Novell
Nsure™ Identity Manager

2

www.novell.com

2004 年 1 月 15 日

POLICY BUILDER とドライバカスタマイズ ガイド

Novell®

法的通知

米国 Novell, Inc.
1800 South Novell Place
Provo, UT 84606
米国

www.novell.com

Policy Builder とドライバカスタマイズガイド
2004 年 1 月 15 日

オンラインドキュメント： この製品およびその他の Novell 製品のオンラインマニュアルや更新情報については、www.novell.com/documentation を参照してください。

Novell の商標

DirXML は、Novell, Inc. の米国およびその他の国における登録商標です。
eDirectory は、米国 Novell, Inc. の商標です。
Novell は、米国 Novell Inc. の米国ならびに他の国における登録商標です。
Nsure は米国 Novell, Inc. の商標です。

Third-Party Materials

すべてのサードパーティの商標は、それぞれの所有者に帰属します。

目次

このガイドについて

Novell® Nsure™ Identity Manager 2 は、DirXML® を使用するデータ共有および同期サービスで、アプリケーション、ディレクトリ、およびデータベース間で情報を共有できます。このサービスは、分散された情報をリンクし、識別情報の変更時に指定システムを自動的に更新するポリシーを設定できます。

Identity Manager は、アカウントプロビジョニング、セキュリティ、シングルサインオン、ユーザセルフサービス、認証、許可、自動化されたワークフロー、および Web サービスの基盤を提供します。Identify Manager を使用すると、分散された識別情報を統合、管理、および制御できるため、適切なユーザに適切なリソースを安全に提供できます。

このガイドでは、Policy Builder および Identity Manager 2 のドライバ設定について詳しく説明します。

追加のドキュメント

DirXML ドライバの使用に関するマニュアルについては、DirXML マニュアルの Web サイト (http://www.novell.com/documentation/lg/dirxmldrivers/index.html) を参照してください。

Identity Manager 2.0 のマニュアルについては、DirXML マニュアルの Web サイト (http://www.novell.com/documentation/lg/dirxml20/index.html) を参照してください。

最新のマニュアル

このマニュアルの最新版については、DirXML マニュアルの Web サイト (http://www.novell.com/documentation/lg/dirxml20/index.html) を参照してください。

マニュアル表記規則

本マニュアルでは、手順に含まれる複数の操作および相互参照パス内の項目を分けるために、左向きの不等号 (>) を使用しています。

商標記号 (®、™ など) は、Novell の商標を示します。アスタリスク (*) はサードパーティの商標を示します。

ユーザコメント

このマニュアルおよび本製品に含まれるその他のマニュアルに関するコメントと提案をお聞かせください。proddoc@novell.com まで電子メールにてご連絡ください。

1 ポリシーとフィルタ

この節では、ポリシーとフィルタの概要、および DirXML® 環境でのそれらの機能について説明します。次のトピックについて説明します。

ポリシーとフィルターについて

ポリシーを使用すると、高いレベルで、DirXML が更新を送受信する方法をカスタマイズできます。

ポリシーを理解するには、ドライバシムが何を実行するように作成されているかについてある程度詳しく知っておくと役に立ちます。

ドライバシムを作成する場合、ドライバを展開する企業で使用される可能性があるすべての情報を同期化する機能を含めようとします。開発者は、接続システム内での関連する変更をすべて検出して、この変更を Novell® eDirectory™ に渡すようにドライバシムを作成します。

この変更は XML 文書に格納され、DirXML の仕様に従ってフォーマットされます。次の抜粋には、これらの XML 文書の 1 つが含まれています。

```
<nds dtdversion="2.0" ndsversion="8.7.3">
<source>
   <product version="2.0">DirXML</product>
   <contact>Novell, Inc.</contact>
</source>

<input>
   <add class-name="User" event-id="0" src-dn="\ACME\Sales\Smith"
   src-entry-id="33071">
      <add-attr attr-name="Surname">
         <value timestamp="1040071990#3" type="string">Smith</value>
      </add-attr>
      <add-attr attr-name="Telephone Number">
         <value timestamp="1040072034#1" type="teleNumber">111-1111</value>
      </add-attr>
   </add>
</input>
</nds>
```

目的によっては、Smith というユーザが電話番号 111-1111 と一緒にシステムに追加されても問題ないかもしれませんが、一部のユーザにとっては問題になる場合があります。

ここで重要なのは、ドライバは関連する変更をすべて報告し、ユーザが必要な変更かを判断してフィルタ処理または変更できるように設計されている点です。どの変更が重要かを判断し、これらの変更を処理する方法のロジックは、ドライバシムでなくエンジンで処理されます。

会社がユーザにそれほど関心がなければ、フィルタを実装して、eDirectory または接続システムのユーザに関する操作をすべてブロックできます。ユーザが会社にとって重要であれば、フィルタを実装して逆の処理を実行できます。

ドライバをカスタマイズする際には、まず、重要でないオブジェクトを同期化しないようにフィルタを定義します。

次に、フィルタでブロックされなかったオブジェクトに対して DirXML が行う処理を定義します。たとえば、前の XML 文書にあった add（追加）操作について考えてみましょう。Smith というユーザと電話番号 111-1111 が接続システムに追加されています。この操作をフィルタしない場合、DirXML はこのユーザの処理を決定する必要があります。

この決定を行うために、DirXML はポリシーセットを特定の順序で適用します（ここでは Transformation Policy（変換ポリシー）については無視します。変換ポリシーは、フィルタが発行者チャネル上で適用される前に実行され、加入者チャネル上の最後のステップです）。

「このオブジェクトがすでにデータストアにあるか」という疑問を解消するのは、最初のポリシーである Matching Policy（一致ポリシー）です。これに答えるには、オブジェクトに固有の特性を定義する必要があります。確認すべき一般的な属性は電子メールアドレスです。電子メールアドレスは通常個人特有のものであるからです。したがって、「2 つのオブジェクトが同じ電子メールアドレスを持っていれば、それらは同一のオブジェクトである」という内容のポリシーを定義します。

一致するオブジェクトが見つかると、DirXML はこのオブジェクトを関連付けと呼ばれる属性に記録します。関連付けとは、DirXML がオブジェクトを接続システムに関連付けられる固有の値です。

一致するオブジェクトが見つからなかった場合、次のポリシーである作成が呼び出されます。Create Policy（作成ポリシー）は、どのような条件の場合にオブジェクトを作成するかを DirXML に指示します。作成ルールでは、一部の属性を必須属性にできます。必須属性が存在しない場合、DirXML は、必要な情報が提供されるまでオブジェクトの作成をブロックします。

オブジェクトが生成されると、3 番目のルールである配置が、オブジェクトを配置する場所を DirXML に指示します。オブジェクトを元のシステムと同一の階層構造に作成するか、それとも属性値に基づいてまったく別の場所に配置するかを指定できます。

ユーザをオブジェクトの場所属性に従って階層に配置し、フルネームに従って名前を付ける場合、Create Policy（作成ポリシー）でこれらの属性を必須にすることができます。このような方法により、この属性が存在することを確認して、配置計画を正しく機能させることができます。

ポリシーを使用すると、さまざまな処理が可能です。Policy Builder を使えば、固有の値の生成、属性の追加と削除、イベントの生成、電子メールの送信など、多くの操作を簡単に実行できます。XSLT を使用して XML 文書を直接変換すると、さらに高度な変換が可能です（変更は eDirectory との間で XML 文書で送受信されます）。

基本は、ポリシーによって DirXML の更新処理を制御できるということです。

13 ページの「ポリシーの概要」でポリシーのさまざまな種類について学んでから、2 章 19 ページの、「Policy Builder を使用したポリシーの定義」で Policy Builder の具体的な操作を習得してください。

Transformation Policy（変換ポリシー）について

Transformation Policy（変換ポリシー）は DirXML と接続システムの間の変換メカニズムとして機能します。システム間でスキーマを変換し、受信した操作には初期変更を、送信する操作には最終変更を加えます。

基本的には、Transformation Policy（変換ポリシー）は前に説明した他のルール (matching(一致)、create(作成)、placement(配置)) を正しく機能させるために使用されます。各ドライバのデフォルト設定には必要な Transformation Policy（変換ポリシー）がすべて含まれているので、最初は変換ポリシーについて考える必要はありません（唯一の例外は Schema Mapping Policy（スキーママッピングポリシー）で、iManager の GUI を使用して簡単に変更できます）。

ポリシーの基本的な種類がわかったら、Transformation Policy（変換ポリシー）を理解すると、基本ポリシーでは不可能なカスタマイズを行うことができます。

DirXML 1.x の用語の変更

DirXML 1.x を使用したことがない場合は、この節を読む必要はありません。

DirXML 1.x では、ルールという用語は、文脈に応じて、一連のルール、そのセットに含まれる個々のルール、および個々のルールに含まれる条件やアクションを指すために使用されていました。しかし、このようにさまざまな状況で同じ用語が使われていると、文脈がわかりにくい場合には混乱を招いてしまいます。

DirXML 2 では、実行される最高レベルの変換を指す場合には、今までのルールに替えてポリシーという用語が使用されるようになりました。1 つまたは複数のポリシーで構成されるポリシーセットを定義し、各ポリシーには 1 つまたは複数のルールを含めることができます。ルールという用語は、複数の条件とアクションからなる個々のセットのみを指すようになりました。

次の表は用語の変更を示します。

意味する内容	DirXML 2 の用語	DirXML 1.x の用語
変換セット	ポリシーセット	ルール
セットに含まれる個々の変換	ポリシー	ルール
個々の変換に含まれる条件とアクション	ルール	ルール

ポリシーの概要

この節では、利用できるポリシーの種類の概要、DirXML でのポリシーの役割、および独自のポリシーを定義する方法を説明します。次のトピックについて説明します。

- 14 ページの「基本ポリシー」
- 15 ページの「Transformation Policy（変換ポリシー）」
- 16 ページの「ポリシーの定義」

基本ポリシー

加入者チャネルと発行者チャネルの両方で定義できるポリシーにはさまざまな種類があります。ポリシーはそれぞれデータ変換の異なる段階で適用されますが、特定のアクションが発生したときにのみ適用されるポリシーもあります。たとえば、Create Policy（作成ポリシー）は新規オブジェクトが作成されたときにのみ適用されます。

ポリシー	説明
Subscriber Matching（加入者一致）	eDirectory 内のオブジェクトに一致するアプリケーション内のオブジェクトを検索して、一致するオブジェクトを相互に関連付ける場合に使用される条件を含むオブジェクト。
Subscriber Create（加入者作成）	アプリケーション内に新規オブジェクトを作成するのに必要な属性の定義を含むオブジェクト。
Subscriber Placement（加入者配置）	新規アプリケーションオブジェクトを作成する場所を決定する条件を含むオブジェクト。
Publisher Matching（発行者一致）	アプリケーション内のオブジェクトに一致する eDirectory 内のオブジェクトを検索して、一致するオブジェクトを相互に関連付ける場合に使用される条件を含むオブジェクト。
Publisher Create（発行者作成）	eDirectory 内に新規オブジェクトを作成するのに必要な属性の定義を含むオブジェクト。
Publisher Placement（発行者配置）	新規 eDirectory オブジェクトを作成する場所を決定する条件を含むオブジェクト。
Schema Mapping（スキーママッピング）	eDirectory とアプリケーション間のスキーママッピングの定義を保持するオブジェクト。

Create（作成）

Create Policy（作成ポリシー）は、新規オブジェクトを作成するために必要な属性の最小限のセットを定義します。

たとえば、eDirectory に新規ユーザを作成して、新規ユーザオブジェクトに名前と ID のみを付けたとします。この作成は eDirectory ツリーでミラーリングされますが、追加は eDirectory に接続しているアプリケーションにすぐには反映されません。これは、より完全な定義を持つユーザオブジェクトのみを許可するよう Create Policy（作成ポリシー）で指定しているからです。

Create Policy（作成ポリシー）は、加入者および発行者の両方に対して同じものを使用することも、異なるものを使用することもできます。

Create Policy（作成ポリシー）は、eDirectory ではドライバ内のオブジェクトとして表されます。

Matching（一致）

Matching Policy（一致ポリシー）は、2 つのオブジェクトを同一と見なす最小限の条件を定義します。

Placement（配置）

Placement Policy（配置ポリシー）は、eDirectory および接続アプリケーション内で新規オブジェクトを作成する場所を決定します。

各ドライバには少なくとも 2 つの Placement Policy（配置ポリシー）が必要です。1つは、外部アプリケーションデータベースが新規オブジェクトを作成したときに新規 eDirectory オブジェクトを配置する場所を指定し、もう 1 つは、eDirectory 内で新規オブジェクトが作成されたときに外部アプリケーションデータベースオブジェクトを作成する場所を指定します。

eDirectory は階層型なので、複数のポリシーがあると複数のコンテナにオブジェクトを作成できるため便利です。ただし、新規オブジェクトをすべて同じコンテナ内に作成しておいて、後で部門コンテナに移動できます。

Schema Mapping（スキーママッピング）

Schema Mapping Policy（スキーママッピングポリシー）は、eDirectory と接続システムの間のスキーママッピングの定義を保持します。

eDirectory スキーマは eDirectory から読み込まれます。アプリケーションスキーマは、接続システムの DirXML ドライバによって提供されます。2 つのスキーマが識別されると、eDirectory と目的のアプリケーションの間に単純なマッピングが作成されます。

DirXML ドライバ設定にスキーママッピングが定義されたら、対応するデータをマッピングすることができます。

Transformation Policy（変換ポリシー）

次のポリシーは、eDirectory とアプリケーションの間でイベントデータフォーマットを変換する場合に使用されます。

ポリシー	説明
Output Transformation（出力変換）	情報が eDirectory からアプリケーションへ渡されるときに使用される変換アクション。
Input Transformation（入力変換）	情報がアプリケーションから eDirectory へ渡されるときに使用される変換アクション。

次のポリシーは、eDirectory とアプリケーションの間でイベントアクションを変換する場合に使用されます。

ポリシー	説明
Subscriber Event Transformation（加入者イベント変換）	あるイベントを別のイベントに変換する場合に使用される変換アクション。
Publisher Event Transformation（発行者イベント変換）	あるイベントを別のイベントに変換する場合に使用される変換アクション。

次のポリシーは、eDirectory とアプリケーションの間でコマンドを変換する場合に使用されます。

ポリシー	説明
Subscriber Command Transformation（加入者コマンド変換）	DirXML エンジンによって eDirectory に送信されたコマンドに対して使用される変換アクション。
Publisher Command Transformation（発行者コマンド変換）	DirXML エンジンによってドライバに送信されたコマンドに対して使用される変換アクション。

ポリシーの定義

ポリシーは次のいずれかの方法で定義できます。

- Policy Builder インタフェースを使って DirXML Script を生成する。既存の非 XSLT ルールは、インポート時に自動的に DirXML Script に変換されます。
- XSLT スタイルシートを使用する。

Policy Builder と DirXML Script

実装するほとんどのポリシーは、Policy Builder インタフェースを使用して定義します。Policy Builder インタフェースではグラフィカルな環境が採用されており、簡単にポリシーを定義および管理できます。

Policy Builder 内におけるルール作成の基礎となる機能は、DirXML Script というカスタムスクリプト言語によって提供されます。

DirXML Script は、テストできるさまざまな条件、実行するアクション、およびポリシーに追加する動的な値で構成されます。これらの各オプションは、インテリジェントなドロップダウンリストを使用して表示され、各ポイントにおいて有効な選択肢と、一般的な値へのリンクを提供します。

Policy Builder を使用すると、DirXML Script を直接操作する必要はありません。

Policy Builder の詳細については、2 章 19 ページの、「Policy Builder を使用したポリシーの定義」を参照してください。

ヒント： Policy Builder を使用する際には必要ありませんが、DirXML Script の完全なリファレンスは、http://developer.novell.com/ndk/doc/dirxml/dirxmlbk/ref/index.html (http://developer.novell.com/ndk/doc/dirxml/dirxmlbk/ref/index.html) から入手できる DirXML Driver Developer Kit に収録されています。

XSLT スタイルシート

より複雑なポリシーを定義するには、XSLT スタイルシートを使用して、ある XML 文書を、必要な変更を含む別の XML 文書に直接変換します。

スタイルシートは非常に柔軟で、Policy Builder 内のルール作成で利用できる定義済みの条件やアクションに変換が合わない場合に使用します。

XSLT スタイルシートを作成するには、XSLT (nds.dtd) と、DirXML エンジンとの間でやり取りされるコマンドとイベントについての深い知識が必要になります。nds.dtd に関する詳しいリファレンスについては、NDS DTD リファレンス (http://developer.novell.com/ndk/doc/dirxml/dirxmlbk/ref/ndsdtd/DTD-TREE.html) および nds.dtd (http://developer.novell.com/ndk/doc/dirxml/dirxmlbk/ref/nds.dtd) を参照してください。

XSLT スタイルシートの詳細については、3 章 135 ページの、「XSLT スタイルシートを使用したポリシーの定義」を参照してください。

フィルタの概要

フィルタは、DirXML エンジンがイベントを処理するオブジェクトクラスと属性を指定します。

加入者および発行者チャネルに対して個々のイベントフィルタが指定されます。イベントフィルタは、フィルタで指定したクラスの 1 つに一致するベースクラスを持つオブジェクトで発生したイベントを渡すだけです。イベントフィルタは、フィルタで指定されているクラスの下位クラスにあたるオブジェクトで発生したイベントは渡しません。ただし、下位クラスも指定されている場合を除きます。

注： eDirectory では、ベースクラスはエントリを作成するために使用されるオブジェクトクラスです。フィルタ内では、ベースクラスの継承元であるスーパークラスではなく、このクラスを指定する必要があります。

たとえば、ユーザクラスが、名字属性と名前属性を持つイベントフィルタで指定されている場合、DirXML エンジンは、これらの属性に対するすべての変更を渡します。ただし、エントリの電話番号属性が変更された場合、電話番号属性はイベントフィルタ内に存在しないため、このイベントは破棄されます。

フィルタは次のトピックを含むように設定する必要があります。

- ルールで必要な属性
- 同期化する属性

フィルタの定義については、4 章 147 ページの、「フィルタの定義」を参照してください。

2 Policy Builder を使用したポリシーの定義

Policy Builder はグラフィカルな完全なインタフェースで、接続システム間のデータ交換を定義するポリシーの作成と管理に使用できます。

この節では、Policy Builder の使用に関する次のトピックについて説明します。

- 19 ページの「Policy Builder タスク」

また、次のトピックの詳細についても説明します。

- 25 ページの「条件」
- 46 ページの「アクション」
- 99 ページの「名詞」
- 124 ページの「動詞」

Policy Builder タスク

この節では、Policy Builder での一般的なタスク実行方法について説明します。

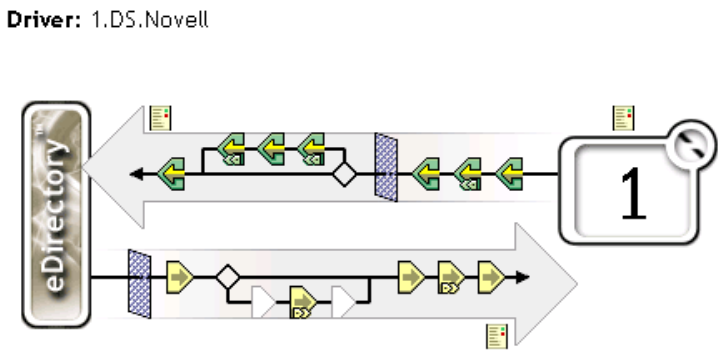
- 19 ページの「Policy Builder の起動」
- 20 ページの「ポリシーの作成」
- 24 ページの「ポリシーの変更」
- 20 ページの「ポリシー内の個々のルールの定義」
- 22 ページの「ルール内の個々の引数の定義」

Policy Builder の起動

1 iManager で、[DirXML Management Role] を展開して [Overview] をクリックします。

2 ドライバセットを指定します。

3 ポリシーを管理するドライバをクリックします。[DirXML Driver Overview] が開きます。



4 [DirXML Driver Overview] からポリシーを管理します。

ポリシーの作成

1 [DirXML Driver Overview] を開き、管理するドライバを見つけます。

2 定義するポリシーを表すアイコンをクリックします。

 は未定義のポリシーを表します。

 は定義済みのポリシーを表します。

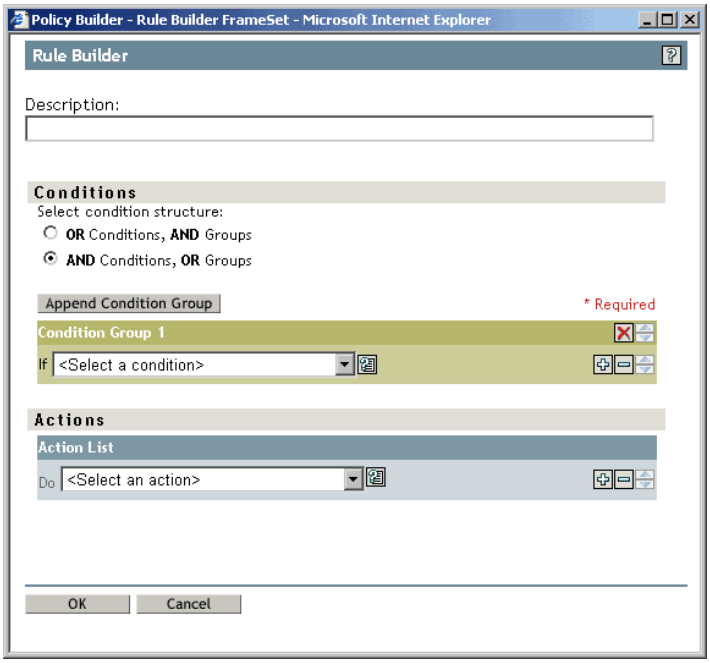
3 [Insert] をクリックします。

4 新規ポリシーの名前を入力し、[Policy Builder] を選択します。

5 ポリシーが表示されます。このポリシーに対して 1 つまたは複数のルールを定義するには、[Append New Rule] をクリックして、20 ページの「ポリシー内の個々のルールの定義」の手順に従います。

ポリシー内の個々のルールの定義

ルールは、Policy Builder の [Rule Builder] ウィンドウで定義します。

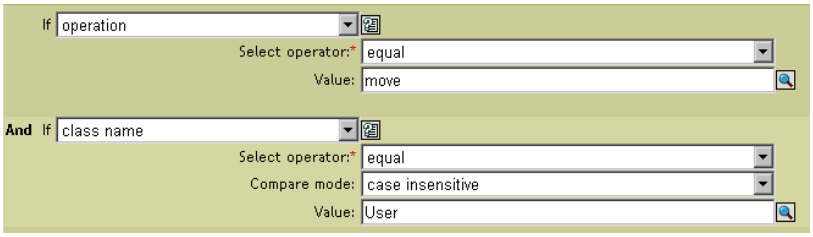


Rule Builder インタフェースでは、インテリジェントなドロップダウンメニューを使用してルールをすばやく作成および変更できます。

Rule Builder で、定義済みアクションを実行する前に満たさなければならない条件のセットを定義します。

たとえば、自分の環境に追加された新規オブジェクトをすべて拒否するルールを作成する必要がある場合は、次のようなルールを定義できます。add（追加）操作が発生すると、操作を拒否します。

