# Novell Identity Manager

3.5.1

www.novell.com

ポリシーの理解

September 28, 2007

Novell®

## 保証と著作権

米国 Novell, Inc. およびノベル株式会社は、この文書の内容または使用について、いかなる保証、表明または約束も行っていません。また文書の商品性、および特定の目的への適合性について、いかなる黙示の保証も否認し、排除します。また、本書の内容は予告なく変更されることがあります。

米国 Novell, Inc. およびノベル株式会社は、すべてのノベル製ソフトウェアについて、いかなる保証、表明または約束も行っていません。またノベル製ソフトウェアの商品性、および特定の目的への適合性について、いかなる黙示の保証も否認し、排除します。米国 Novell, Inc. およびノベル株式会社は、ノベル製ソフトウェアの内容を変更する権利を常に留保します。

本契約の下で提供される製品または技術情報はすべて、米国の輸出規制および他国の商法の制限を受けます。お客様は、すべての輸出規制を遵守し、製品の輸出、再輸出、または輸入に必要なすべての許可または等級を取得するものとします。お客様は、現在の米国の輸出除外リストに掲載されている企業、および米国の輸出管理規定で指定された輸出禁止国またはテロリスト国に本製品を輸出または再輸出しないものとします。お客様は、取引対象製品を、禁止されている核兵器、ミサイル、または生物化学兵器を最終目的として使用しないものとします。ノベル製ソフトウェアの輸出については、「Novell International Trade Services (http://www.novell.com/info/exports/)」の Web ページをご参照ください。弊社は、お客様が必要な輸出承認を取得しなかったことに対し如何なる責任も負わないものとします。



米国 Novell, Inc. は、本文書に記載されている製品に統合されている技術に関する知的所有権を保有します。これらの知的所有権は、「Novell Legal Patents (http://www.novell.com/company/legal/patents/)」の Web ページに記載されている 1 つ以上の米国特許、および米国ならびにその他の国における 1 つ以上の特許または出願中の特許を含む場合があります .

Novell, Inc.
404 Wyman Street, Suite 500
Waltham, MA 02451
U.S.A.
www.novell.com

*オンラインヘルプ:* 本製品とその他の Novell 製品の最新のオンラインヘルプにアクセスする場合は、「Novell Documentation (http://www.novell.com/documentation)」の Web ページをご覧ください。

## Novell の商標

Novell の商標一覧については、「商標とサービスの一覧 (http://www.novell.com/company/legal/trademarks/tmlist.html)」を参照してください。

## サードパーティ資料

サードパーティの商標は、それぞれの所有者に属します。

# 目次

# このガイドについて

Novell® Identity Manager 3.5.1 は、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するためのデータ共有および同期サービスです。このサービスは、分散された情報をリンクし、ユーザは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。

Identify Manager は、アカウントプロビジョニング、セキュリティ、Single Sign-On、ユーザセルフサービス、認証、認可、自動化されたワークフロー、および Web サービスの基盤になります。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、ポリシーとそれらのコンポーネントについて詳しく説明します。

- 5 ページの第 1 章「概要」
- 11 ページの第 2 章「Identity Manager ポリシーのアップグレード」
- 23 ページの第 3 章「ポリシーのタイプの理解」
- 51 ページの第 4 章「ポリシーコンポーネントの理解」
- 57 ページの第 5 章「Identity Manager ポリシーのダウンロード」
- 59 ページの第 6 章「XSLT スタイルシートを使用したポリシーの定義」

ポリシーの管理に関する情報を参照するには、『*Designer 2.1 のポリシー*』または『*iManager for Identity Manager 3.5.1 のポリシー*』を参照してください。

### 対象読者

このガイドは、Identity Manager の管理者を対象にしています。

### フィードバック

本マニュアルおよびこの製品に含まれているその他のマニュアルについて、皆様のご意見やご要望をお寄せください。オンラインマニュアルの各ページの下部にあるユーザコメント機能を使用するか www.novell.com/documentation/feedback.html にアクセスしてコメントを記入してください。

### マニュアルの更新

このマニュアルの最新のバージョンについては、Identity Manager のマニュアルの Web サイト (http://www.novell.com/documentation/idm35) を参照してください。

### 追加のマニュアル

Identity Manager ドライバの使用に関するマニュアルについては、Identity Manager ドライバのマニュアルの Web サイト (http://www.novell.com/documentation/idm35drivers/index.html) を参照してください。

Designer 2.1 の使い方に関するマニュアルについては、Designer 2.1 for Identity Manager 3.5.1 のマニュアルの Web サイト (http://www.novell.com/documentation/designer21/) を参照してください。

Identity Manager で使用されるドキュメントタイプ定義 (DTD) の詳しい説明については、『*Identity Manager 3.5.1 DTD リファレンス*』を参照してください。

### マニュアルの表記規則

このマニュアルでは、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。

# 概要

1

ポリシーは、Identity Manager がさまざまなシステムとデータを同期するために使用するもので、Identity Manager の基盤になります。Identity Manager を効果的に運用するには、ポリシーとその動作のしくみを理解することが重要です。

- 5 ページのセクション 1.1「ポリシーとは」
- 6 ページのセクション 1.2「ポリシーの新機能」

ポリシーに関する管理情報については、次のマニュアルを参照してください。

- *Designer 2.1 のポリシー*
- *iManager for Identity Manager 3.5.1 のポリシー*
- *Identity Manager 3.5.1 用 Novell 資格情報プロビジョニングポリシー*
- *Identity Manager 3.5.1 DTD リファレンス*

## 1.1 ポリシーとは

概して言うと、ポリシーは、Identity Manager が更新を送受信する方法を管理できるようにするルールの集合です。ドライバは接続システムから識別ボールトに変更内容を送信します。識別ボールトでは、データを操作して目的の結果を得るためにポリシーが使用されます。

