセル化し、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編成して再利用すると、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を 1 つずつ変更する代わりに、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。次のものも、より効率的に管理できます。
  - 頻繁に使用され共有される式
  - 頻繁に変更されるリストやビジネスロジック
- **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、Java コード、ロール、またはワークフローを意識する必要はなく、規則の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**。高度な知識を持つ開発者であれば、非常に複雑なロジックの規則を記述できますが、そうでないユーザーの場合、インタフェースの複雑な基盤には注意を向けません。

ユーザーの資格情報や個人情報などの機密データを、承認されていない管理者によるアクセスから守るために、規則をセキュリティー保護することもできます。詳細は、28 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。

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フィールドの値を算定したり、<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、これらの規則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。たとえば、重要なフォーム内で使用される規則の実装を、任意の Identity Manager ユーザーが変更できるとしたら重大な問題を引き起こします。規則のセキュリティー保護については、28 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。

次の例は、役職のリストを返す規則を示しています。

コード例 1-2　　役職リストを返す

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

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フォームを すべて修正する必要はありません。

次の例では、`global.jobTitle` フィールドがコード例 1-2 で定義された `Job Titles` 規則を呼び出して、役職リストを選択ボックスで使用しています。

注　　この例では、`rule` 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 1-3　　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>
```

Identity Manager のフォームでは、別の規則の名前を動的に計算して呼び出す規則もサポートします。次の例は、フォームフィールドによって部門コードを計算する規則を呼び出す方法を示しています。

コード例 1-4 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>
        < 規則 >
          <cond>
            <eq>
               <ref>var1</ref>
               <s>Admin</s>
            </eq>
            <s>AdminRule</s>
            <s>DefaultRule</s>
          </cond>
        </rule>
     </Expansion>
</Field>
```

ロール内での規則の使用

Identity Manager における「ロール」とは、リソースを効率的にグループ化してユーザーに割り当てられるようにするオブジェクトのことです。ロールには、次のような所有者と承認者が指定されます。

- ロールを定義するパラメータに対する変更を承認できるのは、ロールの所有者だけです。
- ロールに対するエンドユーザーの割り当てを承認できるのは、ロールの承認者だけです。

ロールの所有者と承認者は、直接、またはロール割り当て規則を使用して動的にロールに割り当てることができます。

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。Identity Manager によって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。

注 規則の詳細については、『Sun™ Identity Manager 管理ガイド』を参照してください。

次の例は、規則を使用して特定のリソースの属性値を設定する方法を示しています。ユーザーを作成し、この規則をそのユーザーのロールに関連付けると、この規則によって記述値が自動的に設定されます。

コード例 1-5 ユーザーのリソース記述の値の設定

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

ワークフロー内での規則の使用

一般に、Identity Manager の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、定義された手順の規則セットに従って、アクションの関与者から別の関与者に渡されます。関与者は人、マシン、またはその両方の場合があります。

ワークフロー内では、式を使用できる場所ではどこでも規則を使用できます。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
- 別の規則の名前を計算する
- 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

たとえば、手動アクションを使用して、承認リクエストを管理者に送信することができます。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、アクションを終了させて、ワークフローの承認を別の管理者にエスカレーションすることができます。

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。たとえば、次のようにします。

コード例 1-6　　規則名の動的な計算

```
<Activity id='0' name='activity1'>
   <Variable name='ValueSetByRule'>
     < 規則 >
       <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>
```

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。関連する機能をグループ化したいときに、規則を規則ライブラリに追加します。規則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るので、規則の保守が簡単になります。また、フォームやワークフローを設計するときに、有用な規則を簡単に特定して呼び出せるようになります。

注　　規則ライブラリ内の規則の呼び出し方法については、21 ページの「ライブラリ内の規則の呼び出し」を参照してください。

次の例は、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 1-7 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>
```

注 オープンソースの Identity Manager Integrated Development Environment (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、https://identitymanageride.dev.java.net を参照してください。

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、8 ページの「ロール内での規則の使用」および『Sun™ 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、29 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 を使用した規則の作成については、『Sun™ 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。

ベストプラクティス :

規則を設計する際には、経験の少ないユーザーが Identity Manager IDE を使用して規則をさらにカスタマイズすることが、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、Identity Manager の規則は 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
- 静的な値を返す
- 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 1-8 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 1-8 <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>
```

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、<rule name='rulename'> のように **r** が小文字の XPRESS の <rule> 要素を使用します。

静的な値を返す

静的な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。次の例では、文字列のリストを返しています。

コード例 1-9 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™ 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

規則内で <ref> 式を使用すると、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。

- フォームでは、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。
- ワークフローでは、ワークフロープロセス内で定義された任意の変数を参照できます。

次の例では、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 1-10 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>
```

次の例は、ワークフローが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テストする方法を示しています。このワークフローは、ユーザービューを含む `user` 変数を定義します。この規則は、このユーザーにシミュレートされたリソースが割り当てられている場合に `true` を返し、シミュレートされた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 1-11 遷移のテスト

```
<Rule name='Has Simulated Resources'>
   <notnull>
      <ref>user.accountInfo.types[simulated].accounts</ref>
   </notnull>
</Rule>
```

引数を使用した規則の宣言

> ➤|∞∞ ベストプラクティス :
>
> 規則に対して引数を宣言することは必須ではありませんが、宣言することが推奨されています。規則の実行時に「スコープ内」にある変数を使用する規則は、再利用が難しくなります。

規則内で引数を宣言することにより、規則のユーザー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。

<RuleArgument> 要素を使用して規則引数を宣言し、引数名の後に value を指定することによって、引数のデフォルト値を設定できます。
たとえば、次の規則では、規則引数 location のデフォルト値に「Austin」を指定しています。

コード例 1-12 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>
```

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には、<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、規則定義の 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 1-13 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、19 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 1-14 <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たり、値の取得以外のアクションが実行されたりした方がよい場合もあります。規則内で次の XPRESS 式を使用すると、外部変数に値を割り当てることができます。

- <setvar>: 変数の値を指定するために使用します。
- <setlist>: リスト内の指定した位置に値を割り当て、現在の値を上書きするために使用します。
- <putmap>: オブジェクトへのマップ要素を指定するために使用します。

次の例では、規則が 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る方法を示しています。

コード例 1-15 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 で作成し、JavaScript を XPRESS の <script> 要素でラップするほうが都合がよい場合もあります。

次の例では、フォームおよびワークフロー変数の値を参照し、env.get 関数を呼び出して変数名を渡しています。この例では、変数名を割り当てるために env.put 関数を使用しているので、スクリプト内の最後の文の値が規則の値になります。規則は email 変数に値を戻します。

コード例 1-16 <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>
```

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

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
- ライブラリ内の規則の呼び出し
- 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。このインタプリタは、リポジトリから規則を自動的に読み込み、評価します。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 要素を使用して引数 accountId を規則に渡しています。また、引数 value は静的文字列 jsmith として渡されています。

```
<rule name='getEmployeeId'>
   <argument name='accountId' value='jsmith'/>
</rule>
```

式を使用して、次のように引数の値を計算することもできます。
この例では、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。

```
<rule name='getEmployeeId'>
   <argument name='accountId'>
      <ref>user.waveset.accountId</ref>
   </argument>
</rule>
```

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>
```

上記の例では両方とも、表示属性 user.waveset.account の値が引数の値として渡されています。

ライブラリ内の規則の呼び出し

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、ライブラリ内の各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'/>
```

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。

- 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。
- 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。
- 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ができます。
  オプションの <RuleArgument> 要素を使用することができます。この要素については、16 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。

- フォーム内の範囲または明示的な引数の呼び出し
- ワークフロー内のローカル範囲オプションの使用
- 規則引数宣言の使用
- ロックされた引数の使用

フォーム内の範囲または明示的な引数の呼び出し

ここでは、フォーム内で規則引数を解決する方法の例を示します。

次の例は、フォームに規則を追加する方法を示しています。ユーザービューには属性名があるので、このフォームはユーザービューで使用できます。

```
<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`

次の例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 1-17 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、この規則はユーザービューの属性を参照するため、ユーザーフォームからしか呼び出せません。ワークフローでは通常 `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。

次の例は、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 1-18 firstname および lastname 変数を参照する規則

```
<Rule name='generateEmail'>
<RuleArgument name='firstname'/>
<RuleArgument name='lastname'/>
   <concat>
      <ref>firstname</ref>
      <s>.</s>
      <ref>lastname</ref>
      <s>@example.com</s>
   </concat>
</Rule>
```

次の例は、より簡潔で一般的な規則を示しています。この例に示す規則は、ユーザーフォームから呼び出すことを前提としていませんが、明示的な引数を指定して呼び出す必要があります。

コード例 1-19 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。

ワークフロー内のローカル範囲オプションの使用

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。次の例は、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 1-20 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

```
<アクション>
   <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に対して、変数 lastname の値を供給するように要求が行われます。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 オプションを指定するようにしてください。

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 1-21 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 localScope オプションを使用するようにします。

規則引数宣言の使用

> ➤|∞∞ ベストプラクティス :
>
> 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ことは必須ではありませんが、ベストプラクティスとして推奨されています。

引数宣言を使用すると、次のような利点があります。

- 宣言が、規則の呼び出し側にとってドキュメントの役割を果たすことができる
- 宣言によってデフォルト値を定義できる
- Identity Manager IDE で宣言を使用することで、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 で宣言を使用することで、規則呼び出しの設定を単純化できる

たとえば、次のように `generateEmail` 規則を再作成することも可能です。

コード例 1-22 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。

この例では、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。

次の例は、文字列 john.smith@example.com を生成する呼び出しを示しています。

コード例 1-23 john.smith@example.com 文字列の生成

```
<rule name='generateEmail'>
   <argument name='firstname' value='john'/>
   <argument name='lastname' value='smith'/>
</rule>
```

次の例は、文字列 john.smith@yourcompany.com を生成する呼び出しを示しています。

コード例 1-24 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>
```

次の例は、文字列 john.smith@ を生成する呼び出しを示しています。

コード例 1-25 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。

ロックされた引数の使用

引数をデフォルト値で宣言する手法は、規則の開発とカスタマイズを簡単にするのに有用です。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、規則を設定するための簡素化されたユーザーインタフェースが備わっています。引数のデフォルト値を Identity Manager IDE 内で変更することができ、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。

引数を宣言した後、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。ただし、引数値の制御を呼び出し側に渡したくない場合は、`RuleArgument` 要素に `locked` 属性を組み込んで `true` の値を指定することで、引数をロックします。次に例を示します。

コード例 1-26 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、引数の値は常に `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、管理者に必要なの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。管理者が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 ユーティリティーを規則が呼び出す場合は、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る必要があります。

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。

- 適切な組織に規則を配置
- 規則をセキュリティー保護するための認証タイプを使用
- より安全な規則を参照する規則へのアクセス制御

適切な組織に規則を配置

大部分の管理者は利便性を考慮して、副作用なしで計算を実行する、というような単純な規則を `All` という組織に配置し、規則を表示する権限のある人は全員規則にアクセスできるようにします。

ただし、規則のセキュリティー保護を強化したい場合は、次のように操作します。

- 機密情報を含む規則は `All` に配置しない。
- `Top` (または適正な上位レベルの組織) などの適切な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する。

規則をセキュリティー保護するための認証タイプを使用

いくつかの認証タイプ (`AuthType`) を使用して、規則などの所定の Identity Manager objectType のオブジェクトのサブセットの範囲を指定したり、アクセスを制限したりすることもできます。たとえば、ユーザーフォームで選択する規則を設定する場合は、ユーザーが制御範囲内のすべての規則にアクセスできるようにしない方がいいかもしれません。

認証タイプの使用方法の詳細については、『Sun™ Identity Manager 管理ガイド』の「認可タイプを使用したオブジェクトのセキュリティー保護」を参照してください。

より安全な規則を参照する規則へのアクセス制御

安全な規則を参照する規則にアクセスできるユーザーは、その安全な規則の内容を呼び出したり、表示したり、変更することができます。

Identity Manager では承認チェックを実行し、その規則を編集する権限を持つすべてのユーザーをラッパーで呼び出します。承認されたユーザーは、それ以上承認チェックを受けることなく、その規則を使用してほかの規則を呼び出すことができます。このようにして、ユーザーは安全な規則に間接的にアクセスできるようになります。

安全な規則を参照する規則を作成して、セキュリティーの度合いの低い方の規則に対するアクセス権をユーザーに付与する場合、誤って安全な規則に対する不適切なアクセス権をユーザーに付与しないよう注意してください。

注 より安全な規則を参照する規則を作成するには、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。最初の規則を実行する権限と、安全な規則を呼び出す権限も必要です。

デフォルト規則および規則ライブラリのカスタマイズ

ここでは、Identity Manager によって提供されるデフォルト規則と規則ライブラリについて説明します。説明する内容は次のとおりです。

- Identity Manager の規則
- 監査規則
- 監査ポリシー規則
- Service Provider 規則

注 これらの規則と規則ライブラリをカスタマイズするには、Identity Manager IDE を使用します。

Identity Manager の規則

Identity Manager をカスタマイズするには、次の規則および規則ライブラリを使用します。

- AccessEnforcerLibrary
- ActiveSync 規則
- ADRules ライブラリ
- 英数字規則ライブラリ
- Approval Transaction Message
- Approval Transaction Message Helper
- Attestation Remediation Transaction Message
- Attestation Remediation Transaction Message Helper
- Attestation Transaction Message
- Attestation Transaction Message Helper
- CheckDictionaryWord
- DateLibrary
- End User Controlled Organizations
- EndUserRuleLibrary
- ExcludedAccountsRule
- getAvailableServerOptions
- InsertDictionaryWord
- Is Manager
- LoginCorrelationRules
- My Direct Reports
- NamingRules ライブラリ
- NewUsernameRules
- Object Approvers As Attestors
- Object Owners As Attestors
- Organization Names
- OS400UserFormRules
- IS_DELETE

- RACFUserFormRules
- 調整規則
- RegionalConstants ライブラリ
- Remediation Transaction Message
- Remediation Transaction Message Helper
- ResourceFormRules
- Resource Names
- Role Approvers
- Role Notifications
- Role Owners
- Sample On Local Network
- SAP Portal User Form Default Values
- ShellRules
- SIEBEL_NAV_RULE
- TestDictionary
- TopSecretUserFormRules
- ユーザーメンバー規則
- USER_EMAIL_MATCHES_ACCOUNT_EMAIL_CONF
- USER_EMAIL_MATCHES_ACCOUNT_EMAIL_CORR
- USER_FIRST_AND_LAST_NAMES_MATCH_ACCOUNT
- USER_NAME_MATCHES_ACCOUNT_ID
- USER_OWNS_MATCHING_ACCOUNT_ID
- Users Without a Manager
- Use SubjectDN Common Name

AccessEnforcerLibrary

AccessEnforcerLibrary は、特定タイプのオブジェクトを管理できるようにする規則のデフォルトのライブラリです。Access Enforcer リソースアダプタではこれらのオブジェクトを取得できないからです。

入力値 : 表 1-2 を参照してください。

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

AuthType: Library

SubType: `listRules`

返される値 : 表 1-2 を参照してください。

定義済み規則 : 指定しない

次の表は、AccessEnforcerLibrary 規則の例をまとめたものです。

表 1-2 AccessEnforcerLibrary 規則の例

規則名	入力変数	説明
getApplications	• `resName` (リソース名) • 文字列を手動で入力して Access Enforcer オブジェクト名を指定します。	SAP GRC Access Enforcer で使用可能なアプリケーションのリストを返します。`resName` が指定されたら、Access Enforcer からアプリケーションを取得します。それ以外の場合は、静的に指定されたリストを返します。
getRoles	`resName` (リソース名)	SAP GRC Access Enforcer で使用可能なロールのリストを返します。これらのロールは、バックエンドシステムで使用可能なロールと同じです。 これらの値は手動で作成し、SAP GRC Access Enforcer の対応する値と同期する必要があります。
getRequestTypes	なし	SAP GRC Access Enforcer で使用可能な要求タイプのリストを返します。 これらの値は手動で作成し、SAP GRC Access Enforcer の対応する値と同期する必要があります。
getPriorities	なし	SAP GRC Access Enforcer で使用可能な優先度の値のリストを返します。 これらの値は手動で作成し、SAP GRC Access Enforcer の対応する値と同期する必要があります。
getEmployeeTypes	なし	SAP GRC Access Enforcer で使用可能な従業員タイプのリストを返します。 これらの値は手動で作成し、SAP GRC Access Enforcer の対応する値と同期する必要があります。

表 1-2　AccessEnforcerLibrary 規則の例 (続き)

規則名	入力変数	説明
getSLAs	なし	SAP GRC Access Enforcer で使用可能なサービスレベルのリストを返します。 これらの値は手動で作成し、SAP GRC Access Enforcer の対応する値と同期する必要があります。
getSupporttedVersions	`resName` (リソース名)	Identity Manager でサポートされる SAP GRC Access Enforcer バージョンのリストを返します。これらの値は、アダプタファセットの認識する値と同じでなくてはなりません。

ActiveSync 規則

フラットファイル 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。アダプタは処理規則、相関規則、削除規則を使用して、ユーザーの実行内容を決定します。

注　Active Sync 規則には必ず `display.session` ではなく、`context` を使用します。相関規則と削除規則はセッションを取得しませんが、確認規則は取得します。詳細については、59 ページの「相関規則」と 60 ページの「確認規則」を参照してください。

入力値 : `activeSync` 名前空間のリソースアカウント属性を受け入れます。例 : `activeSync.firstname`

カスタムの ActiveSync 規則には、次を指定する必要があります。

AuthType: 指定しない

SubType: 指定しない

名前空間 : `activeSync` 名前空間のリソースアカウント属性を設定します。次に例を示します。

```
activeSync.firstname
```

定義済み規則 : 定義済みの ActiveSync 規則には次のようなものがあります。

- ActiveSync has isDeleted set: `Process deletes as updates` パラメータを `false` に設定した場合のリソースからの移行で使用されます。

注	規則の名前は変更しないでください。別の規則名を使用する必要のある場合は、規則の内容を複製して新しく作成した規則の名前を変更してください。

- No Correlation Rule: 相関が不要な場合は、このデフォルト規則を使用します。
- No Confirmation Rule: 確認が不要な場合は、このデフォルト規則を使用します。

ADRules ライブラリ

ADRules のデフォルトライブラリを使用すると、サーバーのリストを作成できます。

入力値 : なし

カスタムの ADRules 規則には、次を指定する必要があります。

AuthType: 指定しない

SubType: 指定しない

呼び出し :

返される値 : ゼロ以上の文字列値のリスト。

定義済み規則 : なし

表 1-3 ADRules 規則の例

規則名	説明
Exchange Servers	環境内の Exchange サーバーのリストを返します。 このリストを更新して、環境内に Exchange サーバーを追加することができます。
Home Directory Servers	環境内のホームディレクトリサーバーのリストを返します。 このリストを更新して、環境内にホームディレクトリドライブを提供するシステムを追加することができます。
AD Login Scripts	環境内で使用されるユーザーログインスクリプトのリストを返します。 このリストを更新して、環境内にログインバッチスクリプトを追加することができます。
Home Directory Drive Letter	環境内のホームディレクトリのマッピングされたドライブ文字のリストを返します。 このリストを更新して、環境内に共通のホームディレクトリマップドライブ文字を追加することができます。

表 1-3　ADRules 規則の例 (続き)

規則名	説明
Home Directory Volumes	環境内のホームディレクトリボリューム名のリストを返します。 このリストを更新して、環境内に共通のホームディレクトリボリューム名を追加することができます。Identity Manager はこの値とホームディレクトリサーバーの値を使用して、ユーザーのホームディレクトリを作成します。このボリュームは、選択したホームディレクトリサーバー上に存在し、共有されている必要があります。

英数字規則ライブラリ

英数字規則ライブラリは、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るようにする規則のデフォルトライブラリです。

注　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。

入力値 : 表 1-4 を参照してください。

カスタム規則には、次を指定する必要があります。

AuthType: `EndUserRule`

SubType: 指定しない

返される値 : ゼロ以上の文字列のリスト。

次の表は、英数字規則ライブラリの規則の例をまとめたものです。

表 1-4　英数字規則の例

規則名	入力変数	説明
AlphaCapital	なし	英大文字のリストを返します。
AlphaLower	なし	英小文字のリストを返します。
Numeric	なし	数字のリストを返します。
WhiteSpace	なし	空白文字のリストを返します。
SpecialCharacters	なし	共通特殊文字のリストを返します。
legalEmailCharacters	なし	電子メールに適している特殊文字のリストを返します。

表 1-4　英数字規則の例 (続き)

規則名	入力変数	説明
stringToChars	testStr	指定された文字列を文字列の個々の文字で構成されたリストに変換します。
isNumeric	testStr	testStr に数字のみが含まれているかどうかをテストして確認します。
isAlpha	testStr	testStr に英文字のみが含まれているかどうかをテストして確認します。
hasSpecialChar	testStr	testStr 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	testStr	testStr 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	testStr	testStr に電子メールアドレスに適した文字のみが含まれているかどうかをテストして確認します。
StripNonAlphaNumeric	testStr	英数字以外の文字を testStr から削除します。

Approval Transaction Message

Approval Transaction Message 規則は、承認トランザクションのテキストをフォーマットするために使用されるデフォルトの規則です。この規則をカスタマイズして、ユーザーの署名用にさらに多くの情報を設定できます。

入力値 : 次の引数を受け入れます。

- workItemList: 承認される workitems のセット
- variablesList: workitemList 内の各 workitem に対応する変数のセット
- approverName: workitems の承認を求められるユーザー

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

AuthType: 指定しない

SubType: 指定しない

返される値 : workItemList 内の workitems のリスト用にフォーマットされたトランザクションテキスト。

定義済み規則 : なし

Approval Transaction Message Helper

Approval Transaction Message Helper 規則は、単一の workitem の承認用にフォーマットされたトランザクションテキストを返します。

入力値 : 次の引数を受け入れます。

- workItem: 承認される workitem
- variables: workitem 変数

カスタムの Approval Transaction Message Helper 規則には、次を指定する必要があります。

AuthType: 指定しない

SubType: 指定しない

返される値 : 単一の workitem の承認用にフォーマットされたトランザクションテキスト。

定義済み規則 : なし

Attestation Remediation Transaction Message

Attestation Remediation Transaction Message 規則は、アテステーション是正トランザクションのテキストをフォーマットするために使用されるデフォルトの規則です。この規則をカスタマイズして、ユーザーの署名用にさらに多くの情報を設定できます。

入力値 : 次の引数を受け入れます。

- workItemList: 承認される workitems のセット
- variablesList: workitemList 内の各 workitem に対応する変数のセット
- approverName: workitems の承認を求められるユーザー
- action: 値 remediate を想定
- actionComments: 是正の一部として入力されるコメント

カスタムの Attestation Remediation Transaction Message 規則には、次を指定する必要があります。

AuthType: EndUserAuditorRule

SubType: 指定しない

返される値 : フォーマットされたアテステーション是正トランザクションテキスト。

定義済み規則 : なし

Attestation Remediation Transaction Message Helper

Attestation Remediation Transaction Message Helper 規則は、単一の workitem のアテステーション是正用にフォーマットされたトランザクションテキストを返します。

入力値 : 次の引数を受け入れます。

- workItem: 承認される workitem
- variables: workitem 変数

カスタムの Attestation Remediation Transaction Message Helper 規則には、次を指定する必要があります。

AuthType: EndUserAuditorRule

SubType: 指定しない

呼び出し :

返される値 : 単一の workitem のアテステーション是正用にフォーマットされたトランザクションテキスト。

定義済み規則 : なし

Attestation Transaction Message

Attestation Transaction Message 規則は、アテステーショントランザクションのテキストをフォーマットするために使用されるデフォルトの規則です。この規則をカスタマイズして、ユーザーの署名用にさらに多くの情報を設定できます。

入力値 : 次の引数を受け入れます。

- workItemList: 承認される workitems のセット
- variablesList: workitemList 内の各 workitem に対応する変数のセット
- approverName: workitems の承認を求められるユーザー
- action: 値 approved または approve を想定
- actionComments: アテステーションの一部として入力されるコメント

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

AuthType: EndUserAuditorRule

SubType: 指定しない

呼び出し :

返される値 : フォーマットされたアテステーショントランザクションテキスト。

定義済み規則 : なし

Attestation Transaction Message Helper

Attestation Transaction Message Helper 規則は、単一のアテステーション用にフォーマットされたトランザクションテキストを返します。

入力値 : 次の引数を受け入れます。

- `workItem`: 承認される `workitem`
- `variables`: `workitem` 変数

カスタムの Attestation Transaction Message Helper 規則には、次を指定する必要があります。

AuthType: `EndUserAuditorRule`

SubType: 指定しない

呼び出し :

返される値 : 単一のアテステーション用にフォーマットされたトランザクションテキスト。

定義済み規則 : なし

CheckDictionaryWord

CheckDictionaryWord 規則を使用して辞書に対して JDBC クエリーを実行し、その辞書内にパスワードが存在するかどうか確認します。

入力値 : 次の引数を受け入れます。

- `type`
- `driverClass`
- `driverPrefix`
- `url`
- `host`
- `port`
- `database`
- `context`
- `user`
- `password`

- `sql`
- `arg1`

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

AuthType: 指定しない

SubType: 指定しない

呼び出し :

返される値 : ゼロ以上の文字列のリスト。

定義済み規則 : なし

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。

注	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。

入力値 : 表 1-5 を参照してください。

カスタム DateLibrary 規則には、次を指定する必要があります。

AuthType: `Rule`

SubType: 指定しない

返される値 : ブール型の値 (`true` または `false`)。表 1-5 を参照してください。

次の表は、DateLibrary 規則の例をまとめたものです。

表 1-5 DateLibrary 規則の例

規則	入力変数	説明
`Date Validation`	`mm/dd/yy yy`	有効な日付の文字列を決定します。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。 • 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` • 入力された文字列に無効な日付コンポーネントが含まれる場合は `false`
`Validate Day Month Year`	• `month` • `day` • `year`	有効な年、月、日の文字列を決定します。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。 • 入力された文字列が有効な日付である場合は `true` • 入力された文字列が無効な日付である場合は `false`
`Validate Time`	`HH:mm:ss`	有効な時間の文字列を決定します。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。 • 入力された文字列が有効な時間である場合は `true` • 入力された文字列が無効な時間である場合は `false`

End User Controlled Organizations

End User Controlled Organizations の規則は、エンドユーザーインタフェースにログインするユーザーによって制御される組織のセットを決定します。これらの組織とエンドユーザー組織により、ユーザーが EndUser 機能 (AdminGroup) で指定される権限に対する制御の範囲が定義されます。これは規則なので、エンドユーザーインタフェースにログインするユーザーによって、制御の範囲を変えることができます。

入力値 : 認証するエンドユーザーのユーザービュー

カスタムの End User Controlled Organizations の規則には、次を指定する必要があります。

AuthType: EndUserControlledOrganizationsRule

SubType: 指定しない

返される値 : 単一の管理される組織 (string) または管理される組織のリスト。それぞれの値は組織名や ID などです。組織名が返された場合、Top まで完全修飾される必要があります (例 : Top:Marketing:South)。

定義済み規則 : デフォルトでは、ユーザーがメンバーである組織を返します (例 : waveset.organization)。

EndUserRuleLibrary

EndUserRuleLibrary は、Identity Manager がエンドユーザーのアカウント情報を決定または検証するために使用する規則のデフォルトライブラリです。

注 デフォルトでは、Identity Manager のエンドユーザー匿名登録プロセスによって、ユーザーが指定する名 (`firstName`)、姓 (`lastName`)、従業員 ID (`employeeID`) を使用して、`accountId` と `emailAddress` の値が生成されます。匿名登録を行うと、電子メールアドレスとアカウント ID に非 ASCII 文字が表示できるようになります。

匿名登録の処理中に Identity Manager で使用されるアカウント ID や電子メールアドレスが ASCII 文字に保持されるように、国際ユーザーは次の手順を実行する必要があります。

1. 次の EndUserRuleLibrary 規則を変更します。
   - `getAccountId`: `firstName`、`lastName`、および `letter substr` を削除します。`employeeId` のみを使用します。
   - `getEmailAddress`: `firstName`、`lastName`、および `"."` を削除します。`employeeId` のみを使用します。
   - `verifyFirstname`: 長さチェックを `2` から `1` に変更して、単一文字のアジア名を入力できるようにします。
2. エンドユーザーの End User Anon Enrollment Completion form を編集して、`getAccountId` 規則と `getEmailAddress` 規則への呼び出しから、引数 `firstName` および `lastName` を削除します。

注 Identity Manager IDE では、このライブラリは `EndUserRuleLibrary` ライブラリオブジェクトとして表示されます。

入力値 : 表 1-6 と表 1-7 を参照してください。

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

AuthType: `EndUserLibrary`

SubType: 指定しない

次の表は、EndUserRuleLibrary 規則の例をまとめたものです。

表 1-6 EndUserRuleLibrary 規則の例

規則	入力変数	説明
getCallerSession	なし	フォームを実行するユーザー用の内部セッションコンテキスト (Lighthouse コンテキスト) を返します。
getUserView	• resourceTargets リスト • accountId 文字列 • includeAvailableRoleInfos ブール値	指定された accountId のユーザービューを返します。これには、リソースターゲットのリストやロール情報を含むかどうかが含まれます。
getView	• nameOrId 文字列 • type 文字列 • options マップ	名前または GUID、オブジェクトのタイプ、およびオプションのマップによって指定されたオブジェクトのビューを返します。
getUnassignedResources	• roles リスト • currentResources リスト • groups リスト	現在割り当てられていないリソースを特定します。
getDirectReports	• manager 文字列 • options マップ	指定されたマネージャーの直属の部下のリストを返します。たとえば、idmManager 属性が manager という入力変数によって指定されているユーザーのリストです。
getIndirectReports	• manager 文字列 • options マップ	指定されたマネージャーの直属ではない部下のリストを返します。たとえば、manager という入力変数によって指定されているユーザーのレポート構造内のユーザーのうち、直属の部下を除いたリストです。
getResourceObjectParentId	• resourceName 文字列 • resObjectName 文字列 • objType 文字列 • objAttr 文字列	名前、オブジェクトタイプ、オブジェクト属性によって指定されたリソースの親の GenericObject を返します。
getObjectsByType	• type 文字列 • attributeVal 文字列 • attributeName 文字列	タイプによって指定され、attributeName=attributeVal 条件と一致する GenericObject のリストを返します。

表 1-6 EndUserRuleLibrary 規則の例 (続き)

規則	入力変数	説明
getRealName	• accountId 文字列 • addAccountId ブール値	accountId が設定されている場合に、ユーザーの「実際の名前」(FirstName <space> LastName など) を特定します。 • addAccountId 引数が true の場合、Identity Manager は FirstName LastName (accountId) 文字列を返します。 • FirstName または LastName 属性を特定できない場合、規則は accountId のみを返します。 **注 :** • 実際の名前を LastName, FirstName のように表示したい場合、この規則は簡単に変更できます。 • ユーザーには、ほかのユーザーを検索できるような適切な権限が必要です。

次の表は、匿名登録に使用される EndUserRuleLibrary 規則の例をまとめたものです。

表 1-7 匿名登録用の EndUserRuleLibrary 規則の例

規則	入力変数	説明
getAccountId	• firstName 文字列 • lastName 文字列 • employeeId 文字列	名、姓、従業員 ID からアカウント ID を生成します。 First initial + last intial + employee ID **注 :** 匿名登録の処理中に Identity Manager で使用されるアカウント ID や電子メールアドレスが ASCII 文字に保持されるように、国際ユーザーはこの規則を変更する必要があります。手順については、43 ページを参照してください。

表 1-7 匿名登録用の EndUserRuleLibrary 規則の例 (続き)

規則	入力変数	説明
getEmailAddress	• firstName 文字列 • lastName 文字列 • emailDomain 文字列	入力された名、姓、電子メールドメインから電子メールアドレスを生成します。firstname.lastname@emailDomain **注** : 匿名登録の処理中に Identity Manager で使用されるアカウント ID や電子メールアドレスが ASCII 文字に保持されるように、国際ユーザーはこの規則を変更する必要があります。手順については、43 ページを参照してください。
getIdmManager	employeeId 文字列	作成するユーザーの従業員 ID に関連付けられる Identity Manager マネージャーのアカウント ID を返します。この規則は自分の開発環境用にカスタマイズする必要があります。(デフォルトは *configurator*。)
getOrganization	なし	ユーザーが割り当てられる組織の名前を返します。この規則は自分の開発環境用にカスタマイズする必要があります。 (デフォルトは *Top*。)
runValidation	なし	verifyFirstname、verifyLastname、verifyEmployeeId、および verifyEligibility 規則を呼び出します。
verifyFirstname	firstName 文字列	エンドユーザー匿名登録プロセスのためにユーザーが入力した名を検証します。このサンプル規則では、名が NULL でないことを検証しています。この規則は自分の開発環境用にカスタマイズする必要があります。 **注** : 匿名登録の処理中に Identity Manager で使用されるアカウント ID や電子メールアドレスが ASCII 文字に保持されるように、国際ユーザーはこの規則を変更する必要があります。手順については、43 ページを参照してください。
verifyLastname	lastName 文字列	エンドユーザー匿名登録プロセスのためにユーザーが入力した姓を検証します。このサンプル規則では、姓が NULL でないことを検証しています。この規則は自分の開発環境用にカスタマイズする必要があります。

表 1-7 匿名登録用の EndUserRuleLibrary 規則の例 (続き)

規則	入力変数	説明
verifyEmployeeId	employeeId 文字列	エンドユーザー匿名登録プロセスのためにユーザーが入力した従業員 ID を検証します。このサンプル規則では、従業員 ID が有効であることを検証しています。この規則は自分の開発環境用にカスタマイズする必要があります。
verifyEligibility	• firstName 文字列 • lastName 文字列 • employeeId 文字列	エンドユーザー匿名登録プロセスのためにユーザーが入力した従業員 ID を検証するために使用できます。この規則は開発用にカスタマイズする必要があります。

ExcludedAccountsRule

ExcludedAccountsRule 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 : 次の引数を受け入れます。

- accountId: テストされる文字列アカウント ID。

  accountId 引数を、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 引数を使用して、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アカウントはすべて、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)

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

AuthType: ExcludedAccountsRule

SubType: 指定しない

定義済み規則 :

- Microsoft SQL Server Excluded Resource Accounts
- Sun Access Manager Excluded Resource Accounts
- Unix Excluded Resource Accounts
- Windows Excluded Resource Accounts