Rule Builder でこのロジックを実装するには、次の条件を選択できます。

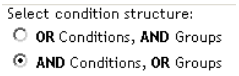


さらに、アクションを選択します。

Rule Builder で利用できる条件とアクションの詳細については、25 ページの「条件」および 46 ページの「アクション」を参照してください。

ヒント

さらに複雑な条件を作成するには、条件と、条件のグループまたはステートメント、あるいはその両方を結合できます。結合する方法を変更するには、条件構造を選択します。

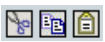


 アイコンをクリックすると、フィールドの値のリストを表示できます。前の例では、このアイコンをクリックすると、有効なクラス名のリストが開きます。

 アイコンをクリックすると、Argument Builder インタフェースを使用して引数を作成できます。

 アイコンをクリックすると、ポリシー、ルール、条件、またはアクションを無効にできます。  アイコンをクリックすると、再び有効にできます。

 アイコンをクリックすると、コメントをポリシーまたはルールに追加できます。コメントはポリシーまたはルール上に直接保存され、長さに制限はありません。

[Cut] / [Copy] / [Paste] アイコン  をクリックすると、Policy Builder のクリップボードを使用できます。[Paste] アイコンは、現在クリップボードに入っている内容が無効な場所では使用できません。

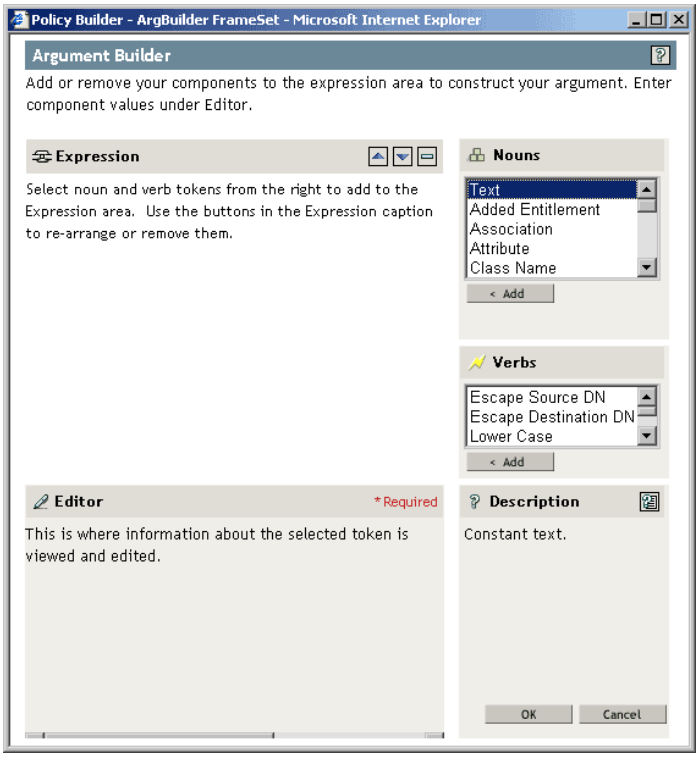
 アイコンを使用すると、条件を追加、削除、および配置できます。

ボタンを使用すると、条件グループを追加できます。

 アイコンを使用すると、条件グループを削除および配置できます。

ルール内の個々の引数の定義

Argument Builder は、Rule Builder 内で使う複雑な引数式を構築できるダイナミックなグラフィカルインタフェースです。

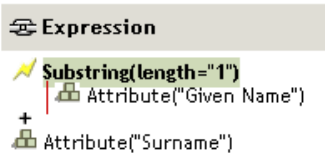


式を定義するには、1 つまたは複数の名詞（値、オブジェクト、変数など）を選択し、動詞（下位文字列、エスケープ、大文字、小文字）と結合して式を構築します。

複数の名詞、動詞、および式を結合して、複雑な引数を構築します。

たとえば、引数を属性値に設定する場合は、属性名詞を選択して属性名を入力または選択するだけです。

この属性の一部だけが必要な場合は、属性名詞を下位文字列動詞と結合できます。



Argument Builder で使用できる名詞および動詞の詳細については、99 ページの「名詞」および 124 ページの「動詞」を参照してください。

ヒント

さらに複雑な条件を作成するには、条件または条件のグループを and/or ステートメントで結合できます。

 アイコンを使うと、名詞および動詞を移動および削除できます。

 アイコンをクリックすると、フィールドの値のリストを表示できます。

名詞または動詞を追加したら、エディタで値を入力して、すぐに他の名詞または動詞を追加できます。[Expression] ペインを更新して変更を適用する必要はありません。次の操作を実行すると、追加した名詞や動詞が表示されます。

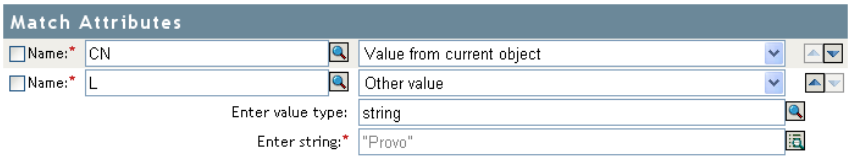
ほとんどの引数はこの標準インタフェースを使って定義できますが、特定の状況では、カスタムの [Argument Builder] ウィンドウを使用して情報を入力します。これらのウィンドウのいくつかでは、値を入力するためにデフォルトの Argument Builder が起動されます。

次の節では、Argument Builder の追加インタフェース、およびこれらを使用する条件とアクションについて説明します。

- 23 ページの「Matching Attribute Builder」
- 23 ページの「Argument Actions Builder」
- 24 ページの「Named String Builder」
- 24 ページの「Argument Value List Builder」

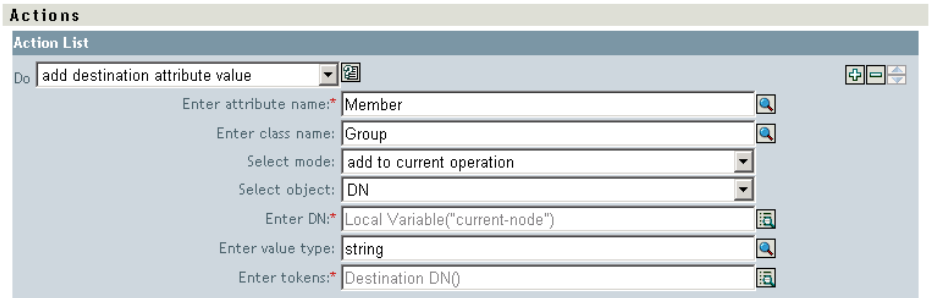
Matching Attribute Builder

Matching Attribute Builder は、(63 ページ) Find Matching Object (一致オブジェクトの検索) アクションを満たす条件を構築するのに使います。



Argument Actions Builder

Argument Actions Builder は、(65 ページ) For Each (繰り返し) などのアクションで実行するアクションのリストを構築するのに使います。



Named String Builder

Named String Builder は、(66 ページ) Generate Event (イベントの生成) や (77 ページ) Send Email (電子メールの送信) などのアクションで使う名前と値のペアを作成するのに使います。

Argument Value List Builder

Argument Value List Builder は、(81 ページ) Set Default Attribute Value (デフォルト属性値の設定) などのアクションの引数を作成するのに使います。この例では、不明な値を持つ文字列引数を作成して、デフォルトの場所を設定しています。

ポリシーの変更

1 [DirXML Driver Overview] を開き、管理するドライバを見つけます。
2 変更するポリシーを示すアイコンをクリックします。
3 変更するポリシーを選択し、[Edit] をクリックします。

ポリシーの削除

1 [DirXML Driver Overview] を開き、管理するドライバを見つけます。
2 削除するポリシーを示すアイコンをクリックします。
3 削除するポリシーを選択し、[Remove] をクリックします。

XML ファイルからのポリシーのインポート

1 [DirXML Driver Overview] を開き、管理するドライバを見つけます。
2 削除するポリシーを示すアイコンをクリックします。
3 既存のポリシーを編集するか、または新規ポリシーを作成します。
4 [Insert] ボタンをクリックし、[Import an XML file containing DirXML Script] を選択します。
5 インポートするポリシーファイルを参照し、[OK] をクリックします。

XML ファイルへのポリシーのエクスポート

1 [DirXML Driver Overview] を開き、管理するドライバを見つけます。

2 削除するポリシーを示すアイコンをクリックします。

3 既存のポリシーを編集するか、または新規ポリシーを作成します。

4 [Save As] ボタンをクリックし、場所を選択して DirXML Script XML ファイルを保存します。

ポリシーリファレンスの作成

ポリシーリファレンスを使用すると、単一のポリシーを作成して複数の場所で参照できます。複数のドライバまたはポリシーによって使われているポリシーがある場合は、リファレンスを作成すると、ポリシーの管理が容易になります。

1 [DirXML Driver Overview] を開き、管理するドライバを見つけます。

2 削除するポリシーを示すアイコンをクリックします。

3 既存のポリシーを編集するか、または新規ポリシーを作成します。

4 [Insert] ボタンをクリックし、[Append a reference to a policy containing DirXML Script] を選択します。

5 参照するポリシーオブジェクトを参照し、[OK] をクリックします。

条件

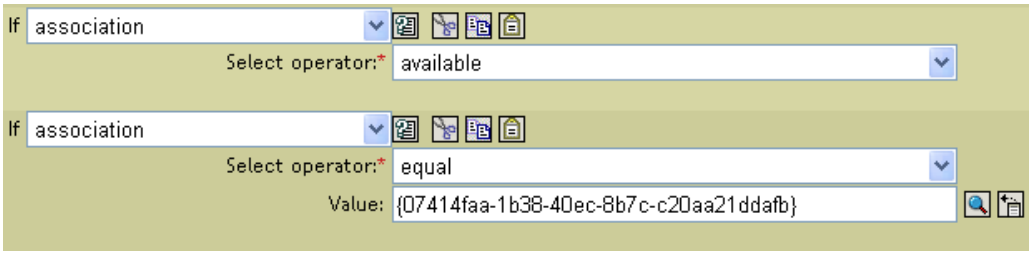
この節では、Policy Builder インタフェースで使用できるすべての条件の詳細について説明します。

(26 ページ) If Association (関連付けの条件)
(27 ページ) If Attribute (属性の条件)
(29 ページ) If Class Name (クラス名の条件)
(30 ページ) If Destination Attribute (宛先属性の条件)
(31 ページ) If Destination DN (宛先 DN の条件)
(33 ページ) If Entitlement (エンタイトルメントの条件)
(35 ページ) If Global Configuration Value (グローバル設定値の条件)
(36 ページ) If Local Variable (ローカル変数の条件)
(37 ページ) If Named Password (名前付きパスワードの条件)
(38 ページ) If Operation Attribute (操作属性の条件)
(40 ページ) If Operation (操作の条件)
(41 ページ) If Operation Property (操作プロパティの条件)
(42 ページ) If Password (パスワードの条件)
(43 ページ) If Source Attribute (ソース属性の条件)
(44 ページ) If Source DN (ソース DN の条件)
(46 ページ) If Xpath Expression (Xpath 式の条件)

If Association（関連付けの条件）

If Association（関連付けの条件）は、現在の操作またはオブジェクトの関連付け値をテストします。

例



条件

演算子	条件が満たされる場合
associated	現在のオブジェクトに対して設定された関連付けがある。
available	現在の操作によって指定された空でない関連付け値がある。
equal	現在の操作によって指定された関連付け値が If Association（関連付けの条件）の内容に完全に等しい。
not-associated	[associated] が False を返す。
not available	[available] が False を返す。
not equal	[equal] が False を返す。

フィールド

演算子

条件テストタイプを選択します。

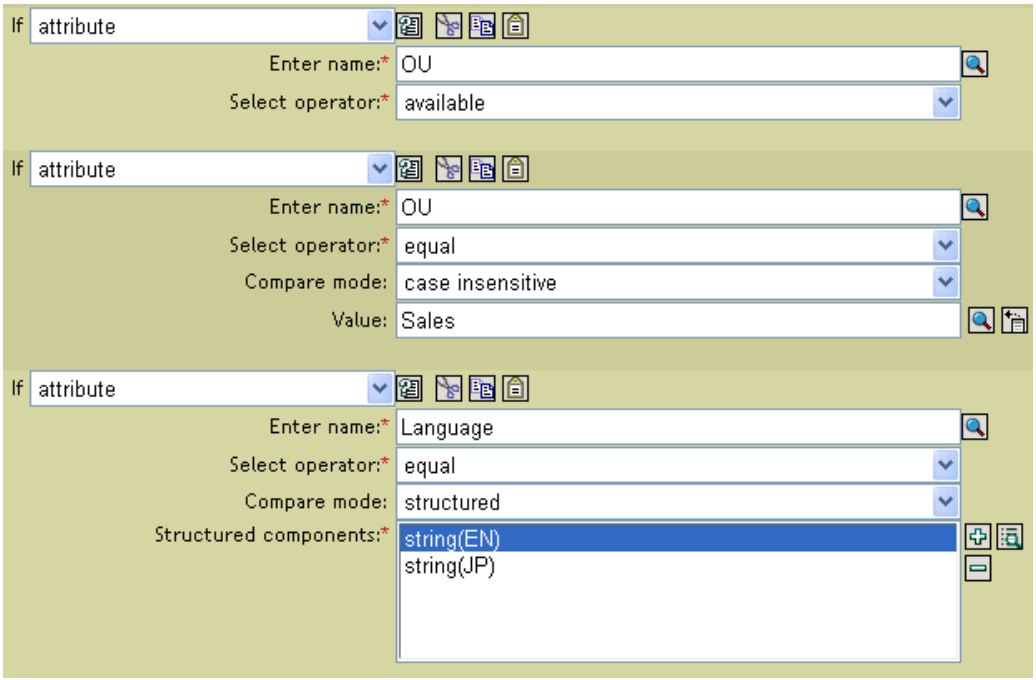
比較モード

比較モードを選択します。133 ページの「比較モード」を参照してください。

If Attribute（属性の条件）

If Attribute（属性の条件）は、現在の操作またはソースのデータストアのいずれかで現在のオブジェクトの関連付け値をテストします。

例



条件

演算子	条件が満たされる場合
available	現在の操作またはソースのデータストアのいずれかで、指定した属性に対する値が使用できる。
equal	現在の操作またはソースのデータストアのいずれかで、指定した属性に対する値が使用でき、指定した比較モードを使って比較した場合に、指定した値に等しい。
not available	[available] が False を返す。
not equal	[equal] が False を返す。

フィールド

名前

選択した条件に対してテストする属性の名前を指定します。

演算子

条件テストタイプを選択します。

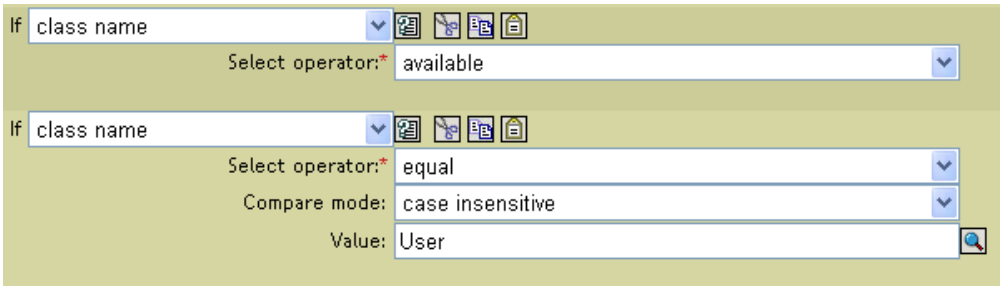
比較モード

比較モードを選択します。133 ページの「比較モード」を参照してください。

If Class Name（クラス名の条件）

If Class Name（クラス名の条件）は、現在の操作のオブジェクトクラス名をテストします。

例



条件

演算子	条件が満たされる場合
available	現在の操作に使用できるオブジェクトクラス名がある。
equal	現在の操作に使用できるオブジェクトクラス名があり、指定した比較モードを使って比較した場合、指定した値に等しい。
not available	[available] が False を返す。
not equal	[equal] が False を返す。

フィールド

演算子

条件テストタイプを選択します。

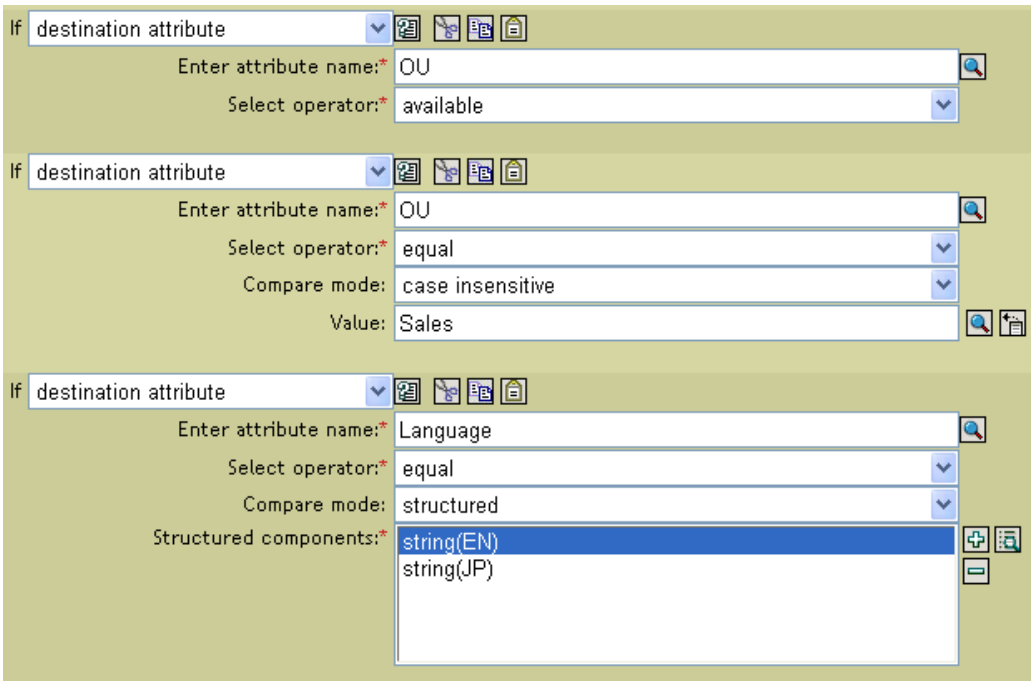
比較モード

比較モードを選択します。133 ページの「比較モード」を参照してください。

If Destination Attribute（宛先属性の条件）

If Destination Attribute（宛先属性の条件）は、宛先データストア内にある現在のオブジェクトの属性値をテストします。

例



条件

演算子	条件が満たされる場合
available	宛先データストアで、指定した属性に対する値が使用できる。
equal	宛先データストアで、指定した属性に対する値が使用でき、指定した比較モードを使って比較した場合、指定した値に等しい。
not available	[available] が False を返す。
not equal	[equal] が False を返す。

フィールド

名前

選択した条件に対してテストする属性の名前を指定します。

演算子

条件テストタイプを選択します。

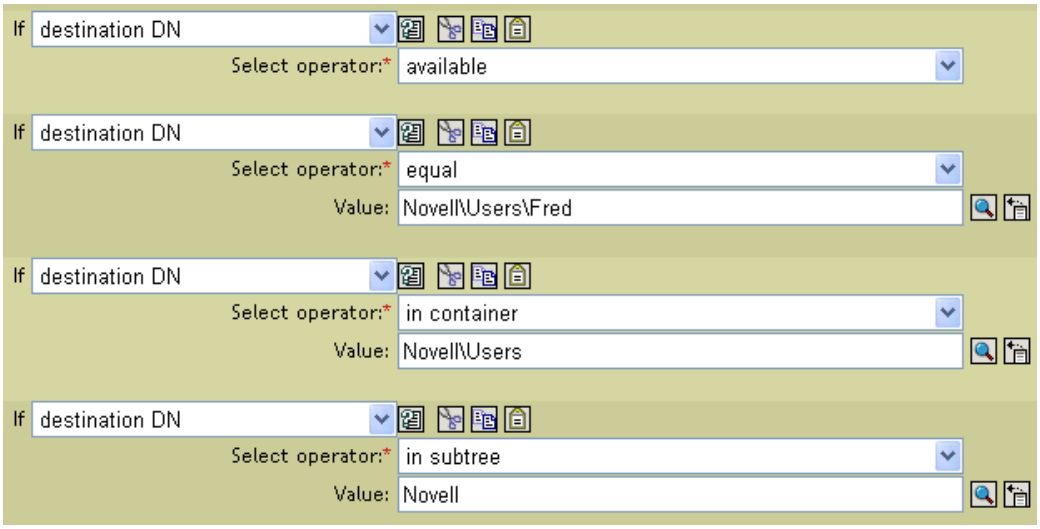
比較モード

比較モードを選択します。133 ページの「比較モード」を参照してください。

If Destination DN（宛先 DN の条件）

If Destination DN（宛先 DN の条件）は、現在の操作の宛先 DN をテストします。

例



条件

演算子	条件が満たされる場合
available	使用可能な宛先 DN がある。
equal	使用可能な宛先 DN があり、宛先データストアの DN フォーマットに適したセマンティックを使って比較した場合、指定した値に等しい。
in container	使用可能な宛先 DN があり、宛先データストアの DN フォーマットに適したセマンティックを使って比較した場合、値によって指定されるコンテナ内のオブジェクトを示す。
in subtree	使用可能な宛先 DN があり、宛先データストアの DN フォーマットに適したセマンティックを使って比較した場合、値によって指定されるサブツリー内のオブジェクトを示す。
not available	[available] が False を返す。
not equal	[equal] が False を返す。
not in container	[in container] が False を返す。
not in subtree	[in subtree] が False を返す。