ポリシーの動作を理解するには、まず、ポリシーのコンポーネントを理解する必要があります。

- ポリシーは複数のルールで構成されています。
- ルールとは条件の集合で、定義したアクション (「“ アクション ”」を参照 ) を実行するには、これらの条件 (「“ 条件 ”」を参照 ) を満たす必要があります。
- アクションは実行時に展開されるトークンから派生する動的な引数を持つことができます。
- トークンは、名詞と動詞の 2 つに分類できます。
  - 名詞トークン (「“ 名詞トークン ”」を参照 ) は、現在の操作、ソースやターゲットのデータストア、または特定の外部ソースから派生した値に展開されます。
  - 動詞トークン (「“ 動詞トークン ”」を参照 ) は、そのトークンに従属する他のトークンを連結した結果を変更します。
- 正規表現 (54 ページのセクション 4.5「正規表現」を参照 ) および XPath 1.0 の式 (55 ページのセクション 4.6「XPath 1.0 の式」を参照 ) は、ポリシーに対し適した結果を作成するためにルールでよく使用されます。
- ポリシーとは XDS ドキュメント上で操作を実行するもので、その主な目的はドキュメントを調べて変更を加えることです。
- 操作とは XDS ドキュメント内の要素のことで、入力要素と出力要素の子になります。要素は Novell® nds.dtd の一部です。詳細については、『*Identity Manager 3.5.1 DTD リファレンス*』の「“NDS DTD”」を参照してください。
- 通常、1 つの操作は 1 つのイベント、コマンド、またはステータスを表します。

- ポリシーは、操作ごとに個別に適用されます。ポリシーが各操作に順番に適用されるのに応じて、その操作が現在の操作になります。各ルールは現在の操作に順次適用されます。直前のルールによって実行されたアクションが原因で、ルールがそれ以降適用されなくなる場合を除き、すべてのルールが現在の操作に適用されます。
- ポリシーはドキュメント外のコンテキストを取得して、結果のドキュメントに反映されない副次的動作を発生させることもできます。

詳細については、このガイドの次の項を参照してください。

- 11 ページの第 2 章「Identity Manager ポリシーのアップグレード」
- 23 ページの第 3 章「ポリシーのタイプの理解」
- 51 ページの第 4 章「ポリシーコンポーネントの理解」
- 57 ページの第 5 章「Identity Manager ポリシーのダウンロード」
- 59 ページの第 6 章「XSLT スタイルシートを使用したポリシーの定義」
- 48 ページのセクション 3.6「ポリシーフローの理解」
- 『*Designer 2.1 for Identity Manager 3.5.1*』の「“ データフロービュー ”」

# 1.2 ポリシーの新機能

Identity Manager 3.5 には、アーキテクチャの変更、リソースオブジェクトという新しいオブジェクト、DirXML® スクリプトの新機能をはじめとして、新しい機能が含まれています。

- 6 ページのセクション 1.2.1「Identity Manager のアーキテクチャの変更」
- 6 ページのセクション 1.2.2「リソースオブジェクト」
- 7 ページのセクション 1.2.3「DirXML スクリプトの新機能」

## 1.2.1 Identity Manager のアーキテクチャの変更

Identity Manager 3.5 では、ドライバがポリシーを参照する方法のアーキテクチャが変更されています。この変更のため、リソースオブジェクトを活用できるようドライバをアップグレードする必要があります。アーキテクチャの変更の詳細については、11 ページの第 2 章「Identity Manager ポリシーのアップグレード」を参照してください。

## 1.2.2 リソースオブジェクト

リソースオブジェクトは、ドライバが使用する情報を格納します。リソースオブジェクトには任意のデータを任意の形式で格納できます。Novell® Identity Manager 3.5 には、ECMA スクリプト、マッピングテーブル、ポリシーライブラリ、一般的なリソースオブジェクトなど、さまざまなタイプのリソースオブジェクトが付属しています。『*Designer 2.1 のポリシー*』の「“ リソースオブジェクトへの情報の格納 ”」または『*iManager for Identity Manager 3.5.1 のポリシー*』の「“ リソースオブジェクトへの情報の格納 ”」を参照してください。

## 1.2.3 DirXML スクリプトの新機能

DirXML スクリプトには、次の新機能が含まれています。



### ポリシーでの変数の使用

DirXML スクリプトでポリシー内で変数を使用できるようになり、定義済み変数が付属しています。詳細については、51 ページのセクション 4.2「変数」および 53 ページのセクション 4.3「変数の拡張」を参照してください。

### Notrace 属性の追加

<rule>、<and>、<or>、<if>、<do>、および <token> のすべてのタグに Notrace 属性が追加されました。Notrace 属性をオンに設定すると、Identity Manager エンジンはこれらのタグに対してトレースメッセージを出力しません。

### If パスワード条件への新しい演算子の追加

“If パスワード ” 条件で、等しいおよび等しくないの演算子がサポートされるようになりました。

### If 関連付け、If 操作、および If パスワード条件でのモードのサポート

“If 関連付け ”、“If 操作 ”、および “If パスワード ” の各条件で、さまざまな比較モードがサポートされるようになりました。

### 一部の条件への新しい演算子の追加

より小さい、より小さくない、より大きい、およびより大きくないの各演算子が次の条件でサポートされるようになりました。

- “If 関連付け ”
- “If 属性 ”
- “If クラス名 ”

- “If ターゲット属性 ”
- “If エンタイトルメント ”
- “If グローバル構成値 ”
- “If ローカル変数 ”
- “If 操作属性 ”
- “If 操作プロパティ ”
- “If パスワード ”
- “If ソース属性 ”

### 一部のアクションでのオプションの Before 属性のサポート

オプションの before 属性を使用すると、元の一致式に対して評価される別の XPath 式の前に XML テキストを挿入できます。Before 属性は次のアクションでサポートされます。

- “XML エレメントの追加 ”
- “XML テキストの追加 ”
- “XPath 式によるクローン ”

### ポリシーの変数への Scope 属性の追加

Scope 属性を使用すると、ポリシー内の変数にスコープを設定できます。次の要素で Scope 属性がサポートされるようになりました。

- “If ローカル変数 ”
- “ ローカル変数の設定 ”
- “ ローカル変数 ”

### 一部のアクションに対するパスワード設定機能の追加

次のアクションでパスワードを使用できるようになりました。

- “ 電子メールの送信 ”
- “ テンプレートから電子メールを送信 ”
- “ ワークフローの開始 ”

### カスタム SMTP ヘッダのサポートの追加

“ 電子メールの送信 ”、および “ テンプレートから電子メールを送信 ” の各アクションで、カスタム SMTP ヘッダを使用できるようになりました。

### 一意の名前への属性の追加

“ 一意の名前 ” トークンの柔軟性を高めるために、counter-use、counter-patter、および on-unavailable の各属性が追加されました。