次の例は、サブタイプの使用方法を示していますが、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 1-27 authType の使用例

```
<Rule name='ExcludedResourceAccounts'
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>
```

次の例は、operation パラメータの使用方法を示しています。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 1-28 operation パラメータの使用例

```
<Rule name='Example Excluded Resource Accounts'
authType='ExcludedAccountsRule'>
<!--
「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>
            </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>
```

コード例 1-28 operation パラメータの使用例 (続き)

```
      </and>
   </or>
</Rule>
```

次の例は、RACF 用の ExcludedAccountsRule を示しています。

コード例 1-29 RACF 用の ExcludedAccountsRule

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE Waveset PUBLIC "waveset.dtd" "waveset.dtd">
<Waveset>
 <Rule name="RACF EAR" authType="ExcludedAccountsRule">
   <RuleArgument name="accountID"/>
   <block>
     <defvar name="excludedList">
       <List>
         <String>irrcerta</String>
         <String>irrmulti</String>
         <String>irrsitec</String>
         <String>IBMUSER</String>
       </List>
     </defvar>
     <cond>
       <eq>
         <containsAny>
           <ref>excludedList</ref>
           <list>
             <upcase>
               <ref>accountID</ref>
             </upcase>
             <ref>accountID</ref>
           </list>
         </containsAny>
         <i>1</i>
       </eq>
       <Boolean>true</Boolean>
       <Boolean>false</Boolean>
     </cond>
   </block>
   <MemberObjectGroups>
     <ObjectRef type="ObjectGroup" id="#ID#Top" name="Top"/>
   </MemberObjectGroups>
 </Rule>
</Waveset>
```

最後の例は、RACF LDAP 用の ExcludedAccountsRule を示しています。

コード例 1-30 RACF LDAP 用のアカウント除外規則

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE Waveset PUBLIC "waveset.dtd" "waveset.dtd">
<Waveset>
<Rule name="Test RACF_LDAP Case Insensitive Excluded Resource Accounts"
authType="ExcludedAccountsRule">
 <RuleArgument name="accountID"/>
 <block>
   <defvar name="excludedList">
     <List>
       <String>irrcerta</String>
       <String>irrmulti</String>
       <String>irrsitec</String>
       <String>IBMUSER</String>
     </List>
   </defvar>
   <defvar name="convertedId">
     <get>
       <split>
         <get>
           <split>
             <ref>accountID</ref>
             <s>,</s>
             </split>
           <i>0</i>
         </get>
         <s>=</s>
       </split>
       <i>1</i>
     </get>
   </defvar>
   <cond>
     <eq>
       <containsAny>
         <ref>excludedList</ref>
         <list>
           <upcase>
             <ref>convertedId</ref>
           </upcase>
           <ref>convertedId</ref>
```

getAvailableServerOptions

getAvailableServerOptions 規則は、指定された同期メカニズムに対して使用できるサーバー設定オプションのリストを決定します。`Waveset.properties` 内の設定の使用は ActiveSync に対してのみ適用され、下位互換性のあるオプションです。

入力値 : `targetObjectType` 引数を受け入れます。

注	`IDMXUser` の場合、リストには `viaWavesetProperties` が返されません。

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

AuthType: 指定しない

SubType: 指定しない

定義済み規則 : なし

InsertDictionaryWord

InsertDictionaryWord 規則を使用して、Identity Manager 辞書に対して JDBC コマンドを実行し、データベースに新しい用語を読み込みます。

入力値 : 次の引数を受け入れます。

- `type`
- `driverClass`
- `driverPrefix`
- `url`
- `host`
- `port`
- `database`
- `context`
- `user`
- `password`
- `sql`
- `arg1`
- `argList`

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

AuthType: 指定しない

SubType: 指定しない

呼び出し :

返される値 : ゼロ以上の文字列のリスト。

定義済み規則 : なし

IS_DELETE

IS_DELETE 規則は、PeopleSoft Active Sync アダプタ用に作成されたサンプル規則で、Active Sync イベントでユーザーを削除する必要があるのかどうかを決定します。

入力値 : なし

カスタム IS_DELETE 規則には、次を指定する必要があります。

AuthType: 指定しない

SubType: 指定しない

定義済み規則 : なし

Is Manager

Is Manager 規則は、指定された `accountIds` がシステム内の他のユーザーのマネージャーであるかどうかをテストします。

入力値 : `managerId` 引数を受け入れます (`<RuleArgument name='managerId'/>`)。

カスタム Is Manager 規則には、次を指定する必要があります。

AuthType: `RoleConditionRule`

SubType: 指定しない

呼び出し :

返される値 : `managerId` がシステム内の任意のユーザーの `idmManager` として宣言されていれば **true** を返し、それ以外の場合は `false` を返します。

この規則は、呼び出し側の `display.session` セッションを使用してリポジトリ内でクエリーを発行します。つまり、この規則はフォームからしか呼び出すことができません。このチェックでは呼び出し側によって制御されている組織内のユーザーのみが一致対象となるので、`managerId` が呼び出し側の制御範囲外のユーザーのマネージャーである場合、この規則は `false` を返す場合があります。

定義済み規則 : なし

LoginCorrelationRules

LoginCorrelationRules は、ユーザーログイン情報を Identity Manager ユーザーにマップします。LoginCorrelationRules では、規則によって Identity Manager ユーザーを検索し、1 つ以上の `AttributeConditions` のリストを返すことができるようなロジックを指定します。

入力値 : なし

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

AuthType: `LoginCorrelationRule`

SubType: 指定しない

呼び出し : LoginModule によって呼び出し、ログイン情報を Identity Manager ユーザーにマップします。

返される値 : 1 つ以上の `AttributeConditions` のリスト。

定義済み規則 :

- Correlate via X509 Certificate SubjectDN
- Correlate via LDAP Uid

My Direct Reports

My Direct Reports 規則は、呼び出し側の直属の部下であるすべての Identity Manager ユーザーの名前を返します。管理機能は通常は階層構造になっていますが、この規則では、呼び出し側が自分のマネージャーに指定されているユーザーの名前のみを返します。この規則によって管理階層を飛び越えることはありません。

入力値 : なし

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

AuthType: `AccessScanRule`

SubType: `USER_SCOPE_RULE`

呼び出し :

返される値 : 呼び出し側が自分のマネージャーに指定されている Identity Manager ユーザー名のリスト。

定義済み規則 : なし

NamingRules ライブラリ

NamingRules ライブラリは、規則の処理後に名前が表示される方法を制御できるようにする規則のデフォルトライブラリです。

注	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。

入力値 : なし

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

AuthType: 指定しない

SubType: 指定しない

呼び出し :

返される値 :

定義済み規則 : なし

次の表は、NamingRules の例をまとめたものです。

表 1-8 NamingRules の例

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 **注 :** メールドメインに AccountName 規則を追加する必要があります。
Fullname - First space Last	Marcus Aurelius
Fullname - First space MI space Last	Marcus A Aurelius
Fullname - Last comma First	Aurelius, Marcus

NewUsernameRules

NewUsernameRule は、標準のリポジトリ初期化ファイルで、ユーザーの識別名 (DN) の一番左側の相対識別名 (RDN) の値を抽出するために使用できます。

入力値 : なし

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

AuthType: `NewUserNameRule`

SubType: 指定しない

呼び出し :

返される値 : 登録時に新しいユーザーに対して提示されるユーザー名。
たとえば、`Use SubjectDN Common Name` によって
`cn=jsmith,ou=engineering,dc=acme,dc=com` から `jsmith` が抽出されます。

定義済み規則 : Use SubjectDN Common Name

Object Approvers As Attestors

Object Approvers As Attestors 規則は、設定された `objectapprovers` パラメータの値が NULL でない場合に、その値を返します。`objectapprovers` リストが設定されていない場合、この規則は新しい規則を作成し、`Configurator` ユーザーを含めます。

入力値 : 次の引数を受け入れます。

- `userEntitlement`: `UserEntitlement` オブジェクトのビュー
- `lhcontext`: 呼び出し側の `LighthouseContext`
- `objectowners`: 所有者であると見なされる Identity Manager ユーザー名のリスト
- `objectapprovers`: 承認者であると見なされる Identity Manager ユーザー名のリスト

カスタムの Object Approvers As Attestors 規則には、次を指定する必要があります。

AuthType: `AccessScanRule`

SubType: `ATTESTORS_RULE`

呼び出し : 実行中のアクセスレビューによる

返される値 :

定義済み規則 : なし

Object Owners As Attestors

Object Owners As Attestors 規則は、設定された `objectowners` パラメータの値が NULL でない場合に、その値を返します。`objectowners` リストが設定されていない場合、この規則は新しい規則を作成し、`Configurator` ユーザーを含めます。

入力値 : 次の引数を受け入れます。

- `userEntitlement`: `UserEntitlement` オブジェクトのビュー
- `lhcontext`: 呼び出し側の `LighthouseContext`
- `objectowners`: 所有者であると見なされる Identity Manager ユーザー名のリスト
- `objectapprovers`: 承認者であると見なされる Identity Manager ユーザー名のリスト

カスタムの Object Owners As Attestors 規則には、次を指定する必要があります。

AuthType: `AccessScanRule`

SubType: `ATTESTORS_RULE`

呼び出し : 実行中のアクセスレビューによる

返される値 : Identity Manager ユーザー名のリスト。

定義済み規則 : なし

Organization Names

Organization Names 規則は、現在のコンテキスト内のすべての組織の表示名のリストを返します。

入力値 : なし

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

AuthType: 指定しない

SubType: 指定しない

呼び出し :

返される値 :

定義済み規則 : なし

OS400UserFormRules

OS400UserFormRules を使用して、OS400 リソースのデフォルトのユーザーフォームを管理します。

入力値 : なし

カスタム OS400UserFormRules 規則には、次を指定する必要があります。

AuthType: `EndUserLibrary`

SubType: 指定しない

呼び出し :

返される値 : 表 1-9 を参照してください。

定義済み規則 : OS400 User Form Default Values

次の表は、OS400UserFormRules の例をまとめたものです。

表 1-9 OS400UserFormRules の例

規則名	説明
Default Password Expiration Interval	パスワードの有効期間のデフォルト値を返します。 返される値は 90 です。
Default Initial Program Call	ユーザーによって呼び出されるデフォルトの初期プログラムを返します。 返される値は *LIB/CCTC00CLP です。
Max Storage List Choices	記憶領域の最大量のデフォルト値を返します。値はキロバイト単位で表され、最大値なし、10M バイト、50M バイト、100M バイトと等しくなります。
Initial Menu Default	初期メニューのデフォルト値を返します。 返される値は *SIGNOFF です。
Language ID Default	デフォルトの言語 ID の値を返します。 返される値は *SYSVAL です。
Country ID Default	デフォルトの国 ID の値を返します。 返される値は *SYSVAL です。
Character Set Default	デフォルトの文字セット値のリストを返します。 返される値は *SYSVAL です。
UID Default	UID のデフォルト値を返します。 返される値は *GEN です。
Home Directory Prepend	ホームディレクトリを形成するために、ユーザー ID の前に付加されるパス。

RACFUserFormRules

RACFUserFormRules を使用して、RACF リソースアカウントのデフォルト設定を指定します。

入力値 : なし

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

AuthType: `EndUserLibrary`

SubType: 指定しない

呼び出し : RACF ユーザーフォームから

返される値 : ゼロ以上の文字列値のリスト。

定義済み規則 : RACF User Form Default Values

次の表は、RACFUserFormRules の例をまとめたものです。

表 1-10 RACFUserFormRules の例

規則名	説明
Prepend RACF Home Dir Path	ホームディレクトリを形成するために `accountId` の前に付加されるパス。
RACF OMVS Program	デフォルトの OMVS プログラム値を指定します。
RACF TSO Command	デフォルトの OMVS TSO 値を指定します。
RACF Master Catalog	デフォルトの OMVS プログラム値を指定します。
RACF User Catalog	デフォルトの OMVS プログラム値を指定します。
RACF Delete TSO Segment	デフォルトの Delete TSO Segment 値を指定します。

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。

- 相関規則
- 確認規則

相関規則

Identity Manager は調整時に相関規則を呼び出して、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

入力値 : `ResourceAdapter#getUser(WSUser)` によって返されるリソースアカウントを表す `WSUser` を受け入れます。

カスタム相関規則には、次を指定する必要があります。

AuthType: 指定しない

SubType: `SUBTYPE_ACCOUNT_CORRELATION_RULE`

名前空間 : 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、次のフォーマットで設定されます。

`account.` *LHS Attr Name*

呼び出し : 調整時

返される値 : 指定したアカウントを所有する可能性のある既存のユーザーを選択するために使用できる条件。相関規則は、次のいずれかのフォームで条件を返すことができます。

- `WSUser NAME` と解釈される文字列
- それぞれが `WSUser NAME` と解釈される文字列要素のリスト
- `com.waveset.object.WSAttribute` 要素のリスト
- `com.waveset.object.AttributeCondition` 要素のリスト

Identity Manager は相関規則によって返される任意の条件セットを使用して、一致するユーザーをリポジトリに問い合わせます。

定義済み規則 : Default Correlation

確認規則

Identity Manager は調整時に確認規則を呼び出して、リソースアカウントを Identity Manager ユーザー (複数可) と比較します

入力値 : 次の引数を受け入れます。

- 既存の IDM ユーザーを表す `WSUser`
- `ResourceAdapter#getUser(WSUser)` によって返されるリソースアカウントを表す `WSUser`

カスタム確認規則には、次を指定する必要があります。

AuthType: なし

SubType: `SUBTYPE_ACCOUNT_CONFIRMATION_RULE`

名前空間 : リソースアカウントのすべての属性値とユーザービューのすべての属性値が、次のフォーマットで設定されます。

- `account.`*LHS Attr Name*
- `user.accounts[*].*`
- `user.waveset.*`
- `user.accountInfo.*`

呼び出し : 調整時

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`)。

定義済み規則 : Default Confirmation

RegionalConstants ライブラリ

RegionalConstants ライブラリは、州、日、月、国、およびカナダの州の表示方法を制御できるようにする規則のデフォルトライブラリです。

注	Identity Manager IDE では、このライブラリは `RegionalConstants Rules` ライブラリオブジェクトとして表示されます。

入力値 : 表 1-11 を参照してください。

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

AuthType: `EndUserRule`

SubType: 指定しない

呼び出し :

返される値 : 文字列のリスト。

定義済み規則 : Regional Constants

次の表は、RegionalConstants 規則の例をまとめたものです。

表 1-11 RegionalConstants 規則の例

規則名	入力変数	説明
US States	なし	米国の州名のリストを返します。
US State Abbreviations	なし	米国の州の標準的な省略名のリストを返します。
Days of the Week	なし	曜日のフルネームのリストを返します。
Work Days	なし	週の 5 労働日のリスト (米国) を返します。
Months of the Year	なし	年間の月のフルネームのリストを返します。
Month Abbreviations	なし	選択した月の標準的な省略名のリストを返します。
Numeric Months of the Year	なし	12 か月のリストを返します。
Days of the Month	なし	31 日 (一か月) のリストを返します。

表 1-11　RegionalConstants 規則の例 (続き)

規則名	入力変数	説明
Smart Days of the Month	• month: 日付の計算対象となる月 • year: 日付の計算対象となる月の年	数字で月と 4 桁の年に基づいたリストを返します。
Countries	なし	世界の国々の名前を英語でリストします。
Canadian Provinces	なし	カナダの州の名前を英語でリストします。

Remediation Transaction Message

Remediation Transaction Message 規則は、是正または受け入れのトランザクションテキストをフォーマットするために使用されるデフォルトの規則です。この規則をカスタマイズして、ユーザーの署名用にさらに多くの情報を設定できます。

入力値 : 次の引数を受け入れます。

- workItemList: 承認される workitems のセット
- variablesList: workitemList 内の各 workitem に対応する変数のセット
- approverName: workitems の承認を求められるユーザー
- action: 値 remediate または mitigate を想定
- Comments: 是正の一部として入力されるコメント
- expiration: 是正の終了日付の ISO 日付文字列。アクションが mitigate される場合のみ必要。

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

AuthType: EndUserAuditorRule

SubType: 指定しない

呼び出し :

返される値 : フォーマットされた是正または受け入れのトランザクションテキスト。

定義済み規則 : なし

Remediation Transaction Message Helper

Remediation Transaction Message Helper 規則は、単一の workitem の是正または受け入れ用にフォーマットされたトランザクションテキストを返します。

入力値 : 次の引数を受け入れます。

- `workItem`: 承認される `workitem`
- `variables`: `workitem` 変数

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

AuthType: `EndUserAuditorRule`

SubType: 指定しない

返される値 : フォーマットされた是正または受け入れのトランザクションテキスト。

定義済み規則 : なし

ResourceFormRules

ResourceFormRules ライブラリは、多くのユーザーフォームで使用される値や選択内容をカスタマイズできるようにする規則のデフォルトライブラリです。これらの規則は、リソースのユーザー属性を選択する場合にもよく使用されます。

入力値 : 表 1-12 を参照してください。

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

AuthType: `EndUserLibrary`

SubType: 指定しない

呼び出し : UserForms によって明確に呼び出される

- `sample\forms\AccessEnforcerUserForm.xml`
- `sample\forms\ADUserForm.xml`
- `sample\forms\AIXUserForm.xml`
- `sample\forms\HP-UXUserForm.xml`
- `sample\forms\NDSUserForm.xml`
- `sample\forms\RedHatLinuxUserForm.xml`

- sample\forms\SolarisUserForm.xml
- sample\forms\SUSELinuxUserForm.xml

返される値 : 文字列のリスト

定義済み規則 : ResourceFormRuleLibrary

次の表は、ResourceFormRules の例をまとめたものです。

表 1-12 ResourceFormRules の例

規則名	入力変数	説明
ListObjects	• resourceType • resourceName • resourceInstance	複数のフォームによって使用できる groups などのリソースオブジェクトのリストを返します。
ListGroups	• resourceName • resourceInstance	複数のフォームによって使用できるグループのリストを返します。**注** : この規則は下位互換性を保持するために提供されています。
getDefaultShell	resourceType	複数のフォームによって使用できる特定の resourceType のデフォルトシェルを返します。各 resourceType が、リソースアダプタで指定されているのと同じデフォルトシェルを持つようにします。
Exchange Servers	なし	Exchange サーバーのリストを返します。 このリストを更新して、環境内に Exchange サーバーを追加することができます。
Home Directory Servers	なし	ユーザーのホームディレクトリを提供するシステムのリストを返します。 このリストを更新して、環境内にホームディレクトリドライブを提供するシステムを追加することができます。
AD Login Scripts	なし	ユーザーログインスクリプトのリストを返します。 このリストを更新して、環境内にログインバッチスクリプトを追加することができます。
Home Directory Drive Letters	なし	ドライブ文字をマップされたホームディレクトリのリストを返します。 このリストを更新して、環境内に共通のホームディレクトリマップドライブ文字を追加することができます。

表 1-12　ResourceFormRules の例 (続き)

規則名	入力変数	説明
Home Directory Volumes	なし	ホームディレクトリのボリューム名のリストを返します。 このリストを更新して、環境内に共通のホームディレクトリボリューム名を追加することができます。Identity Manager はこの値とホームディレクトリサーバーの値を使用して、ユーザーのホームディレクトリを作成します。ボリュームは、選択したホームディレクトリサーバー上に存在し、共有されている必要があります。
HDS Home Directory Servers	なし	ユーザーのホームディレクトリを提供するシステムのリストを返します。 このリストを更新して、環境内にホームディレクトリドライブを提供するシステムを追加することができます。
NDS Home Directory Types	なし	ドライブ文字をマップされたホームディレクトリのリストを返します。 このリストを更新して、環境内に共通のホームディレクトリマップドライブ文字を追加することができます。
NDS Home Directory Volumes	なし	ホームディレクトリのボリューム名のリストを返します。 このリストを更新して、環境内に共通のホームディレクトリボリューム名を追加することができます。Identity Manager はこの値とホームディレクトリサーバーの値を使用して、ユーザーのホームディレクトリを作成します。ボリュームは、選択したホームディレクトリサーバー上に存在し、共有されている必要があります。
NDS Template	• `resourceName` • `ndsTemplate` • `attrList`	NDS テンプレートオブジェクトを返します。
Is Mail User	objectClasses	`objectClasses` リストに次のクラスがすべて含まれている場合には **1** を返し、それ以外の場合は **0** を返します。 • `inetuser` • `ipuser` • `inetmailuser` • `inetlocalmailrecipient` • `userpresenceprofile`
getResourceAttribute	• `resName` • `attrNam`	要求されたリソース属性の値を返します。

Resource Names

Resource Names 規則は、現在のコンテキスト内のリソースのリストを返します。

入力値 : なし

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

AuthType: 指定しない

SubType: 指定しない

呼び出し :

返される値 : リソースのリスト

定義済み規則 : なし

Role Approvers

Role Approvers 規則は、指定したロールの承認者であるユーザーのリストを提供します。

入力値 : `roleName` 引数を受け入れます。

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

AuthType: `RoleUserRule`

SubType: 指定しない

呼び出し :

返される値 : 指定されたロールに対して静的に定義された承認者のリスト。

定義済み規則 : なし

Role Notifications

Role Notifications 規則は、ロールがユーザーに割り当てられたときに通知を受けるように指定されたユーザーのリストを提供します。

入力値 : `roleName` 引数を受け入れます。

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

AuthType: `RoleUserRule`

SubType: 指定しない

呼び出し :

返される値 : 指定されたロールについて通知するように静的に定義された管理者のリスト。

定義済み規則 : なし

Role Owners

Role Owners 規則は、指定したロールの所有者であるユーザーのリストを提供します。

入力値 : `roleName` 引数を受け入れます。

カスタム Role Owners 規則には、次を指定する必要があります。

AuthType: `RoleUserRule`

SubType: 指定しない

呼び出し :

返される値 : 指定されたロールに対して静的に定義された所有者のリスト。

定義済み規則 : なし

Sample On Local Network

Sample On Local Network 規則は、ログインモジュールグループがユーザーログインに適用されるかどうかを決定するために、ログイン時に評価される LoginConstraintRule の例です。

入力値 : なし

カスタムの Sample On Local Network 規則には、次を指定する必要があります。

AuthType: `LoginConstraintRule`

SubType: 指定しない

呼び出し : ログイン処理の実行中に、ログインモジュールグループによって呼び出される

返される値 :

- ユーザー IP アドレスが特定のサブネットと一致してログインモジュールグループが適用される場合に、`1` (true) を返します。
- ユーザー IP アドレスが特定のサブネットと一致しない場合には、`0` (false) を返します。

定義済み規則 : なし

SAP Portal User Form Default Values

SAP Portal User Form Default Values ライブラリは、SAP Portal User Form Default Values を設定する規則のライブラリです。

入力値 : なし

カスタムの SAP Portal User Form Default Values 規則には、次を指定する必要があります。

AuthType: `Library`

SubType: 指定しない

返される値 : 表 1-13 を参照してください。

定義済み規則 : なし

次の表は、SAP Portal User Form Default Values の例をまとめたものです。

表 1-13 SAP Portal User Form Default Values 規則の例

規則名	入力変数	説明
Countries-ISO3166 Map	なし	ISO3166 国コードのマップを返します。
Currency Code Map	なし	通貨コードのマップを返します。
Locale Map	なし	ロケールのマップを返します。
TimeZones	なし	タームゾーン ID のリストを返します。

ShellRules

ShellRules ライブラリは、`getDefaultShell` という名前の 1 つの規則だけで構成されています。複数のフォームが `getDefaultShell` 規則を使用して、特定の Unix `resourceType` のデフォルトシェルを返します。

入力値 : `resourceType` 引数を受け入れます。

有効な `resourceTypes` は Solaris、AIX、HP-UX、Red Hat Linux のみです。

注 各 `resourceType` が、リソースアダプタで指定されているのと同じデフォルトシェルを持つ必要があります。

カスタム ShellRules 規則には、次を指定する必要があります。

AuthType: 指定しない

SubType: 指定しない

返される値 : 特定の `resourceType` のデフォルトシェルを含む文字列。

定義済み規則 : なし

SIEBEL_NAV_RULE

SIEBEL_NAV_RULE は、*AdvancedNavRule* として指定できるサンプルのナビゲーション規則です。詳細は、Siebel CRM マニュアルの「Advanced Navigation」を参照してください。

入力値 : なし

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

AuthType: 指定しない

SubType: 指定しない

定義済み規則 : なし

TestDictionary

TestDictionary 規則を使用して、Identity Manager 辞書に対して JDBC クエリーを実行して接続をテストします。

入力値 : 次の引数を受け入れます。

- `type`
- `driverClass`
- `driverPrefix`
- `url`
- `host`
- `port`
- `database`
- `context`
- `user`
- `password`
- `sql`
- `arg1`

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

AuthType: 指定しない

SubType: 指定しない

呼び出し :

返される値 :

定義済み規則 : なし

TopSecretUserFormRules

TopSecretUserFormRules を使用して、TopSecret リソースアカウントのデフォルト設定を指定します。

入力値 : なし

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

AuthType: `EndUserLibrary`

SubType: 指定しない

呼び出し : TopSecret ユーザーフォームから

返される値 : 表 1-14 を参照してください。

定義済み規則 : なし

次の表は、TopSecretUserFormRules の例をまとめたものです。

表 1-14 TopSecretUserFormRules の例

規則名	説明
TopSecret Default OMVS	デフォルトの OMVS シェルを決定します。
TopSecret Default TSO	デフォルトの TSO プロセスを決定します。
TopSecret Home Prepend Path	ホームディレクトリを作成するために `accountId` の前に付加されるパス。
TopSecret Attribute List	ユーザーに割り当てることのできる属性のリストを返します。

ユーザーメンバー規則

ユーザーメンバー規則を使用すると、ログインしたユーザーに基づいて、単一の組織のユーザーメンバーシップを動的に制御できます。たとえば、ユーザーメンバー規則を `My Employees` という組織に割り当てた場合、この規則はこの組織のユーザーメンバーシップを次のように動的に制御します。

- Bob がログインして My Employees 組織を制御する場合、Bob は My Employees 組織内の自分の従業員のみを表示して管理することができます。
- Mary がログインして My Employees 組織を制御する場合、彼女は自分の従業員のみを表示して管理することができます。Mary は、Bob やその他の人の従業員を表示したり管理したりすることはできません。

入力値 : 認証された管理者ユーザーのユーザービュー、コンテキスト、または認証された管理者ユーザーの Identity Manager セッション

カスタムのユーザーメンバー規則には、次を指定する必要があります。

AuthType: UserMembersRule

SubType: 指定しない

呼び出し :

返される値 :

- リソースのアカウント ID のリスト。

  たとえば、指定したディレクトリ OU 内のすべてのユーザーエントリを返すために、FormUtil.getResourceObjects という呼び出しを実行することでリソースのアカウント ID を返すことができます。

  返されたリソースのアカウント ID は、次のいずれかのフォーマットになります。

  - resourceId:accountId
  - resourceId@accountId

```
<list>
   <s>res1:stevel</s>
   <s>res1:joem</s>
   <s>res1:sallyp</s>
 </list>
```

- 指定した条件に一致するユーザーを Identity Manager リポジトリに問い合わせるために使用する Identity Manager AttributeConditions のリスト。

```
<list>
    <new class='com.waveset.object.AttributeCondition>
      <s>idmManager</s>
      <s>equals</s>
      <ref>waveset.accountId</s>
    </new>
  </list>
```

定義済み規則 : なし

USER_EMAIL_MATCHES_ACCOUNT_EMAIL_CONF

USER_EMAIL_MATCHES_ACCOUNT_EMAIL_CONF 規則は、Identity Manager ユーザーとアカウントを比較する確認規則です。

入力値 : なし

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

AuthType: 指定しない

SubType: `SUBTYPE_ACCOUNT_CONFIRMATION_RULE`

返される値 : `email` 属性値が一致する場合は `True`。

定義済み規則 : なし

USER_EMAIL_MATCHES_ACCOUNT_EMAIL_CORR

USER_EMAIL_MATCHES_ACCOUNT_EMAIL_CORR 規則は、指定したアカウント内の `email` 属性値と一致する `email` 属性値を持つ Identity Manager ユーザーを検索する相関規則です。

入力値 : なし

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

AuthType: 指定しない

SubType: `SUBTYPE_ACCOUNT_CORRELATION_RULE`

返される値 : 属性条件のリスト。

定義済み規則 : なし

USER_FIRST_AND_LAST_NAMES_MATCH_ACCOUNT

USER_FIRST_AND_LAST_NAMES_MATCH_ACCOUNT 規則は、`fullname` 属性を検索することによって、Identity Manager ユーザーとアカウントを比較する確認規則です。

入力値 : なし

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

AuthType: 指定しない

SubType: `SUBTYPE_ACCOUNT_CONFIRMATION_RULE`

返される値 : `first name` の値と `last name` の値が一致する場合は `True`、それ以外は `false` を返します

定義済み規則 : なし

USER_NAME_MATCHES_ACCOUNT_ID

USER_NAME_MATCHES_ACCOUNT_ID 規則は、指定したアカウント内のユーザーと同じ名前の Identity Manager ユーザーを検索する相関規則です。

入力値 : なし

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

AuthType: 指定しない

SubType: `SUBTYPE_ACCOUNT_CORRELATION_RULE`

返される値 : 文字列値を返します

定義済み規則 : なし

USER_OWNS_MATCHING_ACCOUNT_ID

USER_OWNS_MATCHING_ACCOUNT_ID 規則は、指定したアカウントの名前と一致する `accountId` を所有する Identity Manager ユーザーを検索する相関規則です。

入力値 : なし

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

AuthType: 指定しない

SubType: `SUBTYPE_ACCOUNT_CORRELATION_RULE`

返される値 : 属性条件のリストを返します

定義済み規則 :

Users Without a Manager

Users Without a Manager 規則は、管理者である Identity Manager ユーザーを決定します。

入力値 : なし

注	この規則は、呼び出し範囲の `lhcontext` 変数を使用します。

カスタム Users Without a Manager 規則には、次を指定する必要があります。

AuthType: `AccessScanRule`

SubType: `USER_SCOPE_RULE`

呼び出し :

返される値 : 定義されたマネージャーのいないユーザー名のリスト。

定義済み規則 : なし

Use SubjectDN Common Name

Use SubjectDN Common Name 規則によって、主体の DN から主体の共通名を返します。

入力値 : なし

カスタムの Use SubjectDN Common Name 規則には、次を指定する必要があります。

AuthType: `NewUserNameRule`

SubType: 指定しない

呼び出し :

返される値 : 共通名。

定義済み規則 : なし

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 1-15 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 1-15 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。
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。

表 1-15　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
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
- アテスターエスカレーション規則
- 監査ポリシー規則

- 是正ユーザーフォーム規則
- 是正者規則
- レビュー決定規則
- ユーザー範囲規則
- ViolationPriority 規則
- ViolationSeverity 規則
- 監査規則の複数のアカウントタイプのサンプル

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 : 次の引数を受け入れます。

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

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。

AuthType: `AccessScanRule`

SubType: `ATTESTORS_RULE`

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	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 : 次の引数を受け入れます。

- `wfcontext`: WorkflowContext
- `userEntitlement`: ユーザーエンタイトルメントの現在のビュー (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。

AuthType: `AccessScanRule`

SubType: `AttestorEscalationRule`

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、決して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。

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。

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 of duties* 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。

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>
  < リスト >
      <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それぞれ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、その規則に対する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。

注 RULE_EVAL_COUNT 値はポリシースキャン中に評価された規則の数を表しています。Identity Manager は、この値を次のように計算します。

RULE_EVAL_COUNT = スキャンされたユーザー数 x (ポリシー内の規則の数 + 1)

この計算に +1 が含まれているのは、Identity Manager がポリシー規則もカウントするからです。ポリシー規則とは、ポリシーに違反しているかどうかを実際に決定する規則です。ポリシー規則は監査規則の結果を調べ、ブール型ロジックを実行して、ポリシー結果を導き出します。

たとえば、3 つの規則を含むポリシー A と 2 つの規則を含むポリシー B があり、10 人のユーザーをスキャンした場合、次の式から RULE_EVAL_COUNT の値は 70 になります。

10 ユーザー x (3 + 1 + 2 + 1 規則)

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 引数 (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。

AuthType: 指定しない

Subtype: USER_FORM_RULE

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 : 次の引数を受け入れます。

- 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。

AuthType: AccessScanRule

SubType: REMEDIATORS_RULE

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	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、あるいはそのレコードを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。ユーザーエンタイトルメントは完全なユーザービュー (一部のリソースは省略されることがある) と、いくつかの追跡しているデータです。

レビュー決定規則を使用すると、次のようにして、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ます。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。
- 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ように、この規則を設定します。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決定する場合、次の例に示すように、規則がアテスターに対してヒントを提示します。

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

- 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ように規則を設定します。これによって、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ができます。

  引数を追加して、この規則がユーザービューのサブセットを比較できるようにすることができます。たとえば、次のようにします。

```
<set name='viewCompare'>
<!-- ビュー全体を比較 (3 番目の引数でサブパスを指定できます ) -->
<invoke name='compareUserViews' class='com.sun.idm.auditor.ui.FormUtil'>
<ref>userView</ref>
<ref>lastUserView</ref>
<s>accounts</s>
</invoke>
</set>
```

  この引数はユーザービューを比較し、呼び出し側が GenericObject パス式を使用して完全なユーザービューのサブパスを指定できるようにします。特定のアカウントデータのみを比較したい場合には、サブパスでそのデータを指定できます。ユーザービューの accounts サブパスのみを比較した場合、実際のリソースに反映されない相違点は見つけにくくなります。

  ユーザービューの比較で見つかった相違点は、出力マップの reason 要素に返されます。定義済みの Reject Changed Users 規則と同じように、規則によって 0 (アテステーションを却下) または 2 (アテステーションを承認) が返されると、監査ログがこの difference データを捕捉します。

  Reject Changed Users 規則を使用すると、Identity Manager によって異なっていると判断された内容を正確に検証することができ、結果の監査ログレコード内で監査可能な属性を確認できます。

入力値 : 次の引数を受け入れます。

- `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。

AuthType: `AccessScanRule`

SubType: `REVIEW_REQUIRED_RULE`

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 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。この規則は、ユーザービューの accounts セクションのみを比較します。

  不明なユーザービューは、それぞれ手動アテステーション用に転送されます。

- 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。この規則は、ユーザービューの accounts セクションのみを比較します。