フィールド

演算子

条件テストタイプを選択します。

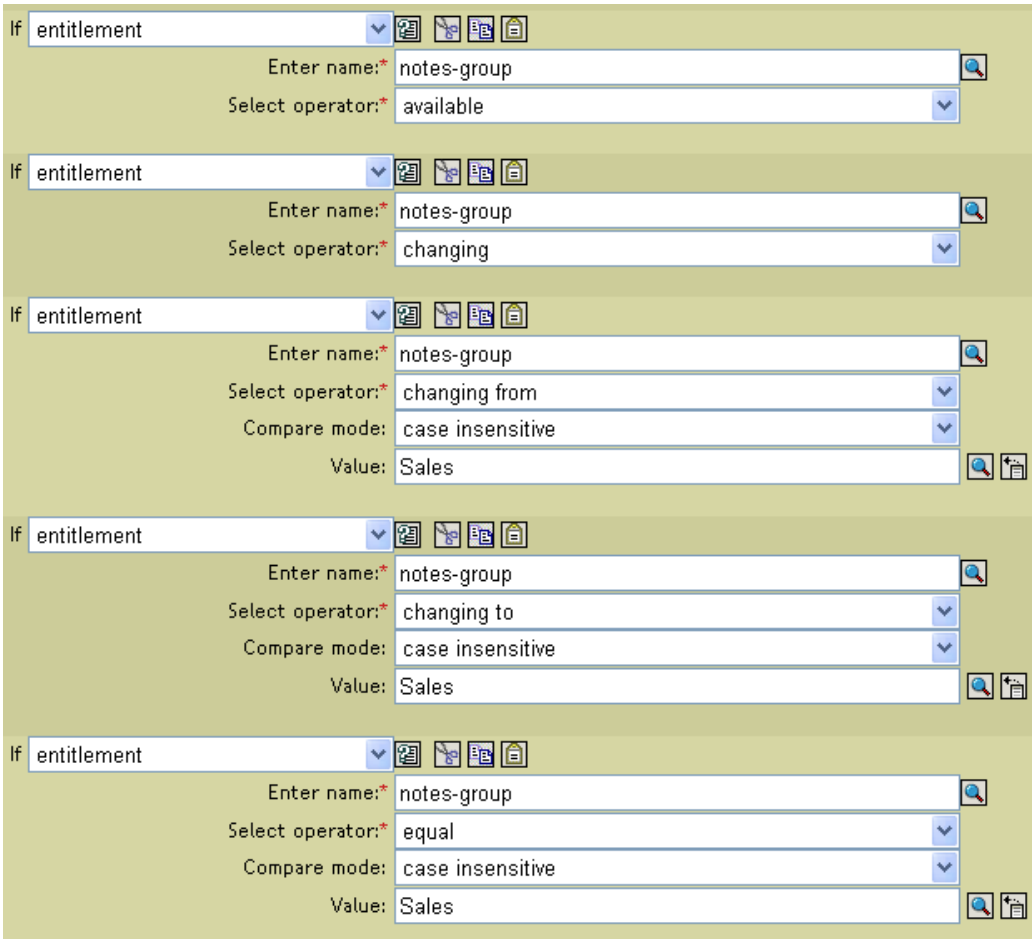
比較モード

比較モードを選択します。133 ページの「比較モード」を参照してください。

If Entitlement（エンタイトルメントの条件）

If Entitlement（エンタイトルメントの条件）は、現在の操作または eDirectory のいずれかで、現在のオブジェクトのエンタイトルメントをテストします。

例



条件

演算子	条件が満たされる場合
available	指定したエンタイトルメントが現在の操作または eDirectory™ データストアで使用できる。
changing	現在の操作に、指定したエンタイトルメントの変更（属性の変更または属性の追加）が含まれる。
changing from	現在の操作に、指定したエンタイトルメントから値を削除する変更（値の削除）が含まれ、指定した比較モードを使って比較した場合、指定した値に等しい値を持つ変更になる。

演算子	条件が満たされる場合
changing to	現在の操作に、指定したエンタイトルメントに値を追加する変更（値の追加または属性の追加）が含まれ、指定した比較モードを使って比較した場合、指定した値に等しい値を持つ変更になる。
equal	宛先データストアで、指定した属性に対する値が使用でき、指定した比較モードを使って比較した場合、指定した値に等しい。
not available	[available] が False を返す。
not changing	[changing] が False を返す。
not changing from	[changing from] が False を返す。
not changing to	[changing to] が False を返す。
not equal	[equal] が False を返す。

フィールド

名前

選択した条件に対してテストするエンタイトルメントの名前を指定します。

演算子

条件テストタイプを選択します。

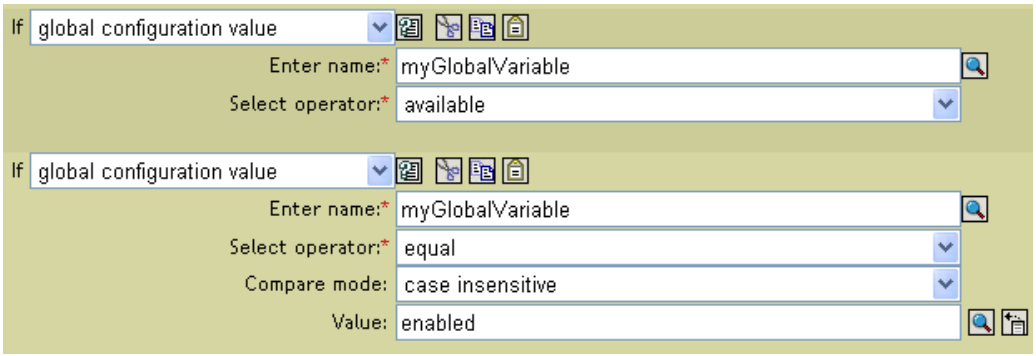
比較モード

比較モードを選択します。133 ページの「比較モード」を参照してください。

If Global Configuration Value（グローバル設定値の条件）

If Global Configuration Value（グローバル設定値の条件）は、グローバル設定変数をテストします。

例



条件

演算子	条件が満たされる場合
available	指定した名前のグローバル設定変数がある。
equal	指定した名前のグローバル設定変数があり、その値が、指定した比較モードを使って比較した場合、指定した値に等しい。
not available	[available] が False を返す。
not equal	[equal] が False を返す。

フィールド

名前

選択した条件に対してテストするグローバル変数の名前を指定します。

演算子

条件テストタイプを選択します。

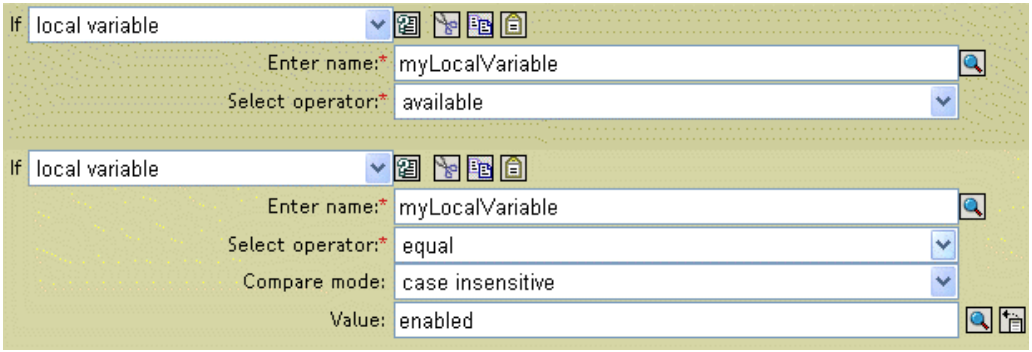
比較モード

比較モードを選択します。133 ページの「比較モード」を参照してください。

If Local Variable（ローカル変数の条件）

If Local Variable（ローカル変数の条件）は、ローカル変数をテストします。

例



条件

演算子	条件が満たされる場合
available	指定された名前を持ち、ポリシー内にある前のルールのアクションによって定義されているローカル変数がある。
equal	指定した名前のローカル変数があり、その値が、指定した比較モードを使って比較した場合、指定した値に等しい。
not available	[available] が False を返す。
not equal	[equal] が False を返す。

フィールド

名前

選択した条件に対してテストするローカル変数の名前を指定します。

演算子

条件テストタイプを選択します。

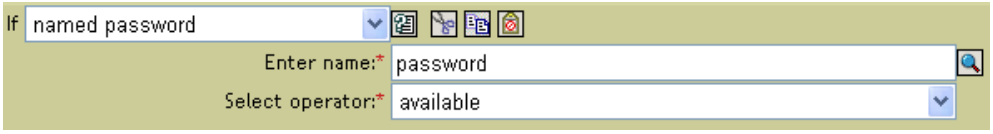
比較モード

比較モードを選択します。133 ページの「比較モード」を参照してください。

If Named Password（名前付きパスワードの条件）

If Named Password（名前付きパスワードの条件）は、現在の操作で指定された名前のパスワードをテストします。実行されるテストタイプは、選択した演算子により異なります。次の表は、各演算子によって実行されるテストタイプについて説明しています。

例



条件

演算子	条件が満たされる場合
available	指定した名前のパスワードが使用できる。
not available	[available] が False を返す。

フィールド

名前

選択した条件に対してテストする、名前付きパスワードの名前を指定します。

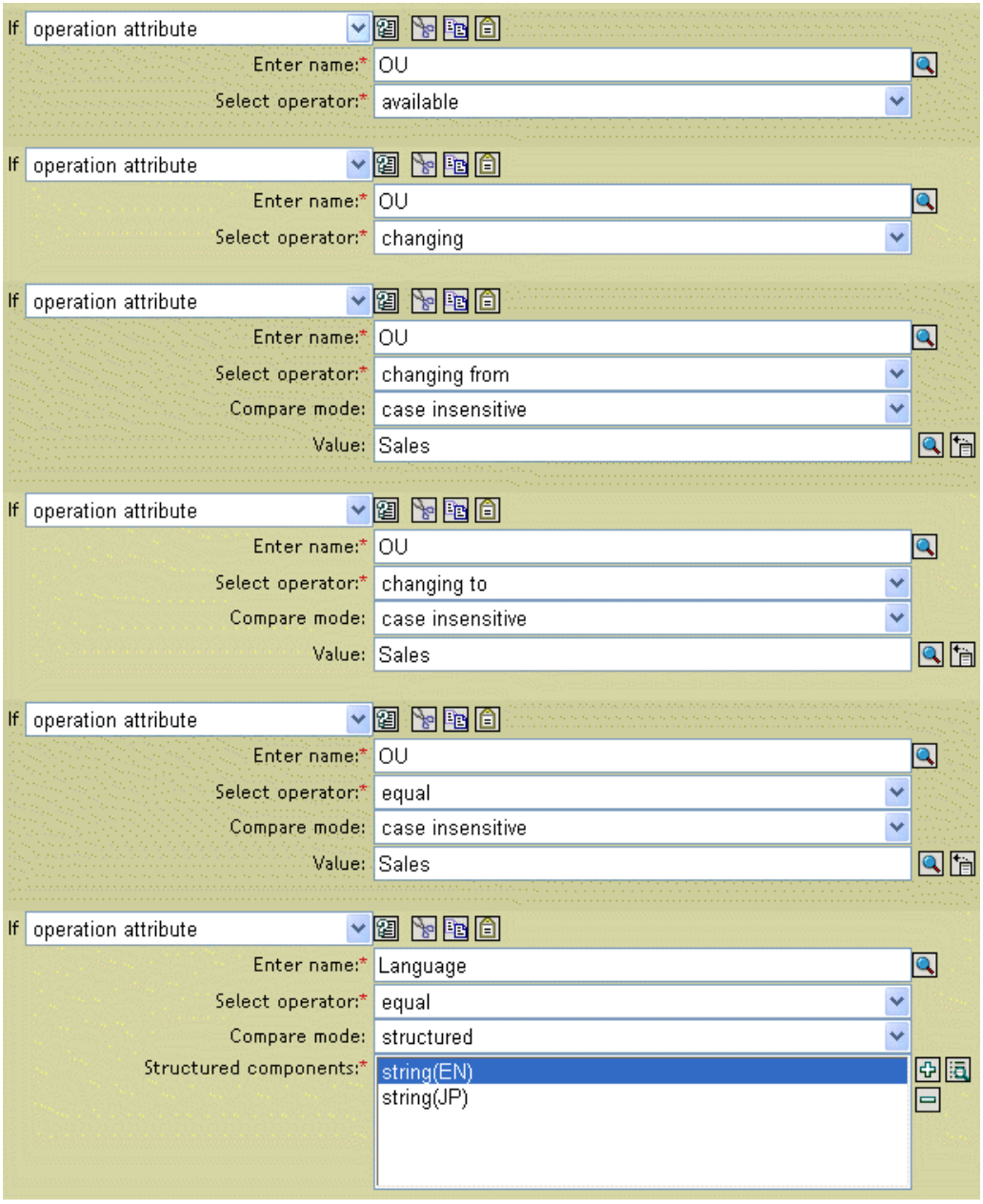
演算子

条件テストタイプを選択します。

If Operation Attribute（操作属性の条件）

If Operation Attribute（操作属性の条件）は、現在の操作の属性値をテストします。

例



条件

演算子	条件が満たされる場合
available	現在の操作（属性の追加、値の追加、属性）で、指定した属性に対する値が使用できる。
changing	現在の操作に、指定した属性の変更（属性の変更または属性の追加）が含まれる。
changing from	現在の操作に、指定した属性から値を削除する変更（値の削除）が含まれ、指定した比較モードを使って比較した場合、指定した値に等しい。
changing to	現在の操作に、指定した属性に値を追加する変更（値の追加または属性の追加）が含まれ、指定した比較モードを使って比較した場合、指定した値に等しい。
equal	現在の操作（値の削除以外）で、指定した属性に対する値が使用でき、指定した比較モードを使って比較した場合、指定した値に等しい。
not available	[available] が False を返す。
not changing	[changing] が False を返す。
not changing from	[changing from] が False を返す。
not changing to	[changing to] が False を返す。
not equal	[equal] が False を返す。

フィールド

名前

選択した条件に対してテストする属性の名前を指定します。

演算子

条件テストタイプを選択します。

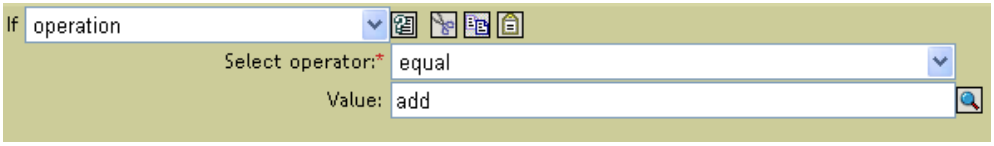
比較モード

比較モードを選択します。133 ページの「比較モード」を参照してください。

If Operation（操作の条件）

If Operation（操作の条件）は、現在の操作の名前をテストします。

例



条件

演算子	条件が満たされる場合
equal	現在の操作の名前が If Operation（操作の条件）の内容に完全に等しい。
not equal	[equal] が False を返す。

フィールド

演算子

条件テストタイプを選択します。

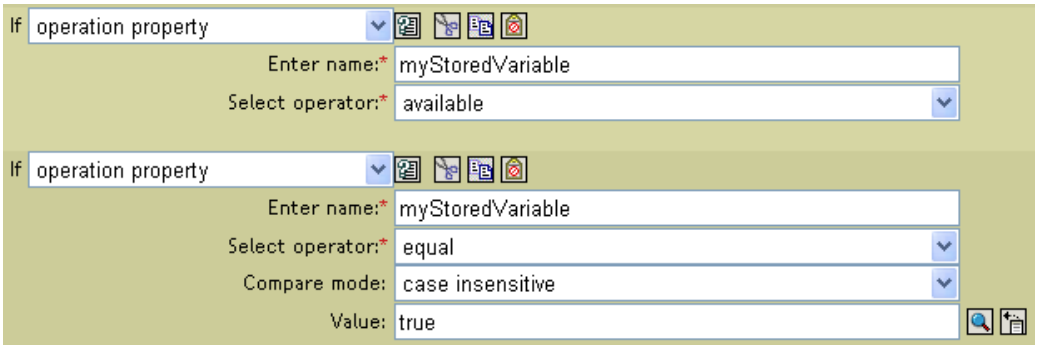
比較モード

比較モードを選択します。133 ページの「比較モード」を参照してください。

If Operation Property（操作プロパティの条件）

If Operation Property（操作プロパティの条件）は、現在の操作に関する操作プロパティをテストします。実行されるテストタイプは、選択した演算子により異なります。次の表は、各演算子によって実行されるテストタイプについて説明しています。

例



条件

演算子	条件が満たされる場合
available	現在の操作に、指定した名前の操作プロパティがある。
equal	現在の操作に、指定した名前の操作プロパティがあり、その値が、指定した比較モードを使って比較した場合、得られる内容に等しい。
not available	[available] が False を返す。
not equal	[equal] が False を返す。

フィールド

名前

選択した条件に対してテストする操作プロパティの名前を指定します。

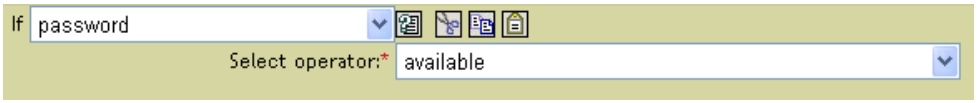
演算子

条件テストタイプを選択します。

If Password（パスワードの条件）

If Password（パスワードの条件）は、現在の操作のパスワードをテストします。

例



条件

演算子	条件が満たされる場合
available	現在の操作に、使用できるパスワードがある。
not available	[available] が False を返す。

フィールド

演算子

条件テストタイプを選択します。

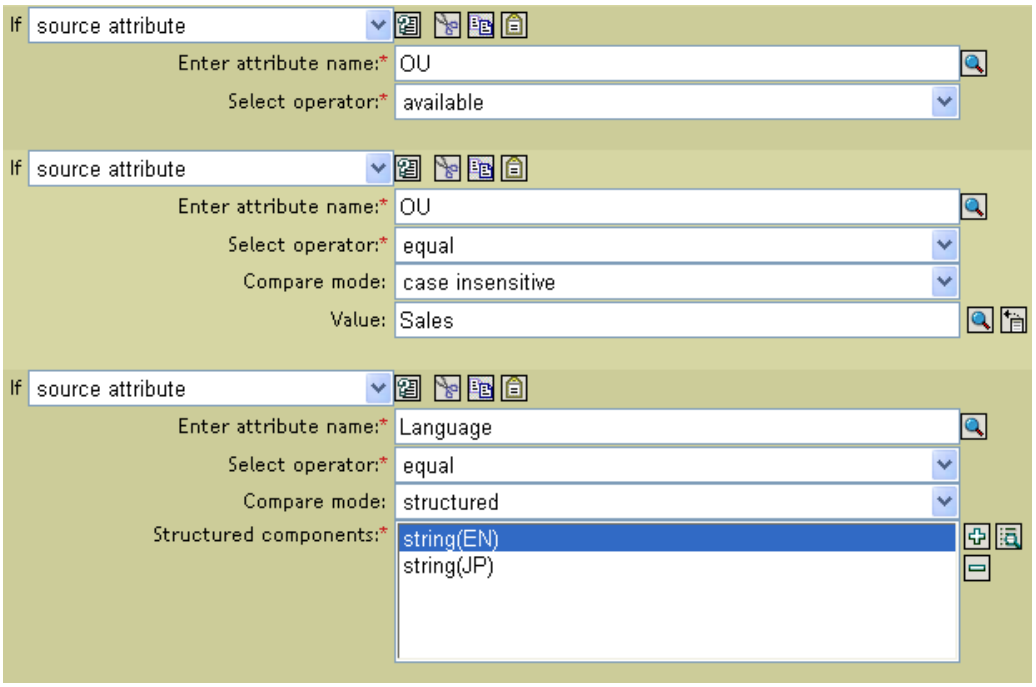
比較モード

比較モードを選択します。133 ページの「比較モード」を参照してください。

If Source Attribute（ソース属性の条件）

If Source Attribute（ソース属性の条件）は、ソースデータストアに含まれる現在のオブジェクトの属性値をテストします。

例



条件

演算子	条件が満たされる場合
available	ソースデータストアで、指定した属性に対する値が使用できる。
equal	ソースデータストアで、指定した属性に対する値が使用でき、指定した比較モードを使って比較した場合、指定した値に等しい。
not available	[available] が False を返す。
not equal	[equal] が False を返す。

フィールド

名前

選択した条件に対してテストするソース属性名を指定します。

演算子

条件テストタイプを選択します。

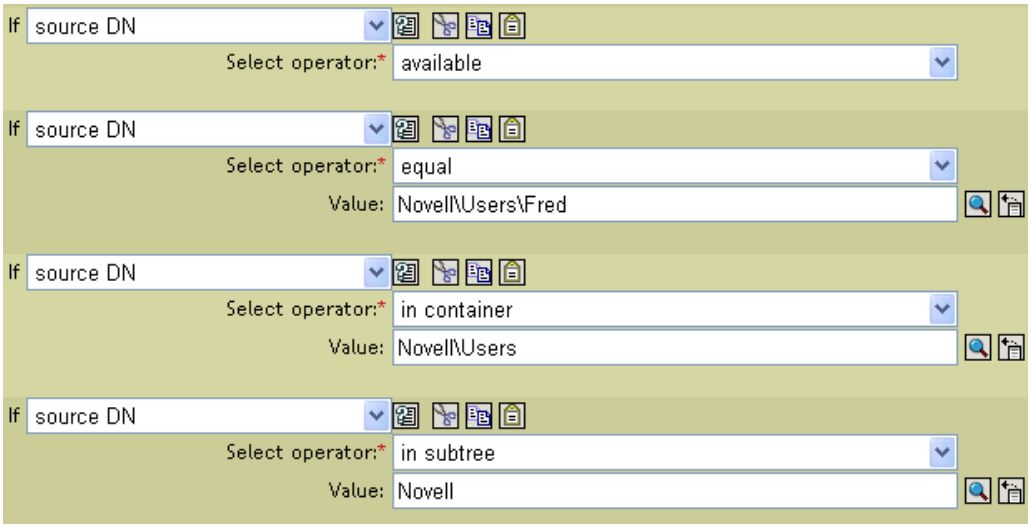
比較モード

比較モードを選択します。133 ページの「比較モード」を参照してください。

If Source DN（ソース DN の条件）

If Source DN（ソース DN の条件）は、現在の操作のソース DN をテストします。

例



条件

演算子	条件が満たされる場合
available	使用可能なソース DN がある。
equal	使用可能なソース DN があり、ソースデータストアの DN フォーマットに適したセマンティックを使って比較した場合、指定した値の内容に等しい。
in container	使用可能なソース DN があり、ソースデータストアの DN フォーマットに適したセマンティックを使って比較した場合、値によって指定されるコンテナ内のオブジェクトを示す。
in subtree	使用可能なソース DN があり、ソースデータストアの DN フォーマットに適したセマンティックを使って比較した場合、値によって指定されるサブツリー内のオブジェクトを示す。
not available	[available] が False を返す。
not equal	[equal] が False を返す。
not in container	[in container] が False を返す。
not in subtree	[in subtree] が False を返す。