### 特定の関連付けまたは DN へのパスワードの追加

“ターゲットパスワードの設定”および“ソースパスワードの設定”の各アクションで、特定の DN または関連付けに対してパスワードを設定できるようになりました。

### DirXML スクリプトの新しい要素

DirXML スクリプトの新機能を使用すると、XSLT スタイルシートの代わりにポリシービルダを使いやすくなります。Designer で DirXML スクリプトの新しい要素を使用する方法の詳細については、次の項を参照してください。

- “If XML 属性 ”
- “SSO 資格情報のクリア ”
- “If”
- “SSO 資格情報の設定 ”
- “SSO パスフレーズの設定 ”
- “ ワークフローの開始 ”
- “While”
- “ 文字 ”
- “ ドキュメント ”
- “ パスワードの生成 ”
- “ クエリ ”
- “ 解決 ”
- “ 時間 ”
- “Base64 デコード ”
- “Base64 エンコード ”
- “ 変換時間 ”
- “ 結合 ”
- “ マップ ”
- “ 分割 ”

# Identity Manager ポリシーのアップグレード

# 2

旧バージョンの Identity Manager がインストールされている場合は、この項を続けて読んでください。Identity Manager 3.5 を初めてインストールした場合は、23 ページの第 3 章「ポリシーのタイプの理解」までスキップします。Identity Manager 3.5 では、ポリシーが相互を参照する方法に新しいアーキテクチャが採用されています。

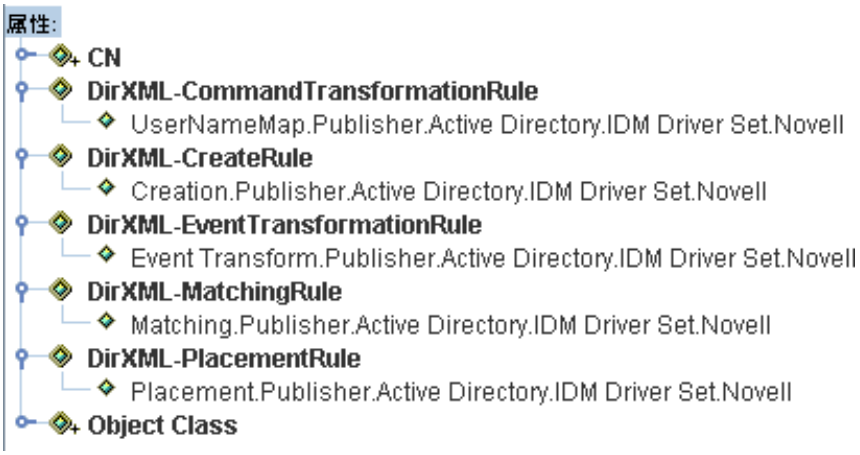


## 2.1 以前の Identity Manager のアーキテクチャ

以前のアーキテクチャでは、別個の eDirectory™ 属性を使用して、個々のポリシー ( ルールまたはスタイルシート ) オブジェクトを指すドライバの各ポリシーセット、発行者オブジェクト、または購読者オブジェクトを表していました。入力変換、出力変換、およびスキーママッピングポリシーセットはドライバオブジェクトに保存されていました。イベント変換、一致、作成、配置、およびコマンド変換ポリシーセットは、発行者オブジェクトおよび購読者オブジェクトに格納されていました。

ポリシータイプの詳細については、23 ページの第 3 章「ポリシーのタイプの理解」を参照してください。11 ページの 図 2-1 は、Active Directory* ドライバの発行者オブジェクトに格納されるすべてのポリシーを示しています。

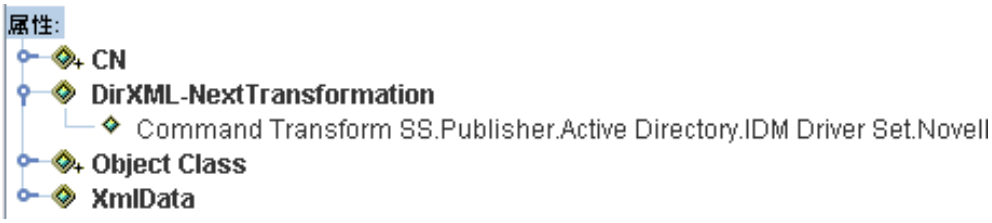
***図 2-1*** *バージョン 3.5 より前の Identity Manager のアーキテクチャ*




ポリシーセットが参照するポリシーまたはスタイルシートの DN は、各属性に格納されています。ポリシーセット内に複数のポリシーがある場合は、ポリシーチェーンが作成されます。DirXML-NextRule 属性は、ポリシーセット内で次に実行する必要があるポリシーを

指します。12 ページの 図 2-2 は、コマンド変換スタイルシートを示しています。これには、DirXML-NextTransformation 属性で次に実行されるコマンド変換スタイルシートまたはポリシーが含まれます。

*図 2-2 ポリシーでの DirXML-NextRule 属性*




この属性を使用すると、ポリシーセット内の順序を便利な方法で表すことができますが、個々のポリシーを共有するにはチェーン内のそれ以降のポリシーもすべて共有しなければならないという欠点があります。

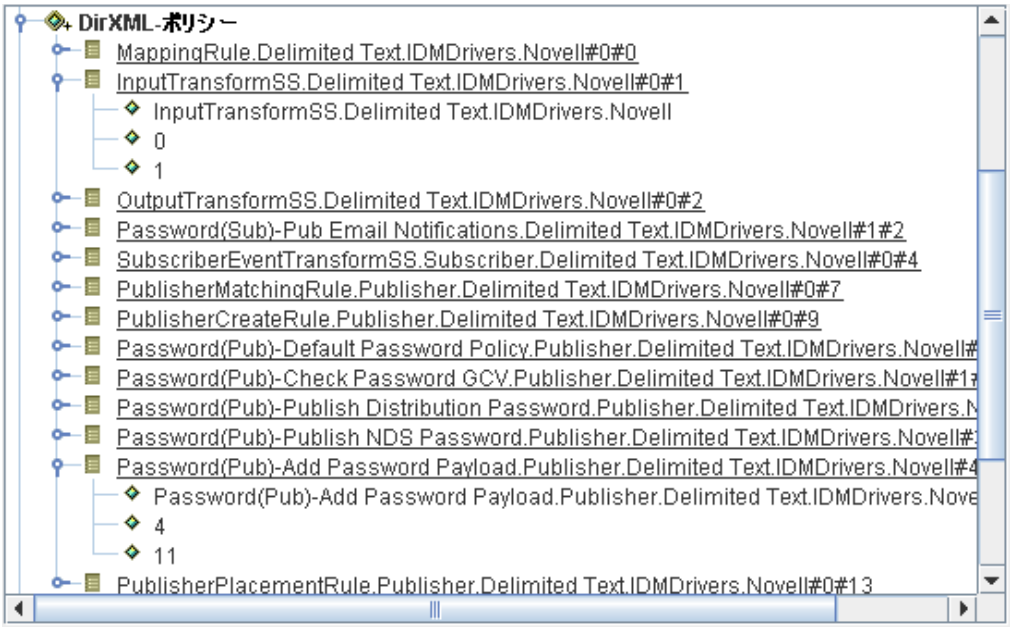
.