  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、手動でアテストする必要があります。

- 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` 引数を受け入れます。

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。

AuthType: `AccessScanRule`

SubType: `USER_SCOPE_RULE`

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。

AuthType: `EndUserAuditorRule`

SubType: 指定しない

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。

AuthType: `EndUserAuditorRule`

SubType: 指定しない

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 1-31 を参照)。

コード例 1-31　　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`

監査ポリシー規則

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。

Service Provider 規則

ここでは、次の Service Provider 規則の例について説明します。

- Service Provider Confirmation 規則
- Service Provider Correlation 規則
- Service Provider Account Locking 規則

Service Provider Confirmation 規則

ここに例示する Service Provider Confirmation 規則は、*candidates* パスの下の accountId 候補のリストと、*view* パスの下のサービスプロバイダユーザービューにアクセスできます。

入力値 : なし

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

AuthType: `SPERule`

SubType: `SUBTYPE_SPE_LINK_CONFIRMATION_RULE`

返される値 : NULL または確認された accountId を表す文字列。

定義済み規則 : なし

次の表は、サービスプロバイダのカスタマイズに使用できる確認規則の例をまとめたものです。

表 1-16 Service Provider Confirmation 規則の例

規則名	説明
Service Provider Example Confirmation Rule Rejecting All Candidates	リンク相関規則によるすべての候補を却下します。NULL を返します。
Service Provider Example Confirmation Rule Returning First Candidate	候補リストの最初の accountId を返します。
Service Provider Example Confirmation Rule Selecting Candidates Using AccountId	ビュー内の accountId と一致する候補を返します。候補リスト内のビューに accountId が見つからなかった場合、規則は `null` を返します。

Service Provider Correlation 規則

ここに例示する Service Provider Correlation 規則は、サービスプロバイダのユーザービューにアクセスできます。

入力値 : なし

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

AuthType: `SPERule`

SubType: `SUBTYPE_SPE_LINK_CORRELATION_RULE`

返される値 : 単一の accountId、accountId のリスト、またはオプションマップ。

- 規則が accountId のリストを返す場合、選択する accountId を決定するために、確認規則を設定する必要があります。
- 規則がオプションマップを返す場合、ビューハンドラは最初に、設定されたオプションマップを使用して `listResourceObjects` コンテキスト呼び出しを実行することで、リソースアダプタからアイデンティティーのリストを取得します。

定義済み規則 : なし

次の表は、サービスプロバイダのカスタマイズに使用できる相関規則の例をまとめたものです。

表 1-17 Service Provider Correlation 規則の例

規則名	説明
Service Provider Example Correlation Rule for LDAP Returning Option Map	オプションマップと LDAP アダプタで使用する検索フィルタを返します。LDAP リソースアダプタによって検索操作の範囲にフィルタを渡すことができます。このフィルタは LDAP 検索フィルタになると想定されています。
Service Provider Example Correlation Rule for Simulated Returning Option Map	オプションマップとシミュレートされたリソースアダプタで使用する検索フィルタを返します。シミュレートされたリソースアダプタによって検索操作の範囲にフィルタを渡すことができます。このアダプタでは、検索フィルタが `AttributeExpression` になると想定されています。
Service Provider Example Correlation Rule Returning List of Identities	ビュー内の accountId で構成される accountId のリストを LDAP DN フォーマットで返します。
Service Provider Example Correlation Rule Returning Single Identity	ビュー内の accountId で構成される単一の accountId を LDAP DN フォーマットで返します。

Service Provider Account Locking 規則

ここに例示する Service Provider Account Locking 規則は、サービスプロバイダのユーザービューにアクセスでき、Sun Directory Server でアカウントをロックまたはロック解除します。

入力値 : 表 1-18 を参照してください。

カスタムの Service Provider Account Locking 規則には、次を指定する必要があります。

AuthType: `SPERule`

SubType: 指定しない

返される値 : なし

定義済み規則 : なし

次の表は、サービスプロバイダのカスタマイズに使用できるアカウントロック規則の例をまとめたものです。

表 1-18 Service Provider Account Locking 規則の例

規則名	入力変数	説明
Service Provider Example Lock Account Rule	`lockExpirationDate`: ロックの期限が切れる `java.util.Date` (NULL も可)。	Sun Directory Server のアカウントをロックします。この規則は、サービスプロバイダのユーザービューのトップレベルの属性を変更します。
Service Provider Example Unlock Account Rule	なし	Sun Directory Server のアカウントをロック解除します。この規則は、サービスプロバイダのユーザービューのトップレベルの属性を変更します。

第 2 章

カスタムアダプタの開発

Identity Manager のオープンアーキテクチャーを利用してカスタムリソースアダプタを作成することにより、Identity Manager の提供するリソースアダプタでは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なります。これらのカスタムアダプタは、Identity Manager からの要求をリソース上で実行される操作に変換するために必要な基本的な特性およびメソッドを定義します。

この章では、Identity Manager カスタムリソースアダプタの作成、テスト、および読み込みを行う方法について説明します。この情報は、次のように構成されています。

注 Identity Manager には、カスタムアダプタの作成基盤として使用可能なサンプルアダプタ (スケルトンアダプタとも呼ばれる) が含まれています。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。

開始する前に

カスタムアダプタの開発を開始する前に、次の節の情報を確認してください。

対象読者

この章では、リソースアダプタの設計および操作に関する基礎的な情報を提供します。

- カスタムリソースアダプタを作成する必要のある開発者
- Identity Manager システムの動作を学んでいる、またはリソースアダプタで発生した問題を解決する必要のある 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「リソース」の章を読んでいることを前提にしています。

重要な注意点

Identity Manager のカスタムリソースアダプタを記述する前に、次の情報を必ずお読みください。

- `com.waveset.adapter` パッケージ内にカスタムアダプタを作成しないでください。代わりに、カスタムアダプタを顧客固有のパッケージ内に作成して、サポートされる公開 API に含まれるパッケージレベルのクラスおよびメソッドをアダプタが確実に使用するようにしてください。たとえば、com.*customer_name*.adapter を使用します。

  また、すべてのパッケージ名を小文字にしてください。

- `import .*` は使用しないでください。この機構は Java でサポートされていますが、`import .*` の使用は、その機構のゆえに望ましくない手法と見なされています。

  - 参照先クラスの実際の場所を読み取り側から隠します。
  - 内部リファクタリング後の特定の状況で、不正またはあいまいな参照 (コンパイラエラーなど) が発生することがあります。

  代わりに、参照されるクラスまたはインタフェースごとに明示的な `import` 文を挿入してください。

関連ドキュメント

この章で提供する情報に加え、リソースアダプタに関する次のマニュアルも参照してください。

表 2-1　関連ドキュメント

マニュアルのタイトル	説明
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™ Identity Manager に読み込んで同期する方法について説明します。
Identity Manager 管理ガイド	Identity Manager の提供するリソースのカスタマイズおよび管理に関する追加情報が含まれています。

これらのマニュアルは、http://docs.sun.com からダウンロードできます。

リソースアダプタについて

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。アダプタは、リソースタイプの基本的な特性を定義します。この情報は、Identity Manager リポジトリ内にリソースオブジェクトとして保存されます。Identity Manager リソースアダプタは、標準または Active Sync 対応のアダプタです。

この節には、次のトピックが含まれています。

- リソースアダプタについて
- Active Sync 対応リソースアダプタについて
- リソースオブジェクトについて
- リソースアダプタクラスについて

リソースアダプタについて

標準リソースアダプタは、Web サーバー、Web アプリケーション、データベース、および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などの Identity Manager のサポートする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。Java の用語では、標準リソースアダプタは `ResourceAdapterBase` クラスを拡張します。

これらのアダプタは、アカウント情報の変更を、Identity Manager から管理されている外部リソースにプッシュし、通常は次の管理アクティビティーを実行します。

- リソースへの接続、およびリソースからの切断
- 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、または取得
- ユーザーの認証
- 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

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、通常、標準リソースアダプタは次の手順に従います。

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するタスクに、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、ブラウザ内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. この要求が作成要求である場合、アダプタは、リソースアカウント情報を使用して Identity Manager ユーザーオブジェクトを更新します。

   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 対応リソースアダプタについて

Active Sync 対応アダプタは、標準リソースアダプタの拡張機能であり、Active Directory などいくつかの一般的なリソースへの Active Sync インタフェースの実装に使用されます。これらのアダプタは、リソースからデータ変更を直接読み込んで、Identity Manager 内で次のアクティビティーを開始します。

- 変更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
- リソース変更の保存
- 進捗に関する情報やエラーのログ記録

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

  Microsoft Active Directory や PeopleSoft などの一部のアプリケーションには、外部インタフェースが用意されています。これらのアプリケーションインタフェースを設定して、特定の変更が発生したときにイベントを監査ログに追加したり、ほかのアプリケーションに通知したりできます。

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、常にトランザクションが監査ログに記録されるように設定できます。このログを 30 分ごとに確認し、変更が行われると Identity Manager でイベントがトリガーされるように、Identity Manager の Active Directory リソースを設定できます。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し、変更が発生するとイベントメッセージを使ってこのアダプタに通知できます。これらのイベントメッセージには、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ったユーザーへの参照が含まれます。

- 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生成して管理できます。このテーブルは、さまざまな方法で生成できます。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、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ます。

- 変更タイムスタンプを含むデータベース

  特定の時刻よりあとに変更されたデータベースエントリを問い合わせる Active Sync 対応クエリーを作成できます。アダプタは、更新を実行してから新しいクエリーをポーリングします。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。前回の一連の変更は実行されているため、リソースに対するこれらの変更だけが処理のために返されます。

- 変更ログエントリが存在するリソース

  大半の LDAP サーバーの提供する変更ログ機構を使用して、変更を追跡できます。必要に応じ、追跡の対象を DIT 内の該当するセクションに限定することもできます。周期的に変更ログエントリを照会することにより、LDAP リソースアダプタは Identity Manager を更新して、検出された作成、削除、更新などの変更を反映させることができます。

通常、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、IAPI オブジェクトを処理し、WavesetResult オブジェクトをアダプタに返します。WavesetResult オブジェクトは、処理が成功したかどうかを Active Sync 対応アダプタに通知します。

   WavesetResult 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。

7. 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。

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、受信した属性を Identity Manager ユーザーにマップします。または、一致するユーザー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、次の規則およびパラメータによって決定されます。

表 2-2 Active Sync 対応アダプタの規則およびパラメータ

パラメータ	説明
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 2-2　Active Sync 対応アダプタの規則およびパラメータ (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。
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。
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。
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。

表 2-2　Active Sync 対応アダプタの規則およびパラメータ (続き)

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。
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。

注

一致が存在する場合、処理規則により、アダプタが IAPIProcess を使用するか、IAPIUser の使用を試みるかが決定されます。(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アダプタが 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、アダプタにより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、IAPIUser により User ビューがチェックアウトされます。
- 削除の場合は、IAPIUser により Deprovision ビューがチェックアウトされます。

ただし、User ビューはチェックアウトされず、IAPIProcess で使用することはできません。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。

リソースオブジェクトについて

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、次の表に示す情報が含まれます。

表 2-3 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

Identity Manager が通信または管理するすべてのリソースについて、Identity Manager でリソースオブジェクトを定義する必要があります。

注 Identity Manager の「デバッグ」ページでリソースオブジェクトを表示できます。

`http://`*host*`:`*port*`/idm/debug/`

各表記の意味は次のとおりです。

- *host* は、Identity Manager が稼働しているローカルサーバーです。
- *port* は、サーバーが待機している TCP ポート番号です。

`session.jsp` ページのオプションを使用して、タイプ `Resource` のオブジェクトをリスト表示できます。詳細については、184 ページの「リソースオブジェクトの表示と編集」を参照してください。

リソースアダプタクラスについて

リソースアダプタクラスは、次の機能を備えるクラスを実装します。

- Identity Manager リポジトリ内のリソースオブジェクトを登録する
- 外部リソースを管理可能にする
- Identity Manager からリソースに情報をプッシュする
- (オプション) リソースから Identity Manager に情報をプルする

  オプションのプル機能は Active Sync と呼ばれます。また、Active Sync 機能を備えたリソースアダプタは Active Sync 対応と呼ばれます。詳細については、101 ページの「Active Sync 対応リソースアダプタについて」を参照してください。

アダプタ開発の準備

カスタムアダプタの記述を開始する前に、いくらかの準備が必要です。この節では、アダプタ開発の準備を行う方法を説明します。準備には、次の作業が含まれます。

- アダプタのソースコードに精通する
- リソースのプロファイル作成
- 含めるクラスおよびメソッドの決定
- REF キットの確認
- ビルド環境の設定

アダプタのソースコードに精通する

カスタムアダプタを作成する前に、リソースアダプタのソースコード内のコンポーネントに精通する必要があります。この節では、大半のアダプタに共通して存在する次のコンポーネントについて説明します。

- 標準の Java ヘッダー情報
- PrototypeXML 文字列
- リソースメソッド

標準の Java ヘッダー情報

標準の Java ヘッダー情報は、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、コンストラクタ、およびインポートされたファイルを識別します。

このヘッダー情報は、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 文字列

アダプタ Java ファイル内の prototypeXML 文字列は、リソースの XML 定義です。この文字列には、Identity Manager ユーザーインタフェースに表示する、リソース名とすべてのリソース属性を含める必要があります。prototypeXML 文字列は、Identity Manager リポジトリに格納されるリソースオブジェクトも定義します。

次の表に、Identity Manager でリソースの定義に使用するさまざまな prototypeXML 情報の種類を示します。

注 これらの情報の種類の中には、Active Sync 対応アダプタ固有のものもあります。

表 2-4 prototypeXML 情報の種類

種類	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。 次のキーワードがあります。 • **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。
リソース属性	<ResourceAttribute> 要素を使用して定義され、Identity Manager がリソースの定義に使用する XML 要素。 詳細は、110 ページの「リソース属性」を参照してください。
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。カスタム属性をマップする場合とは異なる方法で、標準の Identity Manager アカウント属性タイプをマップします。 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、137 ページの「属性のマッピング」を参照してください。

表 2-4 prototypeXML 情報の種類 (続き)

種類	説明
アイデンティティーテンプレート	ユーザーのアカウント名の作成方法を定義します。 このテンプレートの定義には、<Template> タグを使用します。通常、アカウント名は次の形式のどちらかです。 • accountId は、通常、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。 • ユーザーの完全な識別名 (DN) は、cn=accountId,ou=sub-org,ou=org,o=company の形式になります。この形式は、ディレクトリなどの階層構造の名前空間に使用します。 詳細は、118 ページの「アイデンティティーテンプレート」を参照してください。
ログイン設定	(標準リソースアダプタのみ) リソースのパススルー認証をサポートするための値を定義します。 この値の定義には、<LoginConfigEntry> 要素を使用します。 パススルー認証の詳細については、『Sun™ 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、たいていの場合、フォームにより将来の柔軟な変更が可能になります。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できます。

リソース属性

リソースを定義する管理者のみが使用できます。

リソース属性は、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。

カスタムアダプタの記述時に、リソース属性を使用して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。

Active Directory 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。Active Directory 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。リソース属性は、次の例に示すように、アダプタ Java ファイルの prototypeXML 文字列内で、<ResourceAttribute> 要素を使用して定義します。

```
<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。

次の表は、<ResourceAttribute> 要素で使用できるキーワードを示しています。

表 2-5 <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。 **注 :** name キーワードはビュー内の予約語であるため、スキーママップで Identity System ユーザー属性として使用しないでください。
type	使用されるデータ型を特定します。
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 の場合は、複数行ボックスが表示されます。

表 2-5 <ResourceAttribute> 要素のキーワード (続き)

キーワード	説明
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.'
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 (デフォルト値)。 • activesync: Active Sync 対応アダプタのために Active Sync 処理で使用されます。

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。

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、次のいずれかの方法でリソース属性を上書きできます。

- 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。

次の例では、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 2-1 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。次の表は、スケルトンアダプタファイルで提供される必須リソース属性を示しています。

表 2-6 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。この属性は、「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。この属性は、「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。この属性は、「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。

次の表は、ActiveSync クラスの ACTIVE_SYNC_STD_RES_ATTRS_XML 文字列で定義されている必須の Active Sync 固有属性を示しています。

表 2-7 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。この属性は、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。この属性は、「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。 • 0 (ゼロ) を指定した場合、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、この期間が経過すると、アクティブログは切り捨てられて再利用されます。
RA_MAX_AGE_UNIT	seconds、minutes、hours、days、weeks、または months を指定します。この値は、RA_MAX_AGE_LENGTH とともに使用します。
RA_LOG_LEVEL	ログレベル (0: 無効、4: 非常に詳細)。この属性は、Identity Manager 画面内の「ログレベル」フィールドに対応しています。
RA_LOG_PATH	ログファイルの絶対または相対パス。この属性は、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。この属性は、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。たとえば、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 時に実行されます。Active Sync 対応アダプタには必要ありません。

表 2-7 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_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。Active Sync 対応アダプタには必要ありません。

次の表は、ActiveSync クラスの ACTIVE_SYNC_EVENT_RES_ATTRS_XML 文字列で定義されている必須の Active Sync 固有属性を示しています。

表 2-8 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。この属性は、ほかのすべての属性より優先されます。
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、アカウントは作成されません。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。
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。

Identity Manager アカウント属性

リソースを定義する管理者のみが使用できます。

Identity Manager アカウント属性には、リソースでサポートされるデフォルトのユーザー属性が含まれます。

Active Sync 対応アダプタの場合は、アカウント属性を使用して 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。

Identity Manager がサポートするアカウント属性の種類を次に示します。

- `string`
- `integer`
- `boolean`
- `encrypted`
- `binary`

バイナリ属性には、グラフィックファイル、オーディオファイル、または証明書が含まれます。すべてのアダプタが、バイナリアカウント属性をサポートするわけではありません。一般に、特定のディレクトリ、フラットファイル、およびデータベースアダプタがバイナリ属性を処理できます。

注

- 使用するアダプタがバイナリ属性をサポートしているかどうかを確認するには、アダプタのマニュアルで「アカウント属性」の節を参照してください。
- バイナリ属性内で参照されるすべてのファイルは、サイズをできるだけ小さくしてください。たとえば、きわめて大きなグラフィックファイルを読み込むと、Identity Manager のパフォーマンスが影響を受けます。

リソースのスキーママップの `AttributeDefinition` オブジェクト内で Identity Manager アカウント属性を定義し、アダプタファイル内の `prototypeXML` 文字列を使って受信するリソース属性を Identity Manager 内のアカウント属性にマップします。たとえば、LDAP `sn` リソース属性を Identity Manager 内の `lastname` 属性にマップします。Identity Manager アカウント属性には、次の属性が含まれます。

- `accountId`
- `email`
- `firstname`
- `fullname`
- `lastname`
- `password`

標準アダプタのスキーママップ。「アカウント属性」ページまたはスキーママップを使用して、Identity Manager アカウント属性をリソースアカウント属性にマップします。属性のリストは、リソースによって異なります。一般に、使用しないすべての属性をスキーママップページから削除します。属性を追加すると、ユーザーフォームやほかのコードの編集が必要になります。

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。ユーザー用に選択したロールに基づいて、選択したロール内のすべてのリソースの属性の和集合である、一連のアカウント属性の入力が求められます。

注 ユーザーまたはロールの Identity Manager スキーマを表示または編集するには、`IDM Schema Configuration AdminGroup` のメンバーである必要があります。また、`IDM Schema Configuration` 機能を保持している必要もあります。

Active Sync 対応アダプタのスキーママップ。 Active Sync リソーススキーママップは、Active Sync 対応アダプタへの入力を編集可能にする、オプションのユーティリティーです。Active Sync 対応アダプタへの入力は、たいてい、データベースの列名またはディレクトリ属性名になります。スキーママップと Active Sync 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。

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。スキーママップは、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。

リソースアカウントをはじめて作成する場合は、`accountId` 属性が直接リソースの `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。

スキーママップの使用。 リソースインタンスを作成したあとで、管理者はスキーママップを使って次の処理を実行できます。

- 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。

「リソースの編集 / 作成」ページの下部にある「スキーマの編集」ボタンをクリックして、Identity Manager ユーザーインタフェースの「スキーマの編集」ページに Identity Manager アカウント属性を表示できます。

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。

アイデンティティーテンプレート

注 アイデンティティーテンプレートは、リソースを定義する管理者だけが使用できます。

ユーザーまたはロールの Identity Manager スキーマを表示または編集するには、`IDM Schema Configuration AdminGroup` のメンバーである必要があります。また、`IDM Schema Configuration` 機能を保持している必要もあります。

リソース上でアカウントを作成する際に、アイデンティティーテンプレート (またはアカウントの DN) を使用して、ユーザーのデフォルトアカウント名の構文を定義します。アイデンティティーテンプレートは、Identity Manager ユーザーアカウント情報を外部リソース上のアカウント情報に変換します。

アイデンティティーテンプレート内の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表示される属性) を使用できます。また、ユーザーフォーム内でユーザーアイデンティティーテンプレートを上書きすることもできます。これは、組織名を置換する際によく使用される方法です。

Identity Manager ユーザーは、アカウントごとに ID を保持します。この ID は、これらのアカウントのすべてまたは一部で、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、キーとして使用される Identity Manager 内の主要な `accountId` と、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個の `accountId` を保持します。次の表に示すように、`accountId` は `accountId:<resource name>` の形式で表されます。

表 2-9　accountID の例

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

アカウントユーザー名は、次の 2 つの形式のどちらかになります。

- フラットな名前空間
- 階層構造の名前空間

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。

- 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ることを、単純に指定できます。

階層構造の名前空間。階層構造の名前空間を持つシステムには、識別名 (DN) を使用します。DN 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。

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。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アイデンティティー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ます。次の表に、階層構造の名前空間の例および DN の表現方法を示します。

表 2-10　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

たとえば、LDAP のような階層構造の名前空間を持つリソースアイデンティティーテンプレートに対して、次のように指定できます。

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

ログイン設定

ログイン設定は、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 ですが、異なるパラメータを使用するリソースもあります。たとえば、SecurId では user name と passcode が使用されます。

ログイン設定の情報タイプはリソースの定義に役立ちますが、管理者がこれを変更するのは容易ではありません。

パススルー認証の詳細については、149 ページの「リソースタイプのパススルー認証の有効化」および『Sun™ Identity Manager リソースリファレンス』を参照してください。

リソースメソッド

リソースメソッドは、Identity Manager から外部リソースに情報を書き込みます。

注	カスタムメソッドを記述するには、リソースに精通している必要があります。

リソースメソッドをタスクごとに分類します。独自のカスタムアダプタを開発する場合、開発の目標を達成するためにアダプタが必要とするカテゴリを判断する必要があります。次に例を示します。

- アダプタを標準のアダプタにしますか、それとも Active Sync 対応アダプタにしますか。
- 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、どのリソースメソッドを完成する必要があるかが決定されます。

次の表に、リソースメソッドのカテゴリを示します。(各機能カテゴリの追加情報については、この章のあとの方で説明する。)

表 2-11 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。

表 2-11　リソースメソッドのカテゴリ (続き)

カテゴリ	説明
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。リソースの定義に役立ちますが、管理者がこれを変更するのは容易ではありません。

Active Sync 対応アダプタでは、リソースメソッドは次の操作を実行できます。

- リソースから Identity Manager へのフィードを作成する。変更されたリソースを検索したり、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、リソース上の変更を登録または検索する方法、およびリソースとの通信方法を理解している必要があります。
- リソースから Identity Manager へのフィードを実行して、Identity Manager リポジトリ内で更新操作を実行する。

標準リソースアダプタの考慮事項

標準リソースアダプタ内のアカウント属性に固有の考慮事項には、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

注　ユーザーまたはロールの Identity Manager スキーマを表示または編集するには、`IDM Schema Configuration AdminGroup` のメンバーである必要があります。また、`IDM Schema Configuration` 機能を保持している必要もあります。

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するアカウント名を確立します。このテンプレートは、Identity Manager ユーザーアカウント情報を外部リソース上のアカウント情報に変換します。

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。

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、複数のユーザー属性の一部から作成できます。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。この場合、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 2-12 <AuthnProperty> 要素の属性

属性	説明
dataSource	このプロパティーの値のソースを指定します。このプロパティー値のデータソースには、次のものが含まれます。 • user (デフォルト): ログイン時にユーザーが指定した値。 • http attribute: 特定の HTTP セッション属性で指定された値。 • http header: 特定の HTTP ヘッダーで指定された値。 • http remote user: HTTP 要求の remote user プロパティーで指定された値。 • http request: 特定の HTTP 要求パラメータで指定された値。 • resource attribute (Active Directory のみ): 特定のアダプタに追加認証属性を指定可能にする値。この属性は、定義されているリソースでのみ有効です。ユーザーがこの属性を操作することはできません。 • x509 certificate: 値は X509 クライアント証明書です (HTTPS を使って実行された要求でのみ有効)。
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。
doNotMap	LoginConfigEntry にマップするかどうかを指定します。

表 2-12 <AuthnProperty> 要素の属性 (続き)

属性	説明
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。
name	内部の認証プロパティー名を特定します。

フォレスト間のユーザー管理が可能なのは、複数のゲートウェイが存在し、かつ各フォレストに 1 つのゲートウェイが配備されている場合だけです。この場合、ユーザーがドメインを指定せず、定義済のドメインを使用してアダプタごとに認証を実行するように、アダプタを設定できます。次にその方法を示します。

1. リソースオブジェクトの XML 内で、次の認証プロパティーを <AuthnProperties> 要素に追加します。

   ```
   <AuthnProperty name='w2k_domain' dataSource='resource attribute'
   value='MyDomainName'/>
   ```

2. *MyDomainName* を、ユーザーを認証するドメインで置き換えます。

注 このプロパティーの詳細については、『Identity Manager リソースリファレンス』の Active Directory リソースアダプタに関する説明を参照してください。

ほとんどのリソースログインモジュールは、Identity Manager 管理インタフェースとユーザーインタフェースの両方をサポートしています。次の例では、SkeletonResourceAdapter.java で <LoginConfigEntry> 要素を実装する方法を示します。

コード例 2-2 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"+
```

次の例では、サポートされているログインモジュール DATA_SOURCE オプションを定義しています。この例では、LoginConfig エントリは 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。このエントリは、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 2-3 サポートされているログインモジュール DATA_SOURCE 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"+
```

次の例では、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 Login Config エントリを示しています。この場合、値は HTTP 要求ヘッダーから取得されます。

コード例 2-4 Login Config 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"+
```

オブジェクトリソース属性の宣言例

次の例は、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ます。

コード例 2-5 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='AD'>
   </ResourceAttribute>
</ResourceAttributes>
```

Identity Manager 管理インタフェースには、すでに指定したようにデフォルトリソースのリソース属性が表示されます。

リソースのプロファイル作成

以降の節では、標準リソースアダプタおよび Active Sync 対応アダプタの前提条件のプロファイル作成方法および定義方法について説明します。

- 標準リソースアダプタのプロファイル作成
- Active Sync 対応リソースアダプタのプロファイル作成

標準リソースアダプタのプロファイル作成

次の情報を使用して、標準リソースアダプタの前提条件をプロファイル作成および定義します。

- 接続先のリソースタイプにもっとも近い Identity Manager アダプタファイルを選択してください。

  Identity Manager の標準構成に付属のデフォルトの Identity Manager リソースアダプタファイルの簡単な説明については、130 ページの表 2-13 を参照してください。

- ユーザーアカウントの特性の検索する方法、および次のタスクをリモートリソース上で実行する方法を確認します。

  - 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
  - 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
  - `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

- アクションおよびサポートされる属性すべての実行に必要な最小属性を特定する
- 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ることを確認します。

  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、公開された API セットまたは完全なツールキットが付属しています。API 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確認してください。たとえば、データベースへの接続には JDBC を使用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ユーザーを確認します。

  ほとんどのリソースアダプタでは、ユーザーの検索や属性の取得などのタスクの実行に管理アカウントが必須であり、リソースアダプタによりこの管理アカウントが実行されます。このアカウントは一般には、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。

- リソースの内蔵属性が拡張可能かどうかを判定します。

  たとえば、Active Directory および LDAP の両方で、標準の Identity Manager 属性以外の属性である拡張スキーマ属性を作成できます。

  Identity Manager 内で維持する属性を決定して、リソース上での属性名を判別し、Identity Manager 内でその属性に付ける名前を決定します。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。

Active Sync 対応リソースアダプタのプロファイル作成

Active Sync 対応リソースアダプタのプロファイルを作成する際、127 ページの「標準リソースアダプタのプロファイル作成」で説明した考慮事項に加えて、次の情報を使用してください。

- ユーザーアカウントの特性、およびリモートリソース上でこれらのタスクを実行する方法を確認する際、次の操作も実行する必要があります。
  - ユーザーに対する変更を検索する
  - 変更されたユーザーのみを検索する方法を特定する
- イベントを作成するリソース属性またはアクションを判別します。

  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。
- アダプタがソース上でイベントを検出した場合に、Identity Manager が実行するアクションを次の中から決定します。
  - 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
- アダプタを外部リソース内のイベントで駆動するか、特定のポーリング間隔で駆動するかを決定します。

  注 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理解しておく必要があります。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、大半の Identity Manager 配備環境ではハイブリッドな方法が使用されています。

  次のいずれかのアプローチを選択します。
  - ポーリング間隔を設定します。Active Sync マネージャースレッドは、ポーリングインタフェースを、設定可能な間隔または指定したスケジュールで呼び出します。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を設定します。アダプタは、この環境に基づいて LDAP リスナーなどのリスニング接続を設定し、遠隔システムからのメッセージを待機します。何も実行しないように poll メソッドを実装して、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、MQ Series などの保証された配信メカニズム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。
- 外部イベントがスマートポーリングをトリガーし、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する、ハイブリッドソリューションを実装します。

  スマートポーリングでは、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引き下げます。スマートポーリングは、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ります。

  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、効率が向上します。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て、すべての重複を削除します。次にルーチンは完全なオブジェクトを取得して、最新のデータが同期され、更新が効率的に処理されていることを確認します。

含めるクラスおよびメソッドの決定

リソースのプロファイルを作成したあとで、アダプタに必要なクラスとメソッドを判別します。

- 関連する Javadoc を参照して、現状のまま使用可能で、拡張の必要な基底クラスおよびメソッドを判別します。この javadoc は、Identity Manager CD の `REF/javadoc` ディレクトリ内で参照できます。
- 接続先のリソースに基づき、記述する必要があり、かつ Java ファイルに含める必要のあるメソッドのリストを作成します。

  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。

REF キットの確認

Sun Resource Extension Facility キット (REF キット) は、Identity Manager CD またはインストールイメージの /REF ディレクトリに収録されています。この REF キット内のサンプルファイルおよびその他のツールを使用して、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できます。

次の表に、REF キットの内容を示します。

表 2-13 REF キットのコンポーネント

コンポーネント	場所	説明
audit	REF/audit	カスタム監査パブリッシャーのサンプル。
exporter	REF/exporter	ウェアハウスインタフェースコードのソースコード。このソースコードを使用してウェアハウスインタフェースを再構築することで、データエクスポータからウェアハウス関連のデータベース以外にエクスポートできるようになります。
javadoc	REF/javadoc	生成された javadoc。カスタムアダプタの記述が必要なクラスについて記されています。Javadoc を表示するには、ブラウザで次の場所を参照します。 /waveset/image/REF/javadoc/index.html
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。

表 2-13　REF キットのコンポーネント (続き)

コンポーネント	場所	説明
src	REF/src	一般に開発されているリソースアダプタソースファイルおよびスケルトンファイルの例。アダプタの開発およびテスト用の基礎として使用します。 • データベースアカウント用の MySQLResourceAdapter.java • データベーステーブル用の ExampleTableResourceAdapter.java [1] • ファイルベースのアカウント用の XMLResourceAdapter.java • カスタムの LDAP リソースアダプタ開発時に使用する、単純なメソッド用の LDAPResourceAdapter.java • カスタムの LDAP リソースアダプタ開発時に使用する、複雑な変更用の LDAPResourceAdapterBase.java • UNIX アカウント開発用の AIXResourceAdapter.java • 標準リソース用の SkeletonStandardResourceAdapter.java • 標準および Active Sync 対応リソース用の SkeletonStandardAndActiveSyncResourceAdapter.java • Active Sync 専用リソース用の SkeletonActiveSyncResourceAdapter.java • カスタムアダプタのユニットテスト作成用の test.SkeletonResourceTest.java
test	REF/test	カスタムアダプタの基礎として使用可能な、サンプルのリソースアダプタテストソースファイル。
thirdpartysource	REF/ thirdpartysource	
transactionsigner	REF/ transactionsigner	transactionsigner PKCS11KeyProvider のサンプル実装。
BeforeYouBegin. README	REF	アダプタをカスタマイズする前に収集する必要のある概要情報。
build.xml	REF	プロジェクトの構築、テスト、および配布用の Ant ビルドスクリプトのサンプル。
Design-for-Resource-Adapters.htm	REF	リソースアダプタの基本的なアーキテクチャーおよび設計について説明したドキュメント。

表 2-13　REF キットのコンポーネント (続き)

コンポーネント	場所	説明
README	REF	Sun Identity Manager REF キットについて説明したドキュメント。
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。

1. 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、テーブルベースのリソース用アダプタを作成できます。このウィザードの使用の詳細については、『Identity Manager 管理ガイド』の「設定」の章を参照してください。

ビルド環境の設定

この節では、ビルド環境の設定手順を示します。

- Windows の場合
- UNIX の場合

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。詳細は、『Identity Manager リリースノート』の「サポートされているソフトウェアと環境」を参照してください。

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して REF キットをインストールする必要があります。

Windows の場合

Microsoft Windows オペレーティングシステムを使用している場合は、次の手順を実行してビルド環境を設定します。

1. 新しいディレクトリに移動します。
2. ws.bat という名前のファイルを作成します。
3. このファイルに次の行を追加します。

```
set WSHOME=<REF キットがインストールされているパス >
set JAVA_HOME=<JDK がインストールされているパス >
set PATH=%PATH%;%JAVA_HOME%\bin
```

設定する位置 :

- WSHOME は REF キットがインストールされているパスです。
- JAVA_HOME は JDK がインストールされているパスです。

4. ファイルを保存して閉じます。

UNIX の場合

UNIX オペレーティングシステムを使用している場合は、次の手順を実行してビルド環境を設定します。

1. 新しいディレクトリに移動します。
2. ws.sh という名前のファイルを作成します。
3. このファイルに次の行を追加します。