フィールド

演算子

条件テストタイプを選択します。

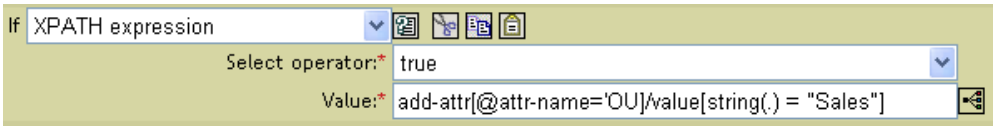
比較モード

比較モードを選択します。133 ページの「比較モード」を参照してください。

If Xpath Expression（Xpath 式の条件）

If Xpath Expression（Xpath 式の条件）は、XPATH 1.0 式の評価結果をテストします。

例



条件

演算子	条件が満たされる場合
true	XPATH 式が True と評価される。
false	[true] が False を返す。

フィールド

演算子

条件テストタイプを選択します。

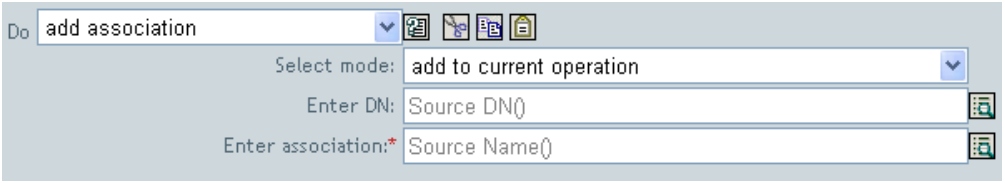
アクション

この節では、Policy Builder インタフェースで使用できるすべてのアクションの詳細について説明します。

Add Association（関連付けの追加）

このアクションは、add association（関連付けの追加）コマンドを eDirectory に送信します。

例



フィールド

モード

このアクションを現在の操作に追加するか、宛先データストアに直接書き込むかを選択します。

DN

Argument Builder を使用して、関連付けを受け取るオブジェクトの DN を指定します。

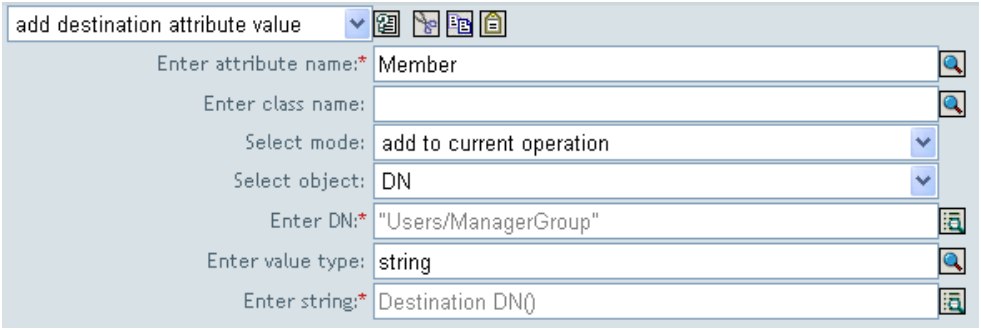
関連付け

Argument Builder を使用して関連付け値を指定します。

Add Destination Attribute Value（宛先属性値の追加）

このアクションは、指定した値を、宛先データストア内のオブジェクトにある名前付き属性に追加します。ターゲットオブジェクトは現在のオブジェクト、DN、または関連付けです。

例



フィールド

属性名

宛先データストアにあるターゲットオブジェクトに追加する属性の名前を指定します。

クラス名

（オプション）宛先データストアにあるターゲットオブジェクトのクラス名を指定します。オブジェクトが現在のオブジェクトと異なる場合、スキーママッピングのためにこの値が必要になることがあります。

モードの選択

このアクションを現在の操作の前または後に追加するか、宛先データストアに直接書き込むかを選択します。

オブジェクトの選択

宛先データストアにある、属性を受け取るオブジェクトを選択します。このオブジェクトは現在のオブジェクトでも、DN または関連付けにより指定されたオブジェクトでもかまいません。

値のタイプ

新しい属性値の構文を選択します。

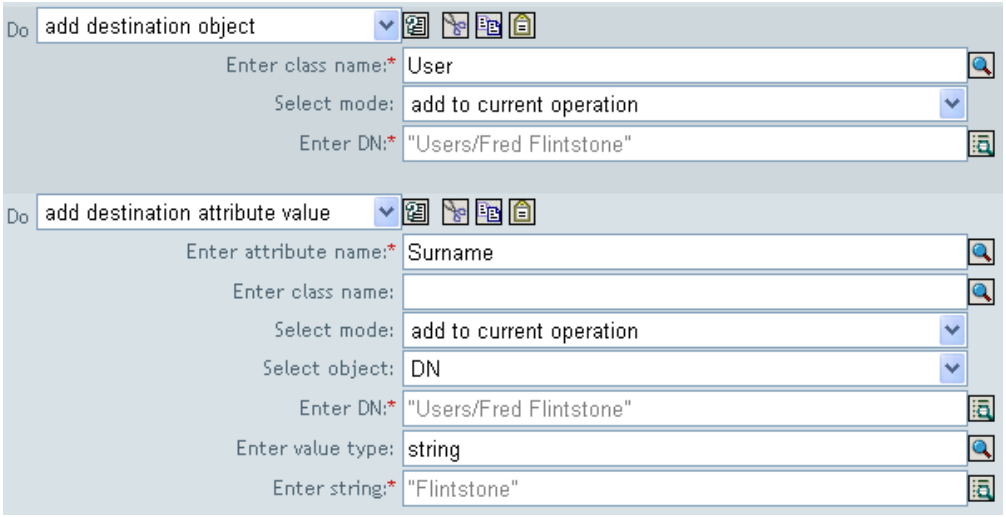
トークン

Argument Builder を使用して新しい属性の値を指定します。

Add Destination Object（宛先オブジェクトの追加）

このアクションは、指定したタイプのオブジェクトを、宛先データストア内に、[Enter DN field] で指定した名前と場所で作成します。オブジェクト作成時に追加する属性値は、以降の（49 ページ）Add Destination Attribute Value（宛先属性値の追加）アクションでは同じ DN を使って追加する必要があります。

例



フィールド

クラス名

宛先データストアに追加するオブジェクトのクラス名を指定します。

モード

このアクションを現在の操作の前または後に追加するか、宛先データストアに直接書き込むかを選択します。

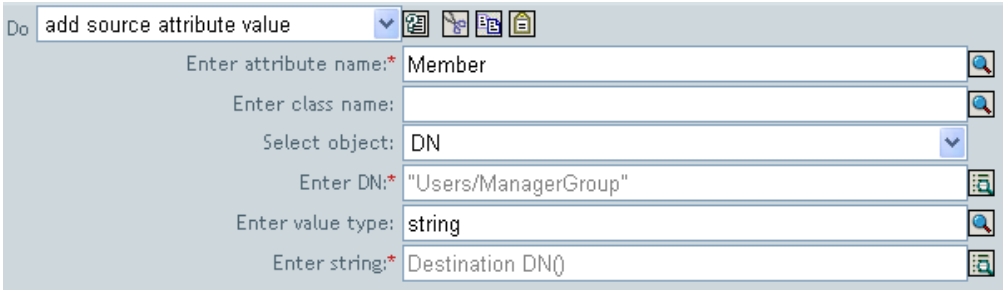
DN

宛先データストアに追加する新規オブジェクトの DN を指定します。

Add Source Attribute Value（ソース属性値の追加）

このアクションは、指定した値を、ソースデータストア内のオブジェクトにある指定した属性に追加します。ターゲットオブジェクトは現在のオブジェクト、DN、または関連付けです。

例



フィールド

属性名

ソースデータストアにあるターゲットオブジェクトに追加する属性の名前を指定します。

クラス名

（オプション）ソースデータストアにあるターゲットオブジェクトのクラス名を指定します。オブジェクトが現在のオブジェクトと異なる場合、スキーママッピングのためにこの値が必要になることがあります。

オブジェクト

ソースデータストアにあるターゲットオブジェクトを選択して属性を受け取ります。このオブジェクトは現在のオブジェクトでも、DN または関連付けにより指定されたオブジェクトでもかまいません。

値のタイプ

新しい属性値の構文を選択します。

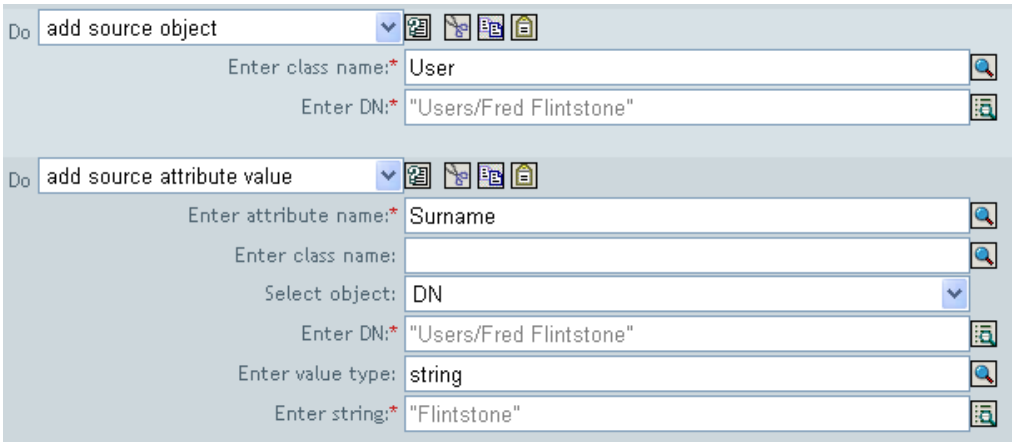
トークン

Argument Builder を使用して新しい属性の値を指定します。

Add Source Object（ソースオブジェクトの追加）

このアクションは、指定したタイプのオブジェクトをソースデータストアに作成します。オブジェクト作成時に追加する属性値は、以降の（51 ページ）Add Source Attribute Value（ソース属性値の追加）アクションでは同じ DN を使って追加する必要があります。

例



フィールド

クラス名

ソースデータストアに追加するオブジェクトのクラス名を指定します。

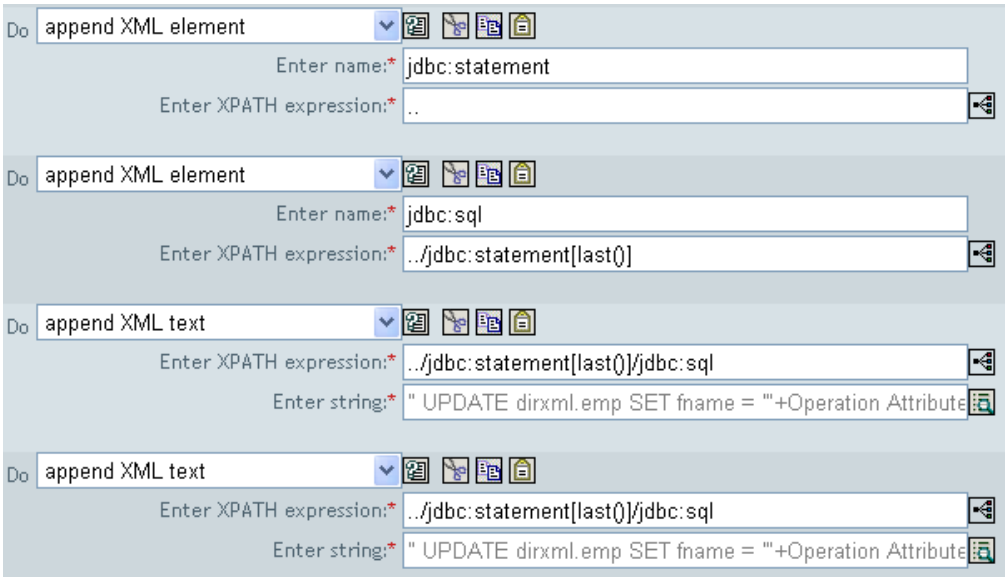
DN

ソースデータストアに追加する新規オブジェクトの DN を指定します。

Append XML Element（XML 要素の追加）

このアクションは、XPATH 式によって選択された要素のセットにカスタム要素を追加します。

例



フィールド

名前

XML 要素のタグ名。このポリシーにプレフィックスが定義されている場合は、名前にネームスペースのプレフィックスを含めることができます。

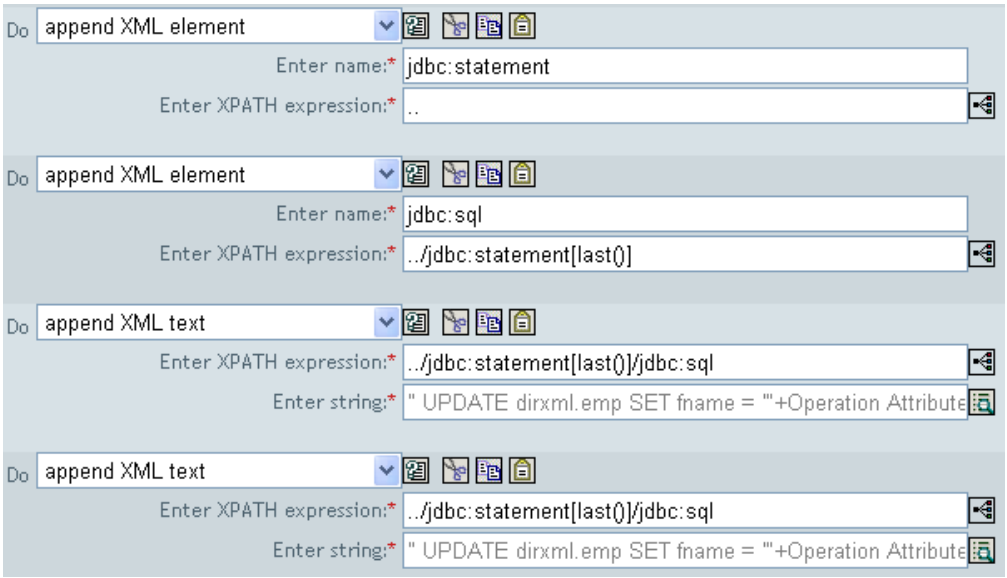
XPATH 式

新規要素の追加先となる要素が含まれるノードセットを返す XPATH 1.0 の式。

Append XML Text（XML テキストの追加）

このアクションは、XPATH 式によって選択された要素のセットに、指定したテキストを追加します。

例



フィールド

XPATH 式

新規要素の追加先となる要素が含まれるノードセットを返す XPATH 1.0 の式。

文字列

式によって選択された要素のセットに追加するテキスト。

Break（ブレーク）

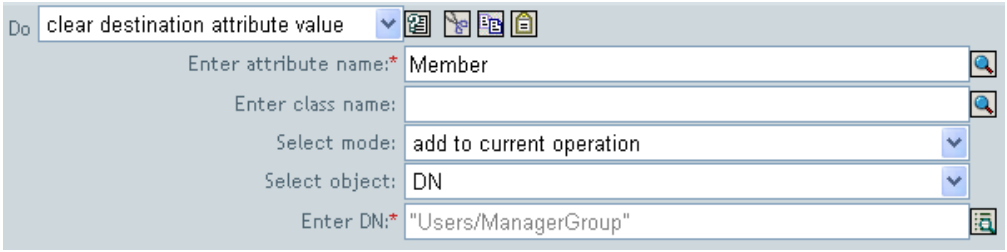
このアクションは、現在の操作が、現在のポリシー内の他のアクションまたはルールによって処理されないようにします。

例

Clear Destination Attribute Value（宛先属性値の消去）

このアクションは、名前付き属性の値すべてを宛先データストア内のオブジェクトから削除します。ターゲットオブジェクトは現在のオブジェクト、DN、または関連付けです。

例



フィールド

属性名

宛先データストアにあるターゲットオブジェクトに追加する属性の名前を指定します。

クラス名

（オプション）宛先データストアにあるターゲットオブジェクトのクラス名を指定します。オブジェクトが現在のオブジェクトと異なる場合、スキーママッピングのためにこの値が必要になることがあります。

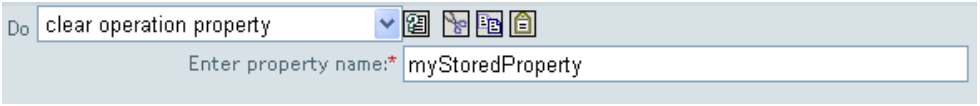
モード

このアクションを現在の操作の前または後に追加するか、宛先データストアに直接書き込むかを選択します。

Clear Operation Property（操作プロパティの消去）

このアクションは、指定した名前を持つ操作プロパティすべてを現在の操作から消去します。

例



フィールド

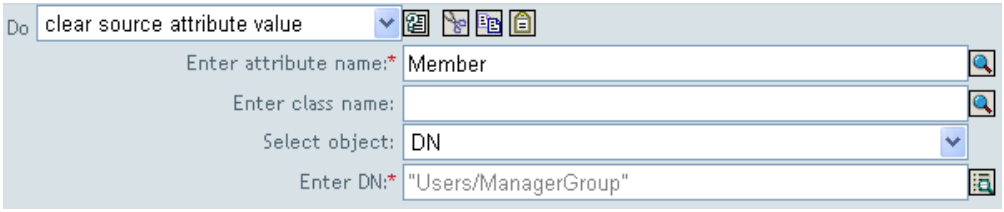
プロパティ名

消去する操作プロパティの名前を指定します。

Clear Source Attribute Value（ソース属性値の消去）

このアクションは、名前付き属性の値すべてをソースデータストア内のオブジェクトから削除します。ターゲットオブジェクトは現在のオブジェクト、DN、または関連付けです。

例



フィールド

属性名

ソースデータストアにあるターゲットオブジェクトに追加する属性の名前を指定します。

クラス名

（オプション）ソースデータストアにあるターゲットオブジェクトのクラス名を指定します。オブジェクトが現在のオブジェクトと異なる場合、スキーママッピングのためにこの値が必要になることがあります。

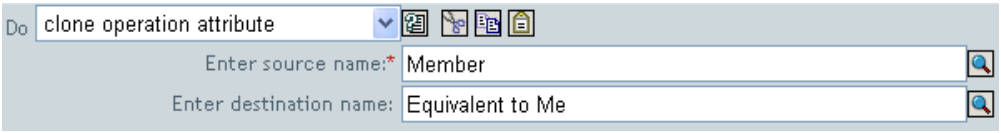
オブジェクトの選択

ソースデータストア内にある、属性を受け取るオブジェクトを選択します。このオブジェクトは現在のオブジェクトでも、DN または関連付けにより指定されたオブジェクトでもかまいません。

Clone Operation Attribute（操作属性のクローン）

このアクションは、指定したソース名に等しい属性名を持ち、現在の操作の子である要素すべてを、指定した宛先名に、属性名を指定した宛先名に設定した状態で複製します。

例



フィールド

ソース名

クローンを作成する属性名を指定します。

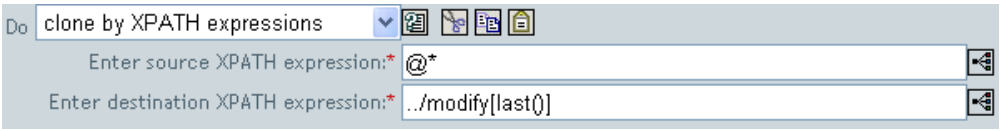
宛先名

クローンに渡す属性名を指定します。

Clone by Xpath Expressions（Xpath 式によるクローン）

このアクションは、ソースフィールドによって指定されたノードの詳細コピーを、宛先フィールドによって指定された要素のセットに追加します。

例



フィールド

Source XPATH Expression

新規要素の追加先となる要素が含まれるノードセットを返す XPATH 1.0 の式。

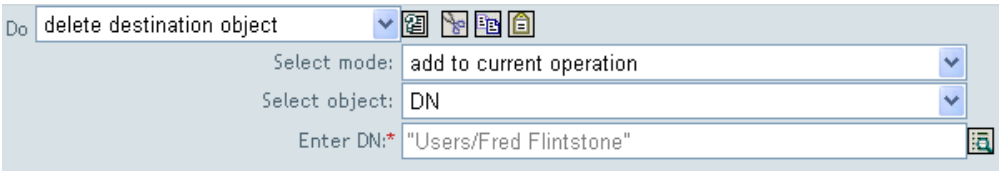
Destination XPATH Expression

クローンを作成するノードが含まれるノードセットを返す XPATH 1.0 の式。

Delete Destination Object（宛先オブジェクトの削除）

このアクションは、宛先データストアのオブジェクトを削除します。ターゲットオブジェクトは現在のオブジェクト、DN、または関連付けです。

例



フィールド

モード

このアクションを現在の操作の前または後に追加するか、宛先データストアに直接書き込むかを選択します。

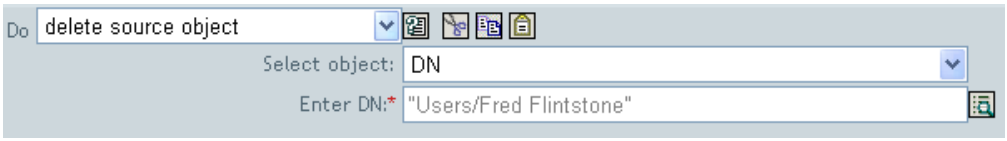
オブジェクト

宛先データストアの中から削除するターゲットオブジェクトを選択します。このオブジェクトは現在のオブジェクトでも、DN または関連付けにより指定されたオブジェクトでもかまいません。

Delete Source Object（ソースオブジェクトの削除）

このアクションは、ソースデータストアのオブジェクトを削除します。ターゲットオブジェクトは現在のオブジェクト、DN または関連付けのいずれかです。

例



フィールド

モード

このアクションを現在の操作の前または後に追加するか、宛先データストアに直接書き込むかを選択します。

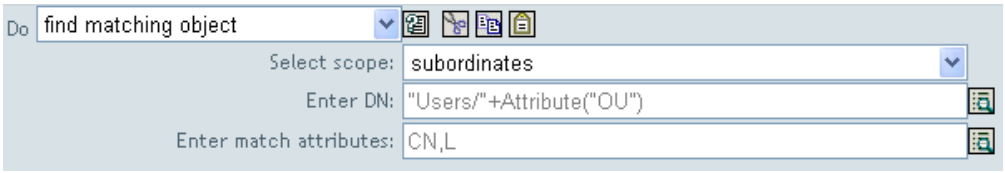
オブジェクト

ソースデータストアの中から削除するターゲットオブジェクトを選択します。このオブジェクトは現在のオブジェクトでも、DN または関連付けにより指定されたオブジェクトでもかまいません。