## 2.2 新しい Identity Manager のアーキテクチャ

Identity Manager 3.5 の新しいアーキテクチャでは、ドライバオブジェクトで DirXML-Policies と呼ばれる 1 つの属性しか使用しません。DirXML-Policies は複数値の属性で、ポリシーを指す DN フィールドと、[*間隔*] および [*レベル*] を指定する 2 つの整数フィールドで構成されます。[*間隔*] フィールドは、ポリシーが属するポリシーセットを示す数値を格納します。[*レベル*] フィールドは、ポリシーセット内のポリシーの相対順序 ( 低～高 ) を示します。

13 ページの 図 2-3 は、区切り付きテキストドライバの DirXML-Policies 属性内にあるすべてのポリシーセットを示しています。パスワードポリシーは購読者コマンド変換ポリシーで、コマンド変換ポリシーの実行時に 4 番目に実行されるポリシーです。

*図 2-3 新しい Identity Manager のアーキテクチャ*




各ポリシーセットには、［整数］フィールドで固有の番号が割り当てられます。これらの値は次のとおりです。

- 0 = スキーママッピング
- 1 = 入力変換
- 2 = 出力変換
- 3 = ECMAScript 関数定義
- 4 = 購読者イベント変換
- 5 = 発行者イベント変換
- 6 = 購読者一致
- 7 = 発行者一致
- 8 = 購読者作成
- 9 = 発行者作成
- 10 = 購読者コマンド変換
- 11 = 発行者コマンド変換
- 12 = 購読者配置
- 13 = 発行者配置

## 2.3 新しいアーキテクチャへのドライバのアップグレード

新しいアーキテクチャを適切に動作させるには、ドライバセット内のすべてのドライバが新しいアーキテクチャを使用する必要があります。1 つのドライバセット内に異なるバージョンのドライバを混在させて、新しいアーキテクチャの機能を利用することはできません。

ドライバをアップグレードするには、Designer 2.0 および iManager 用 Identity Manager 3.5 プラグインをインストールしてから、Designer または iManager を使用します。この手順では、ドライバは新しいアーキテクチャを使用するようにアップグレードされますが、ドライバ環境設定ファイルはアップグレードされません。ドライバ環境設定ファイルをアップグレードするには、17 ページのセクション 2.4「ドライバ環境設定ファイルのアップグレード方法」を参照してください。

- 14 ページのセクション 2.3.1「Designer でのドライバのアップグレード」
- 17 ページのセクション 2.3.2「iManager でのドライバのアップグレード」

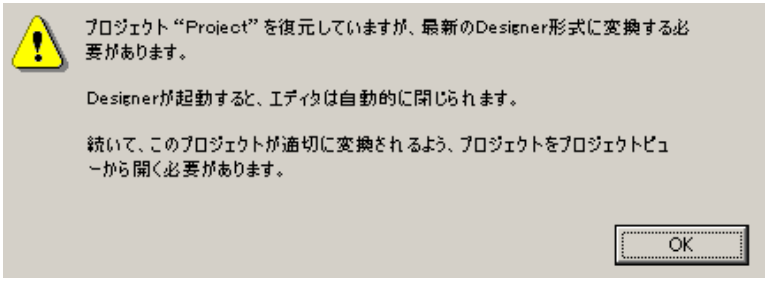
### 2.3.1 Designer でのドライバのアップグレード

Designer でドライバをアップグレードするには :

1. Designer バージョン 2.0 以上をインストールして、Designer を起動します。

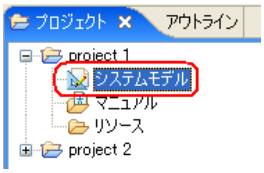
   Designer のアップグレード時に Designer でプロジェクトを開いていた場合は、ステップ 2 に進みます。Designer のアップグレード時に Designer でプロジェクトを開いていなかった場合は、手順ステップ 3 までスキップします。