```
WSHOME=<path_where_REF_is_installed>
JAVA_HOME=<path_where_JDK_is_installed>
PATH=$JAVA_HOME/bin:$PATH

export WSHOME JAVA_HOME PATH
```

設定する位置 :

- WSHOME は REF キットがインストールされているパスです。
- JAVA_HOME は JDK がインストールされているパスです。

4. ファイルを保存して閉じます。

カスタムアダプタの記述

「アダプタ開発の準備」に記載されている準備作業を完了すると、カスタムアダプタの記述を開始できます。

ここでは、カスタムアダプタの記述方法について説明します。次の内容が含まれます。

- 手順の概要
- スケルトンファイルの名前変更
- ソースファイルの編集
- 属性のマッピング
- アイデンティティーテンプレートの指定
- アダプタメソッドの記述
- パススルー認証をサポートするアダプタの設定
- リソースオブジェクトコンポーネントの定義

手順の概要

次の節では、カスタムアダプタを作成するための実行手順についての概要を示します。

- 標準リソースアダプタの記述方法
- Active Sync 対応リソースアダプタの記述方法

標準リソースアダプタの記述方法

この節では、標準アダプタまたは Active Sync 対応アダプタの作成時に実行する手順について説明します。

注	標準アダプタの記述手順は、使用するオペレーティングシステムによって多少異なります。

標準アダプタを作成するには、次の手順に従います。

1. コマンドウィンドウを開いて、次のディレクトリに移動します。

   ```
   \waveset\idm\adapter\src
   ```

2. `SkeletonStandardResourceAdapter.java` スケルトンファイルの名前を任意のファイル名に変更します。詳細については、136 ページの「スケルトンファイルの名前変更」を参照してください。

3. 136 ページの「ソースファイルの編集」に示されている方法で、新規アダプタのソースファイルを編集します。
4. 先に作成したファイルに基づいて、環境を設定します。
   - **Windows の場合**：ソースファイルは `ws.bat` です。
   - **Unix の場合**：ソースファイルは `ws.sh` です。
5. 次のコマンドを入力して、ソースファイルをコンパイルします。
   - **Windows の場合**：`javac -d . -classpath %CLASSPATH%` *yourfile*`.java`
   - **Unix の場合**：`javac -d . -classpath $CLASSPATH` *yourfile*`.java`

Active Sync 対応リソースアダプタの記述方法

この節では、カスタム Active Sync 対応アダプタの作成時に実行する一般的な手順を説明します。

Microsoft Windows オペレーティングシステムを使用している場合は、次の手順を実行してカスタム Active Sync 対応アダプタを作成します。

1. コマンドウィンドウを開いて、次のディレクトリに移動します。

   ```
   \waveset\idm\adapter\src
   ```

2. 次のスケルトンファイルのいずれかを、任意の名前に変更するか、任意の名前を付けてコピーします。詳細については、136 ページの「スケルトンファイルの名前変更」を参照してください。
   - `SkeletonStandardAndActiveSyncResourceAdapter.java` (標準および Active Sync 対応リソースの場合)
   - `SkeletonActiveSyncResourceAdapter.java` (Active Sync のみのリソースの場合)
3. 136 ページの「ソースファイルの編集」に示されている方法で、新規アダプタのソースファイルを編集します。
4. 先に作成したファイルに基づいて、環境を設定します。
   - **Windows の場合**：ソースファイルは `ws.bat` です。
   - **Unix の場合**：ソースファイルは `ws.sh` です。
5. 次のコマンドを入力して、ソースファイルをコンパイルします。
   - **Windows の場合**：`javac -d . -classpath %CLASSPATH%` *yourfile*`.java`
   - **Unix の場合**：`javac -d . -classpath $CLASSPATH` *yourfile*`.java`

スケルトンファイルの名前変更

スケルトンアダプタの名前を、新しいアダプタに適した名前に変更する必要があります。次の操作を実行します。

- サンプルの java ファイルの名前を、新しいクラス名に合わせて変更します。
- ソースコードを編集して、サンプルのクラス名を新しいクラス名で置き換えます。

ソースファイルの編集

スケルトンファイルの名前を変更したあとで、新しいアダプタのソースコードを編集して、特定のテキスト文字列を置き換え、アダプタでサポートするデフォルト値を定義する必要があります。

アダプタのソースファイルを次のように編集します。

1. 次のテキスト文字列を検索し、コード内でアダプタ固有の変更を行う必要のある場所を判別して、置換します。
   - `change-value-here` 文字列は、置換文字列を入力する必要のある場所を示します。
   - `@todo` 文字列は、サポートする特定のシナリオ用のメソッドを書き換える必要のある場所を示します。
2. リソースアダプタタイプに名前を付けます。

   この名前は、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。
3. `prototypeXML` 文字列内のデフォルト値をこのアダプタタイプの独自のデフォルト値で置き換えて、受信したリソース属性を Identity Manager アカウント属性にマップします。たとえば、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。

   詳細については、137 ページの「属性のマッピング」を参照してください。
4. 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 `join`、`leave`、および `move` の操作をサポートするための Java コードを追加します。

   詳細については、139 ページの「アダプタメソッドの記述」を参照してください。
5. アダプタファイルを編集したあとで、Identity Manager に読み込むことができます。

属性のマッピング

一般に、アダプタタイプのオプションを設定するには、受信したリソース属性を標準の Identity Manager アカウント属性にマッピングするか、独自のカスタム属性 (拡張スキーマ属性と呼ばれる) を作成します。

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です。リソースが prototypeXML オブジェクトを提供する必要はありません。

注 SkeletonActiveSyncResourceAdapter 内の属性は必須です。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

受信する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、次の例に示す構文を使用します。

コード例 2-6 リソース属性のマッピング

```
"<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"+
```

各表記の意味は次のとおりです。

- <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 文字列を囲みます。
- <AttributeDefinitionRef> 要素は、Identity Manager アカウント属性を特定します。

次の表に、<AttributeDefinitionRef> element フィールドを示します。

表 2-14 <AttributeDefinitionRef> 要素のフィールド

要素のフィールド	説明
name	リソース属性がマップされている 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内のリソーススキーマページの左側の列。)
mapName	受信したリソース属性の名前を特定します。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	受信した属性タイプ (string、int、または encrypted) を特定します。

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、137 ページの「属性のマッピング」を参照してください。

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には、拡張スキーマ属性を作成する必要があります。次の例は、リソース属性を拡張スキーマ属性にマップする方法を示します。

コード例 2-7 リソース属性の拡張スキーマ属性へのマッピング

```
<AccountAttributeType name='HomeDirectory' type='string'
   mapName='HomeDirectory' mapType='string'>\n"+
</AccountAttributeType>\n"+
```

ObjectRef 型を宣言する必要はありません。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します。

アイデンティティーテンプレートの指定

アイデンティティーテンプレート (またはアカウント DN) を使用して、企業内のすべてのユーザーおよびグループを一意に識別する必要があります。

DN 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

アイデンティティーテンプレートの詳細については、118 ページの「アイデンティティーテンプレート」を参照してください。

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、次の内容を簡潔に説明します。

- 標準リソースアダプタ固有のメソッドの記述方法
- Active Sync 対応アダプタメソッドの記述方法

標準リソースアダプタ固有のメソッドの記述方法

標準リソースアダプタ固有のメソッドは、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。

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

注 カスタムアダプタを記述する際、次の点に留意してください。

- カスタムメソッドにより返される任意の WSUser オブジェクト上で setdisabled() メソッドを呼び出します。
- アダプタが AsynchronousResourceAdapter クラスを実装する場合、このアダプタは部分的に初期化されたユーザーで動作する可能性があることに留意してください。(これらのユーザーは Identity Manager 外部に作成されるが、属性は完全には入力されない。) WSUser がリソースにすでに存在している場合、プロビジョニングツールが作成操作を更新操作に自動的に変換することはありません。リソースアダプタは、この事例を識別する必要があります。

プロトタイプリソースの作成

次の表は、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 2-15 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	通常、リソースアダプタで定義されている、定義済みの prototypeXML 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。 既存のアダプタを拡張する場合は、リソースアダプタを追加し、スーパークラスのプロトタイプリソースに基づく異なるデフォルト値をプログラムで指定できます。

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。

次の表に示すメソッドは、アダプタがリソースと通信していること、および承認されたアカウントにアクセス権があることを確認する際に使用できます。

表 2-16 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 2-17 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 2-18 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 2-18　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。
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。

表 2-18　アカウントの機能 (続き)

機能名	基本で有効	コメント
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 2-19　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 2-20　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 2-21 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 2-22 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 2-23 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 2-24　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 2-25　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。
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。

ベストプラクティス

カスタムアダプタ用の AccountIterator インタフェース実装を記述するときは、次の操作を実行してみてください。

- getAccountIterator() が呼び出されたときに、AccountIterator.next() メソッドが、スキーママップ内のすべての属性を含むユーザーを返すようにします。
  調整サーバーは (getAccountIterator() 要求で使用される複製されたリソース内で) スキーマを調整して、調整サーバーの必要とする属性だけを要求します。一般に、調整サーバーが必要とするのは accountId 属性だけですが、調整サーバーがスキーマ内に追加属性を保持する場合があります。「リソースから読み込み」などの、その他の getAccountIterator() ユーザーは、すべてのスキーマ属性を必要とする可能性があります。

  スキーママップの内容に基づいて正しい処理を実行する「スマート」アダプタの作成を試みます。アダプタの実行可能な処理がアカウントのリスト表示、および要求された情報の取得だけである場合、これらのタスクだけをアダプタが実行するようにしてください。そうしない場合、アダプタがアカウントを取得して必要な属性を取得しなければならなくなる可能性があります。スマートでないアダプタは、常にすべての属性を取得します。
- アダプタを可能な限りスケーラブルにします。通常、これは、アダプタがすべてのアカウントを一度にリストまたは取得しないことを意味します。その代わりに、AccountIterator.next() メソッドの呼び出し時に、アダプタがアカウントに対して処理を繰り返し実行するようにします。アダプタが AccountIterator コンストラクタ内で多くの処理を行うことがないようにしてください。

次の例は、リソースから情報を取得して、Identity Manager が処理可能な情報に変換するコードを示します。

コード例 2-8 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 2-26 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、Active Directory ユーザーと Active Directory 用コンピュータのユーザーマネージャー、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 対応アダプタメソッドの記述方法

Active Sync 固有のメソッドは、Active Sync 対応アダプタの主目的である Identity Manager の更新を実行する機構を提供します。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きます。また、これらのメソッドを使ってアダプタを開始、停止、およびスケジュールできます。

アダプタのこのセクションに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。

以降の節では、Active Sync 対応アダプタメソッドを作成するための一般的なガイドラインについて説明します。

- 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を使用します。

- `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、114 ページの表 2-7 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、文字列定数 ActiveSync.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。

次の表は、いくつかのサンプルのポーリングシナリオを使用したスケジューリングパラメータの使用方法を示します。

表 2-27 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 (a Thursday), 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。

次の例は、属性の取得方法および更新を基底クラスに渡す方法を示します。

コード例 2-9 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。Identity Manager は shutdown メソッドを呼び出します。これは、アダプタがポーリングループにより使用中のオブジェクトすべてをクリーンアップする機会になります。

パススルー認証をサポートするアダプタの設定

Identity Manager はパススルー認証を使用して、ユーザーおよび管理者に 1 つまたは異なる複数のパスワードを使用したアクセス権を付与します。Identity Manager は、パススルー認証の管理に次の実装を利用します。

- ログインアプリケーション (ログインモジュールグループのコレクション)
- ログインモジュールグループ (順序付けされたログインモジュールのセット)
- ログインモジュール (割り当てられたリソースごとに認証を設定して、いくつかある認証の成功要件のいずれかを指定する)

次の方法でパススルー認証をサポートするよう、カスタムアダプタを設定します。

- authenticate() メソッドを正しく実装する
- account.LOGIN 機能を getFeatures() メソッドマップに含める (com.waveset.adapter.ResourceAdapter.ACCOUNT_LOGIN)
- <LoginConfigEntry> セクションをリソースの prototypeXML に含める

カスタムリソースアダプタを設定して対話的なログインをサポートする場合は、ログイン時およびユーザーが初期ログインページを送信したあとで、アダプタを有効にしてユーザーに追加情報を要求する必要があります。

アダプタの authenticate() メソッドは、ログインを対話的にするかどうかを制御します。authenticate() メソッドの戻り値は対話的ログインをトリガーするため、次のログインページの結果により authenticate() が再度呼び出されます。この処理は、authenticate() メソッドがログインについて次の判断を下すまで繰り返されます。

- 例外をスローして失敗する
- 通常通り、認証済みアカウントのアカウント ID を WavesetResult に格納して返すことで成功する

対話的にするには、アダプタの返す WavesetResult が次の条件を満たしている必要があります。

- Constants.AUTHENTICATED_GUID 型や Constants.AUTHENTICATED_IDENTITY 型を持つ ResultItems を含まない
- 次ページのログイン用フォームの動的作成に使用する ResultItems を含む

各 ResultItem がフォーム内のフィールドに対応します。ResultItems に含まれる Constants.CONTINUE_AUTHENTICATION_ATTR 型の値は、次の書式にする必要があります。

label|*attrName*|*displayType*|*prompt*|'*isId*'

各表記の意味は次のとおりです。

- *label* は、ラベルまたは none を含む文字列です。
- *attrName* は、次の authenticate() メソッド呼び出しに loginInfo HashMap 内のキーとして渡されるログイン属性名です。
- *displayType* は、使用するフォームフィールドの種類を示します。*displayType* には、次の値が含まれます。
  - text
  - secret
  - label
  - checkbox

- *prompt* は、フォームフィールドのタイトルまたはラベルに対応します。
- *isId* はオプションの文字列です。

  `isId` 文字列を使用する場合、フォームフィールドの値が、キー `Constants.ACCOUNT_ID` およびフィールドの値とともに `loginInfo HashMap` に追加されます。

続く `ResultItem` 型も「往復」し、次の `authenticate()` 呼び出しで `loginInfo HashMap` に格納されて返されます。

- `Constants.CONTINUE_AUTHENTICATION_ACCOUNT_HANDLE` は、認証処理中のユーザーまたはアカウントを追跡します。
- `Constants.CONTINUE_AUTHENTICATION_PREVIOUS_ATTR` は、以前の認証属性を `loginInfo` から削除します。このため、`loginInfo` に「以前の」認証 `attr` が含まれることはありません。

リソースオブジェクトコンポーネントの定義

この節では、次のリソースオブジェクトコンポーネントの定義方法について説明します。

- リソースオブジェクトクラスの定義
- リソースのオブジェクトタイプの定義
- リソースオブジェクト機能の定義
- リソースオブジェクト属性の定義

リソースオブジェクトクラスの定義

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。

次の例では、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 2-10 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"+
```

次の例では、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 2-11 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"+
```

この例では、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、get、list、および update 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 2-12 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。<ObjectClasses> セクションを取り除いた場合、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。

リソースのオブジェクトタイプの定義

リソースのオブジェクトタイプは、特定のリソースタイプを一意に定義します。オブジェクトタイプは、アダプタの prototypeXML 文字列内で定義します。

XML の <ObjectTypes> 要素は、アダプタの prototypeXML 文字列内部のコンテナです。このコンテナには、そのリソース内で管理する 1 つ以上のオブジェクトタイプ定義が含まれます。この <ObjectTypes> 要素には、リソース固有のオブジェクトに関する完全な記述が含まれています。これらの記述は Identity Manager に提供されます。次の内容が含まれます。

- そのオブジェクトタイプに含まれる特定のオブジェクトクラスのリスト (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

次の表に、<ObjectType> 要素のサポートされている属性を示します。

表 2-28 サポートされている <ObjectType> 要素の属性

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	Identity Manager インタフェース内でこのタイプのオブジェクトに対して表示される .gif ファイルの名前を定義します。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	このタイプのリソースオブジェクトに、同一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られるかどうかを定義します。 • true の場合、このリソースオブジェクトタイプにほかのリソースオブジェクトを含めることができます。 • false の場合、このリソースオブジェクトタイプにほかのリソースオブジェクトを含めることはできません。

次の例に、ObjectType 定義を示します。

コード例 2-13 ObjectType 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"+
```

リソースオブジェクト機能の定義

<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。create および update 機能では、form 属性を指定できます。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 属性を指定しない場合、Identity Manager は create および update 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。

次の表に、オブジェクト機能のマッピングを示します。

表 2-29　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
list	listObjects	いいえ
rename	updateObject	いいえ
saveas	createObject	いいえ
update	updateObject	はい
view	getObject	いいえ

次の例では、<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 2-14 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"+
```

リソースオブジェクト属性の定義

<ObjectAttributes> セクションは、Identity Manager で管理および問い合わせ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と同じにしてください。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。

次の表に、<ObjectAttributes> に必要な属性を示します。

表 2-30 <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。

次の例は、<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 2-15 <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"+
```

次の表は、<ObjectAttribute> 属性について説明しています。

表 2-31 <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。

リソースフォームの定義

次のリソースフォームを指定する必要があります。

- 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 ResourceForm。

  たとえば、*AIX Create Group Form* または *LDAP Create Organizational Unit Form* とします。

- 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 。

たとえば、*AIX Update Group Form* または *LDAP Update Organizational Unit Form* とします。

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もできます。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機構です。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。

次の表に、トップレベルの名前空間に含まれている属性を示します。

注　特に指定されていない限り、すべての値が文字列です。

表 2-32　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。

表 2-32　トップレベルの名前空間の属性 (続き)

属性	説明
<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。

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. 必要に応じて、次のディレクトリを作成します。

   idm/WEB-INF/classes/*package_path*

   ここで、*package_path* はクラスが定義されるパッケージです。次に例を示します。

   com/waveset/adapter/sample

2. *NewResourceAdapter*.class ファイルを、前の手順で作成したディレクトリにコピーします。

3. アダプタを表す、サイズが 18x18 ピクセルで 72 DPI の gif イメージを作成します。Identity Manager により、この .gif ファイルイメージが「リソースのリスト」ページのリソース名の横に表示されます。

   .gif ファイルに名前を付ける際に、次の書式を使用する必要があります。

   YourAdapterName.gif

   アダプタ名のすべての空白を、下線で置き換える必要があります。サンプルとして、次の場所で既存のアダプタ名を確認してください。

   \waveset\idm\web\applet\images

4. gif ファイルを idm/applet/images にコピーします。

5. アプリケーションサーバーを停止してから再起動します。
   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』を参照してください。

6. リソースの HTML ヘルプファイルを作成します。

注	ヘルプファイルの例については、`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。 アプリケーションへのオンラインヘルプの組み込みについては、『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。

7. 管理者インタフェースの「管理するリソースの設定」ページで、「カスタムリソースの追加」ボタンをクリックして、アダプタクラスの完全なクラス名を入力します。たとえば、次のようにします。

   `com.waveset.adapter.sample.`*`NewResourceAdapter`*

8. アダプタを使用して、Identity Manager 内にリソースを作成します。
9. 管理するネイティブのシステムが機能することを確認します。
10. 141 ページの「接続と操作の確認」に記載されている方法で、新しい Identity Manager リソースの接続をテストします。

カスタムアダプタのテスト

カスタムリソースアダプタを記述した後で、そのアダプタの有効性をテストする必要があります。特に、リソースへの接続をテストしてください。

この節には、次のトピックが含まれています。

アダプタのユニットテスト

カスタムアダプタの有効性をユニットテスト (特にリソースへの接続をテスト) するには、次の手順を実行します。

1. アダプタを保存します。
2. 自分のマシンから、そのアダプタのユニットテストを実行します。
3. アダプタを Identity Manager に読み込みます。
4. 次の手順に従って、Identity Manager でアダプタをテストします。
   a. Identity Manager 管理者インタフェースにログインします。

b. 「リソース」>「リソースのリスト」タブをクリックします。

c. 「リソースのリスト」ページで「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。

アダプタの互換性テスト

カスタムリソースアダプタの記述および保守は、手順が非常に複雑になることがあります。カスタムアダプタが期待したとおりに動作をしない、または Identity Manager の期待する機能をアダプタが実行しないことに開発者が気づくことがよくあります。巧みに記述されたリソースアダプタでさえ、外部リソースのアップグレード後にうまく機能しなくなることがあります。

Identity Manager は、カスタムリソースアダプタの品質確認に使用できる互換性テスト機構を備えています。このテスト機構には、次の利点があります。

- カスタムアダプタの記述、公開、および保守が簡単になる
- アダプタの実行および結果の解釈が容易になる
- アダプタのサポートする機能を、アダプタに依存しない方法で、可能な限り集中的にテストできる
- アダプタの問題解決が簡略化される

この節では、Identity Manager の互換性テストスイートの使用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。

互換性テストスイートの動作

Identity Manager の互換性テストスイートは、アダプタのサポートする機能を確認するための一連の標準テストを実行します。特定のテストでアダプタの提供しない機能が必要とされる場合には、Identity Manager はそのテストを省略します。

互換性テストスイートには、リソースアダプタ上で互換性テストを実行するための有効なユーザー名とパスワードなどの、特定の情報が必要になります。通常は、標準の *DataProvider* (Identity Manager に付属) を使ってテストに必要なデータを入手できます。

注 情報を XML の式としてではなくクラス内で提供するなどの特殊な状況では、カスタムの DataProvider を記述できます。

互換性テストの実行方法

Identity Manager 互換性テストスイートを実行するには、次の手順に従います。

1. コマンドウィンドウを開きます。
2. コマンドプロンプトで、`lh` コマンドを次の書式で入力します。

`$WSHOME/bin/lh com.sun.idm.testing.adapter.CompatibilitySuite [`*Options*`] [`*testName*`]`

各表記の意味は次のとおりです。

- [*options*] には、次のオプションを指定します。
  - `-h`: 使用方法を表示する場合に使用します。

    たとえば、次のようにします。

    `Usage: CompatibilitySuite [`*arguments*`]`

    `Valid arguments:`

`-propsFile` *value*	`Path to properties files`
`-formatter` *value*	`Formatter to use for formatting output of tests`
`-user` *value*	`Name of user to execute test as`
`-pass` *value*	`Plain Text password used to log user on`
`-import` *value*	`Comma separated list of files (on server) to import`
`-toDir` *value*	`Directory to put test output in`
`-v`	`Echos all arguments passed in to screen`
`-h`	`Displays usage message`

  - `-propsFile` *file*: プロパティーファイル名の指定に使用します。

- -formatter *type,path*: XML、HTML、プレインテキスト、およびこのファイルを配置するパスの指定に使用します。

- [*testName*] は、実行するテストのコンマ区切りのリストです。

次のプロパティーにより、テストの実行方法が制御されます。

プロパティー	説明
adapter	テストするアダプタのクラス名
dp	カスタム DataProvider の名前
importScript	実行するスクリプトのパスの、コンマ区切りのリスト **注** : これらのスクリプトは、インポートした XML の文字列を返します。
ns	DataProvider の名前空間
includedTests	含めるテストのコンマ区切りのリスト
excludedTests	除外するテストのコンマ区切りのリスト
import	インポートするファイルのコンマ区切りのリスト

これらのプロパティーは、コマンド行から直接指定することも、コマンド行から指定したプロパティーファイルに追加することもできます。次に例を示します。

```
lh -DpropName=propValue
```

プロパティーが競合する場合は、propsFile で指定されたプロパティーファイル内のプロパティーが使用されます。

注 [*testName*] コマンドを使用する場合、互換性テストスイートは includedTests および excludedTests オプションを無視します。

たいていの場合、Identity Manager の提供するフレームワークは、リソースアダプタのテストに必要な柔軟性を備えています。ただし、必要に応じ、次の 2 箇所で機能を簡単に拡張できます。

- DataProvider インタフェースを実装して、カスタム DataProvider を作成できます。カスタム DataProvider を使用すると、任意のソースからのデータを受信できます。

- CompatibilityHelper インタフェースを実装して、CompatibilityHelper を提供できます。CompatibilityHelper を使用すると、テストを実行する前にリソースを初期化できます。

これらのインタフェースの実装および必須の命名規則の詳細については、Javadoc を参照してください。

例 1: デフォルトの DataProvider を使用した互換性テストの実行

この例では、デフォルトの DataProvider を使用して、SimulatedResourceAdapter で互換性テストを実行する方法を示します。

テストの準備を行う

この互換性テストを準備するには、次の手順を実行します。

1. 次のファイルを設定します。

   ```
   sample/compat/example.1/example.properties
   ```

   ```
   sample/compat/example.1/SimulatedCompatibilityConfig.xml
   ```

   注 SimulatedCompatibilityConfig 内のシミュレートしたリソースのデフォルトパスは、/tmp/mySimulatedResource.xml です。

   別の場所を指定する場合は、このパスを編集できます。

2. この例を実行する前に、Apache ant 1.6.5 から $WSHOME/WEB-INF/lib ディレクトリに ant-junit.jar をコピーします。

テストを実行する

互換性テストを実行するには、次の手順を実行します。

1. コマンドウィンドウを開きます。
2. プロンプトで、次のように入力します。

   ```
   cd $WSHOME
   ```

   ```
   bin/lh com.sun.idm.testing.adapter.CompatibilitySuite -propsFile
   sample/compat/example.1/example.properties
   ```

出力は、次の例のようになります。

コード例 2-16 デフォルトの DataProvider を使用した互換性テストの結果

```
TestSuite: com.sun.idm.testing.adapter.CompatibilitySuite
Starting internal database server ...
DB Server @ jdbc:hsqldb:hsql://127.0.0.1:57022/idm
Importing file sample/compat/example.1/SimulatedCompatibilityConfig.xml
'Create(com.sun.idm.testing.adapter.compatibility.Create)' skipped (unknown)
'Authenticate(com.sun.idm.testing.adapter.compatibility.AuthenticateUser)' skipped (unknown)
'DeleteExisting(com.sun.idm.testing.adapter.compatibility.DeleteExisting)' skipped (unknown)
'UpdateExisting(com.sun.idm.testing.adapter.compatibility.UpdateExisting)' skipped (unknown)
'RenameExisting(com.sun.idm.testing.adapter.compatibility.RenameExisting)' skipped (unknown)
'EnableExisting(com.sun.idm.testing.adapter.compatibility.EnableExisting)' skipped (unknown)
'DisableExisting(com.sun.idm.testing.adapter.compatibility.DisableExisting)' skipped (unknown)
'Iterate(com.sun.idm.testing.adapter.compatibility.Iterate)' skipped (unknown)
'DeleteMissing(com.sun.idm.testing.adapter.compatibility.DeleteMissing)' passed (77 ms)

Tests run: 9, failures: 0, errors: 0, skipped: 8, Time elapsed: 10864 ms
```

処理の内容

コード例 2-16 では、`lh` コマンドによる互換性テストの実行には、次の引数が使用されています。

```
-propsFile sample/compat/example.1/example.properties
```

`adapter` と `ns` は両方とも、テストを実行するために必須のプロパティーです。

- `adapter` プロパティーは、テスト対象のアダプタクラス名を提供します。
- `ns` プロパティーは、テスト用の名前空間を提供します。

  DataProvider は、名前空間を使用して複数の設定を行えます。

コード例 2-16 では、`import` プロパティーも使用されています。このプロパティーは、ファイルのリストをリポジトリにインポートします。`import` プロパティーは、`lh import` *filename* と似ています。

互換性テストの開始時に、テスト機構は指定されたプロパティーから `adapter` および `ns` プロパティーを取得します。

デフォルトの DataProvider は、*namespace*`#TestData` 設定オブジェクトの拡張要素からデータを取得します。この例では、拡張要素は `SimulatedCompatibilityConfig#TestData` です。

注	テストの設定時に DataProvider を指定しない場合は、デフォルトの DataProvider が使用されます。

DataProvider は、リポジトリからこの SimulatedCompatibilityConfig#TestData 設定オブジェクトを取得します。

設定オブジェクトをリポジトリ内に取得するには、次のファイル内でオブジェクトを定義する必要があります。これは import プロパティー内で指定されます。

sample/compat/example.1/SimulatedCompatibilityConfig.xml

設定を簡潔にするため、コード例 2-16 では includedTests=DeleteMissing パラメータを使用して 1 つのテストだけが実行されています。

注	使用可能なパラメータ、およびさまざまなテストで必須のパラメータの詳細については、Javadoc を参照してください。

例 2: データの追加

作成テスト、およびユーザーを作成するその他のテストを実行するには、設定オブジェクトにさらにデータを追加する必要があります。次の例では、デフォルトの DataProvider を再度使って XML ファイルをインポートする必要があります。

テストの準備を行う

この互換性テストを準備するには、次の手順を実行します。

1. 次のファイルを設定します。

   sample/compat/example.2/example.properties

   sample/compat/example.2/SimulatedCompatibilityConfig.xml

注	SimulatedCompatibilityConfig 内のシミュレートしたリソースのデフォルトパスは、/tmp/mySimulatedResource.xml です。 別の場所を指定する場合は、このパスを編集できます。

2. この例を実行する前に、Apache ant 1.6.5 から $WSHOME/WEB-INF/lib ディレクトリに ant-junit.jar をコピーします。

テストを実行する

互換性テストを実行するには、次の手順を実行します。

1. コマンドウィンドウを開きます。
2. プロンプトで、次のように入力します。

   ```
   cd $WSHOME
   ```

   ```
   bin/lh com.sun.idm.testing.adapter.CompatibilitySuite -propsFile
   sample/compat/example.2/example.properties
   ```

出力は、次の例のようになります。

コード例 2-17 テスト追加後の互換性テストの結果

```
TestSuite: com.sun.idm.testing.adapter.CompatibilitySuite
Starting internal database server ...
DB Server @ jdbc:hsqldb:hsql://127.0.0.1:57022/idm
Importing file ./sample/compat/example.2/SimulatedCompatibilityConfig.xml
'Authenticate(com.sun.idm.testing.adapter.compatibility.AuthenticateUser)' skipped (unknown)
'UpdateExisting(com.sun.idm.testing.adapter.compatibility.UpdateExisting)' skipped (unknown)
'RenameExisting(com.sun.idm.testing.adapter.compatibility.RenameExisting)' skipped (unknown)
'Iterate(com.sun.idm.testing.adapter.compatibility.Iterate)' skipped (unknown)
'DeleteMissing(com.sun.idm.testing.adapter.compatibility.DeleteMissing)' passed (15 ms)
'EnableExisting(com.sun.idm.testing.adapter.compatibility.EnableExisting)' passed (259 ms)
'DisableExisting(com.sun.idm.testing.adapter.compatibility.DisableExisting)' passed (7 ms)
'DeleteExisting(com.sun.idm.testing.adapter.compatibility.DeleteExisting)' passed (3 ms)
'Create(com.sun.idm.testing.adapter.compatibility.Create)' passed (3 ms)

Tests run: 9, failures: 0, errors: 0, skipped: 4, Time elapsed: 10178 ms
```

処理の内容

次のプロパティーをプロパティーファイル内で設定して、追加テストを要求しました。

```
IncludedTests=DeleteMissing,Create,EnableExisting,DisableExisting,DeleteExisting
```

これらのテストを実行するには、DataProvider からさらにデータを取得する必要がありました。

この新しいデータを提供するため、SimulatedCompatibilityConfig.xml で指定された設定オブジェクトにいくつかの変更が加えられました。

SimulatedCompatibilityConfig.xml ファイルにより、ユーザー名、パスワード、およびユーザー属性のリストを含む create 属性が追加されました。互換性テストにより 1 人のユーザーの作成に必要なユーザー名、パスワード、および属性が要求されると、デフォルトの DataProvider は create 属性を使用します。

SimulatedCompatibilityConfig.xml ファイルにより、スキーママップも追加されました。

例 3: テスト設定の完了

次の例では、テスト設定を完了します。

テストの準備を行う

互換性テスト設定を完了するには、次の手順を実行します。

1. 次のファイルを設定します。

   sample/compat/example.3/example.properties

   sample/compat/example.3/SimulatedCompatibilityConfig.xml

   注 SimulatedCompatibilityConfig 内のシミュレートしたリソースのデフォルトパスは、/tmp/mySimulatedResource.xml です。

   このパスを編集し、ファイル内の 2 つの行を変更することで、別の場所を指定できます。

2. この例を実行する前に、Apache ant 1.6.5 から $WSHOME/WEB-INF/lib ディレクトリに ant-junit.jar をコピーします。
3. リポジトリを初期化して encrypt コマンドを実行する必要があります。

たとえば、lh import sample/init.xml コマンドを使用してリポジトリを初期化します。ここで、元のファイルの内容は次のようになります。

```
<Attribute name="login_infos">
         <List>
           <Object>
             <Attribute name="sim_user" value="ctUser" />
             <Attribute name="sim_password" value="ctPass" />
             <Attribute name="shouldfail" value="no" />
           </Object>
           <Object>
             <Attribute name="sim_user" value="ctUser" />
             <Attribute name="sim_password" value="wrongPass" />
             <Attribute name="shouldfail" value="yes" />
           </Object>
           <Object>
             <Attribute name="sim_user" value="ctUser" />
             <Attribute name="sim_password">
               <!-- result of 'encrypt ctPass' from lh console -->

<EncryptedData>11D1DEF534EA1BE0:-32DFBF32:1165DC91D73:-7FFA|mDBIkSQB3xg=</EncryptedData>
             </Attribute>
             <Attribute name="shouldfail" value="no" />
           </Object>
           <Object>
             <Attribute name="sim_user" value="ctUser" />
             <Attribute name="sim_password">
               <!-- result of 'encrypt wrongPass' from lh console -->

<EncryptedData>11D1DEF534EA1BE0:-32DFBF32:1165DC91D73:-7FFA|m0n9bAaMx+sKpqs5PmH3eQ==
</EncryptedData>
             </Attribute>
             <Attribute name="shouldfail" value="yes" />
           </Object>
         </List>
       </Attribute>
```

4. それぞれの場合に、lh console から encrypt コマンドを実行して、暗号化されたパスワードを取得します。このパスワードは自分の環境内で暗号化解除できます。

   lh console を実行し、コンソールプロンプトで先行する EncryptedData エントリごとに一重引用符内のテキスト (encrypt ctPass など) を入力し、<EncryptedData> と </EncryptedData> の間のテキストを結果で置き換えます。

次の例を参照してください。

```
<!-- result of 'encrypt ctPass' from lh console -->
<EncryptedData>11D1DEF534EA1BE0:-65F64461:1163AB5A7B2:-7FFA|iMm4Tcqck+M=</EncryptedData>

<!-- result of 'encrypt wrongPass' from lh console -->
<EncryptedData>11D1DEF534EA1BE0:-65F64461:1163AB5A7B2:-7FFA|d1/PheqRok+J3uaggtj9Gw==
</EncryptedData>
```