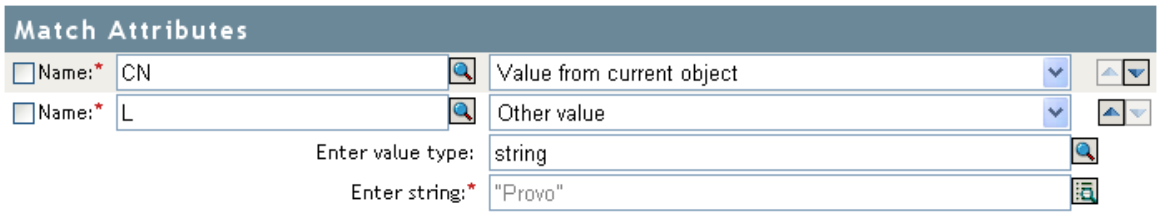
Find Matching Object（一致オブジェクトの検索）

このアクションは、宛先データストア内でクエリを実行して、適切な宛先 DN または適切な宛先関連付けを現在の操作に追加します。

例



次に、Argument Builder を使って一致する属性を検索する例を示します。



フィールド

スコープ

操作のスコープ。エントリ、従属、またはサブツリーを選択します。

DN

選択したスコープを使用して検索する場所の DN。

一致する属性

検索を正しく行うために一致しなければならない属性を設定します。

コメント

DN 引数は、スコープが［entry］の場合は必須ですが、それ以外の場合はオプションです。スコープが［substree］または［subordinates］の場合は、一致する属性が少なくとも 1 つ必要です。

スコープが［entry］の場合にクエリが検索属性をどのように処理するかは定義されていないので、Find Matching Object（一致オブジェクトの検索）の処理も未定義になります。

生成されたクエリには、選択したスコープに基づくスコープ属性が設定され、指定されている場合は、［Enter DN］フィールドの内容に設定された宛先 DN 属性が設定されます。また、現在のオブジェクトのクラス名に基づくクラス名属性と検索クラスも設定されます。

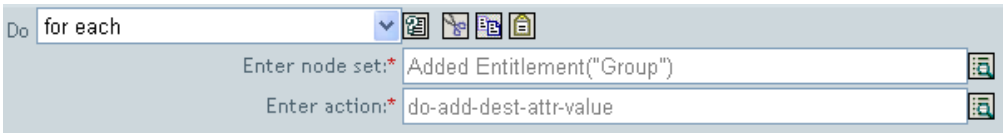
宛先データストアがアプリケーションの場合、関連付けが現在の操作に追加され、一致する検索結果すべてを返します。現在の操作に空でない関連付けがすでに存在する場合、クエリは実行されません。そのため、同じルール内で複数の Find Matching Object（一致オブジェクトの検索）アクションを記述できます。

宛先データストアが eDirectory の場合、現在の操作の宛先 DN 属性が設定されます。現在の操作に空でない宛先 DN 属性がすでに存在する場合、クエリは実行されません。そのため、同じルール内で複数の Find Matching Object（一致オブジェクトの検索）アクションを使用できます。返される結果が 1 つだけで、その結果が関連付けられていない場合、現在の操作の宛先 DN は、一致するオブジェクトのソース DN に設定されます。1 つの結果のみが返され、その結果がすでに関連付けられている場合、現在の操作の宛先 DN は、&#xFFFC という 1 文字に設定されます。複数の結果が返される場合、現在の操作の宛先 DN は、&#xFFFD という 1 文字に設定されます。

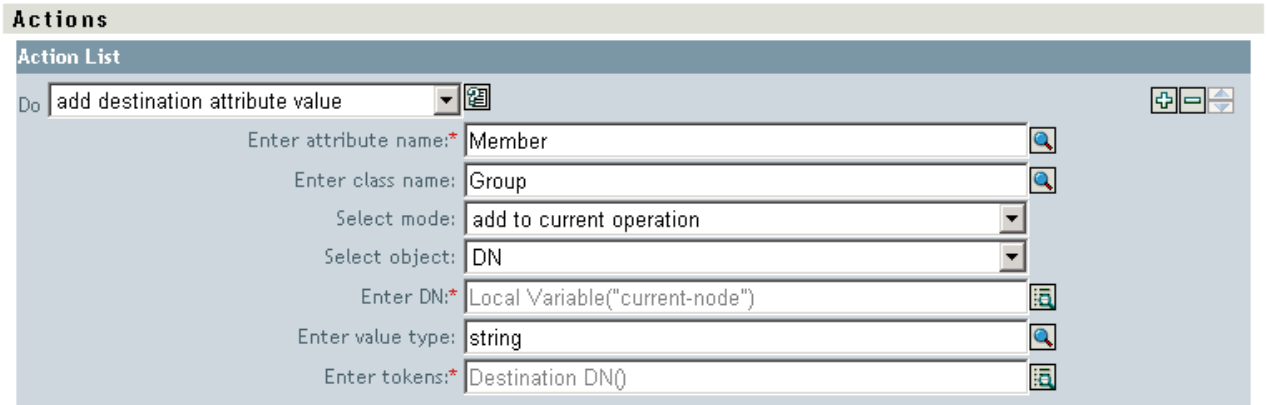
For Each（繰り返し）

このアクションは、指定したノードセット内の各ノードに対して、指定したアクションを 1 回ずつ繰り返します。

例



次に、[Argument Actions Builder] を使用してアクション引数を得る例を示します。



フィールド

ノードセット

指定したアクションを繰り返す対象のノードセット。

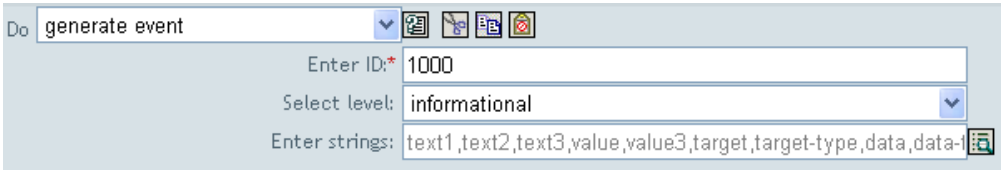
アクション

ノードセット内の各ノードに対して実行するアクション。

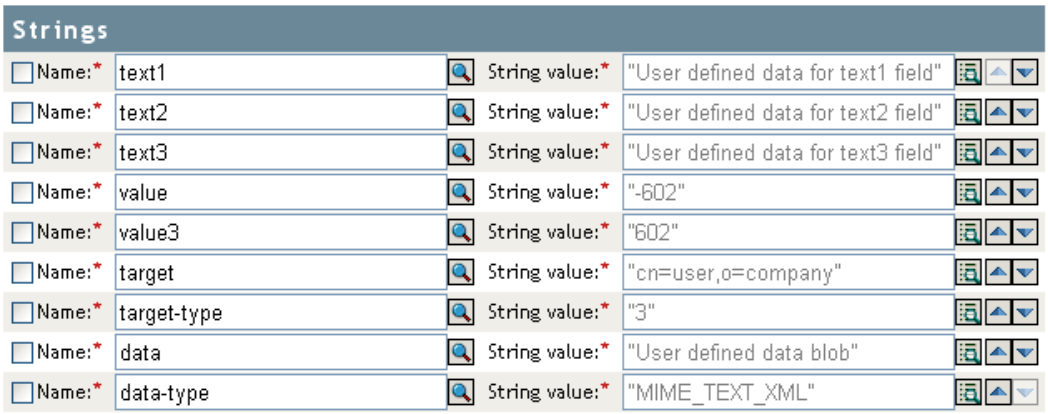
Generate Event（イベントの生成）

このアクションは、DirXML のユーザ定義イベントを Nsure™ Audit に送信します。

例



次に、Named String Builder を使用して文字列引数を得る例を示します。



フィールド

ID

イベントの ID。得られた値は、java.lang.Integer の parseInt メソッドを使って解析した場合に、1000 ～ 1999 の整数にならなければなりません。

レベル

イベントのレベル。

レベル	説明
log-emergency	DirXML エンジンまたはドライバがシャットダウンされるイベント。
log-alert	早急に注意が必要なイベント。
log-critical	DirXML のエンジンまたはドライバの一部が正常に動作しなくなるイベント。
log-error	DirXML またはドライバによって処理できるエラーを説明するイベント。
log-warning	問題を表さないネガティブなイベント。
log-notice	管理者が使い方や操作を理解または向上するのに使用できるイベント（ポジティブまたはネガティブ）。

レベル	説明
log-info	いずれかの重要度を持つポジティブイベント。
log-debug	サポートまたはエンジニアが DirXML エンジンまたはドライバの操作をデバッグするためのイベント。

文字列

イベントとともに含めるユーザ定義の文字列、整数、バイナリ値。これらの値は、Named String Builder を使用して得ることができます。

タグ	説明
target	処理対象のオブジェクト。
target-type	ターゲットに定義済みフォーマットを指定する整数。現在、target-type の定義済み値は次のとおりです。 ◆ 0 = なし ◆ 1 = スラッシュ表記 ◆ 2 = ドット表記 ◆ 3 = LDAP 表記
subTarget	処理対象のターゲットのサブコンポーネント。
text1	ここに入力したテキストは [text1] イベントフィールドに保存されます。
text2	ここに入力したテキストは [text2] イベントフィールドに保存されます。
text3	ここに入力したテキストは [text3] フィールドに保存されます。
value	ここに入力した数はすべて [value] イベントフィールドに保存されます。
value3	ここに入力した数は [value3] イベントフィールドに保存されます。
data	ここに入力したデータは [blob] イベントフィールドに保存されます。
data-type	データの MIME タイプ。すべてのリストについては logevents.h を参照してください。

コメント

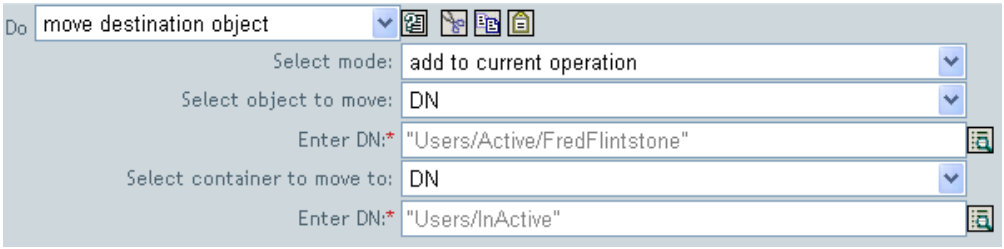
DirXML ユーザ定義イベント ID は、1000 ～ 1999 でなければなりません。有効なイベントレベルは次の表で定義されています。残りのイベントデータフィールドは、4 つの文字列要素と名前属性で提供されます。Nsure Audit イベント構造には、ターゲット、サブターゲット、3 つの文字列 (text1、text2、text3)、2 つの整数 (value、value3) および汎用フィールド (data) が含まれます。テキストフィールドは 256 バイトに制限されていますが、データフィールドには最大 3KB の情報を含めることができます。ただし、現在の環境でより大きなデータフィールドが有効になっている場合を除きます。

Policy Builder を使ったイベントの生成の詳細については、『*Identity Manager 2 Administration Guide (Identity Manager 2 管理者ガイド)*』の「Logging and Reporting Using Nsure Audit (Nsure Audit を使ったログおよびレポート)」の節を参照してください。

Move Destination Object（宛先オブジェクトの移動）

このアクションは、宛先データストアのオブジェクトを移動します。現在のオブジェクト、DN、または関連付けを選択して、DN または関連付けによって指定された別の場所へ移動します。

例



フィールド

モード

このアクションを現在の操作の前または後に追加するか、宛先データストアに直接書き込むかを選択します。

移動するオブジェクト

宛先データストアの中から移動するオブジェクトを選択します。このオブジェクトは現在のオブジェクトでも、DN または関連付けにより指定されたオブジェクトでもかまいません。

コンテナ

オブジェクトを受け取るコンテナを選択します。このコンテナは DN または関連付けによって指定されます。

Move Source Object（ソースオブジェクトの移動）

このアクションは、ソースデータストアのオブジェクトを移動します。現在のオブジェクト、DN、または関連付けを選択して、DN または関連付けによって指定された別の場所へ移動します。

例

フィールド

移動するオブジェクト

ソースデータストアの中から移動するオブジェクトを選択します。このオブジェクトは現在のオブジェクトでも、DN または関連付けにより指定されたオブジェクトでもかまいません。

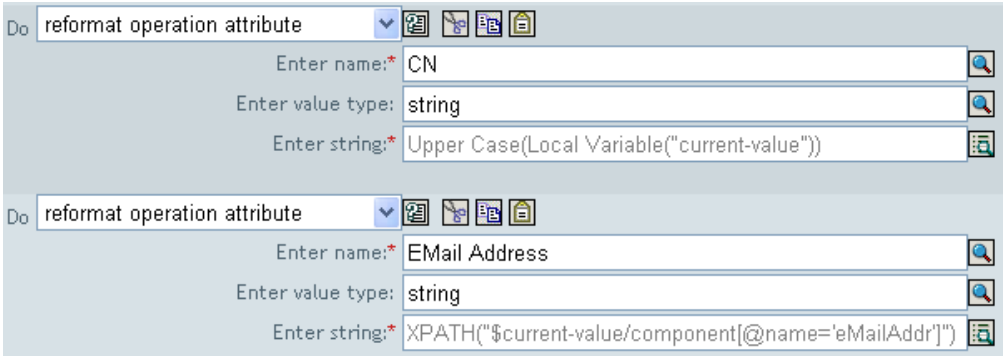
コンテナの選択

オブジェクトを受け取るコンテナを選択します。このコンテナは DN または関連付けによって指定されます。

Reformat Operation Attribute（操作属性の再フォーマット）

このアクションは、現在の操作内にある名前付き属性のすべての値を、指定した値に置き換えます。指定した値は、置き換える各値に対して 1 回評価され、ローカル変数 current-value は元の値に設定されます。

例



フィールド

名前

再フォーマットする属性の名前を指定します。

値のタイプ

新規属性値の構文を指定します。

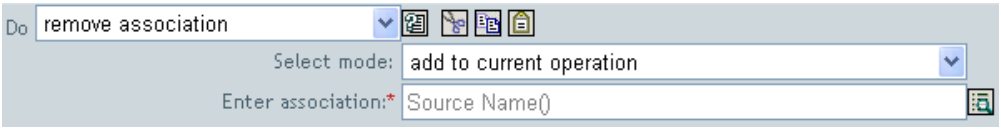
トークン

Argument Builder を使用して属性の新しいフォーマットを指定します。

Remove Association（関連付けの削除）

このアクションは、関連付けの削除コマンドを eDirectory に送信します。

例



フィールド

モード

このアクションを現在の操作の前または後に追加するか、宛先データストアに直接書き込むかを選択します。

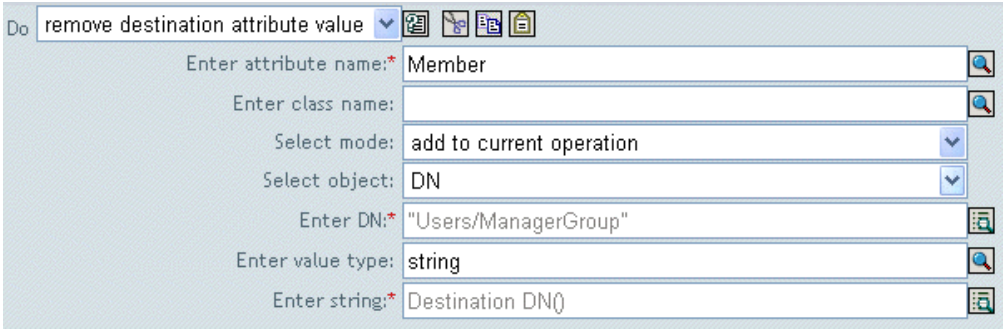
関連付け

Argument Builder を使用して関連付け値を指定します。

Remove Destination Attribute Value（宛先属性の削除）

このアクションは、指定した値を、宛先データストア内のオブジェクトにある名前付き属性から削除します。ターゲットオブジェクトは現在のオブジェクト、DN、または関連付けです。

例



フィールド

属性名

宛先データストアにあるターゲットオブジェクトに追加する属性の名前を指定します。

クラス名

（オプション）宛先データストアにあるターゲットオブジェクトのクラス名を指定します。オブジェクトが現在のオブジェクトと異なる場合、スキーママッピングのためにこの値が必要になることがあります。

モード

このアクションを現在の操作の前または後に追加するか、宛先データストアに直接書き込むかを選択します。

オブジェクトの選択

宛先データストアにあるターゲットオブジェクトを選択します。このオブジェクトは現在のオブジェクトでも、DN または関連付けにより指定されたオブジェクトでもかまいません。

値のタイプ

新規属性値の構文を指定します。

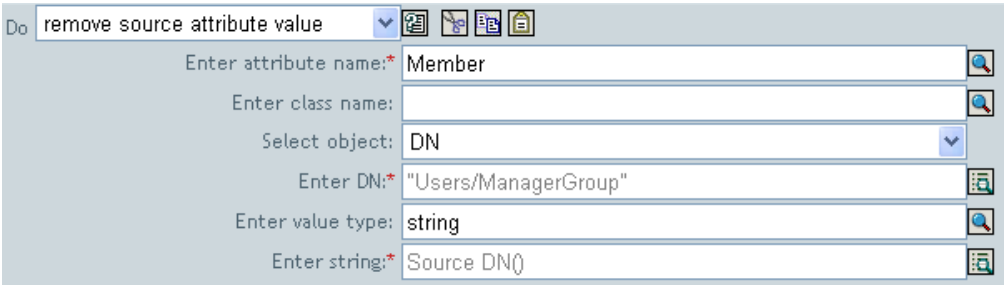
トークン

Argument Builder を使用して新しい属性の値を指定します。

Remove Source Attribute Value（ソース属性値の削除）

このアクションは、指定した値を、ソースデータストア内のオブジェクトにある名前付き属性から削除します。ターゲットオブジェクトは現在のオブジェクト、DN、または関連付けです。

例



フィールド

属性名

ソースデータストアにあるターゲットオブジェクトに追加する属性の名前を指定します。

クラス名

（オプション）ソースデータストアにあるターゲットオブジェクトのクラス名を指定します。オブジェクトが現在のオブジェクトと異なる場合、スキーママッピングのためにこの値が必要になることがあります。

オブジェクトの選択

宛先データストアにある、属性を受け取るオブジェクトを選択します。このオブジェクトは現在のオブジェクトでも、DN または関連付けにより指定されたオブジェクトでもかまいません。

値のタイプ

新規属性値の構文を指定します。

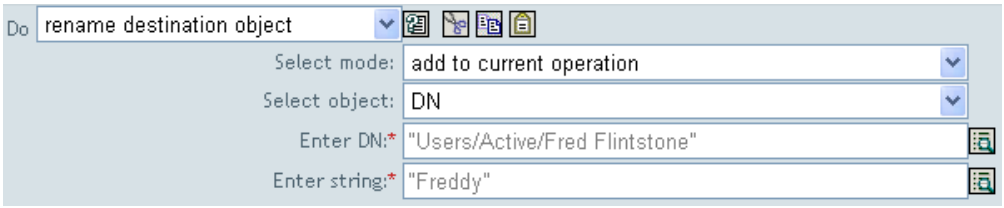
トークン

Argument Builder を使用して新しい属性の値を指定します。

Rename Destination Object（宛先オブジェクトの名前変更）

このアクションは、宛先データストアのオブジェクトを名前変更します。ターゲットオブジェクトは現在のオブジェクト、DN、または関連付けです。

例



フィールド

モード

このアクションを現在の操作の前または後に追加するか、宛先データストアに直接書き込むかを選択します。

オブジェクト

宛先データストアにあるターゲットオブジェクトを選択します。このオブジェクトは現在のオブジェクトでも、DN または関連付けにより指定されたオブジェクトでもかまいません。

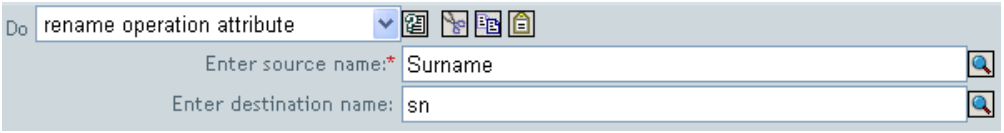
文字列

Argument Builder を使用して、宛先データストア内のオブジェクトの新しい名前を指定します。

Rename Operation Attribute（操作属性の名前変更）

このアクションは、指定した名前に等しい指定した属性を持ち、現在の操作の子である要素すべてで、宛先属性名に、指定した属性を設定します。

例



フィールド

ソース名

ソースデータストアでの属性名を指定します。

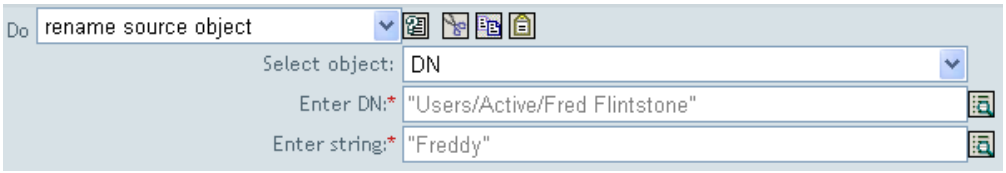
宛先名

宛先データストアでの属性名を指定します。

Rename Source Object（ソースオブジェクトの名前変更）

このアクションは、ソースデータストアのオブジェクトを、指定した名前に名前変更します。ターゲットオブジェクトは現在のオブジェクト、DN、または関連付けです。

例



フィールド

Select Object

ソースデータストアにあるターゲットオブジェクトを選択します。このオブジェクトは現在のオブジェクトでも、DN または関連付けにより指定されたオブジェクトでもかまいません。

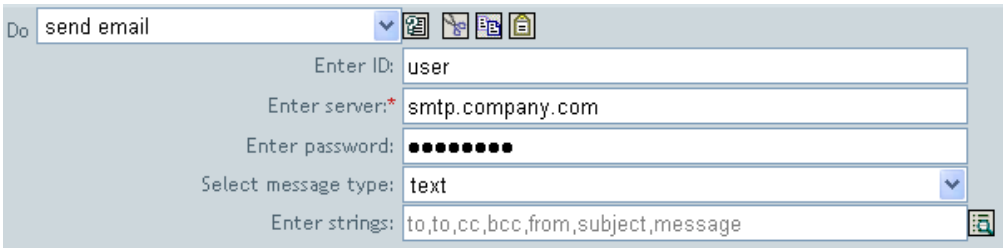
文字列

Argument Builder を使用して、ソースデータストアにあるオブジェクトの新しい名前を指定します。

Send Email（電子メールの送信）

このアクションは、指定したサーバに電子メール通知を送信します。SMTP サーバでの認証に使用するオプションの資格情報は、ID とパスワードで指定します。

例



次に、Named String Builder を使用して文字列引数を得る例を示します。

フィールド

ID

（オプション）メッセージを送信する SMTP システムのユーザ ID。

サーバ

SMTP サーバ名。

パスワード

（オプション）SMTP サーバアカウントパスワード。

警告： パスワードの属性値はクリアテキストで保存されます。

タイプ

電子メールのメッセージタイプを選択します。

文字列

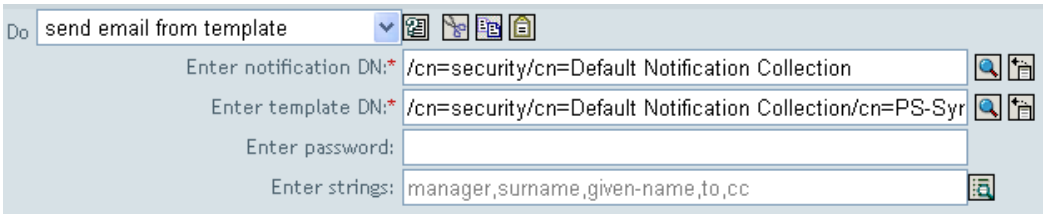
これらの値には、さまざまな電子メールアドレス、件名、およびメッセージが含まれます。次の表は、有効な指定文字列引数を示します。