2. Designer のアップグレード時にプロジェクトを開いていた場合は、次の警告メッセージが表示されます。アップグレードに関するメッセージを読んで、［*OK*］をクリックします。



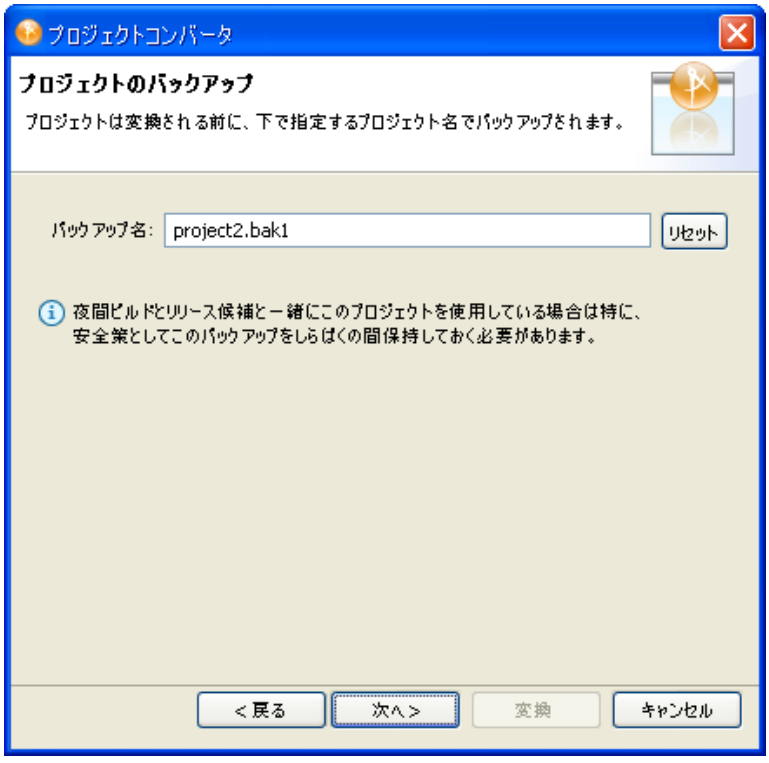

   プロジェクトが閉じられ、アップグレードが実行されます。

3. プロジェクトビューで、［*システムモデル*］をダブルクリックし、プロジェクトを開いて変換します。

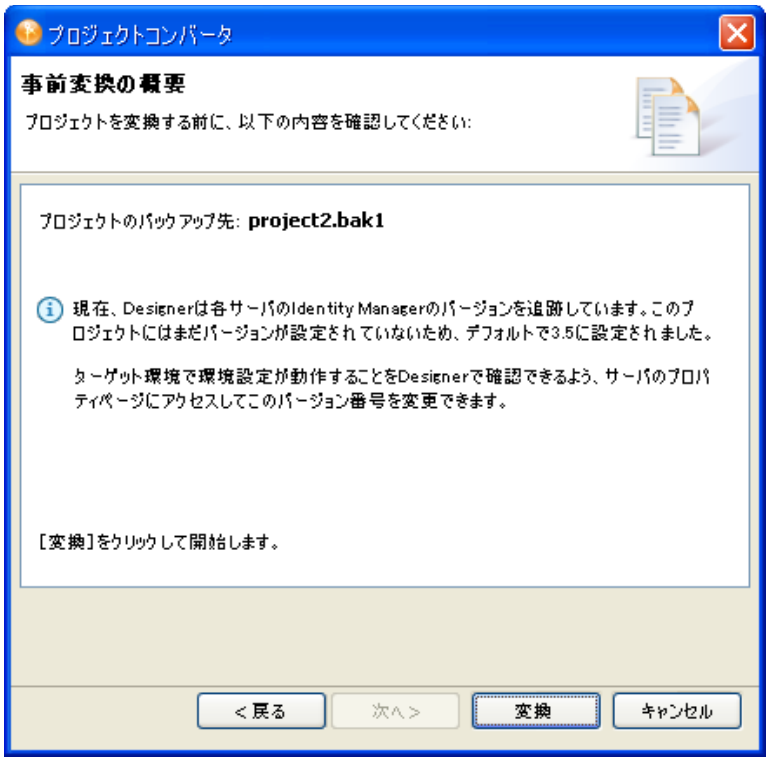



4. プロジェクトコンバータのメッセージを読みます。このメッセージは、プロジェクトがバックアップされて新しいフォーマットに変換され、変更内容がファイルに記録されてから新しいプロジェクトが開かれることを説明しています。続いて、［*次へ*］をクリックします。

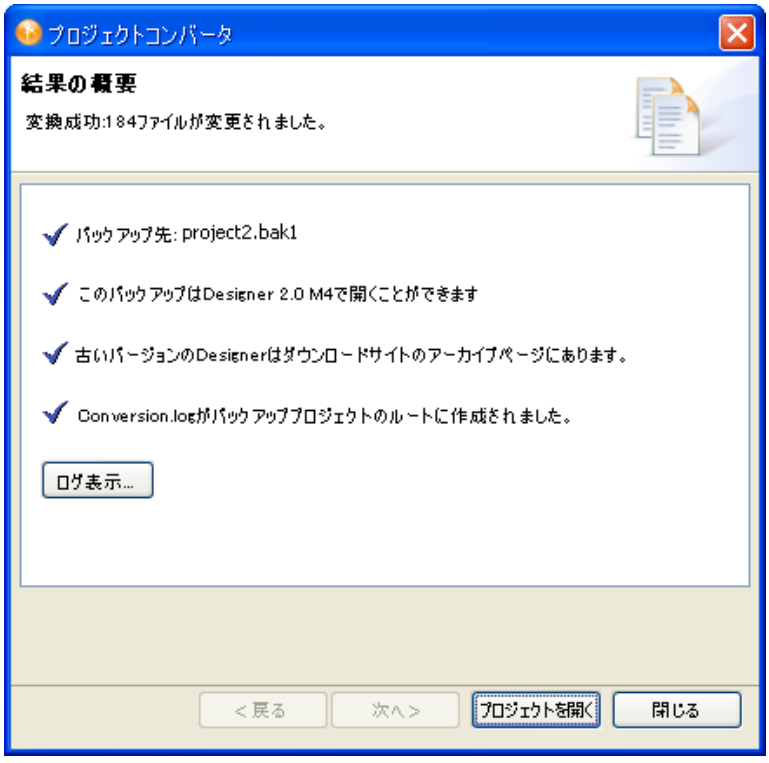

5 プロジェクトのバックアップ名を指定して、［*次へ*］をクリックします。




6 プロジェクトの変換の概要を読み、［*変換*］をクリックします。




**7** プロジェクトの変換結果の概要を読み、[*プロジェクトを開く*] をクリックします。

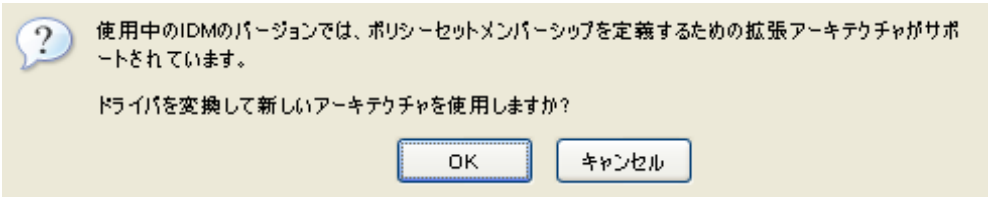



生成されたログファイルを参照する場合は、[*ログ表示*] をクリックします。

### 2.3.2 iManager でのドライバのアップグレード

iManager でドライバをアップグレードするには :

1. Identity Manager 3.5 と最新のプラグインがインストールされていることを確認してから、iManager を起動します。
2. *［Identity Manager］* > *［Identity Manager の概要］* の順にクリックします。
3. ［*検索*］をクリックしてドライバセットオブジェクトを検索し、アップグレードするドライバをクリックします。
4. 表示されるメッセージを読み、［*OK*］をクリックします。