また、次に示すようにブロック全体をコメントにすることで、DataProvider に 2 つの `login info` エントリをスキップさせることも可能です。

```
<!-- commented out
       <Attribute name="login_infos">
         <List>
           <Object>
             <Attribute name="sim_user" value="ctUser" />
             <Attribute name="sim_password" value="ctPass" />
             <Attribute name="shouldfail" value="no" />
           </Object>
           <Object>
             <Attribute name="sim_user" value="ctUser" />
             <Attribute name="sim_password" value="wrongPass" />
             <Attribute name="shouldfail" value="yes" />
           </Object>
           <Object>
             <Attribute name="sim_user" value="ctUser" />
             <Attribute name="sim_password">

<EncryptedData>11D1DEF534EA1BE0:-32DFBF32:1165DC91D73:-7FFA|mDBIkSQB3xg=</EncryptedData>
             </Attribute>
             <Attribute name="shouldfail" value="no" />
           </Object>
           <Object>
             <Attribute name="sim_user" value="ctUser" />
             <Attribute name="sim_password">

<EncryptedData>11D1DEF534EA1BE0:-32DFBF32:1165DC91D73:-7FFA|m0n9bAaMx+sKpqs5PmH3eQ==
</EncryptedData>
             </Attribute>
             <Attribute name="shouldfail" value="yes" />
           </Object>
         </List>
       </Attribute>
-->
```

5. 次に、新しいデータをコピーし、<EncryptedData> タグ内に貼り付けて、古いデータを置き換えます。タグ内に不要な空白や改行が存在しないことを確認してください。

テストを実行する

テストを再度実行するには、次の手順に従います。

1. コマンドウィンドウを開きます。
2. プロンプトで、次のように入力します。

```
cd $WSHOME

bin/lh com.sun.idm.testing.adapter.CompatibilitySuite -propsFile
sample/compat/example.3/example.properties
```

出力は、次の例のようになります。

コード例 2-18 テスト設定完了後の互換性テストの結果

```
TestSuite: com.sun.idm.testing.adapter.CompatibilitySuite
Starting internal database server ...
DB Server @ jdbc:hsqldb:hsql://127.0.0.1:57022/idm
Importing file ./sample/compat/example.3/SimulatedCompatibilityConfig.xml
'Create(com.sun.idm.testing.adapter.compatibility.Create)' passed (31 ms)
'Authenticate(com.sun.idm.testing.adapter.compatibility.AuthenticateUser)' passed (12 ms)
'DeleteExisting(com.sun.idm.testing.adapter.compatibility.DeleteExisting)' passed (1 ms)
'DeleteMissing(com.sun.idm.testing.adapter.compatibility.DeleteMissing)' passed (1 ms)
'UpdateExisting(com.sun.idm.testing.adapter.compatibility.UpdateExisting)' passed (33 ms)
'RenameExisting(com.sun.idm.testing.adapter.compatibility.RenameExisting)' passed (5 ms)
'EnableExisting(com.sun.idm.testing.adapter.compatibility.EnableExisting)' passed (10 ms)
'DisableExisting(com.sun.idm.testing.adapter.compatibility.DisableExisting)' passed (5 ms)
'Iterate(com.sun.idm.testing.adapter.compatibility.Iterate)' passed (352 ms)

Tests run: 9, failures: 0, errors: 0, skipped: 0, Time elapsed: 10262 ms
```

処理の内容

テストスイート全体を実行するはずの、含まれるテストを指定する行が example.properties ファイルから削除されました。

残りのテストを実行するには追加データが必要であるため、SimulatedCompatibilityConfig.xml ファイルが update、rename、および iterate 属性を含むように変更されました。これらの属性により、ユーザーの変更、ユーザー名の変更、および処理を繰り返すユーザーセットの作成が行われます。また、login_info 属性も追加されました。この属性は、リソースアダプタが認証をサポートしている場合に、ユーザー認証に使用する項目のリストを指定します。

最後に、login info エントリの下に指定されている shouldfail 属性により、ネガティブテストが許可されます。これで、このスイート内のテストは、エラーもスキップされるテストもなく終了するはずです。

例 4: Javascript または Beanshell スクリプトの実行

互換性テストを使用すると、javascript または beanshell スクリプトを実行して、スクリプトの結果をリポジトリにインポートできます。いずれかのスクリプトが、インポートする XML を含む文字列を返す必要があります。

Identity Manager には、Apache Velocity テンプレートの例、およびこのテンプレートを使用するいくつかのサポート beanshell スクリプトが含まれています。beanshell スクリプトは、必須の変数を入力するためだけに作成されました。このスクリプトを使用すると、デフォルトの DataProvider の操作が非常に簡単になります。

テストの準備を行う

beanshell スクリプトで実行する互換性テストの準備を行うには、次の手順に従います。

1. 次のファイルを設定します。

   ```
   sample/compat/example.4/example.properties
   ```

   ```
   sample/compat/example.4/SimulatedCompatibilityConfig.bsh
   ```

   注 SimulatedCompatibilityConfig 内のシミュレートしたリソースのデフォルトパスは、/tmp/mySimulatedResource.xml です。

   別の場所を指定する場合は、このパスを編集できます。

   このファイル内の 2 つの行を変更する必要があります。

2. この例を実行する前に、Apache ant 1.6.5 から $WSHOME/WEB-INF/lib ディレクトリに ant-junit.jar をコピーします。

テストを実行する

互換性テストを実行するには、次の手順を実行します。

1. コマンドウィンドウを開きます。
2. プロンプトで、次のように入力します。

   ```
   cd $WSHOME
   ```

   ```
   bin/lh com.sun.idm.testing.adapter.CompatibilitySuite -propsFile
   sample/compat/example.4/example.properties
   ```

出力は、次の例のようになります。

コード例 2-19 Beanshell スクリプト実行後の互換性テストの結果

```
TestSuite: com.sun.idm.testing.adapter.CompatibilitySuite
Starting internal database server ...
DB Server @ jdbc:hsqldb:hsql://127.0.0.1:57022/idm
Executing script
/opt/build/dv207518/adapterTestsTemp/waveset/export/pipeline/./sample/compat/example.4/Simul
atedCompatibilityConfig.bsh
Importing results
'Create(com.sun.idm.testing.adapter.compatibility.Create)' passed (25 ms)
'Authenticate(com.sun.idm.testing.adapter.compatibility.AuthenticateUser)' passed (11 ms)
'DeleteExisting(com.sun.idm.testing.adapter.compatibility.DeleteExisting)' passed (5 ms)
'DeleteMissing(com.sun.idm.testing.adapter.compatibility.DeleteMissing)' passed (4 ms)
'UpdateExisting(com.sun.idm.testing.adapter.compatibility.UpdateExisting)' passed (4 ms)
'RenameExisting(com.sun.idm.testing.adapter.compatibility.RenameExisting)' passed (3 ms)
'EnableExisting(com.sun.idm.testing.adapter.compatibility.EnableExisting)' passed (11 ms)
'DisableExisting(com.sun.idm.testing.adapter.compatibility.DisableExisting)' passed (5 ms)
'Iterate(com.sun.idm.testing.adapter.compatibility.Iterate)' passed (22 ms)

Tests run: 9, failures: 0, errors: 0, skipped: 0, Time elapsed: 11354 ms
```

処理の内容

DataProvider により `importScript` プロパティーが提供されたため、`SimulatedCompatibilityConfig.bsh` スクリプトが実行されました。このスクリプトの返す XML 文字列は、リポジトリ内に設定オブジェクトとしてインポートされます。スクリプトには必要な項目が指定されており、Velocity テンプレートにより文字列が作成されます。

次のいずれかの方法で、`import` スクリプトをデバッグできます。

- `lh console` を使ってトレースをオンに設定して、生成されたログファイルでスクリプトの戻り値を確認します。たとえば、次のように入力します。

  ```
  trace 4 com.sun.idm.testing.adapter.CompatibilitySuite
  ```

- `excludedTests` プロパティーを使って各テストを除外します。テストは実行されませんが、スクリプトは実行されます。

注 この例では beanshell スクリプトが使用されていますが、Javascript を使用することもできます。

Apache Velocity テンプレートエンジンを使用してスクリプティングを容易にする複数の beanshell ヘルパーが、sample/compat/beanshell ディレクトリ内に用意されています。

注	beanshell ヘルパーの使用を支援する目的で、コメント付きの例が含まれています。

テンプレートを使用するには、次のコードを beanshell スクリプトの先頭に追加します。

```
// import helpers
String wavesetHome = Util.getWavesetHome();

if(wavesetHome != null) {
    if ( wavesetHome.startsWith("file:" ) ) {
       wavesetHome = wavesetHome.substring("file:".length());
    }

    addClassPath(wavesetHome + "./sample/compat/");
}

importCommands("beanshell");
```

ヘルパーの使用は任意です。

スクリプトを使用する際の要件は、スクリプトが XML を含む文字列を返すことだけです。スクリプトは、_params を使用して CompatibilitySuite に渡された任意のパラメータにアクセスできます。

ここで、_params には、次のパラメータのいずれかを含めることができます。

プロパティー	説明
adapter	テストするアダプタのクラス名
dp	カスタム DataProvider の名前
importScript	実行するスクリプトのパスの、コンマ区切りのリスト **注**: これらのスクリプトは、インポートした XML の文字列を返します。
ns	DataProvider の名前空間

プロパティー	説明
includedTests	含めるテストのコンマ区切りのリスト
excludedTests	除外するテストのコンマ区切りのリスト
import	インポートするファイルのコンマ区切りのリスト

注 これらのプロパティーは、プロパティーファイル内で設定するか、コマンド行で -D コマンドを使用して _params マップに追加しない限り、_params マップ内では提供されません。

beanshell では、params.get("parameter_name") の呼び出しを使用してこれらのパラメータを取得できます。

beanshell スクリプトが設定オブジェクトの名前を作成できるように、namespace パラメータの設定方法をスクリプトに知らせる必要がある場合は、次の方法でスクリプトが取得されます。

```
String namespace = _params.get("ns");
```

例 5: Web コンテナ内部からのテストの実行

次の手順を使用して、Web コンテナの内部から互換性テストを実行します。

テストの準備を行う

この例を実行する前に、次の手順を実行します。

1. Apache ant 1.6.5 から $WSHOME/WEB-INF/lib ディレクトリに ant-junit.jar をコピーします。
2. web.xml ファイルの次の部分のコメントを解除して、com.sun.idm.testing.adapter.compatibility.CTServlet を有効にします。
   - サーブレット定義のコメント解除 :

```
<servlet>
    <servlet-name>CompatibilityTests</servlet-name>

<servlet-class>com.sun.idm.testing.adapter.compatibility.CTServlet
</servlet-class>
    <load-on-startup>1</load-on-startup>
  </servlet>
```

- サーブレットマッピングのコメント解除 :

```
<servlet-mapping>
    <servlet-name>CompatibilityTests</servlet-name>
    <url-pattern>/servlet/CTServlet</url-pattern>
  </servlet-mapping>
```

注

- 新しいサーブレットを使用するために、コンピュータの再起動が必要な場合があります。
- サーブレットは、特定のパラメータを伴う POST 要求を受け入れます。インポートしたファイルなど、一部のパラメータを使用することで複数の指定が可能になります。

3. 次のパラメータを互換性スイートに指定できます。

プロパティー	説明
adapter	テストするアダプタのクラス名
dp	カスタム DataProvider の名前
excludedTests	除外するテストのコンマ区切りのリスト
import	インポートするファイルのコンマ区切りのリスト
importScript	実行するスクリプトのパスの、コンマ区切りのリスト **注** : これらのスクリプトは、インポートした XML の文字列を返します。
includedTests	含めるテストのコンマ区切りのリスト
ns	DataProvider の名前空間
pass	ユーザーのログオンに使用するプレーンテキストパスワード 注 : このパスワードはプレーンテキストで送信されます。これにより、サーブレットを有効にするかどうかの決定が影響を受けることがあります。
user	テストを実行するユーザーの名前

さらに、リモート専用のパラメータには次が含まれます。

プロパティー	説明
importXMLText	インポートする XML を含む文字列
importScriptText	実行するスクリプトを含む文字列
importScriptSuffix	スクリプトが beanshell の場合は、bsh を指定します。 スクリプトが javascript の場合は、js を指定します。 **注** : 複数のスクリプトをサーブレットに指定する場合、すべてのスクリプトを javascript にするか、beanshell にする必要があります。javascript と beanshell を混在させることはできません。

サーブレットへのアクセスには、debug/CompatTests.jsp またはコマンド行の Java プログラム CTContainerTest.java を使用できます。

4. リモートでのテストの実行準備として、ファイル idmtesting.jar および sample/compat 内のサンプルフォルダをリモートシステムにコピーします。

テストを実行する

CompatTests.jsp ページからテストを実行するには、次の手順を実行します。

1. ブラウザを開いて、*idm instance*/debug/CompatTests.jsp に移動します。次に例を示します。

   http://*example.com*:8080/idm/debug/CompatTests.jsp

2. この例を実行するには、次の値を指定する必要があります。

   Namespace = SimulatedAdapterTests

   Adapter = com.waveset.adapter.SimulatedResourceAdapter

   User Name to Run Test as = configurator

   Password for User to Run Test as = *configurator's password*

   Script type = Beanshell radio button

   Import Result of this Script Text = *SimulatedCompatibilityConfig.bsh file contents*

3. sample/compat/example.4 ディレクトリから SimulatedCompatibilityConfig.bsh ファイルの内容をコピーして、このテキストフィールドに貼り付けます。

注 このスクリプトでは例 4 を実行していますが、同じ方法でほかの例を実行できます。ほかのパラメータも使用できますが、jsp ファイル内で名前が若干変更されています。

CTContainerTest というコマンド行の Java プログラムを使って、互換性テストをリモートで実行することもできます。次に使用方法を示します。

CTContainerTest -url url [-v] [*-parm1_name parm1_value -parm2_name parm2_value ... -parmx_name parmx_value*]

各表記の意味は次のとおりです。

- パラメータ名は、サーブレットの受け入れるパラメータと同じです。
- パラメータ値は、サーブレットの受け入れる値と同じです。

サーブレットでは、次のパラメータをコマンド行引数として使用することはできません。

- importScriptText

  importLocalScriptFile パラメータを使用して、importScriptText パラメータをサーブレットに送信できます。importLocalScriptFile オプションは、パラメータ値で指定されたファイルの内容を読み取り、importScriptText パラメータ名を使ってそのファイルの内容をサーブレットに送信します。

- importXMLText

  importLocalXMLFile パラメータを使って importXMLText パラメータをサーブレットに送信できます。importLocalXMLFile オプションは、パラメータ値で指定されたファイルの内容を読み取り、importXMLText パラメータ名を使ってそのファイルの内容をサーブレットに送信します。

使用可能なパラメータおよびその使用方法の詳細については、CTContainerTest プログラムを引数なしで実行してください。次にその実行方法を示します。

java -cp idmtesting.jar com.sun.idm.testing.adapter.CTContainerTest

次の例は、このコマンドを実行するさまざまな方法を示します。

注 これらの例を Windows 環境で実行するには、*hostname* および *port* を調整し、クラスパスの区切り文字をコロン (:) からセミコロン (;) に変更し、パスの区切り文字をバックスラッシュ (/) からスラッシュ (\) に変更する必要があります。

コード例 2-20 デフォルトの DataProvider を使用した互換性テストの実行

```
java -cp idmtesting.jar:idmcommon.jar
com.sun.idm.testing.adapter.CTContainerTest -url
"http://host:port/idm/servlet/CTServlet" -adapter
com.waveset.adapter.SimulatedResourceAdapter -ns SimulatedAdapterTests
-importLocalXMLFile ./example.1/SimulatedCompatibilityConfig.xml
-includedTests DeleteMissing
```

コード例 2-21 追加テストを含む互換性テストの実行

```
java -cp idmtesting.jar:idmcommon.jar
com.sun.idm.testing.adapter.CTContainerTest -url
"http://host:port/idm/servlet/CTServlet" -adapter
com.waveset.adapter.SimulatedResourceAdapter -ns SimulatedAdapterTests
-importLocalXMLFile ./example.2/SimulatedCompatibilityConfig.xml
-includedTests
DeleteMissing,EnableExisting,DisableExisting,DeleteExisting,Create
```

コード例 2-22 テスト設定完了後の互換性テストの実行

```
java -cp idmtesting.jar:idmcommon.jar
com.sun.idm.testing.adapter.CTContainerTest -url
"http://host:port/idm/servlet/CTServlet" -adapter
com.waveset.adapter.SimulatedResourceAdapter -ns SimulatedAdapterTests
-importLocalXMLFile ./example.3/SimulatedCompatibilityConfig.xml
```

コード例 2-23　Beanshell スクリプト実行後の互換性テストの実行

```
java -cp idmtesting.jar:idmcommon.jar
com.sun.idm.testing.adapter.CTContainerTest -url
"http://host:port/idm/servlet/CTServlet" -adapter
com.waveset.adapter.SimulatedResourceAdapter -ns SimulatedAdapterTests
-importLocalScriptFile ./example.4/SimulatedCompatibilityConfig.bsh
```

リソースオブジェクトのテスト

この節では、次に示すリソースオブジェクトのテスト方法について説明します。

- リソースオブジェクトの表示と編集
- Identity Manager でのリソースオブジェクトのテスト

リソースオブジェクトの表示と編集

リポジトリ内の生の XML を表示して、リソースの設定を確認できます。

次の手順を使用して、リソースオブジェクトを表示および編集できます。

1. 管理者ユーザーインタフェースにログインします。
2. ブラウザに `http://host:port/idm/debug` と入力して、Identity Manager の「Debug」ページを開きます。
3. 「List Objects」ボタンの横のプルダウンメニューで、「Resource」を選択します。
4. 「List Objects」ボタンをクリックします。

   「List Objects of Type: Resource」ページが開き、すべてのリソースアダプタおよび Active Sync 対応アダプタのリストが表示されます。

   注　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。

5. 表示するリソースオブジェクトを見つけます。
   - リソースオブジェクトを表示するには、「View」リンクをクリックします。
   - リソースオブジェクトを編集するには、「Edit」リンクをクリックします。
6. 完了したら、「Back」をクリックします。

Identity Manager でのリソースオブジェクトのテスト

Identity Manager 管理インタフェースの「リソースの検索」および「リソースのリスト」ページを使用して、リソースオブジェクトの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 2-33 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 2-34 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。
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。

カスタムアダプタのトラブルシューティング

Identity Manager の「デバッグ」ページを使って、カスタムアダプタ内のメソッドをトレースできます。最初にトレースを有効にして、トレースが要求されるメソッドを特定する必要があります。また、カスタムアダプタ内に新しいメソッド用のログエントリを作成する呼び出しを提供する必要もあります。

アダプタのデバッグを実行するには、アダプタの生成するログファイルを確認する必要があります。トレースを有効にして、トレースするメソッドを特定した場合、アダプタはそのリソース設定をログファイルに書き込みます。この情報を使用して、アダプタが起動したこと、およびすべての設定変更が保存されたことを確認します。

注 カスタムアダプタのトレースおよびデバッグの詳細については、『Identity Manager Tuning, Troubleshooting, and Error Messages』を参照してください。

カスタムアダプタの保守

新しい Identity Manager パッチまたはサービスパックをインストールするたびに、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。アダプタを変更または拡張して、新しいリリースで加えられた変更にアダプタを適合させることが必要な場合があります。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。

新しいリリースにアップグレードする場合、ターゲットの Identity Manager バージョンによっては、すべてのカスタムリソースアダプタの再コンパイルが必要になることがあります。Identity Manager API を使用するすべてのカスタム Java (カスタムリソースアダプタを含む) は、アップグレード時に再コンパイルが必要です。さらに、Identity Manager ライブラリを使用するほかの Java クラスについても考慮してください。

アップグレードの詳細については、『Identity Manager Upgrade』を参照してください。

注　現在の Identity Manager のインストール内にカスタムの要素が大量に含まれる場合は、Sun のアカウント担当者または Sun のカスタマサポートに相談してください。

第 3 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 4 章

Identity Manager Web サービスでの SPML 1.0 の使用

Service Provisioning Markup Language (SPML) 1.0 は、サービスプロビジョニングアクティビティーと通信するオープンインタフェースを提供するための OASIS 標準です。Identity Manager の Web サービスには、HTTP 用の SPML 要求を使用してアクセスします。

この章では、Identity Manager および Identity Manager Service Provider 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に関する情報が含まれます。

説明する内容は次のとおりです。

注 Identity Manager は、SPML version 1.0 と version 2.0 の両方をサポートします。

この章で説明されている概念は、特に SPML 1.0 に関連していますが、この章の内容は第 5 章「Identity Manager Web サービスでの SPML 2.0 の使用」で説明される概念を理解するための基礎としても役立ちます。

開始する前に

Identity Manager Web サービスの操作を開始する前に、以下の節を確認してください。

対象読者

この章は、アプリケーション開発者および Identity Manager の配備、手続き型ロジックの実装、SPML 1.0 クラスを使用したサービスプロビジョニング要求メッセージのフォーマットや応答メッセージの解析などを担当する開発者を対象としています。

重要な注意点

SPML 1.0 を操作する前に、次の事柄に注意する必要があります。

- Identity Manager Web サービスインタフェースの操作で最適なパフォーマンスを得るには、Identity Manager に同梱されている 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。
- Service Provider REF キットには、Service Provider 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 1.0 を使用して Identity Manager Service Provider (サービスプロバイダ) 機能にアクセスできます。これらの機能は、SPML version 2.0 では使用できません。

  サービスプロバイダ SPML インタフェースは Identity Manager SPML インタフェースに非常に似ています。設定および操作上の相違点は、この章の該当する箇所に説明されています。

関連ドキュメントと Web サイト

SPML の使用方法に関する詳細については、この章で提供する情報に加えて、この節で示すマニュアルおよび Web サイトも参照してください。

推奨ドキュメント

SPML version 2.0 の使用方法の詳細については、本書の第 5 章「Identity Manager Web サービスでの SPML 2.0 の使用」を参照してください。

有用な Web サイト

OpenSPML の使用方法の詳細について参照したり、OpenSPML 1.0 Toolkit をダウンロードしたりするには、次の Web サイトにアクセスしてください。

http://www.openspml.org

SPML の設定

SPML インタフェースを公開するには、特定のリポジトリオブジェクトをインストールおよび変更し、`Waveset.properties` ファイルを編集することにより、Identity Manager サーバーを正しく設定する必要があります。

SPML インタフェースを設定する手順については、次の節で説明します。

- リポジトリオブジェクトのインストールと変更
- Waveset.properties ファイルの編集
- 設定オブジェクトの編集

リポジトリオブジェクトのインストールと変更

次の表は、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 4-1 SPML の設定に使用されるリポジトリオブジェクト

Object	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 フォームを定義します。
`Configuration:IDM Schema Configuration`	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、Identity Manager ユーザーオブジェクトのクエリー可能かつ概要の属性であるユーザー属性を定義します。 • SPML フィルタで使用する属性に対して、クエリー可能な属性を定義します。 • 最適化された検索で返す属性に対して、概要の属性を定義します。
`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 には、`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が含まれています。`sample/spml1.xml` 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た作成中の標準スキーマを追跡するために、`person` クラスが定義されています。このクラスをカスタマイズしないでください。次の場合を除き、`person` クラスの標準スキーマとの一貫性を維持してください。

サービスプロバイダ SPML インタフェースを設定する場合は、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、サービスプロバイダ 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 要求または SPML 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。たとえば、view はクライアントと Identity Manager の間で直接渡されます。

次の (デフォルト) パスを使用して、サービスプロバイダ SPML インタフェースにアクセスします。

```
/servlet/spespml
```

たとえば、*host*:*port* 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://host:port/idm/servlet/spespml
```

各表記の意味は次のとおりです。

- *host* は Identity Manager を実行しているマシンです。
- *port* は サーバーが待機している TCP ポートの番号です。

注 標準 SPML スキーマの最新情報については、SPML 1.0 仕様 (http://www.openspml.org/) を参照してください。

Waveset.properties ファイルの編集

次の表は、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 4-2 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
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。

soap.username でプロキシユーザーを定義し、次のパスワードプロパティーのいずれか 1 つだけを指定できます。

- soap.password の指定がもっとも簡単ですが、この方法では、properties 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 の指定の方が安全なオプションですが、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。

Web サービスでは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。この設定は、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 応答するサーバーが存在する HTTP ポートが一般にアクセス可能な場合、プロキシユーザーの使用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する Identity Manager 操作を設定できます。

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。関連するいくつかの Web 標準規格が認証に使用できますが、これらの標準が一般的に使用されるようにはなっていません。現時点では、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 の使用です。Identity Manager は、SSL の設定方法を指示していません。

プロキシユーザーまたは SSL を使用できない場合、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。資格情報の指定のサポートを含む SpmlClient クラスの拡張である LighthouseClient クラスを使用して、ログイン要求を実行し、すべての SPML 要求内のセッショントークンを渡すことができます。

注

サービスプロバイダ SPML インタフェースは、認証と承認をサポートしませんが、Identity Manager SPML インタフェースで、サービスプロバイダ SPML の代わりに `IDMXUser` ビューを使用するよう設定することができます。

サービスプロバイダ 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。サービスプロバイダ 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 サービスプロバイダ 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作成

暗号化パスワードを作成するには、次のいずれかの方法を使用します。

- Identity Manager コンソールを開き、`encrypt` コマンドを使用します。
- Identity Manager の「デバッグ」ページまたはコンソールを開き、XML のプロキシユーザーを表示します。`password` 属性の値に対する `WSUser` 要素を検索し、`soap.epassword` プロパティーの値として使用します。

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設定する必要があります。

- Configuration:SPML オブジェクト
- Configuration:SPMLPerson オブジェクト
- Configuration:IDM Schema Configuration オブジェクト
- TaskDefinition:SPMLRequest オブジェクト
- 配備記述子

注	サービスプロバイダ 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。

Configuration:SPML オブジェクト

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。

次の属性は、`GenericObject`、`schemas`、および `classes` で定義されます。

- 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 つのオブジェクトを定義します。

当初は、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 スキーマを定義していない場合、Identity Manager は、スキーマ要求メッセージを受信したあとに、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。

> ➤|∞∞ **ベストプラクティス :**
>
> SPML スキーマの記述はベストプラクティスと見なされるので、汎用のツール (OpenSPML ブラウザなど) を使用して要求を作成できます。

デフォルトの SPML 設定

次の例は、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 4-1 デフォルトの SPML 設定

```
<Configuration name='SPML' authType='SPML'>
<Extension>
<Object>
  <Attribute name='classes'>
    <List>
      <Object name='person'>
        <Attribute name='type' value='User'/>
        <Attribute name='form' value='SPMLPerson'/>
        <Attribute name='default' value='true'/>
        <Attribute name='identifier' value='uid'/>
      </Object>
      <!-- 'user' クラスはフォームを定義しないので、組み込み型の簡素化されたスキーマをデフォルトに設定する。この処理は実際には好ましくないが、現在 SimpleRpc がこれに依存している。
      -->
      <Object name='user'>
        <Attribute name='type' value='User'/>
        <Attribute name='identifier' value='waveset.accountId'/>
      </Object>
      <!-- 'userview' クラスは、未変更のビューを渡す "view" フォームを定義する
      -->
      <Object name='userview'>
        <Attribute name='type' value='User'/>
        <Attribute name='form' value='view'/>
        <Attribute name='identifier' value='waveset.accountId'/>
        <Attribute name='multiValuedAttributes'>
          <List>
            <String>waveset.resources</String>
            <String>waveset.roles</String>
            <String>waveset.applications</String>
          </List>
```

コード例 4-1 デフォルトの SPML 設定 (続き)

```
        </Attribute>
      </Object>

      <Object name='role'>
        <Attribute name='type' value='Role'/>
        <Attribute name='form' value='SPMLRole'/>
        <Attribute name='default' value='true'/>
        <Attribute name='identifier' value='name'/> <!--  属性 ... 暫定的に ? -->
      </Object>
</Configuration>

</Waveset>
```

この例では、次の 2 つのクラスが定義されています。

- 標準の `person`
- `request` という名前の Identity Manager 拡張

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。`name` 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、この値は必須ではありません。この名前を追加要求または検索要求での `objectclass` 属性の値として使用できます。
- 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、この属性は `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。この属性には、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` に指定されている場合は、この属性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、1 つのクラスをデフォルトとして指定する必要があります。
- 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 検索要求をクラスに対して評価する場合は、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。このアプローチは、User 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、通常 User 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、CDATA でマークされたセクションを使用する方が便利です。Identity Manager が spml.xml ファイルを正規化すると、CDATA でマークされたセクションは < および > 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と一般的なアカウント管理操作のためのさまざまな拡張要求が含まれています。

Configuration: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。

次の例は、標準の person クラスがフォームを参照する方法を示します。

コード例 4-2 標準の Person クラスでのフォームの参照

```
<Configuration name='SPMLPerson'>
   <Extension>
      < フォーム >
         <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>
         <Field name='gn'>
            <Derivation><ref>global.firstname</ref></Derivation>
         </Field>
         <Field name='global.firstname'>
            <Expansion><ref>gn</ref></Expansion>
         </Field>
```

コード例 4-2 標準の Person クラスでのフォームの参照 (続き)

```
          <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

- <Display> 要素を含まない
- データ変換のためにのみ定義される
- 対話型編集としては使用されない

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 name を内部 name 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。

Configuration:IDM Schema Configuration オブジェクト

SPML 検索フィルタで属性を使用する場合は、これらの属性を 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。Identity Manager は、その属性の値が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、拡張属性をリポジトリ内に格納します。

拡張属性の数は最小限に抑えるようにしてください。拡張属性の数が多すぎると、リポジトリのサイズが増加するだけでなく、Identity Manager 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があります。Identity Manager クエリーで属性を使用するには、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。

ユーザーの概要の属性のセットに属性を含める場合は、これら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、代わりにもっとも重要なユーザー属性の一部のみ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、概要の属性が返されます。

次の例では、`firstname`、`lastname`、`fullname`、`description`、および `telephone` が `IDMAttributeConfigurations` で定義されたあとにユーザーの `IDMObjectClassConfiguration` に存在する拡張属性です。`firstname`、`lastname`、および `telephone` のみがクエリー可能かつ概要の属性です。

コード例 4-3 拡張属性として宣言された `telephone` と `description`

```
<Configuration name="IDM Schema Configuration"
                  id='#ID#Configuration:IDM_Schema_Configuration'
                  authType='IDMSchemaConfig'>
       <IDMSchemaConfiguration>
           <IDMAttributeConfigurations>
               <!-- これは標準のセットである -->
               <IDMAttributeConfiguration name='firstname'
                                          syntax='STRING'/>
               <IDMAttributeConfiguration name='lastname'
                                          syntax='STRING'/>
               <IDMAttributeConfiguration name='fullname'
                                          syntax='STRING'/>
               <!-- これらは、SPML の拡張である -->
               <IDMAttributeConfiguration name='description'
                                          syntax='STRING'/>
               <IDMAttributeConfiguration name='telephone'
                                          syntax='STRING'/>
           </IDMAttributeConfigurations>
           <IDMObjectClassConfigurations>
               <IDMObjectClassConfiguration name='User'
                                            extends='Principal'
                                            description='User description'>
                   <IDMObjectClassAttributeConfiguration name='firstname'
                                                         queryable='true'
                                                         summary='true'/>
                   <IDMObjectClassAttributeConfiguration name='lastname'
                                                         queryable='true'
                                                         summary='true'/>
                   <IDMObjectClassAttributeConfiguration name='fullname'/>
                   <IDMObjectClassAttributeConfiguration name='description'/>
```

コード例 4-3 拡張属性として宣言された telephone と description (続き)

```
                <IDMObjectClassAttributeConfiguration name='telephone'
                                                      queryable='true'
                                                      summary='true'/>
            </IDMObjectClassConfiguration>
        </IDMObjectClassConfigurations>
    </IDMSchemaConfiguration>
</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、同じ Configuration: 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。

TaskDefinition: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。その後サーバーが、あとの状態要求に対する SPML 応答で、タスクインスタンスのリポジトリ ID を返します。

```
<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 値は変更することができます。非同期要求が完了すると、通常システムは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、その結果の値を指定された期間保持します。これらの結果が保持される期間は、サイト固有の値です。

正の resultLimit 値を使用して、タスクが完了したあとにシステムが結果を保持できる時間 (秒単位) を指定できます。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、そのタスク名を別の値に変更しないかぎり、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 サービスプロバイダ 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、SPML 要求を受信するサーブレットの宣言が含まれるように編集する必要があります。

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 ファイル内にあるサーブレット宣言を調べてください。次の例は、サーブレット宣言を示しています。

コード例 4-4 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`

各表記の意味は次のとおりです。

- ここで、*host* は Identity Manager を実行しているマシンです。
- *port* はサーバーが待機する TCP ポートの番号です。

<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。

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。

ブラウザを起動するには、次の手順に従います。

1. コマンドウィンドウを開きます。
2. コマンドプロンプトで、次のコマンドを入力します。

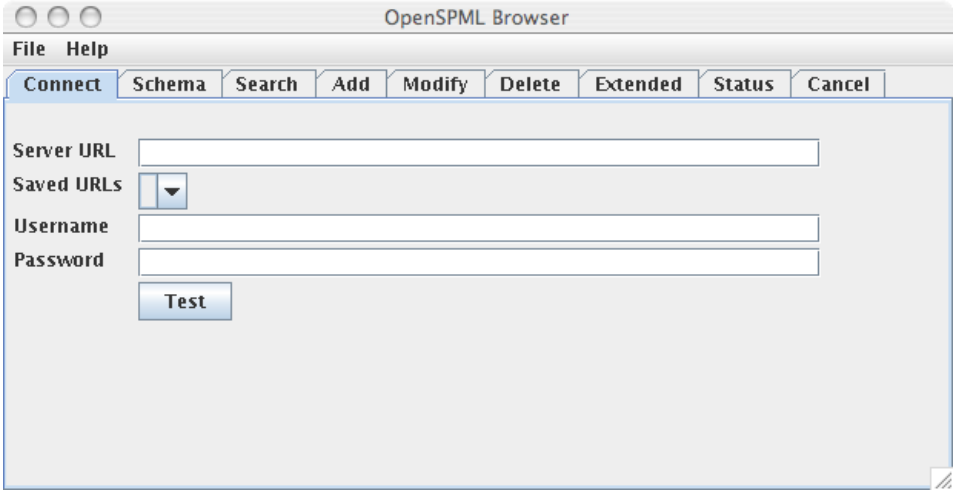
```
lh spml
```

Identity Manager サーバーへの接続

Identity Manager サーバーに接続するには、次の手順に従います。

1. OpenSPML ブラウザを開き、「**接続**」タブを選択します。

図 4-1　　OpenSPML ブラウザの例



2. 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://host:8080/idm/servlet/rpcrouter2`