文字列名	説明
to	アドレスを電子メール受信者リストに追加します。複数のインスタンスを追加できます。
cc	アドレスを電子メール受信者リストの cc に追加します。複数のインスタンスを追加できます。
bcc	アドレスを電子メール受信者リストの bcc に追加します。複数のインスタンスを追加できます。
from	送信者の電子メールアドレスとして使用するアドレスを指定します。
reply-to	電子メールメッセージの返信アドレスとして使用するアドレスを指定します。
subject	電子メールの件名を指定します。
message	電子メールメッセージの内容を指定します。
encoding	電子メールメッセージに使用する文字エンコードを指定します。

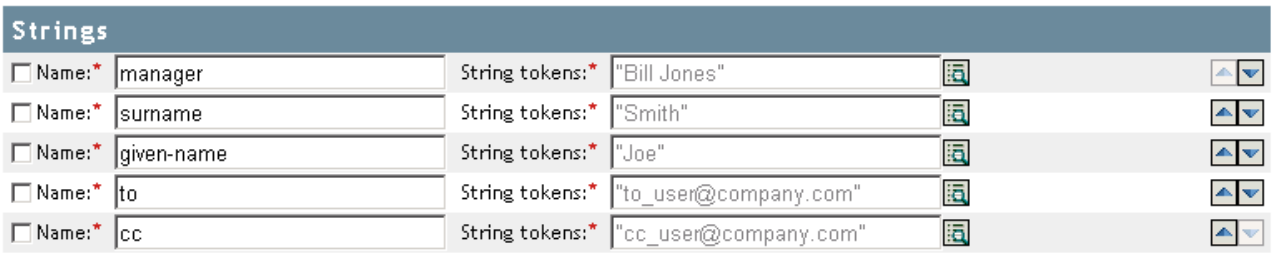
Send Email From Template（テンプレートからの電子メールの送信）

このアクションは、SMTP 通知設定オブジェクト、電子メールテンプレートオブジェクト、および置換トークンを使用して電子メール通知を生成します。

例



次に、Named String Builder を使用して文字列引数を得る例を示します。



フィールド

通知 DN

SMTP 通知設定オブジェクトのスラッシュ形式の DN。

テンプレート DN

電子メールテンプレートオブジェクトのスラッシュ形式の DN。

パスワード

（オプション）SMTP サーバアカウントパスワード。

警告： パスワードの属性値はクリアテキストで保存されます。

文字列

電子メールメッセージの置換トークン。次の表は、さまざまな電子メールアドレスを指定する予約済みの置換トークンを示します。

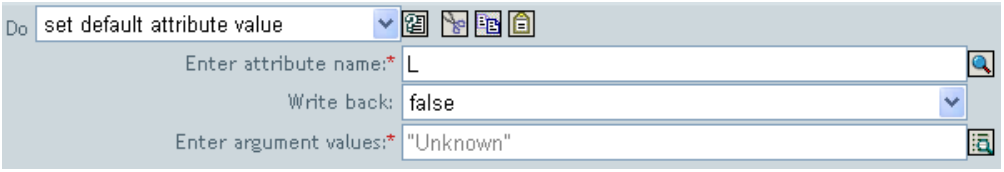
文字列名	説明
to	アドレスを電子メール受信者リストに追加します。複数のインスタンスを追加できます。
cc	アドレスを電子メール受信者リストの cc に追加します。複数のインスタンスを追加できます。

文字列名	説明
bcc	アドレスを電子メール受信者リストの bcc に追加します。複数のインスタンスを追加できます。
reply-to	電子メールメッセージの返信アドレスとして使用するアドレスを指定します。
encoding	電子メールメッセージに使用する文字エンコードを指定します。

Set Default Attribute Value（デフォルト属性値の設定）

このアクションは、該当する属性に値が存在しない場合でも、名前付き属性に対して指定した値を現在の操作に追加します。このアクションは、現在の操作が［add］の場合にのみ有効です。［write-back］が［true］の場合は、ソースオブジェクトにもデフォルト値が書き込まれます。

例



フィールド

属性名

宛先データストアにあるターゲットオブジェクトに追加する属性の名前を指定します。

ライトバック

ライトバックを［true］に設定した場合、デフォルト値はソースオブジェクトにも書き込まれます。

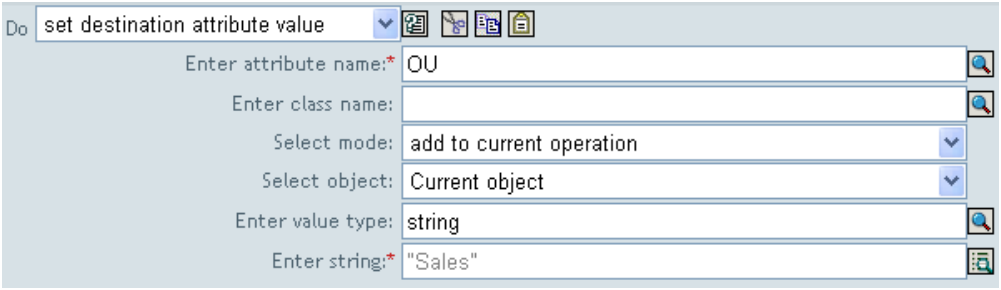
値

Argument Builder を使用して属性のデフォルト値を指定します。

Set Destination Attribute Value（宛先属性値の設定）

このアクションは、指定した値を、宛先データストアにあるオブジェクトの名前付き属性に追加し、その属性の他の値はすべて削除します。ターゲットオブジェクトは現在のオブジェクト、DN、または関連付けです。

例



フィールド

属性名

宛先データストアにあるターゲットオブジェクトに追加する属性の名前を指定します。

クラス名

（オプション）宛先データストアにあるターゲットオブジェクトのクラス名を指定します。オブジェクトが現在のオブジェクトと異なる場合、スキーママッピングのためにこの値が必要になることがあります。

モード

このアクションを現在の操作の前または後に追加するか、宛先データストアに直接書き込むかを選択します。

オブジェクト

ソースデータストアにある、属性を受け取るターゲットオブジェクトを選択します。このオブジェクトは現在のオブジェクトでも、DN または関連付けにより指定されたオブジェクトでもかまいません。

値のタイプ

属性値の構文を選択します。

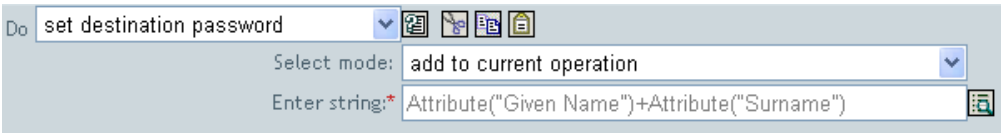
トークン

Argument Builder を使用して属性の値を指定します。

Set Destination Password（宛先パスワードの設定）

このアクションは、指定した値を、宛先データストア内の現在のオブジェクトのパスワードとして設定します。

例



フィールド

モード

このアクションを現在の操作の前または後に追加するか、宛先データストアに直接書き込むかを選択します。

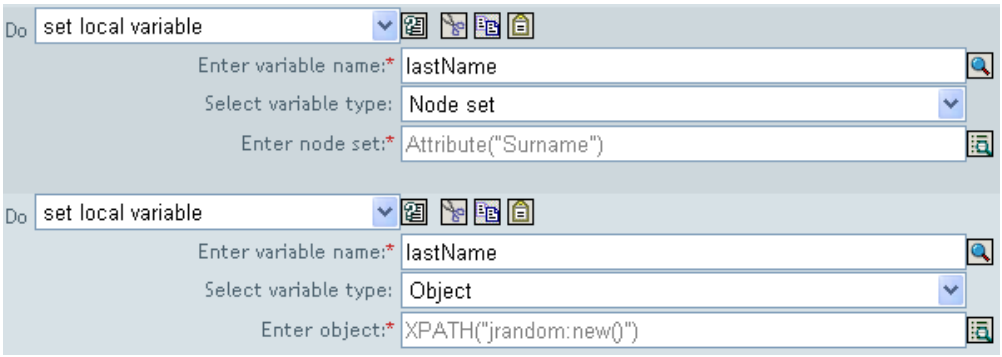
文字列

Argument Builder を使用してパスワードの値を指定します。

Set Local Variable（ローカル変数の設定）

このアクションは、指定した名前を持つローカル変数を、指定した文字列値、指定した XPATH 1.0 ノードセット、または指定した Java* オブジェクトに設定します。

例



フィールド

変数名

新規ローカル変数の名前を指定します。

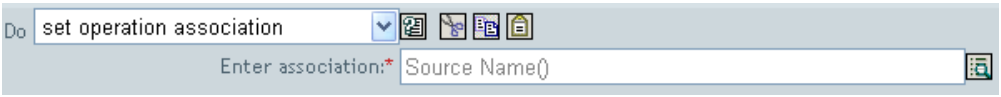
変数タイプ

追加するローカル変数のタイプを選択します。タイプは文字列、XPATH 1.0 ノードセット、または Java オブジェクトのいずれかです。

Set Operation Association（操作の関連付けの設定）

このアクションは、現在の操作の関連付け値を指定した値に設定します。

例



フィールド

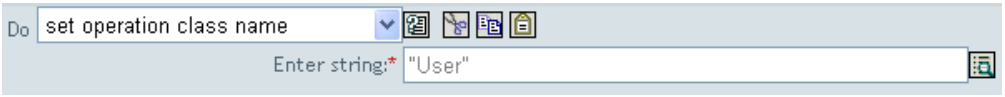
関連付け

新規関連付け値を設定します。

Set Operation Class Name（操作クラス名の設定）

このアクションは、現在の操作のオブジェクトクラス名を指定した値に設定します。

例



フィールド

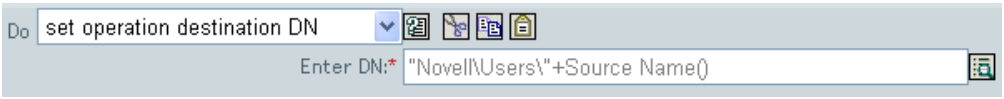
文字列

新規クラス名を指定します。

Set Operation Destination DN（操作の宛先 DN の設定）

このアクションは、現在の操作の宛先 DN を指定した値に設定します。

例



フィールド

DN

新規宛先 DN を指定します。

Set Operation Property（操作プロパティの設定）

このアクションは、現在の操作に、指定した名前と値で操作プロパティを作成します。操作プロパティは操作内に保存される名前付きの値で、通常は、操作結果を処理するポリシーで必要な追加コンテキストを提供するために使用されます。

例

フィールド

プロパティ名

新規操作プロパティの名前を設定します。

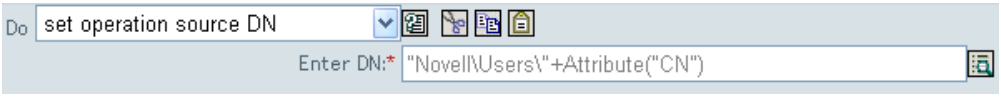
文字列

新規操作プロパティの値を設定します。

Set Operation Source DN（操作のソース DN の設定）

このアクションは、現在の操作のソース DN を指定した値に設定します。

例



フィールド

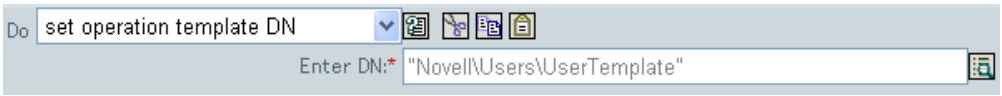
DN

新規ソース DN を設定します。

Set Operation Template DN（操作のテンプレート DN の設定）

このアクションは、現在の操作のテンプレート DN を指定した値に設定します。このアクションは現在の操作が［add］の場合にのみ有効です。

例



フィールド

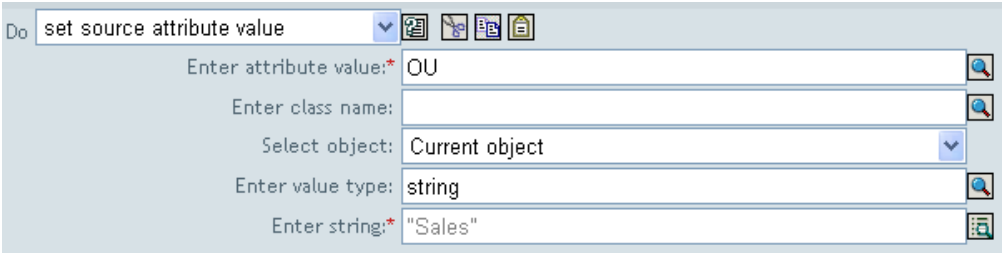
DN

新規テンプレート DN を指定します。

Set Source Attribute Value（ソース属性値の設定）

このアクションは、指定した値を、ソースデータストア内のオブジェクトにある名前付き属性に追加し、その属性の他の値はすべて削除します。ターゲットオブジェクトは現在のオブジェクト、DN、または関連付けです。

例



フィールド

属性名

ソースデータストアにあるターゲットオブジェクトに追加する属性の名前を指定します。

クラス名

（オプション）ソースデータストアにあるターゲットオブジェクトのクラス名を指定します。オブジェクトが現在のオブジェクトと異なる場合、スキーママッピングのためにこの値が必要になることがあります。

オブジェクト

ソースデータストアにある、属性を受け取るターゲットオブジェクトを選択します。このオブジェクトは現在のオブジェクトでも、DN または関連付けにより指定されたオブジェクトでもかまいません。

値のタイプ

属性値の構文を選択します。

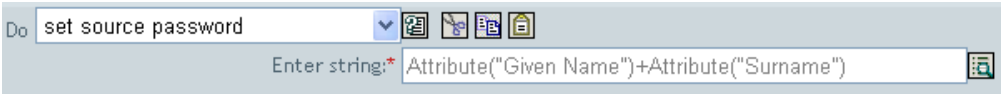
トークン

Argument Builder を使用して属性の値を指定します。

Set Source Password（ソースパスワードの設定）

このアクションは、指定した値を、ソースデータストア内の現在のオブジェクトのパスワードとして設定します。

例



フィールド

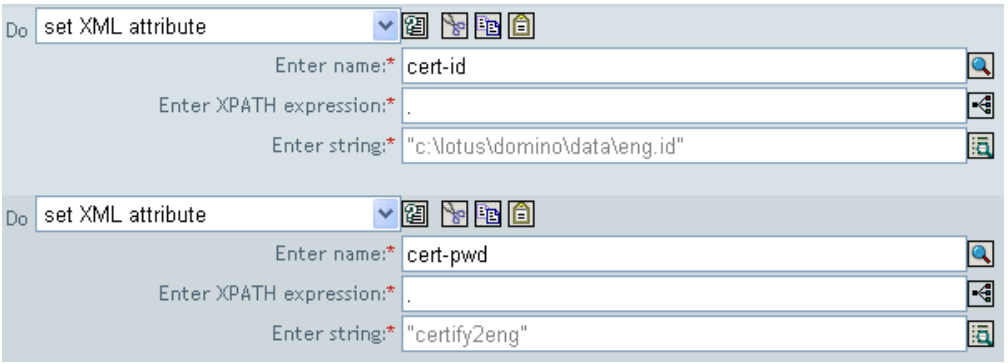
文字列

Argument Builder を使用してソースパスワードの値を指定します。

Set XML Attribute（XML 属性の設定）

このアクションは、名前属性で指定されているカスタム XML 属性を、XPATH 式によって選択された要素のセットに設定します。

例



フィールド

名前

XML 属性のタグ名。このポリシーにプレフィックスが定義されている場合は、名前にネームスペースのプレフィックスを含めることができます。

XPATH 式

XML 属性を設定する要素が含まれるノードセットを返す XPATH 1.0 の式。

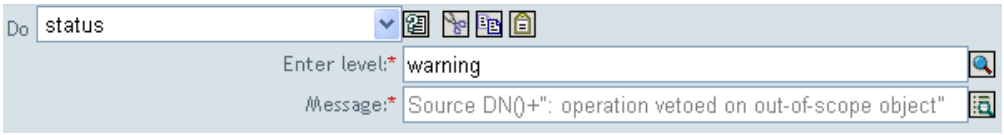
文字列

Argument Builder を使用して XML 属性値を指定します。

Status（ステータス）

このアクションは、指定したレベルとメッセージを使用してステータス通知を生成します。

例



フィールド

レベル

通知のステータスレベルを指定します。

メッセージ

Argument Builder を使用してステータスメッセージを指定します。

コメント

レベルが［retry］の場合、ポリシーはすぐに入力文書の処理を停止し、現在処理中のイベントの再試行をスケジュールします。

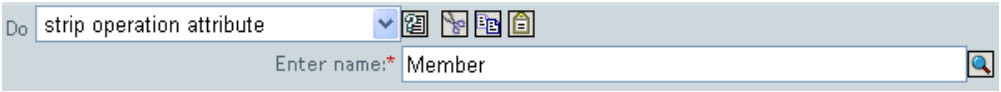
レベルが［fatal］の場合、ポリシーはすぐに入力文書の処理を停止し、ドライバのシャットダウンを開始します。

現在の操作にイベント ID があれば、そのイベント ID がステータス通知に使用されます。イベント ID がない場合、イベント ID はレポートされません。

Strip Operation Attribute（操作属性の除去）

このアクションは、指定した名前に等しい指定属性名を持ち、現在の操作の子であるすべての要素を現在の操作から除去します。

例



フィールド

名前

現在の操作から除去する属性の名前を指定します。

Strip Xpath（Xpath の除去）

このアクションは、XPATH 1.0 式によって選択されたノードを現在の操作から削除します。式はノードセットに評価される必要があります。

例

フィールド

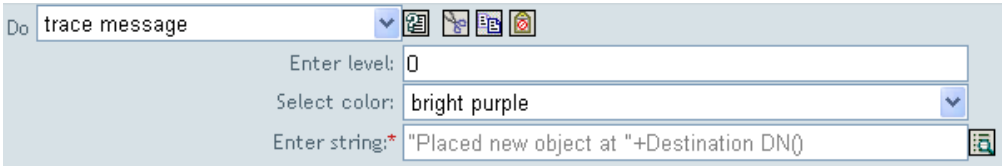
XPATH 式

現在の操作から削除する要素が含まれるノードセットを返す XPATH 1.0 の式。

Trace Message（トレースメッセージ）

このアクションは、指定した文字列を DSTRACE に選択した色で送信します。メッセージを表示するには、指定したトレースレベルが現在 DSTRACE で選択されているレベルと同じか、またはそれ以下でなければなりません。

例



フィールド

レベル

メッセージのトレースレベルを入力します。デフォルトは 0 です。

色

トレースメッセージの色を選択します。

文字列

トレースメッセージの値を指定します。

Veto（拒否）

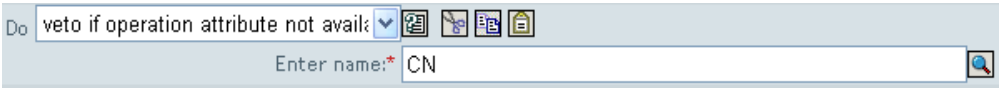
このアクションは、現在の操作をキャンセルします。

例

Veto If Operation Attribute Not Available（属性値が利用できない場合は拒否）

このアクションは、現在の操作で指定した属性が使用できない場合、現在の操作をキャンセルします。

例



フィールド

名前

拒否を実行する前に使用可能かどうかを確認する属性の名前を指定します。

名詞

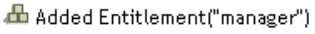
この節では、Policy Builder インタフェースで使用できるすべての名詞の詳細について説明します。

(100 ページ) Added Entitlement（追加されたエンタイトルメント）
(101 ページ) Association（関連付け）
(102 ページ) Attribute（属性）
(103 ページ) Class Name（クラス名）
(104 ページ) Destination Attribute（宛先属性）
(105 ページ) Destination DN（宛先 DN）
(106 ページ) Destination Name（宛先名）
(107 ページ) Entitlement（エンタイトルメント）
(108 ページ) Global Configuration Value（グローバル設定値）
(109 ページ) Local Variable（ローカル変数）
(110 ページ) Named Password（名前付きパスワード）
(111 ページ) Operation（操作）
(112 ページ) Operation Attribute（操作属性）
(113 ページ) Operation Property（操作プロパティ）
(114 ページ) Password（パスワード）
(115 ページ) Removed Attribute（削除された属性）
(116 ページ) Removed Entitlement（削除されたエンタイトルメント）
(117 ページ) Source Attribute（ソース属性）
(118 ページ) Source DN（ソース DN）
(119 ページ) Source Name（ソース名）
(120 ページ) Text（テキスト）
(121 ページ) Unique Name（固有名）
(123 ページ) Unmatched Source DN（一致しないソース DN）
(124 ページ) XPath

Added Entitlement（追加されたエンタイトルメント）

この名詞は、現在の操作に追加された名前付きエンタイトルメントの値まで拡張します。

例



フィールド

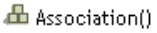
名前

エンタイトルメントの名前。

Association（関連付け）

この名詞は、現在の操作で指定された関連付け値まで拡張します。

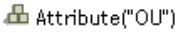
例



Attribute（属性）

この名詞は、現在の操作で指定された属性値まで拡張します。

例



フィールド

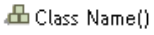
名前

属性の名前。

Class Name（クラス名）

この名詞は、現在の操作で指定されたオブジェクトクラス名まで拡張します。

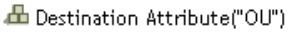
例



Destination Attribute（宛先属性）

この名詞は、指定された属性値まで拡張します。

例



フィールド

クラス名

宛先データストア内の、読み込むオブジェクトのクラス名。これは、オブジェクトが現在のオブジェクトと異なる場合に必要になることがあります。

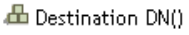
名前

属性の名前。

Destination DN（宛先 DN）

この名詞は、現在の操作または操作の一部で指定された宛先 DN まで拡張します。

例



フィールド

変換

［true］は、ソースデータストアの DN フォーマットに変換します。

開始

次のいずれかで始まるセグメントインデックス。

- 0 は最もルートに近いセグメント
- >0 は最もルートに近いセグメントからのオフセット
- -1 は最もリーフに近いセグメント
- <-1 は最もリーフに近いセグメントから最もルートに近いセグメントへのオフセット