5. ドライバセット内に複数のドライバがある場合は、各ドライバに対してステップ 2 ～ステップ 4 を繰り返します。

## 2.4 ドライバ環境設定ファイルのアップグレード方法

既存のドライバとそのポリシーをアップグレードするには、さまざまな方法があります。Identity Manager にはカスタマイズしたポリシーをマージするためのマージプロセスがないため、簡単な方法はありません。ドライバをアップグレードすると、新しいドライバ内にあるポリシーと同じ名前のポリシーは上書きされます。ポリシーをカスタマイズしている場合、それらのポリシーは上書きされ、カスタマイズの内容は失われます。

この問題を解決してアップグレードする方法は数多くありますが、この項では 2 種類のアップグレード方法について説明します。どちらの方法にも長所と短所があります。

- 17 ページのセクション 2.4.1「新しいドライバをインストールして古いドライバから既存のポリシーを移動する方法」
- 18 ページのセクション 2.4.2「新しいドライバ環境設定ファイルを既存のドライバにオーバーレイする方法」

### 2.4.1 新しいドライバをインストールして古いドライバから既存のポリシーを移動する方法

この方法の長所は次のとおりです。

- 既存のポリシーが上書きされない。

この方法の短所は次のとおりです。

- 同期したオブジェクトの関連付けがすべて失われ、再作成、展開、および再ロードする必要がある。

- 関連付けを再度作成するために時間がかかる。特定の関連付けに依存するポリシーがある場合、そのポリシーは動作しなくなります。
- ポリシーとルールが正確に復元されたことを確認する作業が複雑である。

### 2.4.2 新しいドライバ環境設定ファイルを既存のドライバにオーバーレイする方法

この方法による影響は、ポリシーの設定内容に応じて変わります。

長所は次のとおりです。

- ドライバ環境設定ファイル内のポリシーとは異なる名前がポリシーに付いている場合、それらのポリシーは上書きされない。
- 同期したオブジェクトの関連付けは現状のまま残り、再作成する必要がない。

短所は次のとおりです。

- ドライバ環境設定ファイル内のポリシーと同じ名前がポリシーに付いている場合、それらのポリシーは上書きされる。

このアップグレードオプションを推奨します。ただし、このアップグレード方法を機能させるには、ポリシーを作成するために特定の方法を用意する必要があります。

- ポリシーの開発時には、ポリシーのアップグレード時と同じ手順に従う必要があります。
- 既存の Novell ポリシーまたはルールは変更しないでください。
- デフォルトのポリシーを使用しない場合は、ポリシーを無効にしますが、削除はしないでください。
- ビジネスニーズに合った目的の結果を達成する新しいポリシーまたはルールを作成します。
- ポリシーに名前を付ける際には、会社の標準の命名モデルを使用します。
- ポリシーの名前には、そのポリシーが保存されているポリシーセットを示すプレフィックスを付けます。これにより、ポリシーの接続先のポリシーセットを判断できます。

これらの方法を用意したら、18 ページのセクション 2.5「ドライバ環境設定の推奨アップグレード手順」に従ってドライバ環境設定をアップグレードします。

## 2.5 ドライバ環境設定の推奨アップグレード手順

これは、Novell が推奨するドライバ環境設定のアップグレード手順です。必ず実験環境で手順を実行してください。この手順は、Designer でも iManager でも実行できます。

## 2.5.1 Designer でのドライバ環境設定のアップグレード

このアップグレード手順では、次に示す 3 つの異なるタスクを完了する必要があります。

- 19 ページの 「ドライバのエクスポートの作成」
- 19 ページの 「既存のドライバへの新しいドライバ環境設定ファイルのオーバーレイ」
- 19 ページの 「ドライバへのカスタムポリシーとルールの復元」

### ドライバのエクスポートの作成

ドライバのエクスポートを作成して、現在の環境設定のバックアップを作成します。アップグレードする前に、バックアップを作成してあることを確認してください。

1. Designer のプロジェクトが最新バージョンのドライバを使用していることを確認します。方法については、『*Designer 2.1 for Identity Manager 3.5.1* 』の「“ 識別ボールトからのドライバセットまたはドライバのインポート ”」を参照してください。
2. モデラーで、アップグレードするドライバのドライバ行を右クリックします。
3. ［*環境設定ファイルのエクスポート*］を選択します。
4. 環境設定ファイルを保存する場所を参照して、［*保存*］をクリックします。
5. ［結果］ページで［*OK*］をクリックします。

### 既存のドライバへの新しいドライバ環境設定ファイルのオーバーレイ

1. モデラーで、アップグレードするドライバのドライバ行を右クリックします。
2. ［*環境設定ウィザードの実行*］を選択します。
3. 警告のページが表示されたら、［*はい*］をクリックします。

   この警告は、ドライバ設定とポリシーがすべてリセットされることを通知しています。

   **重要 :** データが失われることがないように、カスタマイズした各ポリシーにデフォルトのポリシーとは異なる名前が付いていることを確認します。
4. アップグレードするドライバのドライバ環境設定を参照して選択し、［*実行*］をクリックします。
5. ドライバの情報を指定して、［*次へ*］をクリックします。

   情報を指定するページは複数表示される場合があります。
6. ［結果］ページで［*OK*］をクリックします。

### ドライバへのカスタムポリシーとルールの復元

次の 2 つの方法で、ポリシーをポリシーセットに追加できます。

- 19 ページの 「アウトラインビューによるカスタマイズしたポリシーの追加」
- 20 ページの 「ポリシーフローの表示ビューによるカスタマイズしたポリシーの追加」