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」タブを選択し、「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」タブを選択し、「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 が正しいかどうか確認します。
- 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、206 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、SPML アプリケーションに 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になります。Java アプリケーションの場合は、OpenSPML ツールキットを使用して、このメカニズムを設定します。

注　Identity Manager Web サービスインタフェースの操作で最適なパフォーマンスを得るには、Identity Manager に同梱されている 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。

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

次の表は、OpenSPML 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 4-3 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。 オブジェクトのタイプは、objectclass 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必要があります。SPML では、標準スキーマがまだ定義されていませんが、ほとんどすべてのスキーマをサポートするように Identity Manager を設定できます。
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。
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。
ModifyRequest	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。
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。
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。
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。
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が含まれます。要求のクラスごとに、対応する応答クラスが存在します。たとえば、AddResponse および ModifyResponse などがあります。
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。

サービスプロバイダ REF キットには、サービスプロバイダ 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 サービスプロバイダ 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://host:port/idm/spespml`

各表記の意味は次のとおりです。

- ここで、*host* は Identity Manager サービスプロバイダ を実行しているマシンです。
- *port* はサーバーが待機する TCP ポートの番号です。

サービスプロバイダ のトレースを有効にすると、サービスプロバイダ SPML メッセージが標準出力に出力されます。

ExtendedRequest の例

次の表は、クライアントとの間でメッセージを送受信するために使用できる、さまざまな ExtendedRequest クラスを示しています。

表 4-4 メッセージを送受信するための ExtendedRequest クラス

ExtendedRequest	説明
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。
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。
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。
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。
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、ExtendedRequests をビュー操作に変換します。

次の各節で、これらのクラスの標準的な形式を使用する例が提供されています。

ExtendedRequest の例

次の例は、ExtendedRequest の標準的な形式を示しています。

コード例 4-5 ExtendedRequest の形式

```
ExtendedRequest req = new ExtendedRequest();
req.setOperationIdentifier("changeUserPassword");
req.setAttribute("accountId", "exampleuser");
req.setAttribute("password", "xyzzy");
req.setAttribute("accounts","Lighthouse,LDAP,RACF");
ExtendedResponse res = (ExtendedResponse) client.send(req);
```

ほとんどの SPML ExtendedRequests は、次の引数を受け入れます。

- accountId ー Identity Manager ユーザー名を特定します。
- accounts ー リソース名をコンマ区切りのリストで提供します。

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 (Identity Manager のユーザーアカウントを含む) が更新されます。accounts を渡す場合は、指定した SPML 操作は指定したリソースのみを更新します。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 の例

次の例は、deleteUser 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)。

注　この要求をカスタマイズする場合、副作用を伴う可能性があります。

コード例 4-6　deleteUser 要求

```
ExtendedRequest req = new ExtendedRequest();
req.setOperationIdentifier("deleteUser");
req.setAttribute("accountId","exampleuser");
req.setAttribute("accounts","Lighthouse,LDAP,RACF");
ExtendedResponse res = (ExtendedResponse) client.send(req);
```

disableUser の例

次の例は、disableUser 要求の標準的な形式を示しています。
(ビュー > 「無効化」 ビュー)。

コード例 4-7　disableUser 要求

```
ExtendedRequest req = new ExtendedRequest();
req.setOperationIdentifier("disableUser");
req.setAttribute("accountId","exampleuser");
req.setAttribute("accounts","Lighthouse,LDAP,RACF");
ExtendedResponse res = (ExtendedResponse) client.send(req);
```

enableUser の例

次の例は、enableUser 要求の標準的な形式を示しています。
(ビュー > 「有効化」 ビュー)。

コード例 4-8　enableUser 要求

```
ExtendedRequest req = new ExtendedRequest();
req.setOperationIdentifier("enableUser");
req.setAttribute("accountId","exampleuser");
req.setAttribute("accounts","Lighthouse,LDAP,RACF");
ExtendedResponse res = (ExtendedResponse) client.send(req);
```

launchProcess の例

次の例は、launchProcess 要求の標準的な形式を示しています。
(ビュー > 「プロセス」 ビュー)。

コード例 4-9 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。

- launchProcess: カスタムプロセスの起動。
- process: 起動する Identity Manager リポジトリ内の TaskDefinition オブジェクトの名前。
- taskName: ワークフローの起動に必要なタスクの名前。

  task instance オブジェクトは、プロセスの実行時の状態を保持します。

残りの属性は任意であり、これらの属性はタスクに渡されます。

listResourceObjects の例

次の例は、listResourceObjects 要求の標準的な形式を示しています。

コード例 4-10 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

resetUserPassword の例

次の例は、resetUserPassword 要求の標準的な形式を示しています (ビュー > 「ユーザーパスワードのリセット」ビュー)。

コード例 4-11 resetUserPassword 要求

```
ExtendedRequest req = new ExtendedRequest();
req.setOperationIdentifier("resetUserPassword");
req.setAttribute("accountId","exampleuser");
req.setAttribute("accounts","Lighthouse,LDAP,RACF");
ExtendedResponse res = (ExtendedResponse) client.send(req);
```

runForm の例

次の例は、runForm 要求の標準的な形式を示しています。

コード例 4-12 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

次の例は、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すフォームを示します。

コード例 4-13 クエリーフォーム

```
<Configuration name='SPML Get Object Names'>
  <Extension>
    < フォーム >
      <Field name='roles'>
        <Derivation>
          <invoke class='com.waveset.ui.FormUtil'>
            <ref>display.session</ref>
            <s>Role</s>
          </invoke>
        </Derivation>
      </Field>
      <Field name='resources'>
        <Derivation>
          <invoke class='com.waveset.ui.FormUtil'>
            <ref>display.session</ref>
            <s>Resource</s>
          </invoke>
        </Derivation>
      </Field>
      <Field name='organizations'>
        <Derivation>
          <invoke class='com.waveset.ui.FormUtil'>
            <ref>display.session</ref>
            <s>ObjectGroup</s>
          </invoke>
        </Derivation>
      </Field>
    </Form>
  </Extension>
</Configuration>
```

runForm 要求を使用し、Identity Manager Session API を呼び出すことで、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、searchRequest を使用してそれらの名前をクエリーできます。ただし、この設定では、情報を収集するために 4 つの searchRequests が必要になります。SPML 要求の数を減らすには、単一の runForm 要求を使用してクエリーをフォーム内にコード化したあと、クエリーを実行し、結合された結果を返します。

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 には、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含まれています。

SPML のトレースの詳細については、『Identity Manager Tuning, Troubleshooting, and Error Messages』の「Tracing and Troubleshooting Identity Manager」の章を参照してください。

SPML を実装するためのメソッドの例

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、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

コード例 4-14 に、追加要求の例を示します。

コード例 4-14 追加要求

```
SpmlClient client = new SpmlClient();
   client.setURL("http://example.com:8080/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。

コード例 4-15 認証された SPML 要求

```
SpmlClient client = new SpmlClient();
   client.setURL("http://example.com:8080/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 4-16 LighthouseClient を使用して認証された SPML 要求

```
LighthouseClient client = new LighthouseClient();
   client.setURL("http://example.com:8080/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");
```

これらの例の唯一の違いは、コード例 4-16 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。たとえば、この例を使用して、`Waveset.properties` 内でのプロキシユーザーの設定を回避できます。この結果、プロキシユーザーの代わりに、指定されたユーザーを反映する監査ログを取得できます。

この例では、要求が送信されたときに、`client.setUser` と `client.setPassword` によって認証されます。

検索要求

検索要求の例をコード例 4-17 に示します。

コード例 4-17 検索要求

```
SpmlClient client = new SpmlClient();
   client.setURL("http://example.com:8080/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);
   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 5 章

Identity Manager Web サービスでの SPML 2.0 の使用

この章では、Identity Manager 8.0 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 4 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。

開始する前に

Identity Manager Web サービスの操作を開始する前に、以下の節を確認してください。

対象読者

この章は、アプリケーション開発者および Identity Manager の配備、手続き型ロジックの実装、SPML 2.0 クラスを使用したサービスプロビジョニング要求メッセージのフォーマットや応答メッセージの解析などを担当する開発者を対象としています。

重要な注意点

SPML 2.0 を操作する前に、次の事柄に注意する必要があります。

- Identity Manager Web サービス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 に同梱されている 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。
- SPML 2.0 を実装するときは、スキーマに `spml2ObjectClass` 属性を追加するように設定を変更する必要があります。以前のリリースで提供された `objectclass` 属性値は、現在 `spml2ObjectClass` 属性として維持されています。
- SPML 2.0 を使用して Identity Manager Service Provider (サービスプロバイダ) 機能にアクセスすることはできません。これらの機能は、SPML version 1.0 で使用できます。

関連ドキュメントと Web サイト

SPML の使用方法に関する詳細については、この章で提供される情報のほかに、この節で示すマニュアルおよび Web サイトを参照してください。

推奨ドキュメント

SPML version 1.0 の使用方法の詳細については、本書の第 4 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。

有用な Web サイト

OpenSPML の使用方法を参照したり、SPML 2.0 仕様を読んだり、OpenSPML 2.0 Toolkit をダウンロードしたりするには、次の Web サイトにアクセスしてください。

http://www.openspml.org

概要

この節では、SPML 2.0 に関する以下の一部の基本的な概念について説明します。

- SPML 2.0 と SPML 1.0 の比較
- SPML 2.0 の概念の Identity Manager へのマッピング
- サポートされる SPML 2.0 の機能

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 4 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 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、ユーザーが機能を拡張し、将来新しい機能を追加できるようにする追加の機能 (表 5-1 を参照) が用意されています。

表 5-1 SPML の機能

SPML 1.0	SPML 2.0
追加	追加
Modify	Modify
Delete	Delete
Lookup	Lookup
SchemaRequest	ListTargets
検索	「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 • 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を説明するために、独自の用語が使用されています。

注 OpenSPML 2.0 仕様 (http://www.openspml.org/) を参照してください。

この節では、次の SPML 2.0 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
- PSO
- PSOIdentifier
- オープンコンテンツとオペレーショナル属性

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、233 ページのコード例 5-6 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 Identity Manager は、ユーザーのみを管理し、`spml2ObjectClass` と呼ばれるユーザーの拡張属性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。次に、この属性は、次のフォームを検索するために使用されます。

- 後で追加ターゲットをサポートするために提供される `repoType`。
- Identity Manager ビューとの間で属性をマップするフォーム。

PSOIdentifier

SPML には、*PsoID* と呼ばれるオブジェクト ID が存在します。

OASIS SPML 2.0 仕様は、PSOIdentifiers (PsoID) を要求元 (クライアント) から隠すことを推奨しています。このため、Identity Manager は、システムに PSO を追加するときに、PsoID としてリポジトリ ID (repoID) を使用します。

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、要求が行われたときに、この属性が存在しない場合、SPMLUser などの「デフォルト」objectclass を指定し、使用することができます。内部的には、objectclass の値はユーザー用に spml2ObjectClass 属性として維持されます。Identity Manager では、この属性はユーザー拡張属性である必要があります。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、spml2ObjectClass 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 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 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。

この情報は次の各節で構成されています。

- Password 機能
- Suspend 機能

注

Identity Manager では、Reference 機能、Search 機能、および Updates 機能、または CapabilityData クラスはサポートされません。

サポートされるどの機能でも CapabilityData クラスは使用されません。このため、Identity Manager は、このクラスをサポートしません。CapabilityData クラスはカスタム機能を実装するために使用されます。

OpenSPML 2.0 Toolkit では、整列化、非整列化などで CapabilityData がサポートされます。

コア機能

Identity Manager は、次のコア機能をサポートしています。

表 5-2 コア機能

機能	説明	相違点
AddRequest	指定された PSO をシステムに追加します。	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。	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。
ListTargetsRequest	Identity Manager を通して利用可能なターゲットをリストします。	• 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。また、Waveset.properties を使用することもできます。 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

表 5-2　コア機能 (続き)

機能	説明	相違点
ModifyRequest	指定された PSO 属性を変更します。	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。

- `ListTargetsRequest` 要求に対して、`username` および `password` 値を指定できます。これらの値は、`ListTargetsRequest` 応答で返される `session token` 値によって識別される `session` を確立するための資格情報として使用されます。このセッションは、`session token` 値をオペレーショナル属性として含む、以下のすべての要求のコンテキストです。
  セッションを設定する別の方法は、`soap.username` および `soap.password` 属性を `Waveset.properties` に指定する方法です。この場合は、`session token` は不要です。
- Identity Manager は DSML Profile のみをサポートします。

`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。

次の例は、Identity Manager の `SessionAwareSpml2Client` クラスを通して `ListTargetsRequest` を呼び出す `.jsp` です。

コード例 5-1　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://host:port/idm/servlet/openspml2";
%>

<html>
<head><title>SPML2 Test</title></head>
```

コード例 5-1 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
 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();
 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 5-2 は、送信される SPML 2.0 の要求を示します。

コード例 5-2 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>
```

コード例 5-2 要求 XML の例 (続き)

```
    <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 5-3 は、クライアントに返される SPML 要求の本体を示します。

コード例 5-3 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 5-4 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 5-4 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.*"%>
```

コード例 5-4 クライアントコードの例 (続き)

```
<%@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://host:port/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 (ListTargetsRequest を送信する )
 Response res = client.login("configurator", "password");

%>
<%= res.toString()%>
</body>
</html>
```

コード例 5-5 は、送信される SPML 要求の本体を示します。

コード例 5-5 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 5-6 は、クライアントが受信する (クライアントに返される) SPML 要求の本体を示します。

コード例 5-6　　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'>
    <スキーマ>
      <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'/>
            <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>
    <機能>
      <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

Identity Manager は、表 5-3 で説明した Async 機能をサポートします。

表 5-3　　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	

表 5-3　Async 機能 (続き)

機能	説明	オペレーショナル属性	相違点
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

Identity Manager は、表 5-4 で説明した Batch 機能をサポートします。

表 5-4　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

Identity Manager は、表 5-5 で説明した Bulk 機能をサポートします。

表 5-5　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

Identity Manager は、表 5-6 で説明した Password 機能をサポートします。

表 5-6　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 (1 またはそれ以下) 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 5-7 は ResetPasswordRequest の例です。

コード例 5-7 ResetPasswordRequest の例

```
ResetPasswordRequest rpr = new ResetPasswordRequest();
...
PSOIdentifier psoId = new PSOIdentifier(accountId, null, null);
rpr.setPsoID(psoId);
...
```

SetPasswordRequest の例

コード例 5-8 は SetPasswordRequest の例です。

コード例 5-8 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 5-9 は ValidatePasswordRequest の例です。

コード例 5-9 ValidatePasswordRequest の例

```
ValidatePasswordRequest vpr = new ValidatePasswordRequest();
...
PSOIdentifier psoId = new PSOIdentifier(accountId, null, null);
vpr.setPsoID(psoId);
vpr.setPassword("apassword");
...
```

Suspend 機能

Identity Manager は、表 5-7 で説明した Suspend 機能をサポートします。

表 5-7 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。

SPML 2.0 を使用するための Identity Manager の設定

この節では、SPML 2.0 を Identity Manager で使用するための設定方法について説明します。次のトピックを扱います。

管理する属性の決定

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 `objectclass` 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`
- 資格情報
- `firstname`
- `lastname`
- `emailAddress`

これらの属性をユーザービューにマップする必要があります。

またこの節では、Identity Manager での SPML 2.0 サポートを使用して、PSO の管理方法を説明する、簡単な例も示します。

Identity Manager は、sample/spml2.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。sample/spml2.xml 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。詳細については、このファイルの内容を参照してください。

注 デフォルトでは、spml2ObjectClass 属性は User スキーマに存在しません。この属性がまだ有効になっていない場合、Identity Manager を SPML 2.0 サーバーとして機能させるには、スキーマに spml2ObjectClass 属性を手動で追加する必要があります。

spml2ObjectClass 属性は Identity Manager で提供される schema.xml に定義されていますが、この属性を設定に追加するセクションはコメントアウトされています。本稼働用のスキーマがその元のスキーマから取得されたファイル内にあることを前提として、その元のセクションのコメントを解除し、スキーマファイルをインポートするか、再インポートし、Identity Manager を再起動して SPML 2.0 機能を使用可能にすることができます。

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 設定オブジェクトの設定

sample/spml2.xml ファイルには、SPML 2.0 サポートの初期状態の設定が含まれています。このファイルまたはこのファイルから取得したファイルをインポートして、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することができます。

SPML2 設定タイプ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うことができます。

注 拡張に関する詳細については、242 ページの「システムの拡張」を参照してください。

web.xml の設定

Tomcat などのサーブレットコンテナを使用している場合は、web.xml を使用して、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 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。

web.xml 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使用できます。このウィンドウを使用して、SPML 2.0 メッセージのフローを監視できます。これは、デバッグの際に便利です。

init-param の追加方法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>
```

次の例は、コメント付きのセクションで、その他の init-params についての情報が含まれています。

コード例 5-10 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>
```

コード例 5-10 コメント付きの例 (続き)

```
    </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。
    -->

    <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 トレースの設定

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、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。

SPML のトレースの詳細については、『Identity Manager Tuning, Troubleshooting, and Error Messages』の「Tracing and Troubleshooting Identity Manager」の章を参照してください。

システムの拡張

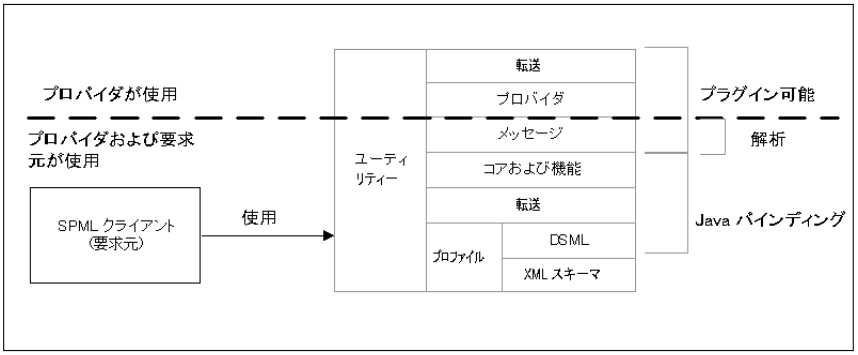
設定オブジェクトを変更することによって、スキーマを拡張します。セクションを変更することにより、要求の executor を追加できます。フォームを使用して、DSML とビューの間でマッピングを行うことができます。

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。次の図は、OpenSPML 2.0 Toolkit のアーキテクチャーを示しています。

図 5-1　　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。このアダプタを出発点として使用し、内容を変更して、Identity Manager 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と通信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 `/REF` に格納されているインストールイメージの 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

注	ビジネスプロセスエディタ (BPE) は非推奨となり、次の Identity Manager リリースでは削除される予定です。代わりに、Identity Manager IDE を使用してください。

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、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、フォームベースでグラフィカルに表示します。

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。

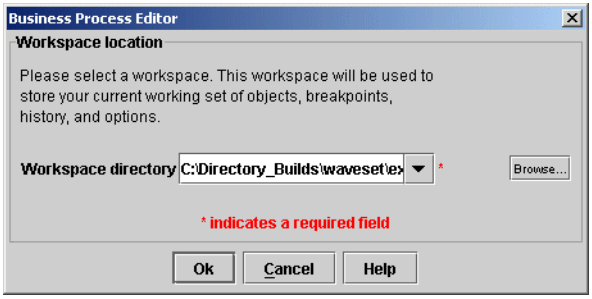
   ```
   set WSHOME=<Path_to_idm_directory>
   set JAVA_HOME=<path_to_jdk>
   ```

   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。

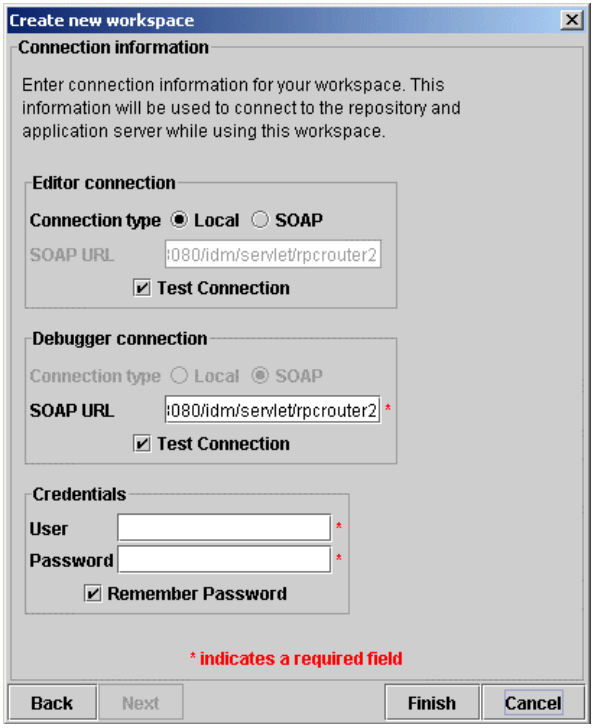
1. 「Workspace location」ダイアログで、新規ワークスペースの一意の名前を「Workspace Directory」フィールドに入力して「OK」をクリックします。

   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://*host*:*port*/*idm*/servlet/rpcrouter2」と入力します。ここで、<*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://*host*:*port*/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。

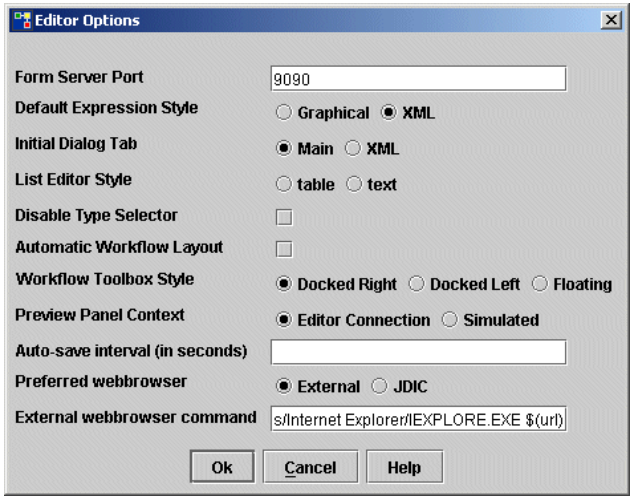
JDIC の有効化

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ません。
- 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

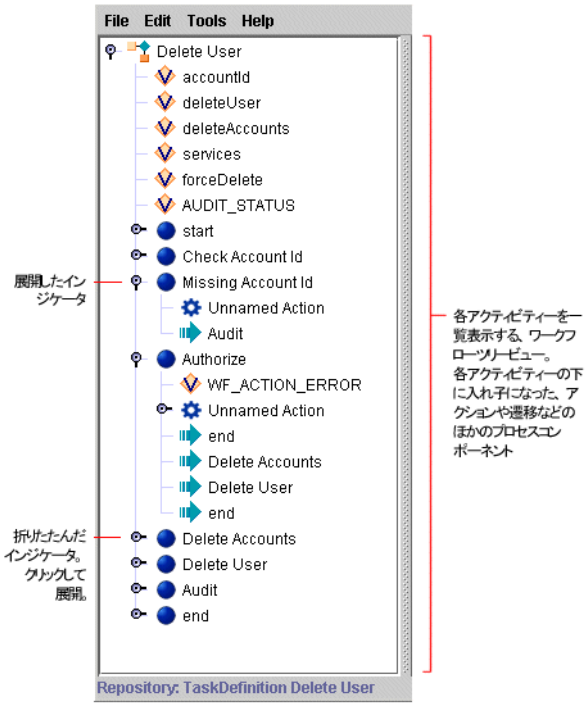
BPE のインタフェースには、メニューバーと、選択のためのダイアログが含まれています。主な表示は 2 つのメイン区画に分かれています。

- 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。

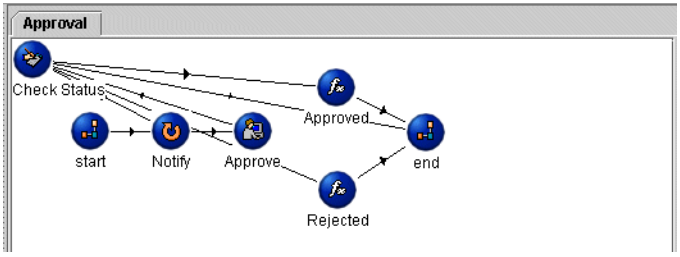
これらのビューについては、次の節で説明します。

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、261 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。

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。

   「Select objects to edit」ダイアログが表示され、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
   - 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。

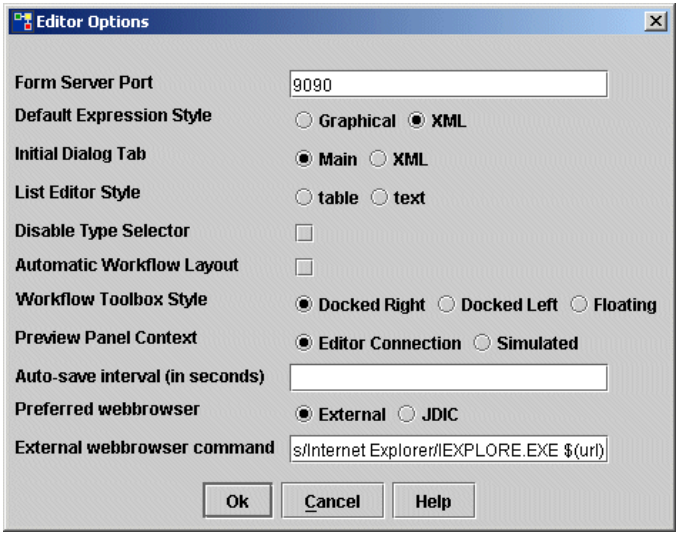
3. オブジェクトタイプをダブルクリックすると、そのタイプに対して表示アクセス権のあるオブジェクトがすべて表示されます。
4. プロセスまたはオブジェクトを選択して「OK」をクリックします。

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-7　「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 秒です。
- 「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。

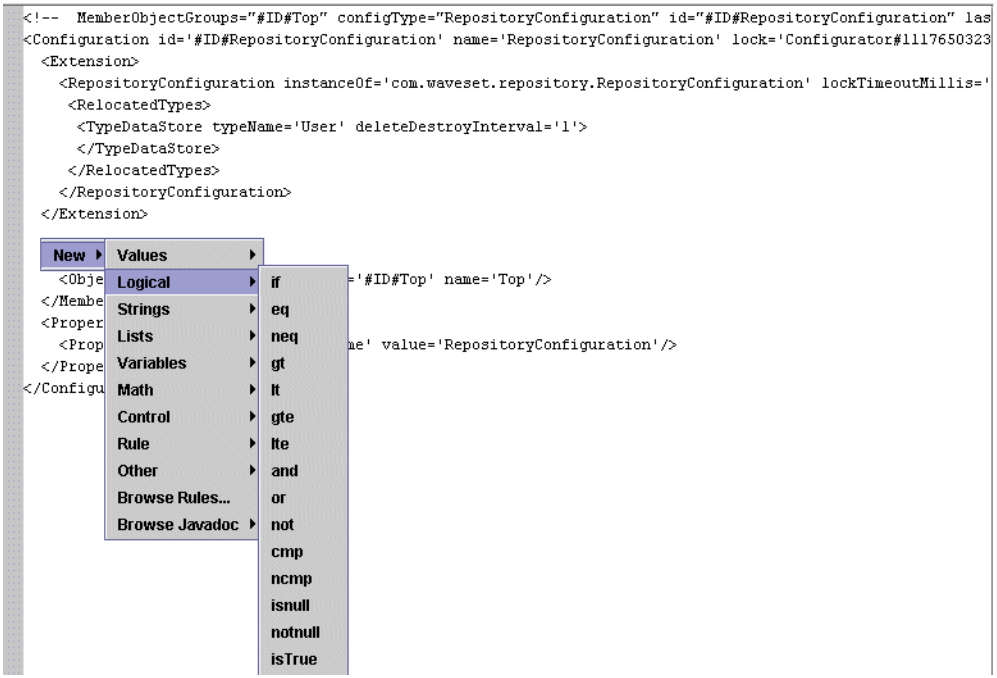
XPRESS の挿入

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。

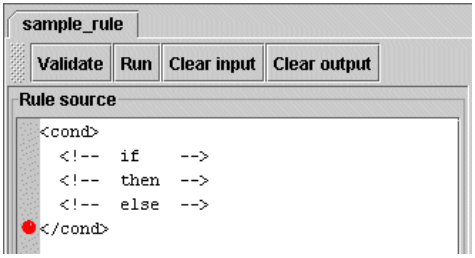
図 A-8　　XML への XPRESS 関数挿入メニュー



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、「ワークフローリビジョンの検証」を参照してください。

図 A-9　　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

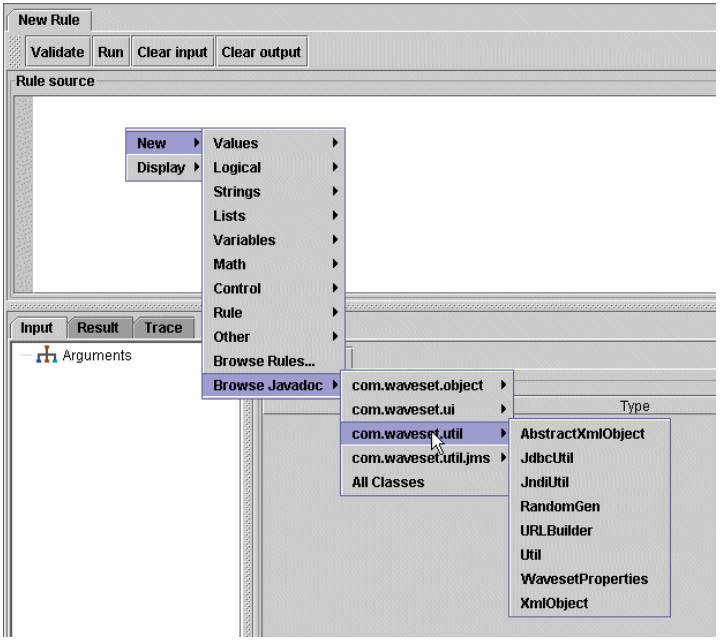
キーボートコマンド / キー	アクション
Ctrl-C	コピー
Ctrl-O	開く (リポジトリオブジェクト)
Ctrl-R	ソースの更新
Ctrl-S	保存 (リポジトリオブジェクト)
Ctrl-V	貼り付け
Ctrl-X	切り取り
削除	削除
F5	現在の行の選択
F6	ステップアウト
F7	ステップイン
F8	ステップオーバー
F9	続行 (デバッグ)

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-10　　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。

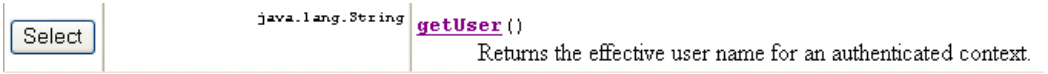
- 「All Classes」: Javadoc クラスのフレームビューを表示します。ブラウザ内で、このビューから各クラスの JavaDoc にジャンプできます。

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-11　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。

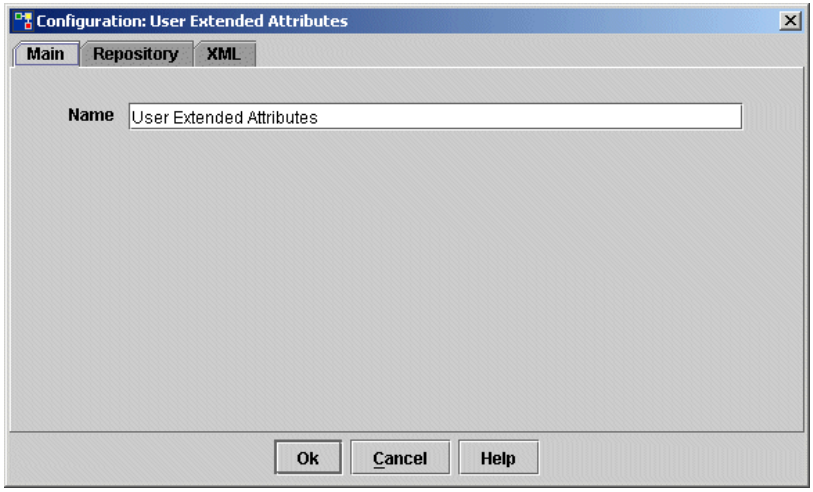
BPE では、次の図に示すように、これらの各種設定オブジェクトのリストが左区画のツリービューに表示されます。

図 A-12　　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。

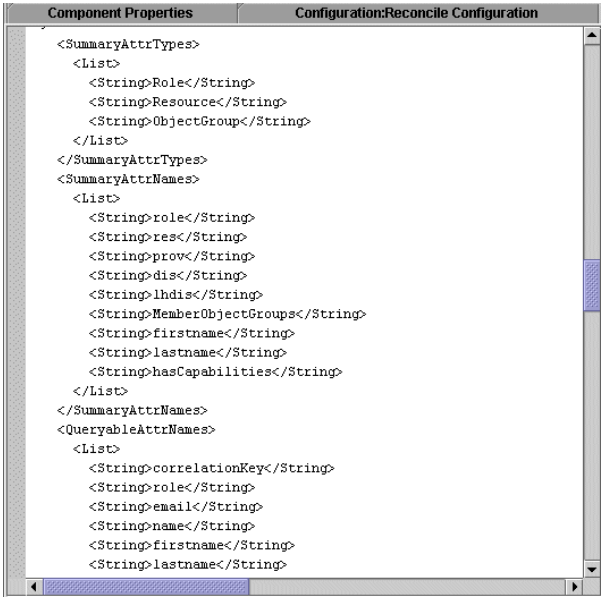
たとえば、ツリービューで「User Extended Attributes」をダブルクリックすると、次のダイアログが表示されます。