長さ

含める DN セグメントの数。負の数は、（セグメントの総数 + 長さ）+ 1 と解釈されます（たとえば、5 つのセグメントで構成される DN の場合、長さ -1 であれば (5 + (-1)) + 1 = 5)、長さ -2 であれば (5 + (-2)) + 1 = 4 など）。

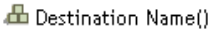
コメント

開始および長さがデフォルト値 {0、-1} に設定されている場合は、DN 全体が使用されます。それ以外の場合は、開始および長さで指定された DN の一部のみが使用されます。DN のフォーマットは、ソース DN フォーマットへの変換が［true］に設定されている場合、自動的にソースデータストアのフォーマットに変換されます。

Destination Name（宛先名）

これは、現在の操作で指定された宛先 DN の非修飾の相対識別名（RDN）まで拡張します。

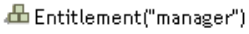
例



Entitlement（エンタイトルメント）

この名詞は、現在のオブジェクトに対する指定エンタイトルメントの値まで拡張します。

例



フィールド

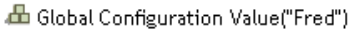
名前

エンタイトルメントの名前。

Global Configuration Value（グローバル設定値）

この名詞は、指定したグローバル設定変数の値まで拡張します。

例



フィールド

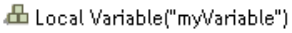
名前

グローバル設定値の名前。

Local Variable（ローカル変数）

この名詞は、名前付きローカル変数の値まで拡張します。

例



フィールド

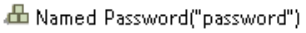
名前

ローカル変数の名前。

Named Password（名前付きパスワード）

この名詞は、ドライバから指定パスワードまで拡張します。

例



フィールド

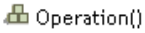
名前

パスワードの名前。

Operation（操作）

この名詞は、現在の操作の名前まで拡張します。

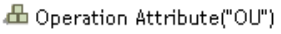
例



Operation Attribute（操作属性）

この名詞は、現在の操作（属性の追加、値の追加、または属性）から指定した属性の値まで拡張します。

例



フィールド

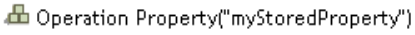
名前

現在の操作からの属性名。

Operation Property（操作プロパティ）

この名詞は、現在の操作に関する指定された操作プロパティの値まで拡張します。

例



フィールド

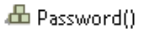
名前

プロパティの名前。

Password（パスワード）

この名詞は、現在の操作で指定されたパスワードまで拡張します。

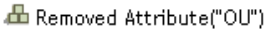
例



Removed Attribute（削除された属性）

この名詞は、現在の操作（削除属性）で削除された指定属性値まで拡張します。

例



フィールド

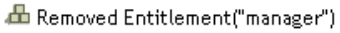
名前

削除された属性の名前。

Removed Entitlement（削除されたエンタイトルメント）

この名詞は、現在の操作で削除された指定エンタイトルメントの値まで拡張します。

例



フィールド

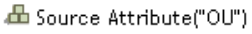
名前

エンタイトルメントの名前。

Source Attribute（ソース属性）

この名詞は、ソースデータストアにある、現在のオブジェクト、DN、または関連付けからの指定した属性値まで拡張します。

例



フィールド

クラス名

ソースデータストア内の、読み込むオブジェクトのクラス名。これは、オブジェクトが現在のオブジェクトと異なる場合に必要になることがあります。

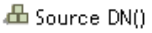
名前

属性の名前。

Source DN（ソース DN）

この名詞は、現在の操作または操作の一部で指定されたソース DN まで拡張します。

例



フィールド

変換

［true］は、宛先データストアの DN フォーマットに変換します。

開始

次のいずれかで始まるセグメントインデックス。

- 0 は最もルートに近いセグメント
- >0 は最もルートに近いセグメントからのオフセット
- -1 は最もリーフに近いセグメント
- <-1 は最もリーフに近いセグメントから最もルートに近いセグメントへのオフセット

長さ

含める DN セグメントの数。負の数は、（セグメント総数 + 長さ）+ 1 と解釈されます（たとえば、5 つのセグメントで構成される DN の場合、長さ -1 であれば (5 + (-1)) + 1 = 5)、長さ -2 であれば (5 + (-2)) + 1 = 4 など）。

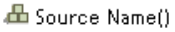
コメント

開始および長さがデフォルト値 {0、-1} に設定されている場合は、DN 全体が使用されます。それ以外の場合は、開始および長さで指定された DN の一部のみが使用されます。DN のフォーマットは、変換の属性が［True］に設定されている場合、宛先データストアのフォーマットに変換されます。

Source Name（ソース名）

これは、現在の操作で指定されたソース DN の非修飾の相対識別名（RDN）まで拡張します。

例



Text（テキスト）

この名詞は、指定したテキストまで拡張します。

例

フィールド

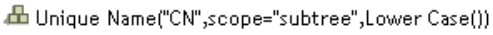
テキスト

テキスト値を指定します。

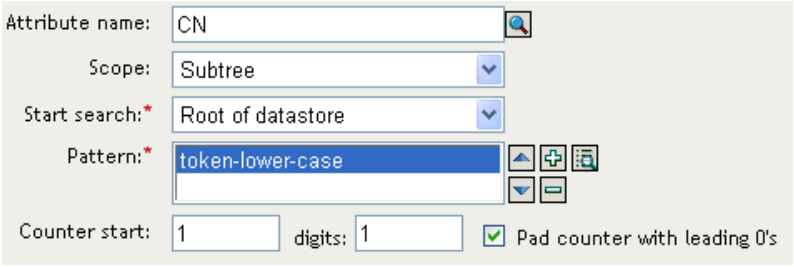
Unique Name（固有名）

この名詞は、指定した条件に従って宛先データストアで固有のパターンベースの名前まで拡張します。

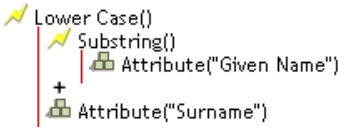
例



Unique Name（固有名）引数を構築する［Editor］ウィンドウの例は次のとおりです。



固有名を指定するために次のパターンを構築しています。



このパターンでは固有な名前が生成されない場合、カウンタ開始から 1 桁の数字が、指定した桁数まで追加されます。この例では、エラーが発生する前に、追加された桁によって 9 個の追加固有名が生成されます (pattern1 ～ pattern9)。

フィールド

名前

固有かどうかを確認するための属性の名前。

スコープ

固有かどうかを確認するスコープ。デフォルトのスコープはサブツリーです。

検索開始

検索の開始点を選択します。開始点はデータストアのルート、あるいは DN または関連付けによって指定されたルートのいずれかになります。

パターン

Argument Builder を使用して、固有な値を生成するのに使うパターンを指定します。

カウンタ開始

カウンタを開始する数値で、デフォルトは 1 です。

桁

カウンタの桁数で、デフォルトは 1 です。[Pad counter with leading 0's] をチェックすると、先頭に 0 を付けて桁長を一致させます。たとえば、桁数が 3 の場合、最初の固有値には 001 が追加され、次の固有値には 002 というようになります。

コメント

指定した各パターンで、宛先データストアに対して名前属性の値のクエリが実行されます。これは、DN、関連付け、またはクエリのベースとなるデータストアのルート、および選択したスコープを使用して行われます。

指定した各パターンは、インスタンスを返さない値が見つかるまで、指定した順序で試行されます。

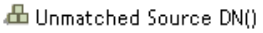
指定した値がすべて試行されると、最後の値にはカウンタが追加され、その値が、クエリがインスタンスを返さなくなるまで繰り返し試行されます (試行されるたびにカウンタが増えます)。デフォルトでは、カウンタは 1 から始まり、パッドされていません。[Counter start] フィールドを使用すると、カウンタが異なる数値から始まるよう設定できます。カウンタには、[digits] フィールドで指定された桁数が使用されます (デフォルトは 1)。桁数が指定数より少ない場合、カウンタはゼロでパッドされます。桁数が指定数を超えた場合、固有名は生成されず、それを囲むルールによってエラーステータスが返されます。

宛先データストアが eDirectory で、名前が省略されている場合、検索は擬似属性「[Entry.rdn]」に対して行われます。これは、命名属性がどれであるかにかかわらずオブジェクトの RDN を表します。宛先データストアがアプリケーションの場合、名前は必須です。

Unmatched Source DN（一致しないソース DN）

この名詞は、このルールの条件（短絡評価を考慮）で、If Source DN（ソース DN の条件）条件の最新の一致によって一致しなかった DN の一部に対応する、現在の操作にあるソース DN の一部まで拡張します。

例



フィールド

変換

［true］は、宛先データストアの DN フォーマットに変換します。

コメント

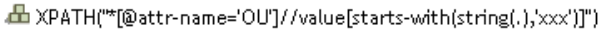
一致がない場合は、DN 全体が使用されます。変換属性が［true］に設定されている場合、DN のフォーマットは宛先データストアのフォーマットに変換されます。

このトークンは DirXML 1 の <copy-path-prefix> に相当し、主に下方互換性を確保するために存在します。

XPath

この名詞は、XPATH 1.0 の式の評価結果まで拡張します。

例



フィールド

式

評価する XPATH 1.0 の式。

動詞

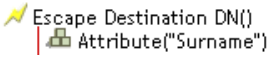
この節では、Policy Builder インタフェースで使用できるすべての動詞の詳細について説明します。

(125 ページ) Escape Destination DN (宛先 DN のエスケープ)
(126 ページ) Escape Source DN (ソース DN のエスケープ)
(127 ページ) Lower Case (小文字)
(128 ページ) Parse DN (DN の解析)
(130 ページ) Replace All (すべて置換)
(131 ページ) Replace First (最初のエントリを置換)
(132 ページ) Substring (下位文字列)
(133 ページ) Upper Case (大文字)

Escape Destination DN（宛先 DN のエスケープ）

この動詞は、宛先 DN フォーマットのルールに従って、囲まれた値をエスケープします。

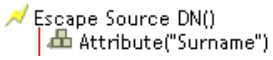
例



Escape Source DN（ソース DN のエスケープ）

この動詞は、ソース DN フォーマットのルールに従って、囲まれた値をエスケープします。

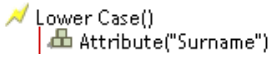
例



Lower Case（小文字）

この動詞は、囲まれた名詞と動詞を小文字に変換します。

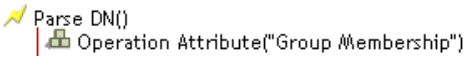
例



Parse DN（DN の解析）

この名詞は、囲まれたトークンの連結拡張により指定された DN のバージョンまで拡張します。

例



フィールド

宛先 DN 区切り記号

宛先 DN のカスタム区切り記号を指定します。

宛先 DN フォーマット

解析された DN の出力に使用するフォーマットを指定します。

長さ

含める DN セグメントの数。負の数は、（セグメントの総数 + 長さ）+ 1 と解釈されます（たとえば、5 つのセグメントで構成される DN の場合、長さ -1 であれば (5 + (-1)) + 1 = 5)、長さ -2 であれば (5 + (-2)) + 1 = 4 など）。

ソース DN 区切り記号

ソース DN のカスタム区切り記号を指定します。

ソース DN フォーマット

ソース DN の解析に使用するフォーマットを指定します。

開始

次のいずれかで始まるセグメントインデックス。

- 0 は最もルートに近いセグメント
- >0 は最もルートに近いセグメントからのオフセット
- -1 は最もリーフに近いセグメント
- <-1 は最もリーフに近いセグメントから最もルートに近いセグメントへのオフセット

コメント

DN は、ソース DN フォーマットで指定されたフォーマットに従って解析されます。開始および長さで指定された DN の部分は、宛先 DN フォーマットにより指定されたフォーマットに変換されます。

カスタム DN フォーマットを指定するには、パラメータを使用します。区切り記号を構成する 8 文字は、次のように定義されます。

1. Typed Name ブールフラグ：「0」は名前がタイプ指定されていないことを示します。「1」は名前がタイプ指定されていることを示します。

2. Unicode No-Map Character ブールフラグ：「0」は、マッピングできない Unicode 文字をエスケープ記号付きの 16 進文字列（¥FEFF など）で出力または解釈しないことを意味します。eDirectory で使用できない Unicode 文字は、0xfeff、0xfffe、0xfffd、および 0xffff です。

3. 相対 RDN 区切り記号

4. RDN 区切り記号

5. 名前分割記号

6. 名前値区切り記号

7. ワイルドカード文字

8. エスケープ文字

RDN 区切り記号と相対 RDN 区切り記号が同じ文字の場合、名前の向きはルートの右に、それ以外の場合はルートの左になります。

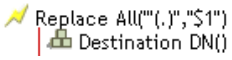
区切り記号セットの文字が 8 文字を超える場合、余分な文字はエスケープする必要がある文字と見なされますが、それ以外には特別な意味はありません。

開始および長さがデフォルト値 {0、-1} に設定されている場合は、DN 全体が使用されます。それ以外の場合は、開始および長さで指定された DN の一部のみが使用されます。

Replace All（すべて置換）

この動詞は、囲まれたすべての名詞と動詞で、指定した正規表現エントリをすべて置換します。

例



フィールド

正規表現

置換する下位文字列に一致する正規表現。

置換文字列

置換文字列を指定する正規表現。

コメント

一致する各インスタンスは、[Replace with] フィールドで指定された値によって指定される文字列に置換されます。

正規表現の作成については、次の Web サイトを参照してください。

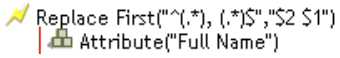
- http://java.sun.com/j2se/1.4/docs/api/java/util/regex/Pattern.html (http://java.sun.com/j2se/1.4/docs/api/java/util/regex/Pattern.html)
- http://java.sun.com/j2se/1.4/docs/api/java/util/regex/Matcher.html#replaceAll (java.lang.String) (http://java.sun.com/j2se/1.4/docs/api/java/util/regex/Matcher.html#replaceAll (java.lang.String))

パターンオプション CASE_INSENSITIVE、DOTALL、UNICODE_CASE が使われますが、適切な埋め込みエスケープを使って戻すこともできます。

Replace First（最初のエントリを置換）

この動詞は、指定した正規表現の最初のエントリを置換します。

例



フィールド

正規表現

置換する下位文字列に一致する正規表現。

置換文字列

置換文字列を指定する正規表現。

コメント

一致するインスタンスは、[Replace with] フィールドで指定された値によって指定される文字列に置換されます。

正規表現の作成については、次の Web サイトを参照してください。

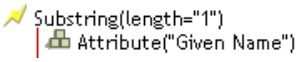
- http://java.sun.com/j2se/1.4/docs/api/java/util/regex/Pattern.html (http://java.sun.com/j2se/1.4/docs/api/java/util/regex/Pattern.html)
- http://java.sun.com/j2se/1.4/docs/api/java/util/regex/Matcher.html#replaceAll (java.lang.String) (http://java.sun.com/j2se/1.4/docs/api/java/util/regex/Matcher.html#replaceAll (java.lang.String))

パターンオプション CASE_INSENSITIVE、DOTALL、UNICODE_CASE が使われますが、適切な埋め込みエスケープを使って戻すこともできます。

Substring（下位文字列）

この動詞は、[Length] フィールドで指定した数の文字を含む文字列まで拡張します。囲まれた名詞および動詞は、下位文字列の動詞が適用される前に連結されます。

例



フィールド

開始

連結の開始場所。

- 0 は最初の文字
- >0 は文字列の開始からのオフセット
- -1 は最後の文字
- <-1 は最後の文字から文字列の開始へのオフセット

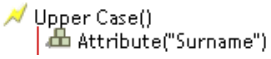
長さ

下位文字列に含める開始からの文字数。負の数は、(文字の総数 + 長さ) + 1 と解釈されます (たとえば、5 文字の文字列の場合、長さ -1 であれば (5 + (-1)) + 1 = 5)、長さ -2 であれば (5 + (-2)) + 1 = 4 など)。

Upper Case（大文字）

この動詞は、囲まれた名詞と動詞を大文字に変換します。

例



値

この節では、Policy Builder の一般的な値のリストを示します。

比較モード

モード	説明
case	文字ごとに大文字と小文字を区別する比較。
nocase	文字ごとに大文字と小文字を区別しない比較。
regex	文字列全体の正規表現一致。デフォルトでは大文字と小文字が区別されませんが、表現内でエスケープを使用すると変更できます。 http://java.sun.com/j2se/1.4/docs/api/java/util/regex/Pattern.html (http://java.sun.com/j2se/1.4/docs/api/java/util/regex/Pattern.html) および http://java.sun.com/j2se/1.4/docs/api/java/util/regex/Matcher.html#matches() (http://java.sun.com/j2se/1.4/docs/api/java/util/regex/Matcher.html#matches()) を参照してください。 パターンオプション CASE_INSENSITIVE、DOTALL、UNICODE_CASE が使われますが、適切な埋め込みエスケープを使って戻すこともできます。
src-dn	ソースデータストアの DN フォーマットに適したセマンティックを使った比較。
dest-dn	宛先データストアの DN フォーマットに適したセマンティックを使った比較。
numeric	数値で比較。
octet	オクテット（Base64 エンコード）の値を比較。
structured	属性の構造化構文の比較ルールに従って構造化属性を比較。

3 XSLT スタイルシートを使用したポリシーの定義

スタイルシートは、XSLT 変換ルールを定義します。DirXML® エンジンの XSLT プロセッサは、1999 年 11 月 16 日の W3C 勧告に準拠しています。仕様については、次を参照してください。

- XSL Transformations (XSLT) (http://www.w3.org/TR/1999/REC-xslt-19991116)
- XML Path Language (XPath) (http://www.w3.org/TR/1999/REC-xpath-19991116)

スタイルシートは次の場所で使うことができます。

- 入力変換ルール
- 出力変換ルール
- イベント変換ルール
- 一致、作成、または配置のルール
- マッピングルール

次の節では、DirXML でスタイルシートを使用する実装の詳細について説明します。

- 136 ページの「制限」
- 137 ページの「識別情報変換からの開始」
- 138 ページの「DirXML が渡すパラメータの使用」
- 140 ページの「拡張関数の使用」
- 141 ページの「DirXML 外でのスタイルシートのテスト」
- 142 ページの「パスワードの作成例：作成ルール」
- 143 ページの「eDirectory ユーザ作成例：作成ルール」

iManager での XSLT スタイルシートの管理

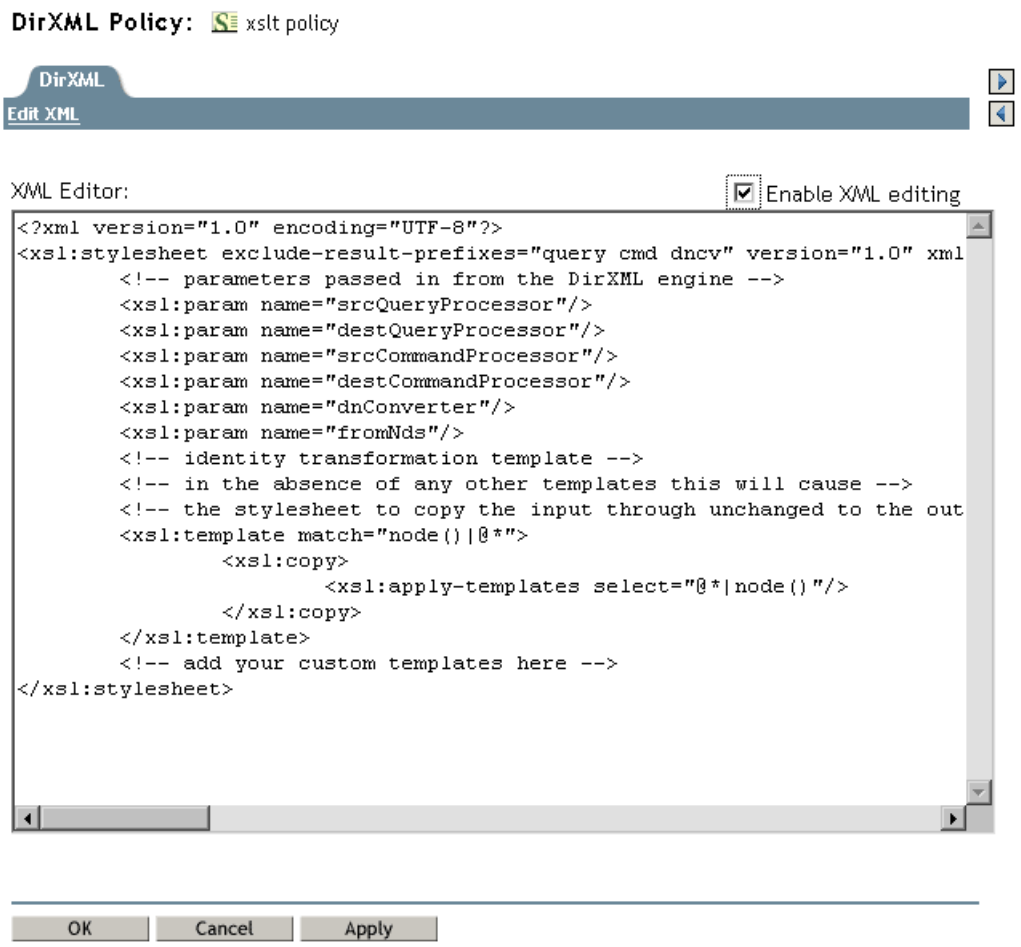
XSLT ポリシースタイルシートは、iManager を使用して追加、変更、および削除します。次の節では、iManager で XSLT スタイルシートを使用する方法の詳細について説明します。

- 135 ページの「XSLT ポリシーの追加」

XSLT ポリシーの追加

1. [DirXML Driver Overview] を開き、管理するドライバを見つけます。
2. 定義するポリシーを表すアイコンをクリックします。
3. [Insert] をクリックします。
4. 新規ポリシーの名前を入力し、[XSLT] を選択して [Enter] をクリックします。

5 XSLT ポリシーを定義し、[OK] をクリックします。



制限

3 つのルールタイプ (matching（一致）、create（作成）、および placement（配置））では、XML 文書を使用することもできます。これらのルールをスタイルシートとして記述した場合、次の制限を受けます。

Matching（一致）ルール制限

Matching（一致）ルールを XSLT スタイルシートとして記述した場合、次の制限を受けます。

- 複数の一致が見つかったときに通知するには、単一の Unicode 文字 0xFFFD という特別な値を使用する。
- 追加イベントのみを操作できる。
  - 加入者チャネルでは、DirXML ドライバは、アプリケーションで見つかった一致の <association> 要素を追加する必要がある。
  - 発行者チャネル上では、eDirectory™ 内で一致が見つかった場合、DirXML ドライバで、<add> 要素の dest-dn 属性を埋める必要がある。