#### アウトラインビューによるカスタマイズしたポリシーの追加

1. *アウトラインビュー*で、アップグレードしたドライバを選択して、*ポリシーセットビュー*を表示します。

2 カスタマイズしたポリシーをドライバに復元する必要があるポリシーセットを右クリックして、*[新規作成] > [コピー]* の順に選択します。

3 カスタマイズしたポリシーを参照して選択し、[*OK*] をクリックします。

4 カスタマイズしたポリシーの名前を指定し、[*OK*] をクリックします。

5 ファイルの競合を示すメッセージが表示されたら、[*はい*] をクリックしてプロジェクトを保存します。

6 ポリシービルダでポリシーが開いたら、コピーしたポリシーの情報が正しいことを確認します。

7 ドライバに復元する必要があるカスタマイズした各ポリシーに対して、ステップ 2 ～ステップ 6 を繰り返します。

8 ドライバを起動してテストします。

9 ポリシーが動作することを確認したら、ドライバを運用環境に移します。

#### ポリシーフローの表示ビューによるカスタマイズしたポリシーの追加

1 *アウトラインビュー*で、アップグレードしたドライバを選択して、[*ポリシーフローを表示*] アイコンをクリックします。

2 カスタマイズしたポリシーをドライバに復元する必要があるポリシーセットを右クリックして、*[ポリシーの追加] > [既存の項目をコピー]* の順に選択します。

3 カスタマイズしたポリシーを参照して選択し、[*OK*] をクリックします。

4 カスタマイズしたポリシーの名前を指定し、[*OK*] をクリックします。

5 ファイルの競合を示すメッセージが表示されたら、[*はい*] をクリックしてプロジェクトを保存します。

6 ポリシービルダでポリシーが開いたら、コピーしたポリシーの情報が正しいことを確認します。

7 ドライバに復元する必要があるカスタマイズした各ポリシーに対して、ステップ 2 ～ステップ 6 を繰り返します。

8 ドライバを起動してテストします。

9 ポリシーが動作することを確認したら、ドライバを運用環境に移します。

### 2.5.2 iManager でのドライバ環境設定のアップグレード

このアップグレード手順では、次に示す 3 つの異なるタスクを完了する必要があります。

- 20 ページの 「ドライバのエクスポートの作成」
- 21 ページの 「既存のドライバへの新しいドライバ環境設定ファイルのオーバーレイ」
- 21 ページの 「ドライバへのカスタムポリシーとルールの復元」

#### ドライバのエクスポートの作成

ドライバのエクスポートを作成して、現在の環境設定のバックアップを作成します。アップグレードする前に、バックアップを作成してあることを確認してください。

1 iManager で、*[Identity Manager] > [Identity Manager の概要]* の順に選択します。

2 [*検索*] をクリックして、アップグレードするドライバを格納するドライバセットオブジェクトを見つけます。

3 アップグレードするドライバをクリックして、［エクスポート］をクリックします。

4 ［次へ］をクリックして、［環境設定にリンクされているかどうかにかかわらず、含まれるすべてのポリシーをエクスポート］を選択します。

5 ［次へ］をクリックし、［名前を付けて保存］をクリックします。

6 ［Save to Disk ( ディスクに保存)］を選択し、［OK］をクリックします。

7 ［完了］をクリックします。

### 既存のドライバへの新しいドライバ環境設定ファイルのオーバーレイ

1 iManager で、［Identity Manager］> ［Identity Manager の概要］の順に選択します。

2 ［ドライバの追加］をクリックし、新規ドライバウィザードのページで［次へ］をクリックします。

3 オーバーレイするドライバ環境設定を選択して、［次へ］をクリックします。

4 ［既存のドライバ］フィールドで、アップグレードするドライバを参照して選択します。

5 ドライバの情報を指定して、［次へ］をクリックします。

6 ［概要］ページで、［このドライバおよびポリシーライブラリに関するすべてを更新する］を選択します。

**重要 :** データが失われることがないように、カスタマイズしたポリシーにデフォルトとは異なる名前が付いていることを確認します。

7 ［次へ］をクリックし、［概要］ページで［完了］をクリックします。

### ドライバへのカスタムポリシーとルールの復元

1 iManager で、［Identity Manager］> ［Identity Manager の概要］の順に選択します。

2 ［検索］をクリックしてドライバセットオブジェクトを見つけ、アップグレードしたドライバをクリックします。

3 カスタマイズしたポリシーをドライバに復元する必要があるポリシーセットを選択して、［挿入］をクリックします。

4 ［既存のポリシーを使用する］を選択し、カスタムポリシーを参照して選択します。

5 ［OK］をクリックし、［閉じる］をクリックします。

6 ドライバに復元する必要がある各カスタムポリシーに対して、ステップ 3 ～ステップ 5 を繰り返します。

7 ドライバを起動してテストします。

8 ポリシーが動作することを確認したら、ドライバを運用環境に移します。

# ポリシーのタイプの理解

# 3

この項では、ポリシーとフィルタの概要、および Identity Manager 環境でのそれらの機能の概要について説明します。次のトピックについて説明します。

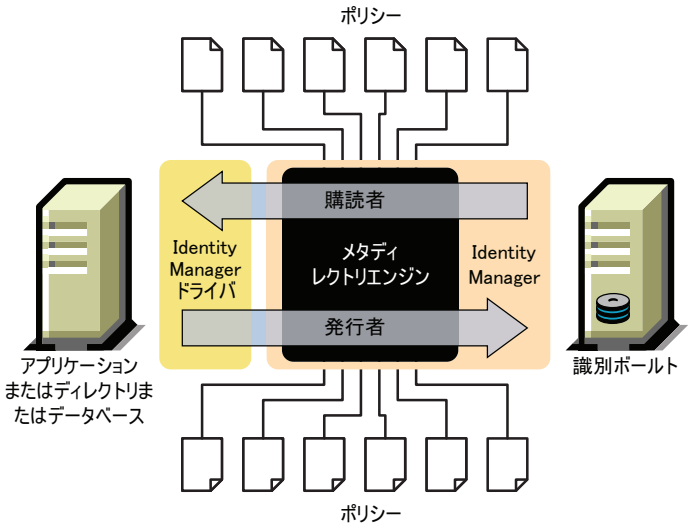


## 3.1 ポリシーに関する Identity Manager のアーキテクチャ

Identity Manager は、識別ボールトと任意のアプリケーション、ディレクトリ、またはデータベースとの間で安全にデータを移動することができます。この処理を実現するために、Identity Manager は、eDirectory™ のデータとイベントを XML 形式に変換する洗練されたインタフェースを備えています。このインタフェースにより、ディレクトリとの間でデータが流れることができます。

次の図は、Identity Manager の基本コンポーネントとそれらの関係を示しています。

***図 3-1*** *Identity Manager のコンポーネント*




Metadirectory エンジンは Identity Manager アーキテクチャの重要なモジュールです。このエンジンは、Identity Manager ドライバが識別ボールトと情報を同期化できるインタフェースを提供し、異なるデータシステムでも接続してデータを共有できるようにします。

メタディレクトリエンジンは、XML 形式を使用して、識別ボールトのデータと識別ボールトのイベントを提供します。メタディレクトリエンジンは、ルールプロセッサとデータ変換エンジンを採用して、2 つのシステム間のデータフローを操作します。ルールプロセッサと変換エンジンにアクセスするには、ポリシーセットと呼ぶ制御ポイントを使用します。ポリシーセットにはゼロ個以上のポリシーを含めることができます。