図 A-13　　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。

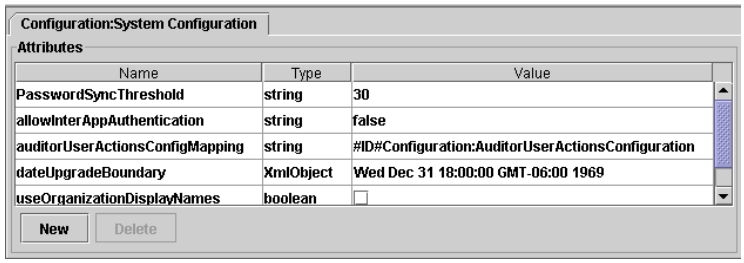
図 A-14　BPE での調整設定オブジェクトの XML 表示



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-15　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。

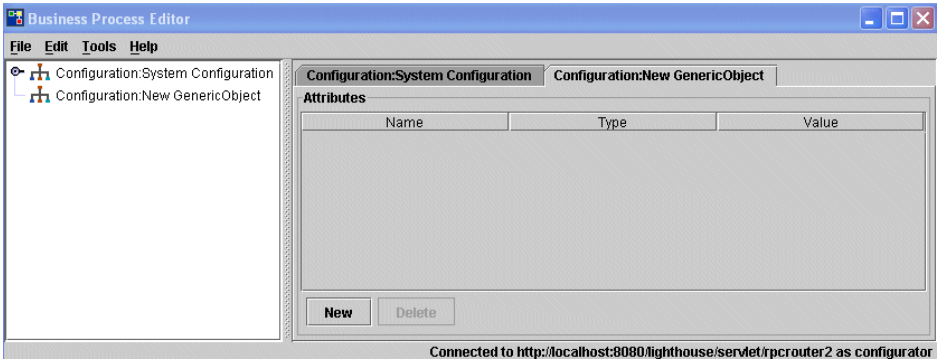
   「Configuration:New GenericObject」または「Configuration:New Configuration」ダイアログが開き、メインパネルが表示されます。

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。

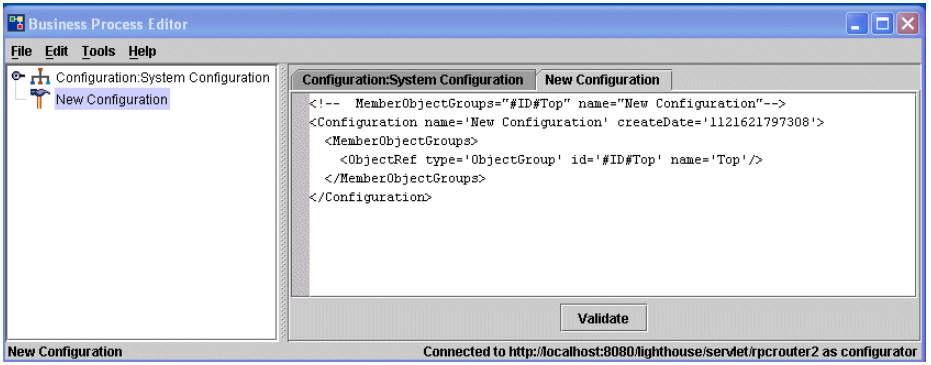
   - 汎用オブジェクトを作成した場合は、空の「Attributes」区画が次のように表示されます。

図 A-16　　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-17　　BPE での新規設定オブジェクトの表示

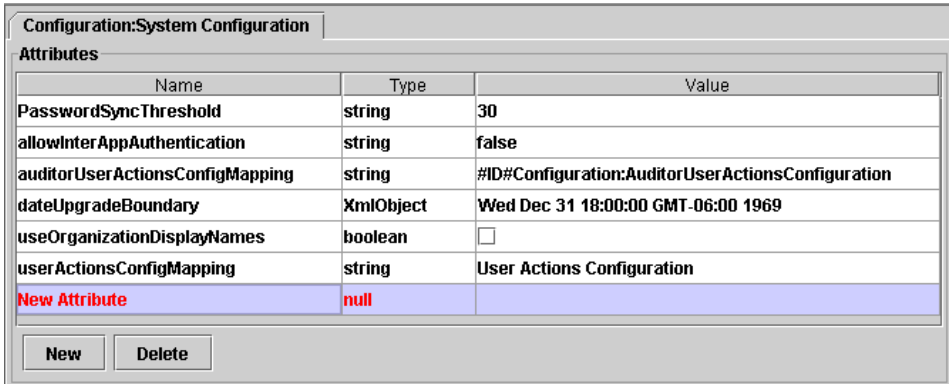


3. 汎用オブジェクトを作成する場合は、次のように属性を追加し、必要に応じて手順を繰り返します。

   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 Attributes」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-18　　BPE での汎用オブジェクト新規属性の表示



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、246 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

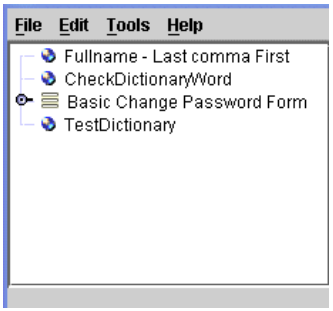
規則を操作するとき、BPE のインタフェースには次の表示区画があります。

- 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-19　　ツリービューでの規則表示



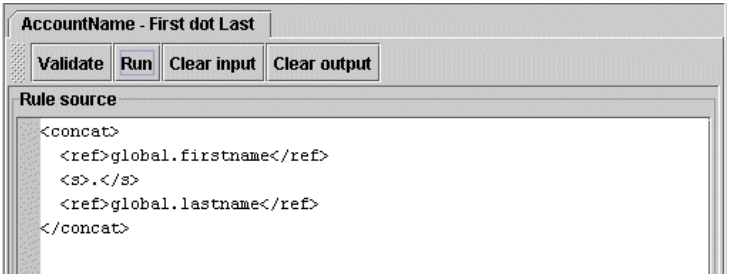
一般に、ツリービューにはタスク、フォーム、またはビューの階層が表示されます。階層では各要素が順番に表示され、親要素の下にサブ要素が入れ子にされます。

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-20　　「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

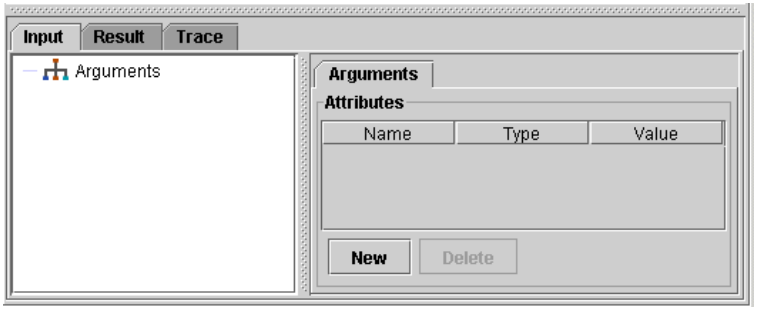
この区画の上にあるボタンを使用して、次の操作を実行することもできます。

- 「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-21　「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 (266 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

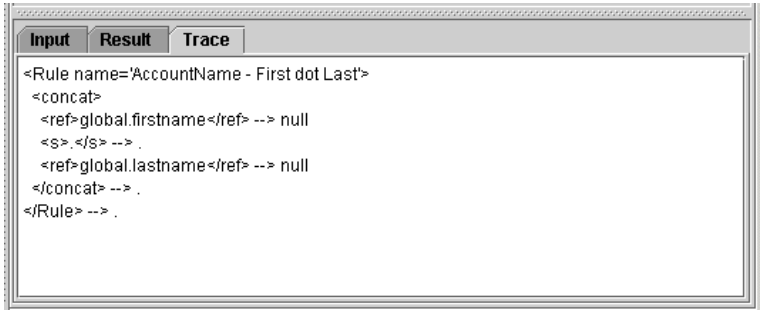
「Result」タブを選択し、「Rule source」区画の上にある「Run」をクリックすると、選択した規則を実行できます。「Result」タブ区画には、規則の戻り値が XML 形式で表示されます。

図 A-22　「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-23　「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。

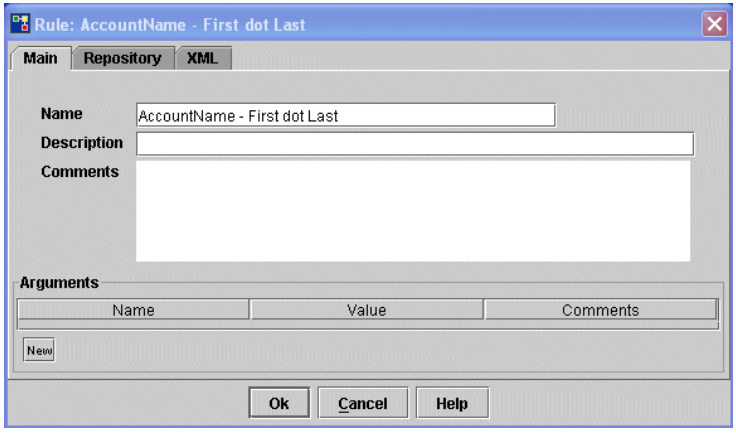
ツリービューまたはダイアグラムビューで項目を選択して右クリックすると、その項目に対して実行できる操作がメニュー項目として表示されます。

「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れています。

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-24　「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。

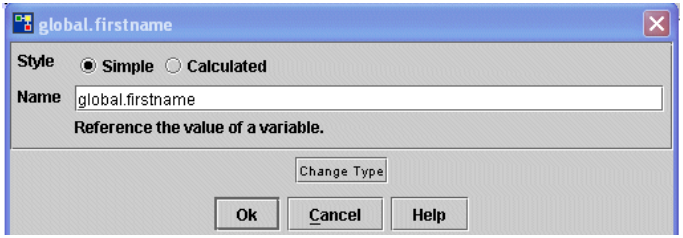
- 「Name」: 選択した規則の名前が自動的に表示されます。これは、Identity Manager のインタフェースに表示される名前です。
- 「Description」(省略可能): 規則の目的を説明するテキストを指定します。
- 「Comment」: <Comment> 要素を使用して規則の本体に挿入されるテキストを指定します。
- 「Arguments」: 必要な引数を指定します。

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-25　「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

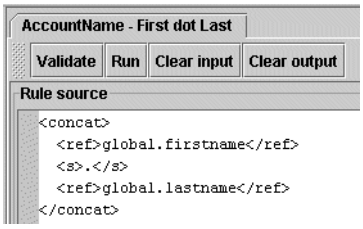
ダイアグラムビューでの情報表示形式を変更するには、次の手順に従います。

1. 右クリックして操作メニューを表示します。
2. 「Display」> <*view_type*> 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。

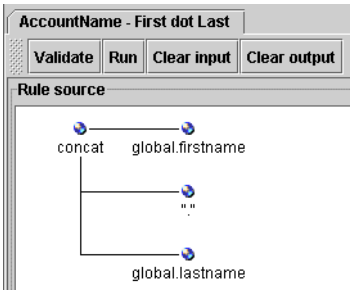
図 A-26　XML 表示



- 「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-27 には選択した規則の一部のみを示しています。

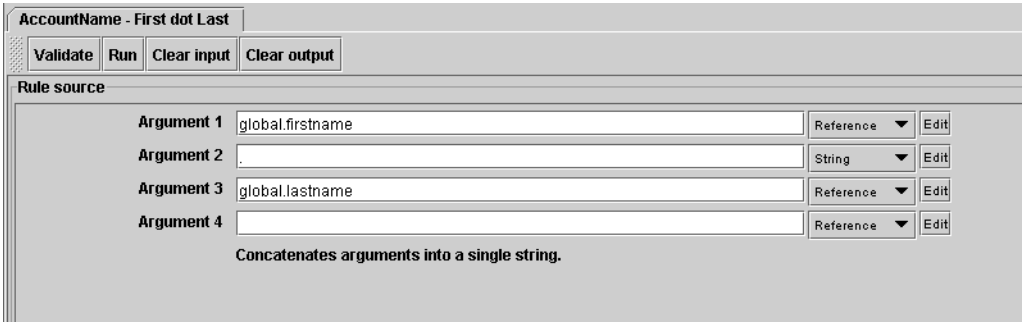
図 A-27 グラフィカル表示



- 「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。

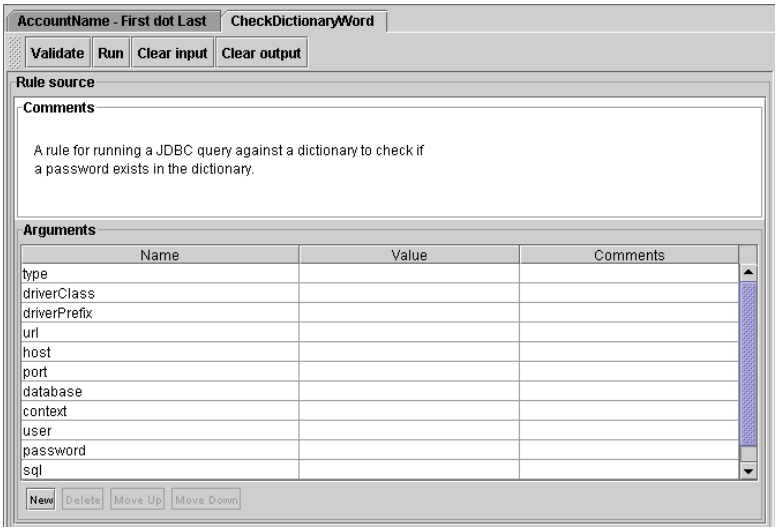
  その他のプロパティーについては、別のダイアログを開くことが必要な場合があります。
  新しい式を作成するときは、「Property Sheet」表示タイプを使用すると効率的に作業できます。このビューでは、グラフィカルビューを使用する場合と比べて、式の引数をすばやく入力できます。

図 A-28 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-29 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-29　　設定表示



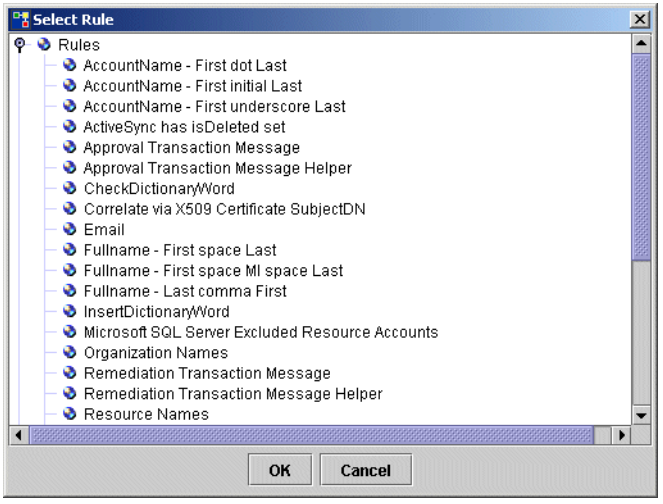
規則の参照

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-30) 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-30　「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。

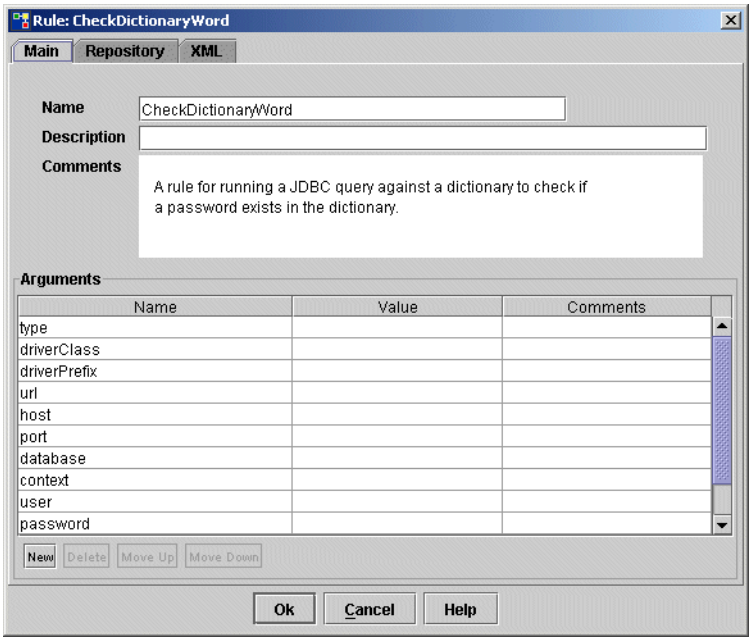
- Main
- Repository
- XML

「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-31 を参照) と同じ情報が表示されます。

図 A-31　「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-32　「Repository」タブ表示

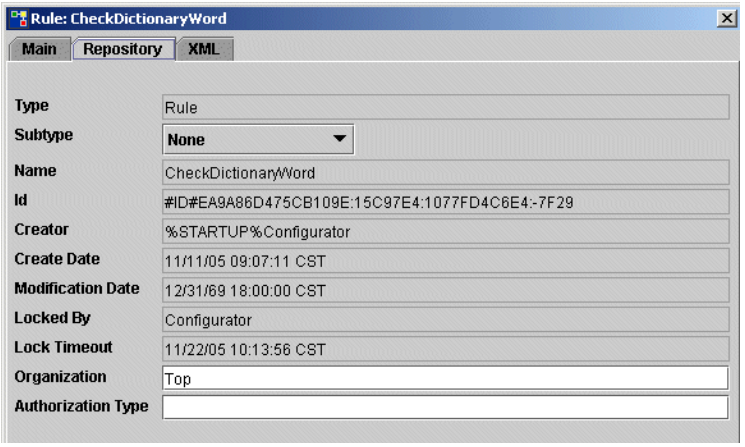


表 A-2　「Repository」タブのフィールド

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「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。

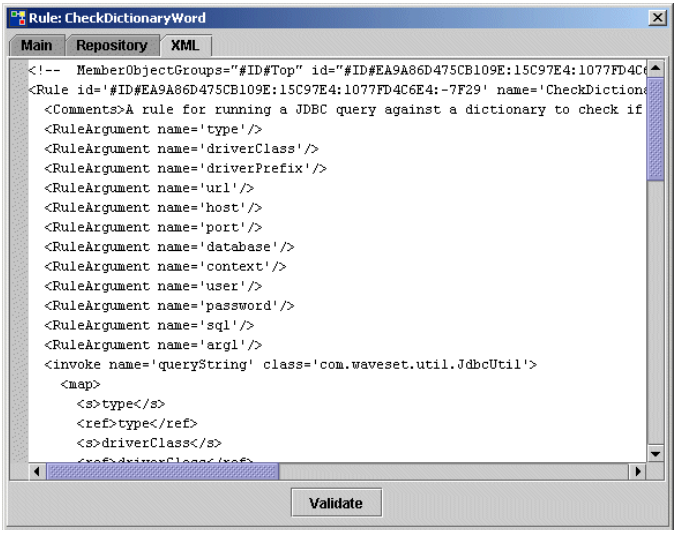
- 「Organization」: 新しい組織割り当てをテキストフィールドに入力します。
- 「Authorization Type」: 新しい認証タイプをテキストフィールドに入力します。

注　規則のサブタイプの例については、47 ページの「ExcludedAccountsRule」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-33　「XML」タブ表示

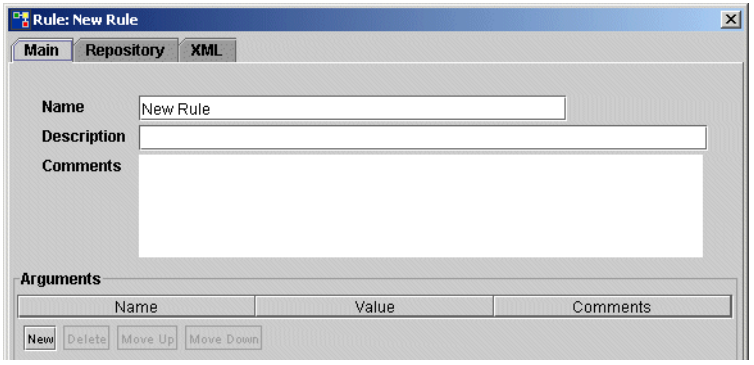


新しい規則の作成

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-34　「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。

- **「Argument」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。

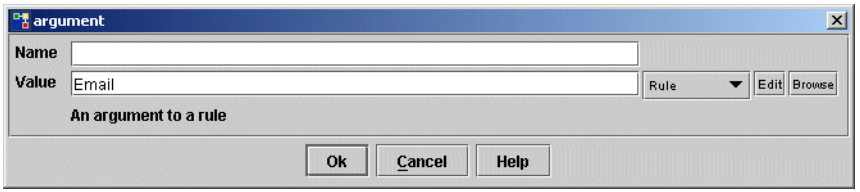
以降の節では、これらの各ダイアログについて詳しい情報を示します。

注 規則の構造の詳細は、13 ページの「規則構文について」を参照してください。

「Argument」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。

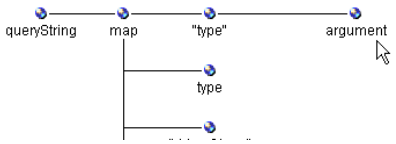
図 A-35　「Argument」ダイアログ



「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-36　引数ノードのダブルクリック



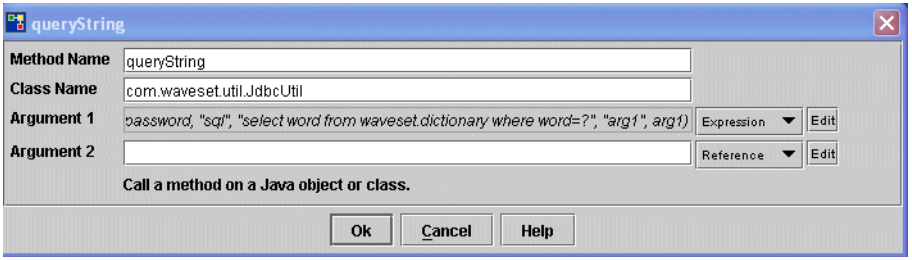
「Argument」ダイアログには、次の基本オプションがあります。

- 「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-37　「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-38　「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。

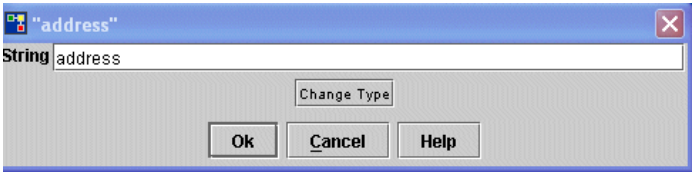
表 A-3　有効な引数の型 (続き)

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-39　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、<element_type> 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

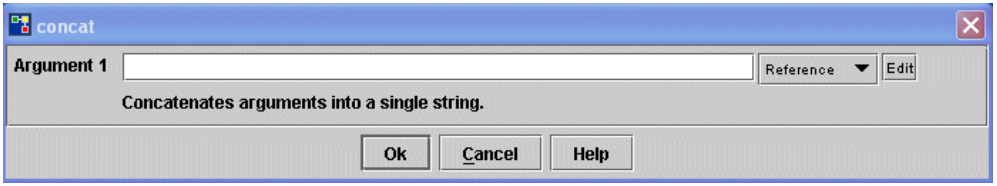
メニューオプション	XPRESS 関数 / 呼び出し可能な追加操作 ...
Values	`string`、`integer`、`list`、`map`、`message`、`null`
Logical	`if`、`eq`、`neq`、`gt`、`lt`、`gte`、`lte`、`and`、`or`、`not`、`cmp`、`ncmp`、`isnull`、`notnull`、`isTrue`、`isFalse`
String	`concat`、`substr`、`upcase`、`downcase`、`indexOf`、`match`、`length`、`split`、`trim`、`ltrim`、`rtrim`、`ztrim`、`pad`
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`
規則	• 新しい規則を作成する • 引数を作成する
Other (functions, object access, diagnostics)	その他のオプションを表示する : • 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-40　「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。

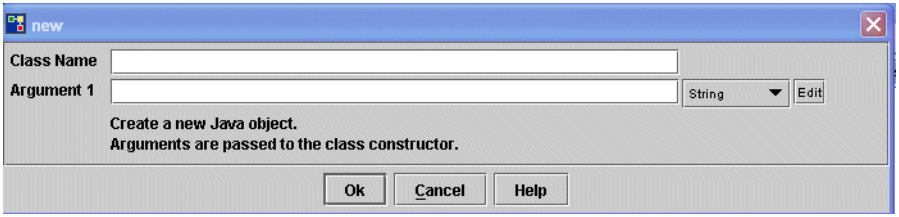
操作オプションとしては、表 A-5 で説明されているオプションのいずれかを選択できます。

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-41　「new」ダイアログ

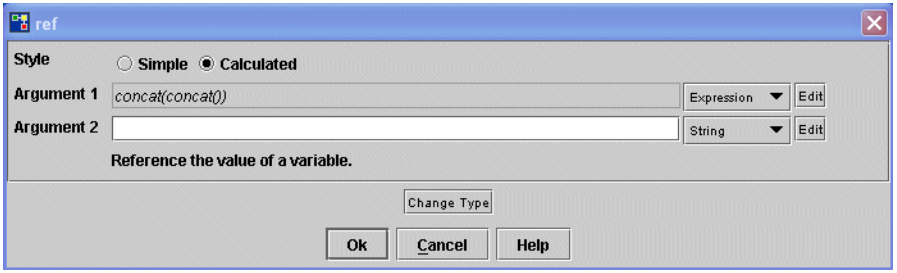


要素詳細の編集

「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-42 は、`ref` 文の変数ウィンドウを示します。

図 A-42　「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。
トレース	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

   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 *Filename*.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

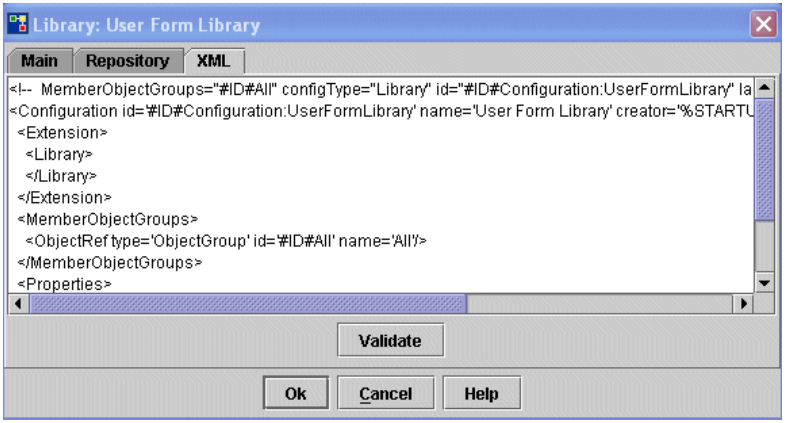
表示または編集する規則ライブラリを選択するには、次の手順に従います。

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-43　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

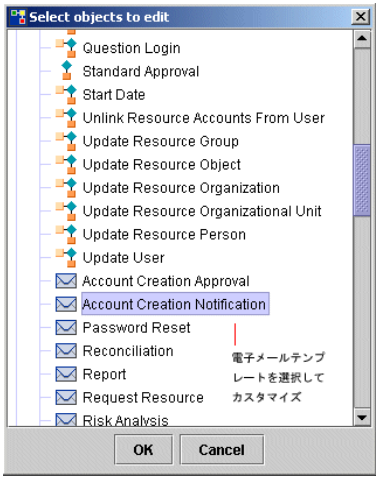
- この例で示す画面例は、プロセスを読み込むときのものとは多少異なる場合があります。結果として、コンポーネントの位置が違っていたり、プロセスの操作の一部が省略されていたりする場合がありますが、この例の目的にとって重要な違いではありません。
- 多くのタスクはツリービューまたはダイアグラムビューから実行できますが、この例では主に BPE のツリービューを使用します。

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-44) 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-44 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。

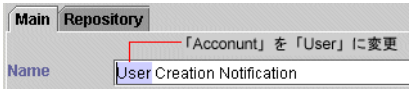
   電子メールテンプレートのコピーがリストビューに表示されます。

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-45　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-46　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

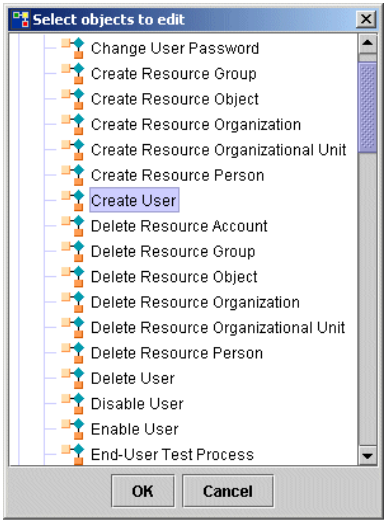
次の手順に従って、新しい電子メールテンプレートを使用するように「Create User」ワークフロープロセスを変更します。

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。

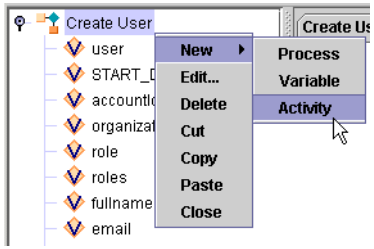
図 A-47　　ワークフロープロセスのロード



「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-48　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。

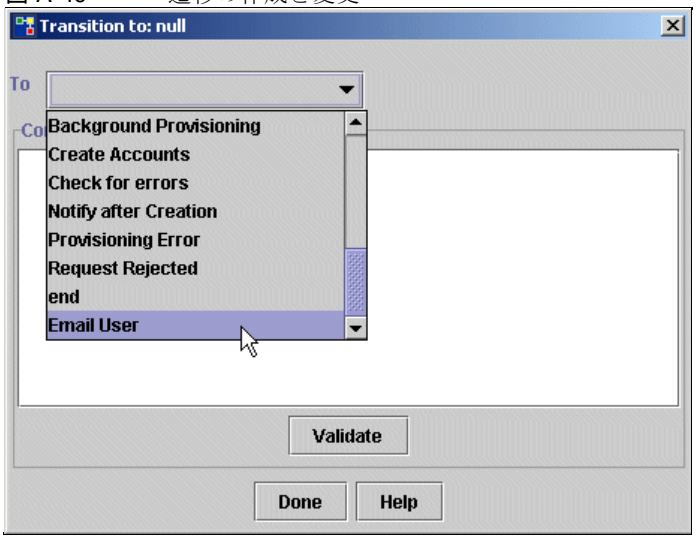
4. activity1 をダブルクリックして「Activity」ダイアログを開きます。
5. 「Name」フィールドに「**Email User**」と入力して、アクティビティー名を変更します。

   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。

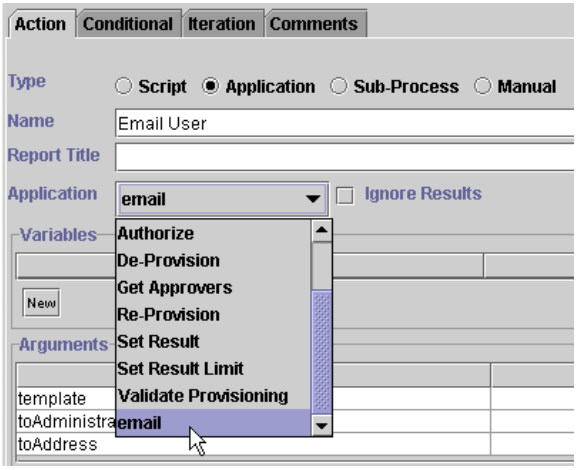
図 A-49　遷移の作成と変更



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。

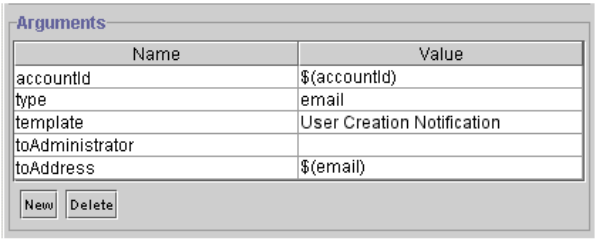
15. 「Application」メニューから「**email**」を選択します。

図 A-50　　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-51　　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
- デバッガのメインウィンドウの使用
- 実行プロセスのステップスルー
- はじめに
- ワークフローのデバッグ
- フォームのデバッグ

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、312 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` プロパティーを無効にする必要があります。

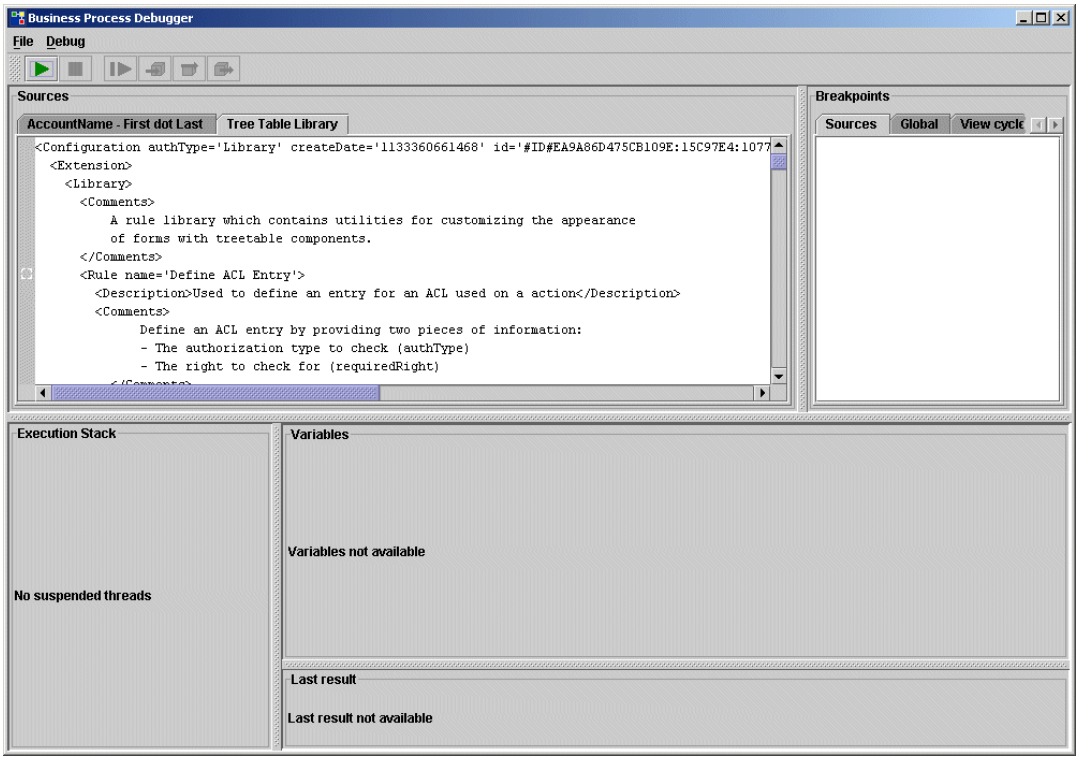
デバッガを無効にするには、システム設定オブジェクトの `serverSettings.<server>.debugger.enabled` プロパティーを `false` に設定します。