- イベントを削除できる。
- 追加のイベントを生成できない。
- イベントタイプを変更できない。

属性およびクラスの名前は、eDirectory ネームスペースに存在します。

Create（作成）ルール制限

Create（作成）ルールを XSLT スタイルシートとして記述した場合、次の制限を受けます。

- 追加イベントのみを操作できる。
- 属性および値を <add> 要素に追加できる。
- イベントを削除できる（これが追加イベントを拒否する方法になります）。

属性およびクラスの名前は、eDirectory ネームスペースに存在します。

Placement（配置）ルール制限

配置ルールを XSLT スタイルシートとして記述した場合、次の制限を受けます。

- 追加イベントのみを操作できる。
- <add> 要素の dest-dn 属性を埋める必要がある。
- イベントを削除できる。

属性およびクラスの名前は、eDirectory ネームスペースに存在します。

識別情報変換からの開始

DirXML フォーマットとはまったく異なる XML フォーマットとの間で変換するのでなければ、識別情報の変換を実装するテンプレートからスタイルシートを始めることもできます。これらのテンプレートを使用すると、文書内のイベントを特に傍受または変更しなくても、修正なしに通過させることができます。

次の 2 つのテンプレートは、識別情報変換を実装しています。

```
<xsl:template match="/" >
    <xsl:apply-templates select="node()|@*"/>
</xsl:template>

<xsl:template match="node()|@*" >
    <xsl:copy>
       <xsl:apply-templates select="node()|@*"/>
    </xsl:copy>
</xsl:template>
```

DirXML が渡すパラメータの使用

DirXML エンジンは、スタイルシートで使用できる次のパラメータをルールスタイルシートに渡します。DirXML 1.1 では、クエリプロセッサパラメータが schema mapping（スキーママッピング）ルール、input transformation（入力変換）ルール、および output transformation（出力変換）ルールに渡されるようになっています。コマンドプロセッサパラメータはすべてのルールに渡されます。

- fromNds：これはブール値です。ルールが加入者チャネルによって処理されている場合は True、発行者チャネルによって処理されている場合は False になります。
- srcQueryProcessor：これは、XdsQueryProcessor インタフェースを実装する Java オブジェクトです。これにより、スタイルシートはイベントソースの詳細をクエリできます。
- destQueryProcessor：これは、XdsQueryProcessor インタフェースを実装する Java オブジェクトです。これにより、スタイルシートはイベントターゲットの詳細をクエリできます。
- srcCommandProcessor：これは、XdsCommandProcessor インタフェースを実装する Java オブジェクトです。これにより、スタイルシートはコマンドをイベントソースに「ライトバック」できます。DirXML 1.0 では使用できません。
- destCommandProcessor：これは、XdsCommandProcessor インタフェースを実装する Java オブジェクトです。これにより、スタイルシートは、他のルールのほとんどをバイパスして、コマンドを宛先に直接発行できます。DirXML 1.0 では使用できません。

これらのパラメータを使用するには、スタイルシートに次の行を含めます。

```
<xsl:param name="fromNds"/>
<xsl:param name="srcQueryProcessor"/>
<xsl:param name="destQueryProcessor"/>
<xsl:param name="srcCommandProcessor"/>
<xsl:param name="destCommandProcessor"/>
```

DirXML 1.1 では、プロセッサは、クエリまたはコマンド要素を最高レベル要素として受け入れ、必要に応じて <input> および <nds> で折り返します。

クエリおよびコマンドパラメータをスキーママッピングルールで使用する場合、入力変換ルールおよび出力変換ルールには次の制限が適用されます。

1. アプリケーションシムによって発行されるクエリは、アプリケーションシムが使用できる形式でなければなりません。つまり、スキーマ名がアプリケーションネームスペース内に存在し、クエリはシムがネイティブに使う XML ボキャブラリに従う必要があります。関連付けリファレンスはクエリに追加されません。
2. アプリケーションシムからの応答は、シムによって返された形式になります。変更やスキーママッピングは実行されておらず、関連付けリファレンスも解決されていません。
3. NDS に発行されるクエリは、NDS が使用できる形式でなければなりません。つまり、スキーマ名が NDS ネームスペース内に存在し、クエリは XDS でなければなりません。関連付けリファレンスは解決されません。
4. アプリケーションシムからの応答は、シムによって返された形式になります。変更やスキーママッピングは実行されていません。

クエリプロセッサ

クエリプロセッサの使用は、拡張関数の Novell XSLT 実装により異なります。クエリを作成するには、XdsQueryProcessor インタフェースのネームスペースを宣言する必要があります。これを行うには、次の行をスタイルシートの <xsl:stylesheet> または <xsl:transform> 要素に追加します。

```
xmlns:query="http://www.novell.com/nxsl/java/
com.novell.nds.dirxml.driver.XdsQueryProcessor"
```

次の例は、クエリプロセッサの 1 つを使用しています (極端に長い行は折り返されているので、< で始まっていません)。

```
<!-- Query object name queries NDS for the passed object -->
<!-- name.Ideally, this would not depend on "CN":to do -->
<!-- this, add another parameter that is the name of the   -->
<!-- naming attribute.-->

<xsl:template name="query-object-name">
   <xsl:param name="object-name"/>

<!-- build an xds query as a result tree fragment -->
   <xsl:variable name="query">
      <nds ndsversion="8.5" dtdversion="1.0">
         <input>
            <query>
               <search-class class-name="{ancestor-or-self:
:add/@class-name}"/>

<!-- NOTE:depends on CN being the naming attribute -->
               <search-attr attr-name="CN">
                   <value><xsl:value-of select="$object-name"/
                        ></value>
               </search-attr>
<!-- put an empty read attribute in so that we don't get -->
<!-- the whole object back                               -->
               <read-attr/>
            </query>
         </input>
      </nds>
   </xsl:variable>

<!-- query NDS -->
<xsl:variable name="result" select="query:query($destQuery
     Processor,$query)"/>

<!-- return an empty or non-empty result tree fragment -->
<!-- depending on result of query                      -->
   <xsl:value-of select="$result//instance"/>
</xsl:template>
```

コマンドパラメータ

作成ルールによって追加されたデフォルト属性をチャネルにライトバックできるようにするために、作成ルールの <required-attr> 要素に、write-back という名前の新しい XML の属性が追加されました。この属性が存在していて true に設定されている場合、作成ルールは srcCommandProcessor を変更コマンドとともに呼び出し、デフォルト値をソースにライトバックします。

次の例は、コマンドパラメータを使用して、ライトバック処理を行います。

```
<?xml version="1.0"?>
<xsl:transform
       version="1.0"
       xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
       xmlns:cmd="http://www.novell.com/nxsl/java
       com.novell.nds.dirxml.driver.XdsCommandProcessor"
>
<xsl:param name="srcCommandProcessor"/>

<xsl:template match="node()|@*">
     <xsl:copy>
          <xsl:apply-templates select="@*|node()"/>
     </xsl:copy>
</xsl:template>

<xsl:template match="add">
     <xsl:copy>
          <xsl:apply-templates select="@*|node()"/>
     </xsl:copy>

<!-- on a user add, add Engineering department to the source object -->
<xsl:variable name="dummy">
     <modify class-name="{@class-name} "dest-dn="{@src-dn}">
          <xsl-copy-of select="association"/>
          <modify-attr attr-name="OU">
               <add-value>
                    <value type="string">Engineering</value>
                </add-value>
          </modify-attr>
</modify>
</xsl:variable>
     <xsl:variable name="dummy2"
         select="cmd:execute($srcCommandProcessor, $dummy)"/>
</xsl:template>

</xsl:transform>
```

拡張関数の使用

XSLT は、いくつかの変換には優れたツールですが、自明でない文字列の操作や反復処理などの変換には適していません。しかし、Novell XSLT プロセッサには拡張関数が実装されているので、スタイルシートで、Java で実装されている関数を呼び出したり、拡張によって、JNI を通じてアクセスできる他の言語に実装されている機能を呼び出したりできます。

具体的な例については、クエリプロセッサを使った前の例、および Java を使った文字列操作を示す次の例を参照してください（極端に長い行は折り返されているので、< で始まっていません）。

```
<!-- get-dn-prefix places the part of the passed dn that -->
<!-- precedes the last occurrence of '\' in the passed dn -->
<!-- in a result tree fragment meaning that it can be    -->
<!-- used to assign a variable value                     -->

<xsl:template name="get-dn-prefix" xmlns:jstring="http://
     www.novell.com/nxsl/java/java.lang.String">

   <xsl:param name="src-dn"/>

<!-- use java string stuff to make this much easier -->
   <xsl:variable name="dn" select="jstring:new($src-dn)"/>
   <xsl:variable name="index" select="jstring:lastIndexOf
       ($dn,'\')"/>
   <xsl:if test="$index != -1">
      <xsl:value-of select="jstring:substring($dn,0,$index)
          "/>
   </xsl:if>
</xsl:template>
```

DirXML 外でのスタイルシートのテスト

DirXML エンジンでの XSLT 処理はコマンドラインから起動できます。これを使用して、スタイルシートを DirXML にインストールする前に、より制御された環境でスタイルシートをテストできます。

次のバッチファイルを使用して、NT または Windows 2000 上で XSLT プロセッサを起動できます。

```
@echo off
setlocal
rem TODO - edit the following line to point to directory where NDS and DirXML are installed

set DIRXML_HOME=c:\novell\nds
set COMMON_JARS=%DIRXML_HOME%\lib%DIRXML_HOME%\jre\bin\java -classpath%COMMON_JARS%\xp.jar;
%COMMON_JARS%\collections.jar; %COMMON_JARS%\nxsl.jar com.novell.xsl.nxsl %1 %2 %3 %4 %5 %6 %7
%8 %9

endlocal
```

引数なしでプロセッサを起動すると、プロセッサのコマンド構文に関する最新の情報が出力されます。

この場合、XSLT 処理は DirXML 外で実行されているので、srcQueryProcessor および destQueryProcessor は使用できません。この制限を避けるには、クエリプロセッサを使うコードを一時的にコメントアウトし、クエリから受け取る応答の明示的な割り当てに置き換えます。たとえば、次のようになります。

```
<!-- query NDS -->
<!-- <xsl:variable name="result" select="query:query($destQueryProcessor, $query)"/> -->

<!-- simulate query results -->

<xsl:variable name="result">
   <nds dtdversion="1.0" ndsversion="8.5">
      <output>
        <instance class-name="User" src-dn="\MY_TREE \MY_ORG\Fred"/>
        <status event-id="" level="success"></status>
      </output>
   </nds>
<xsl:variable>
```

パスワードの作成例：作成ルール

次のスタイルシートは作成ルールに使用できます。このスタイルシートは、ユーザを作成して、ユーザの Surname 属性と CN 属性からユーザのパスワードを生成し、識別情報変換を実行します（傍受または変換しようとするイベントを除き、文書に記述されたすべての情報を渡します）。

```
<?xml version="1.0" encoding="ISO-8859-1"?>

<!--   This stylesheet has an example of how to replace a create rule with
        an XSLT stylesheet and supply an initial password for "User" objects.-->

<xsl:transform xmlns:xsl="http://www.w3.org/1999/XSL/Transform
     "version="1.0">

<!-- ensure we have required NDS attributes -->
<xsl:template match="add">
   <xsl:if test="add-attr[@attr-name='Surname'] and
                 add-attr[@attr-name='CN']">
      <!-- copy the add through -->
      <xsl:copy>
         <xsl:apply-templates select="@*|node()"/>
         <!-- add a <password> element -->
         <xsl:call-template name="create-password"/>
      </xsl:copy>
   </xsl:if>

<!-- if the xsl:if fails, we don't have all the required attributes
      so we won't copy the add through, and the create rule will veto the add -->

</xsl:template>

<xsl:template name="create-password">
   <password>
      <xsl:value-of select="concat(add-attr[@attr-name='Surname']/value,
           '-',add-attr[@attr-name='CN']/value)"/>
   </password>
</xsl:template>

<!-- identity transform for everything we don't want to change -->

<xsl:template match="@*|node()">
    <xsl:copy>
        <xsl:apply-templates select="@*|node()"/>
    </xsl:copy>
</xsl:template>

</xsl:transform>
```

eDirectory ユーザ作成例：作成ルール

このスタイルシートは作成ルールに使用できます。外部アプリケーションで作成されたエントリから eDirectory ユーザを作成する方法を示します。この例は、新しく採用された社員をまず Human Resources データベース内に作成し、その後ネットワーク上に作成するような場合です。ユーザの名前と姓を取り出し、eDirectory ツリーに固有の CN を生成します。eDirectory では、CN はコンテナ内で固有であれば良いことになっていますが、このスタイルシートは、eDirectory ツリーにあるすべてのコンテナで CN を固有にします。

```
<?xml version="1.0" encoding="ISO-8859-1"?>

<!--  This stylesheet is an example of how to replace a create rule with an
      XSLT stylesheet and that creates the User name from the Surname and
      given Name attributes -->

<xsl:transform
   xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0"
   xmlns:query="http://www.novell.com/nxsl/java/com.novell.nds.dirxml.driver.
XdsQueryProcessor"
   >

<!-- This is for testing the stylesheet outside of DirXML so things
     are pretty to look at -->
<xsl:strip-space elements="*"/>
<xsl:preserve-space elements="value,component"/>
<xsl:output method="xml" indent="yes"/>

<!-- dirxml always passes two stylesheet parameters to an XSLT rule:
     an inbound and outbound query processor -->
<xsl:param name="srcQueryProcessor"/>
<xsl:param name="destQueryProcessor"/>

<!-- match <add> elements -->
<xsl:template match="add">

   <!-- ensure we have required NDS attributes we need for the name -->
   <xsl:if test="add-attr[@attr-name='Surname'] and
                 add-attr[@attr-name='Given Name']">

      <!-- copy the add through -->
      <xsl:copy>
         <!-- copy any attributes through except for the src-dn -->
         <!-- we'll construct the src-dn below so that the placement rule will work -->
         <xsl:apply-templates select="@*[string(.) != 'src-dn']"/>

        <!-- call a template to construct the object name and place the result in a variable -->
         <xsl:variable name="object-name">
            <xsl:call-template name="create-object-name"/>
         </xsl:variable>

         <!-- now create the src-dn attribute with the created name -->
         <xsl:attribute name="src-dn">
            <xsl:variable name="prefix">
               <xsl:call-template name="get-dn-prefix">
                  <xsl:with-param name="src-dn" select="string(@src-dn)"/>
               </xsl:call-template>
            </xsl:variable>
            <xsl:value-of select="concat($prefix,'\',$object-name)"/>
         </xsl:attribute>
```

```
         <!-- if we have a "CN" attribute, set it to the constructed name -->
         <xsl:if test="./add-attr[@attr-name='CN']">
            <add-attr attr-name="CN">
               <value type="string"><xsl:value-of select="$object-name"/></value>
            </add-attr>
         </xsl:if>

         <!-- copy the rest of the stuff through, except for what we have already copied -->
         <xsl:apply-templates select="*[name() != 'add-attr' or @attr-name != 'CN'] |
                                    comment() |
                                     processing-instruction() |
                                      text()"/>

         <!-- add a <password> element -->
         <xsl:call-template name="create-password"/>

      </xsl:copy>
   </xsl:if>
   <!-- if the xsl:if fails, it means we don't have all the required attributes
        so we won't copy the add through, and the create rule will veto the add -->
</xsl:template>

<!-- get-dn-prefix places the part of the passed dn that precedes the   -->
<!-- last occurrance of '\' in the passed dn in a result tree fragment   -->
<!-- meaning that it can be used to assign a variable value               -->
<xsl:template name="get-dn-prefix" xmlns:jstring="http://www.novell.com/nxsl/java/
java.lang.String">
   <xsl:param name="src-dn"/>

   <!-- use java string stuff to make this much easier -->
   <xsl:variable name="dn" select="jstring:new($src-dn)"/>
   <xsl:variable name="index" select="jstring:lastIndexOf($dn,'\')"/>
   <xsl:if test="$index != -1">
      <xsl:value-of select="jstring:substring($dn,0,$index)"/>
   </xsl:if>
</xsl:template>

<!-- create-object-name creates a name for the user object and places the   -->
<!-- result in a result tree fragment                                       -->
<xsl:template name="create-object-name">

   <!-- first try is first initial followed by surname -->
   <xsl:variable name="given-name" select="add-attr[@attr-name='Given Name']/value"/>
   <xsl:variable name="surname" select="add-attr[@attr-name='Surname']/value"/>
   <xsl:variable name="prefix" select="substring($given-name,1,1)"/>
   <xsl:variable name="object-name" select="concat($prefix,$surname)"/>

   <!-- then see if name already exists in NDS -->
   <xsl:variable name="exists">
      <xsl:call-template name="query-object-name">
         <xsl:with-param name="object-name" select="$object-name"/>
      </xsl:call-template>
   </xsl:variable>

   <!-- if exists, then try 1st fallback, else return result -->
   <xsl:choose>
      <xsl:when test="$exists != ''">
         <xsl:call-template name="create-object-name-2"/>
      </xsl:when>
      <xsl:otherwise>
         <xsl:value-of select="$object-name"/>
      </xsl:otherwise>
   </xsl:choose>
```

```
</xsl:template>

<!-- create-object-name-2 is the first fallback if the name created by      -->
<!-- create-object-name already exists                                       -->
<xsl:template name="create-object-name-2">

   <!-- first try is first name followed by surname -->
   <xsl:variable name="given-name" select="add-attr[@attr-name='Given Name']/value"/>
   <xsl:variable name="surname" select="add-attr[@attr-name='Surname']/value"/>
   <xsl:variable name="object-name" select="concat($given-name,$surname)"/>

   <!-- then see if name already exists in NDS -->
   <xsl:variable name="exists">
      <xsl:call-template name="query-object-name">
         <xsl:with-param name="object-name" select="$object-name"/>
      </xsl:call-template>
   </xsl:variable>

   <!-- if exists, then try last fallback, else return result -->
   <xsl:choose>
      <xsl:when test="$exists != ''">
         <xsl:call-template name="create-object-name-fallback"/>
      </xsl:when>
      <xsl:otherwise>
         <xsl:value-of select="$object-name"/>
      </xsl:otherwise>
   </xsl:choose>

</xsl:template>

<!-- create-object-name-fallback recursively tries a name created by      -->
<!-- concatenating the surname and a count until NDS doesn't find         -->
<!-- the name.There is a danger of infinite recursion, but only if      -->
<!-- there is a bug in NDS                                                  -->
<xsl:template name="create-object-name-fallback">
   <xsl:param name="count" select="1"/>

   <!-- construct the a name based on the surname and a count -->
   <xsl:variable name="surname" select="add-attr[@attr-name='Surname']/value"/>
   <xsl:variable name="object-name" select="concat($surname,'-',$count)"/>

   <!-- see if it exists in NDS -->
   <xsl:variable name="exists">
      <xsl:call-template name="query-object-name">
         <xsl:with-param name="object-name" select="$object-name"/>
      </xsl:call-template>
   </xsl:variable>

   <!-- if exists, then try again recursively, else return result -->
   <xsl:choose>
      <xsl:when test="$exists != ''">
         <xsl:call-template name="create-object-name-fallback">
            <xsl:with-param name="count" select="$count + 1"/>
         </xsl:call-template>
      </xsl:when>
      <xsl:otherwise>
         <xsl:value-of select="$object-name"/>
      </xsl:otherwise>
   </xsl:choose>
```

```
</xsl:template>

<!-- query object name queries NDS for the passed object-name.Ideally, this would   -->
<!-- not depend on "CN":to do this, add another parameter that is the name of the   -->
<!-- naming attribute.-->
<xsl:template name="query-object-name">
   <xsl:param name="object-name"/>

   <!-- build an xds query as a result tree fragment -->
   <xsl:variable name="query">
      <nds ndsversion="8.5" dtdversion="1.0">
         <input>
            <query>
               <search-class class-name="{ancestor-or-self::add/@class-name}"/>
               <!-- NOTE:depends on CN being the naming attribute -->
               <search-attr attr-name="CN">
                  <value><xsl:value-of select="$object-name"/></value>
               </search-attr>
             <!-- put an empty read attribute in so that we don't get the whole object back -->
               <read-attr/>
            </query>
         </input>
      </nds>
   </xsl:variable>

   <!-- query NDS -->
   <xsl:variable name="result" select="query:query($destQueryProcessor,$query)"/>

   <!-- return an empty or non-empty result tree fragment depending on result of query -->
   <xsl:value-of select="$result//instance"/>
</xsl:template>

<!-- create an initial password -->
<xsl:template name="create-password">
   <password>
      <xsl:value-of select="concat(add-attr[@attr-name='Surname']/value,'-',add-attr[@attr-
name='CN']/value)"/>
   </password>
</xsl:template>

<!-- identity transform for everything we don't want to mess with -->
<xsl:template match="@*|node()">
    <xsl:copy>
        <xsl:apply-templates select="@*|node()"/>
    </xsl:copy>
</xsl:template>

</xsl:transform>
```

4 フィルタの定義

フィルタを使用して、Nsure™ Identity Manager が同期化するオブジェクトと属性を指定できます。

この節では、フィルタに関する次のトピックを説明します。

- 147 ページの「フィルタのタスク」

フィルタのタスク

この節では、iManager で一般的なフィルタ関連タスクを実行する方法について説明します。

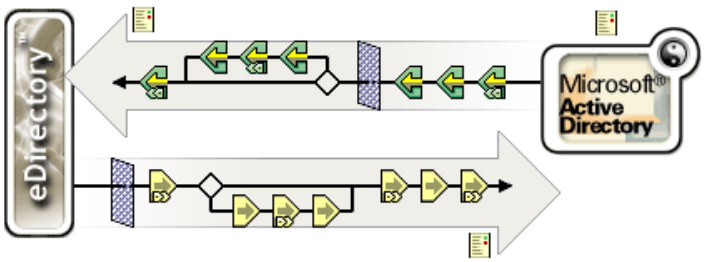
- 147 ページの「フィルタの管理」
- 148 ページの「フィルタの表示および変更」

フィルタの管理

1 iManager で［DirXML Management Role］を展開し、［Overview］をクリックします。

2 ドライバセットを指定します。

3 フィルタを管理するドライバをクリックします。［DirXML Driver Overview］が開きます。



4 ［DirXML Driver Overview］でフィルタを管理します。

フィルタの表示および変更

1 [DirXML Driver Overview] を開き、管理するドライバを見つけます。

2 発行者チャネルまたは加入者チャネル上で定義するフィルタを示すアイコンをクリックします。

3 [Filter] ウィンドウが開き、現在定義されているフィルタが表示されます。[Filter] ウィンドウを使ってフィルタを変更します。[Filter] ウィンドウにある [Help] アイコンをクリックすると、詳細を参照できます。