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-52　　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、261 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

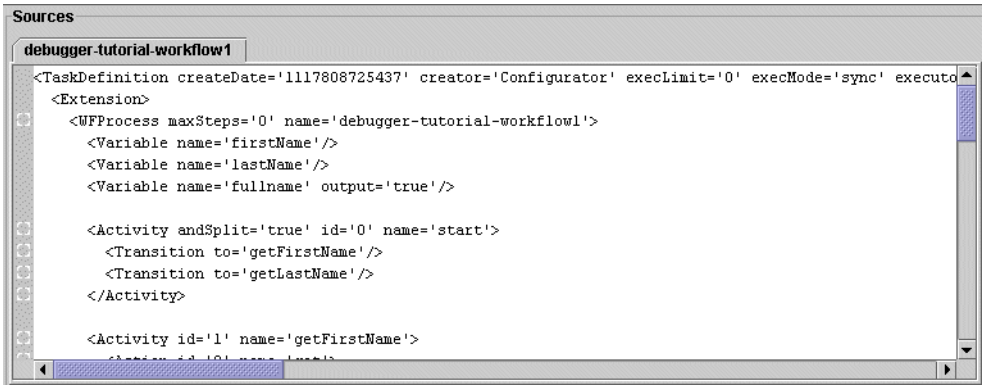
- 「Last Result Not Available」領域
- ブレークポイントの設定

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-53　　BPE デバッガのメインウィンドウの「Source」パネル

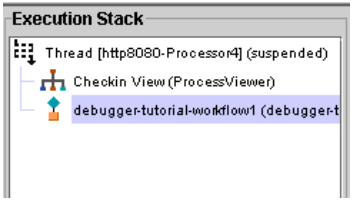


実行スタック

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。

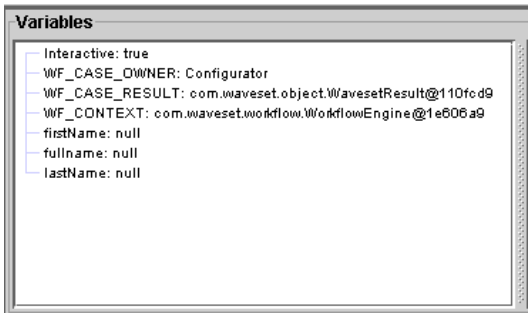
図 A-54　　BPE デバッガのメインウィンドウの「Execution Stack」パネル



「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-55　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-56　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。

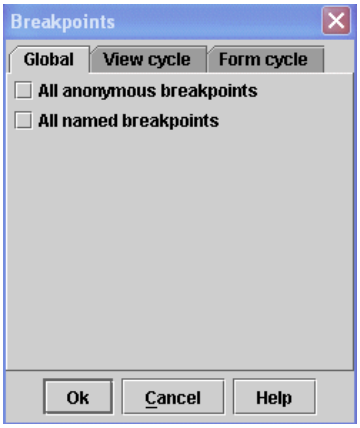
    ブレークポイントは、ステップオーバーおよびステップアウト機能よりも優先されます。その結果、この設定を有効にした場合、実質的にはステップオーバーおよびステップアウト処理をステップイン処理に変えたことになります。一般的な用途では、「All named breakpoints」は、特定のページがどの

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。

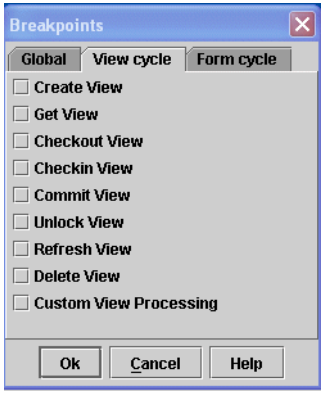
(省略可能) グローバル設定を選択して「OK」をクリックします。

図 A-57　BPE デバッガの「Breakpoints」パネル : 「Global」タブ



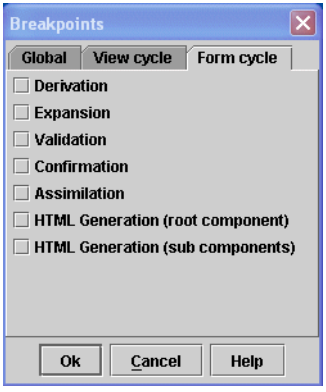
- プロセス実行中に発生するビュー処理に基づいて、コードにブレークポイントを設定するには、「View cycle」タブを選択します。このダイアログには、最も頻繁に呼び出されるビュー操作が一覧表示されます。一覧表示されたそれぞれのビュー操作は、各ビューで使用可能です。

図 A-58　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-59　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。ステップインは必ず、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-60　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。

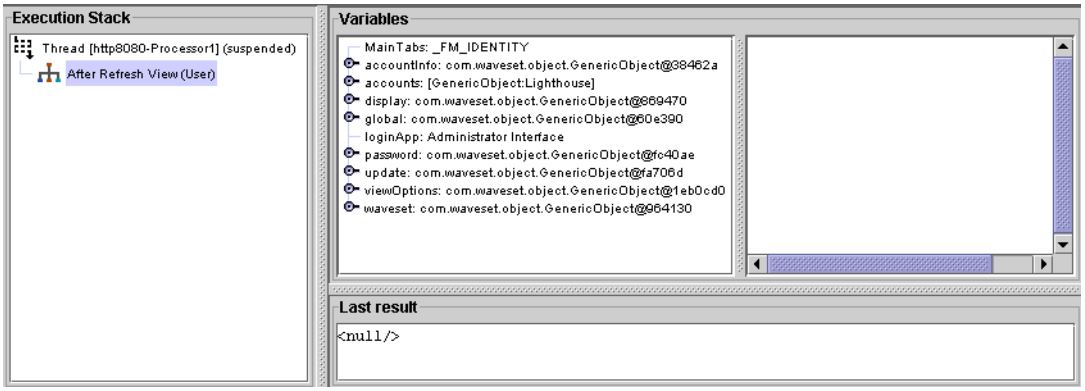
「After Refresh View」結果の表示

「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-61　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。

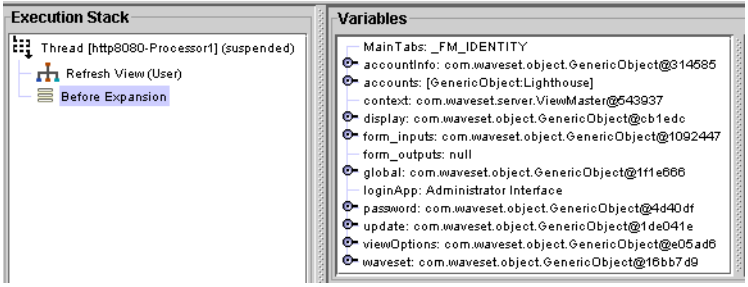
フォームのステップスルー

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-62　例 1:「Before First Expansion」パスでの中断のデバッグ



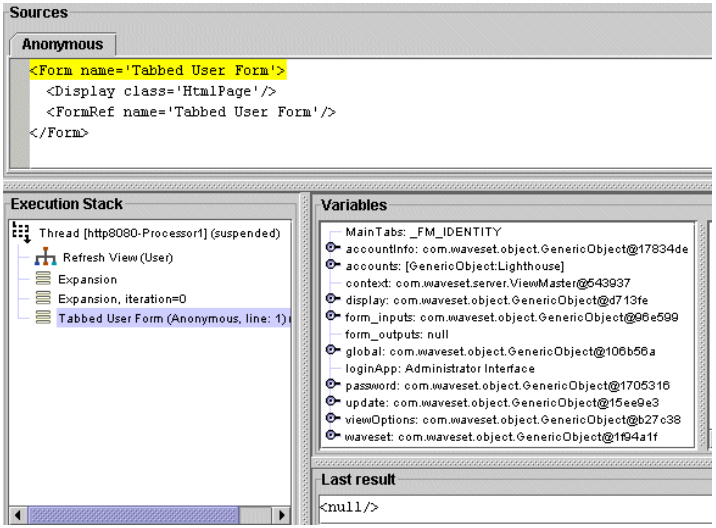
2. 「step-into」をもう一度クリックします。

   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-63 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。

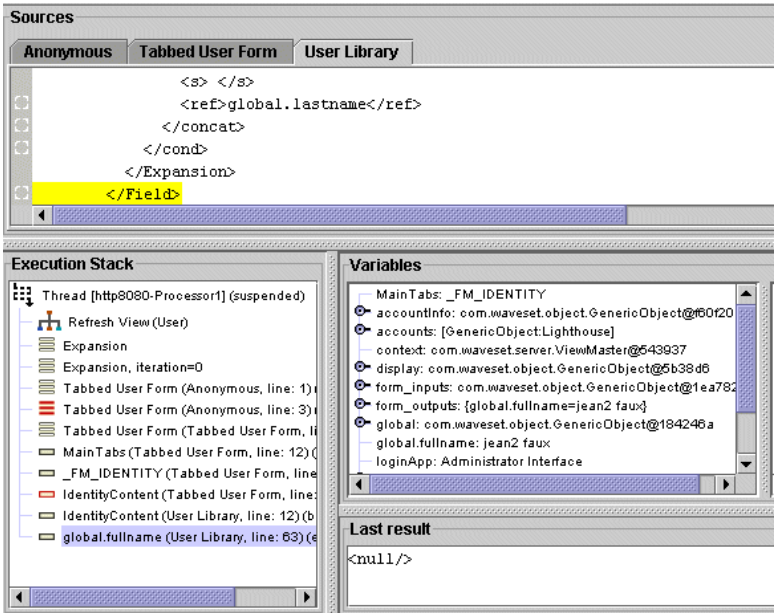
5. 「`<Field name='global.fullName'>`」に達するまで「step-into」をクリックし続けます (約 20 ～ 30 回のステップイン操作)。

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-64　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します。
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、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をデバッグする

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

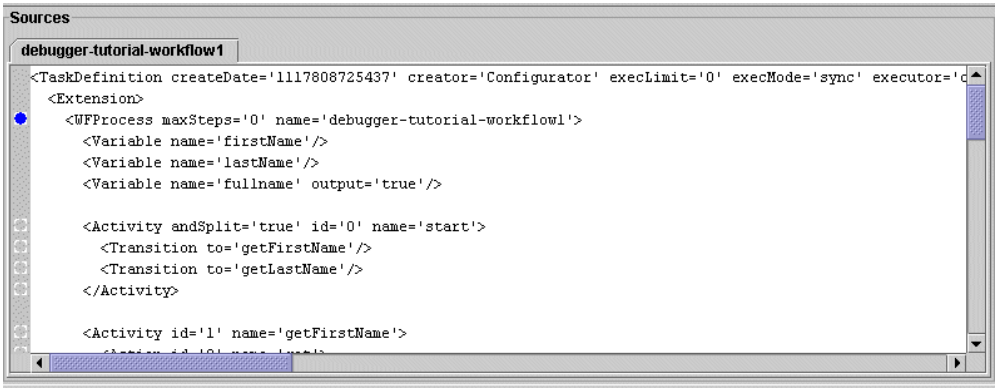
ステップ 1: プロセスの起動

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-65　　最初のブレークポイントの設定

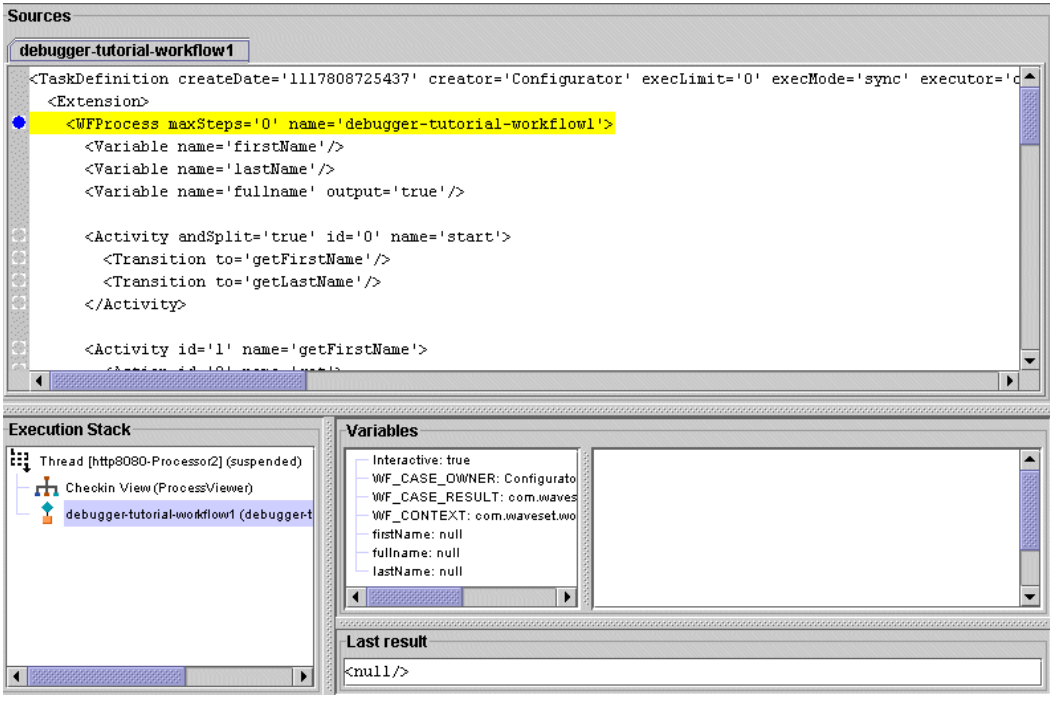


3. <WFProcess> タグのすぐ近くにあるボックスをクリックして、ワークフローの開始位置にブレークポイントを設定します。
4. Identity Manager にログインし、「Tasks」>「Run Tasks」の順に選択します。

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-66　　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です。

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。

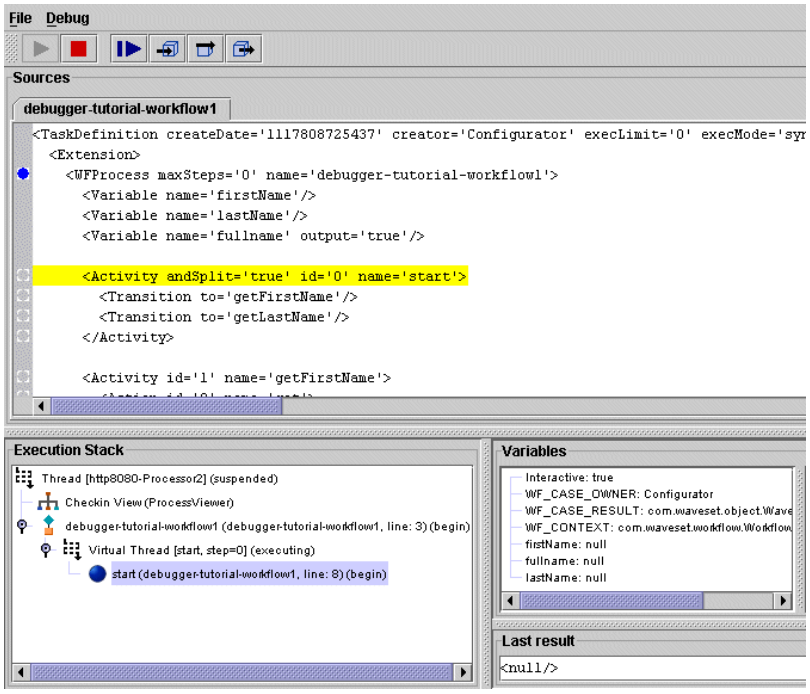
ステップ 2: 実行の開始

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-67　　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。

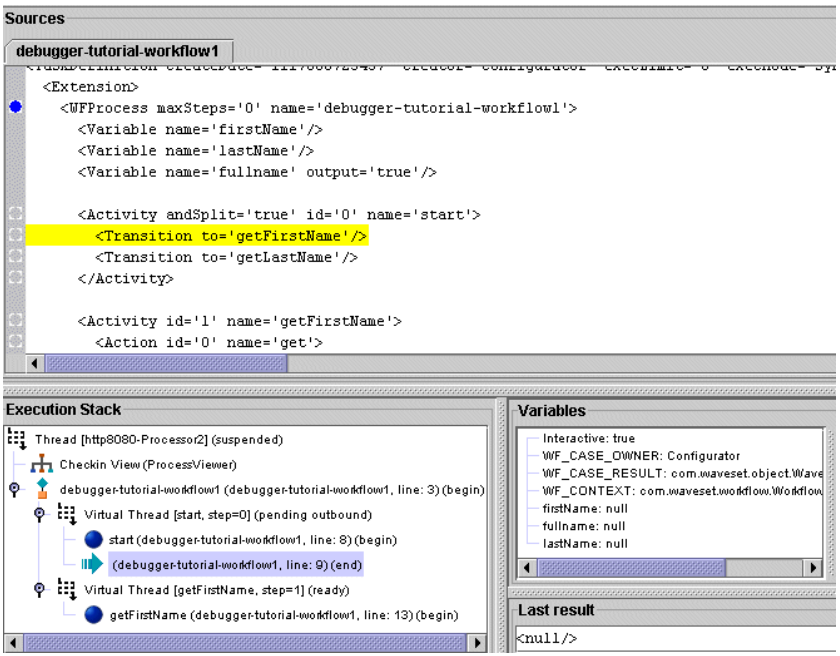
1. 「step-into」をクリックします。

   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ります。

図 A-68　　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。

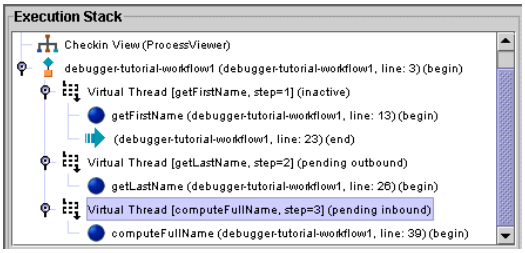
7. 「step-into」をクリックします。

   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」になります。

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-69　　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。

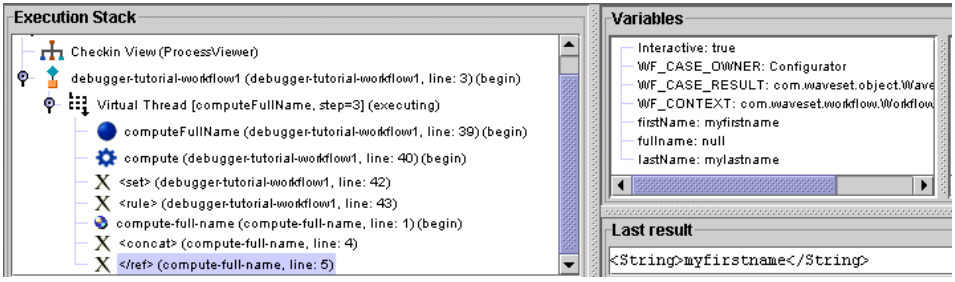
2. 「step-into」をクリックします。

   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-70　　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。

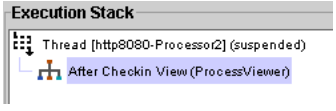
7. 「step-into」をあと 4 回クリックします。`end` の状態が、準備完了、実行中、と順に変化します。

   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-71　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

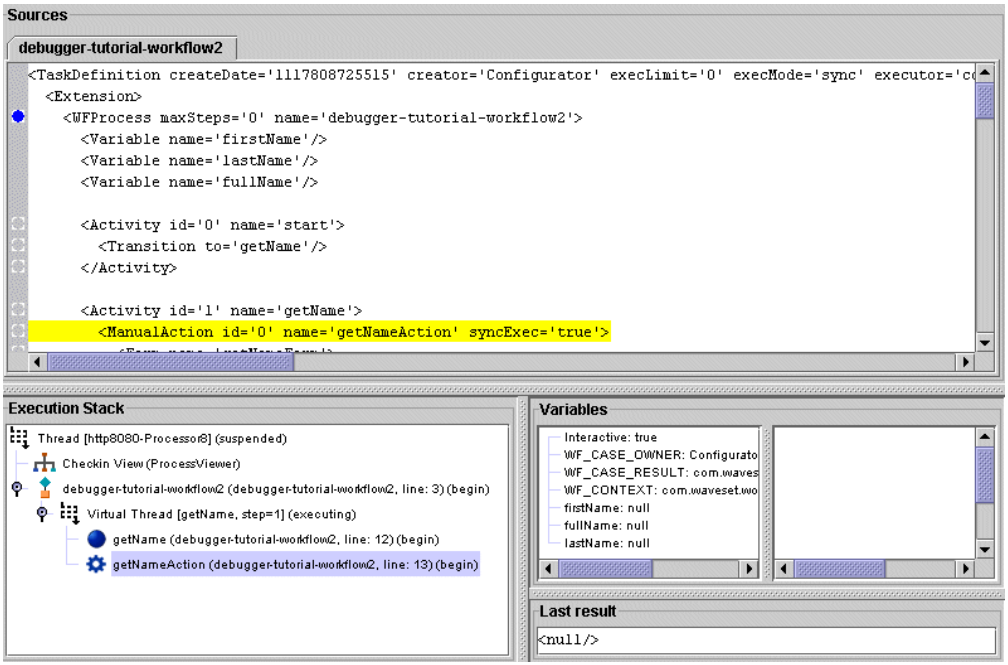
ここで示す例では、手動アクションとフォームを含む、サンプルワークフローのデバッグ方法を説明します。

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。

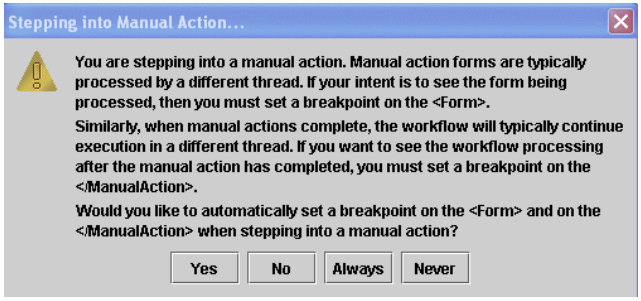
   設定したブレークポイントでデバッガが停止します。
6. 「step-into」を 6 回、つまり、デバッガが「`<ManualAction... name='getNameAction'>)`」に達するまでクリックします。

図 A-72　　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。

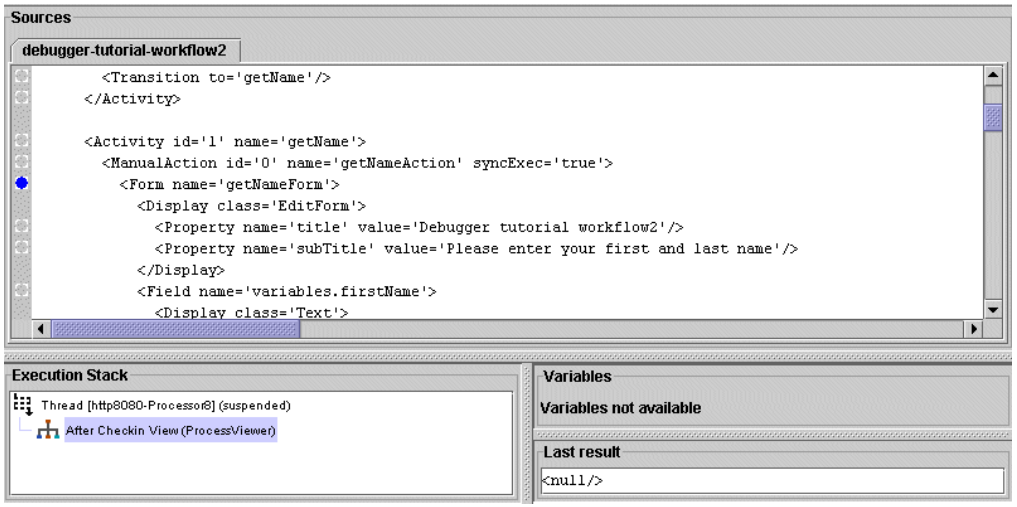
図 A-73　　「Stepping Into Manual Action」ダイアログ



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。

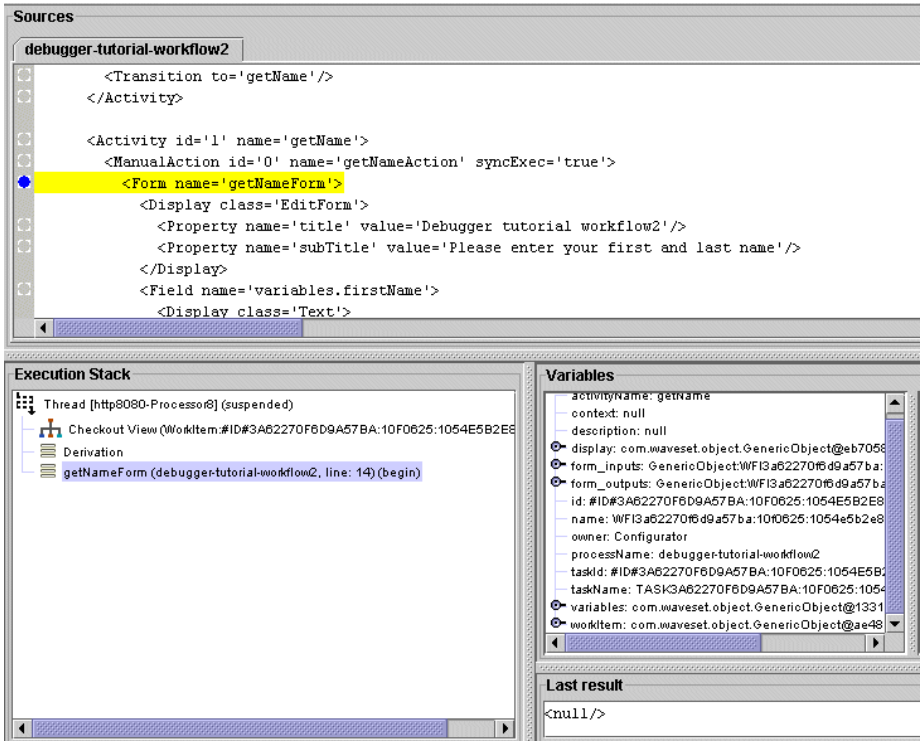
   図 A-74　　フォームの開始を示すブレークポイント



   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-75　　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。

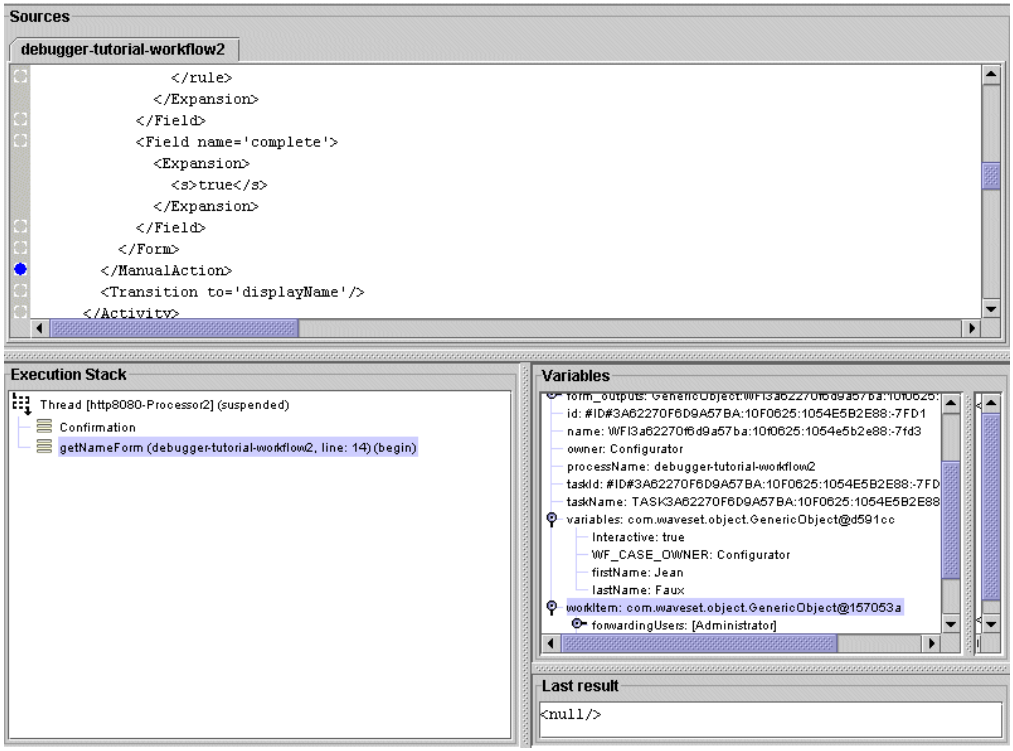
3. ブラウザウィンドウに戻り、入力を求められたら姓と名を入力して「Save」をクリックします。

   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-76　　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。

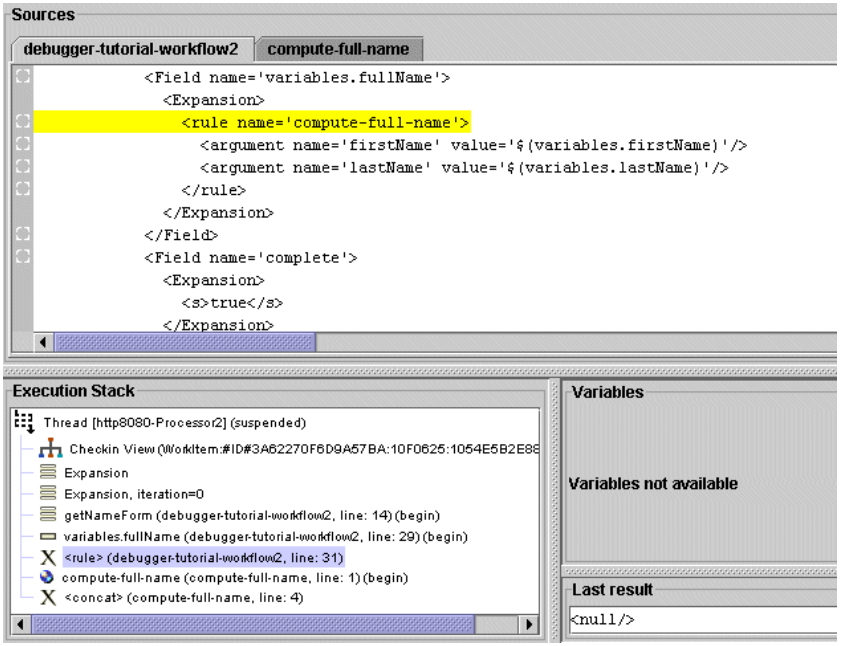
1. 「Continue」をクリックして検証フェーズをスキップします。

   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-77　規則処理へのステップイン



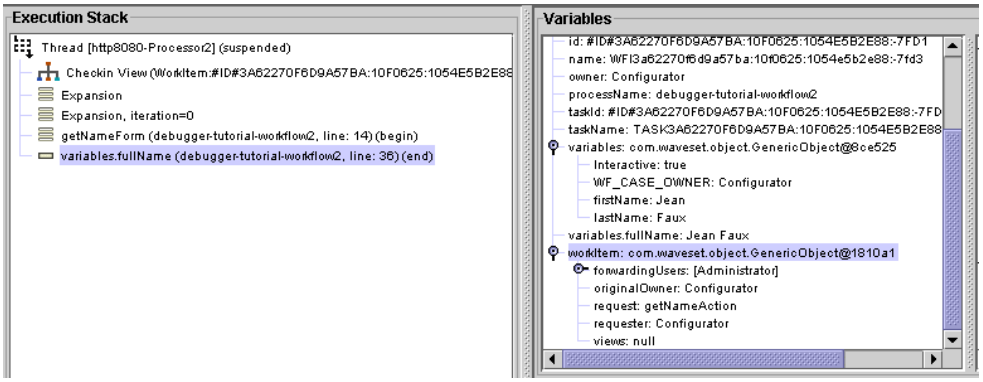
3. 「step-into」を 5 回クリックします。デバッガは <rule> 要素にステップインした状態になります。
4. 「step-into」を 7 回、またはデバッガが </Rule> 要素に達するまでクリックします。

   「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-78　　デバッガでの variable.fullName の実行完了の表示

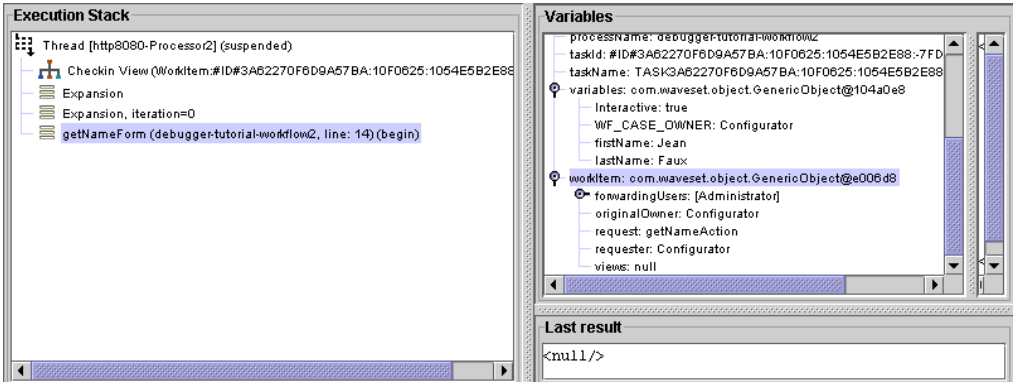


一般に、form_inputs はビューを特定し、form_outputs にはビューに同化されるデータが含まれます。ただし、Active Sync フォームのように、必ずしもすべてのフォームがビューに結び付けられるわけではありません。フォームエンジンは一般的なデータマッピングエンジンであり、フォーム入力からフォーム出力へのマッピングを行います。ビューハンドラは、フォームエンジンにビューを渡す処理と、出力をビューに戻して反映する処理を受け持ちます。

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-79　　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」をクリックします。
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 :

    実行が再開されたため、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。
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 :

実行が再開されたため、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。

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パスの名前があります。

Derivation

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。

Expansion

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

U

V

W

X

あ

い

え

お

か

き

く

け

こ

さ

し

す

せ

そ

ち

つ

て

と

な

に

ね

は

ひ

ふ

へ

ほ

ま

む

め

ゆ

よ

ら

り

れ

ろ

わ

わ