ポリシーは、主にチャネル入力上のイベントをチャネル出力上のコマンドセットに変換することによって、ビジネスルールとプロセスを実装します。各ドライバがデータとイベントを同期する方法は、管理者が一連のポリシーを使用して設定します。たとえば、ユーザオブジェクトの名前属性に対して必ず値を指定するように作成ポリシーで指定されている場合、名前を値を指定せずにユーザオブジェクトを作成しようとすると、拒否されます。

## 3.2 フィルタの使用

フィルタでは、メタディレクトリエンジンがイベントを処理するオブジェクトクラスや属性、およびそれらのクラスおよび属性に対する変更の処理方法を指定します。

フィルタは、ベースクラスがフィルタで指定されたクラスのいずれかと一致するオブジェクトで発生するイベントだけを通過させます。フィルタで指定されたクラスの従属クラスであるオブジェクトで発生するイベントは通過させません。ただし、従属クラスも指定されている場合を除きます。チャネルごとに個別のフィルタを設定して、各クラスおよび属性の同期方向および信頼されるデータソースを制御できます。

---

**注 :** eDirectory™ では、エントリの作成に使用されるオブジェクトクラスがベースクラスです。フィルタでは、ベースクラスの継承元であるスーパークラス、または追加属性の取得元である補助クラスではなく、このクラスを指定する必要があります。

---

たとえば、名字属性および名前属性を持つユーザクラスがフィルタで同期対象として設定されている場合、メタディレクトリ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。しかし、エントリの電話番号属性が変更されている場合は、電話番号属性がフィルタに設定されていないため、メタディレクトリエンジンはこのイベントをドロップします。

フィルタの設定には次を含める必要があります。

- 同期する属性
- まだ同期されていないが、何らかのアクションを実行するポリシーをトリガするために使用される属性

フィルタの定義の詳細については、『*Designer 2.1 のポリシー*』の「“ フィルタを使用したオブジェクトのフローの制御 ”」または『*iManager for Identity Manager 3.5.1 のポリシー*』の「“ フィルタを使用したオブジェクトのフローの制御 ”」を参照してください。

## 3.3 ポリシーの動作のしくみ

高度なレベルにおいては、ポリシーは、Identity Manager が更新を送受信する方法をカスタマイズできるようにするルールの集合です。ドライバは接続システムから識別ボールトに変更内容を送信します。識別ボールトでは、データを操作して目的の結果を得るためにポリシーが使用されます。


- 25 ページのセクション 3.3.1「変更の検出と識別ボールトへの変更の送信」
- 25 ページのセクション 3.3.2「情報のフィルタリング」

- 26 ページのセクション 3.3.3「ポリシーを使用した変更の適用」

### 3.3.1 変更の検出と識別ボールトへの変更の送信

ドライバを作成すると、ドライバを展開している企業が使用する可能性のあるすべての情報を同期するための機能を組み込もうとします。開発者は、接続システム内で関連する変更を検出し、それを識別ボールトに送信するようにドライバを作成します。

変更内容は、Identity Manager の仕様に応じた形式で XML ドキュメントに格納されます。次に、このような XML ドキュメントの抜粋を示します。

```
<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>
```

### 3.3.2 情報のフィルタリング

ドライバは、関連する変更をすべて報告してから、ユーザが情報をフィルタできるように設計されています。これにより、必要な情報だけを現在の環境に取り込むことができます。

たとえば、ある企業でグループに関する情報が必要ない場合は、識別ボールトまたは接続システムに、グループに関する操作をすべてブロックするフィルタを実装できます。この企業でユーザとグループに関する情報が必要になった場合は、識別ボールトと接続システム間で両方のタイプのオブジェクトを同期できるようにフィルタを実装できます。

必要なオブジェクトだけを同期できるようにフィルタを定義するのは、ドライバのカスタマイズの第 1 歩です。

次の段階として、フィルタを通過したオブジェクトに対して、Identity Manager が何を行うかを定義します。例として、前に挙げた XML ドキュメントの追加操作を参照してください。接続システムに、名前が「Smith」、電話番号が「111-1111」のユーザが追加されています。この操作が許可されているとすると、Identity Manager では、このユーザに対する処理内容を決定する必要があります。

### 3.3.3 ポリシーを使用した変更の適用

変更を加える場合、Identity Manager は、一連のポリシーを特定の順序で適用します。

まず、一致ポリシーが、「このオブジェクトはすでにデータストア内にあるか?」という質問に答えます。このためには、オブジェクトに固有の特性を定義する必要があります。一般的にチェックされる属性は、電子メールアドレスです。これは通常、固有であるためです。「2 つのオブジェクトの電子メールアドレスが同じ場合、それらは同じオブジェクトである」というポリシーを定義できます。

一致するものがあった場合、Identity Manager は、これを関連付けと呼ばれる属性に記録します。関連付けは、Identity Manager が接続システム内のオブジェクトを関連付けられるようにする一意の値です。

一致しているものがない場合は、作成ポリシーが呼び出されます。作成ポリシーは、オブジェクトの作成条件を Identity Manager に通知します。作成ルールでは特定の属性の存在を必須にすることができます。これらの属性が存在しない場合、Identity Manager 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。

オブジェクトが作成されると、配置ポリシーによって、オブジェクトの配置場所が Identity Manager に通知されます。オブジェクトを、それが元々あったシステムと同じ階層構造で作成するように指定できます。また、オブジェクトを属性値に基づいてまったく別の場所に配置することもできます。

オブジェクトの位置属性に従ってユーザを階層に配置し、フルネーム属性に従って名前を付ける場合は、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。これによって、属性は確実に存在するようになり、配置の方針どおりに正しく機能します。

ポリシーの利用法は、他にも多数あります。ポリシービルダを使用すると、一意の値の生成、属性の追加および削除、イベントおよびコマンドの生成、電子メールの送信など、多くのことを簡単に実行できます。XSLT を使用すると、アプリケーション間で情報を移動する XML ドキュメントを直接変換するといったさらに高度な変換が可能です。

23 ページの第 3 章「ポリシーのタイプの理解」に進んでさまざまなタイプのポリシーについてさらに詳しく学んだ上で、『*Designer 2.1 のポリシー*』または『*iManager for Identity Manager 3.5.1 のポリシー*』を参照してポリシービルダの使い方を学んでください。

## 3.4 ポリシータイプ

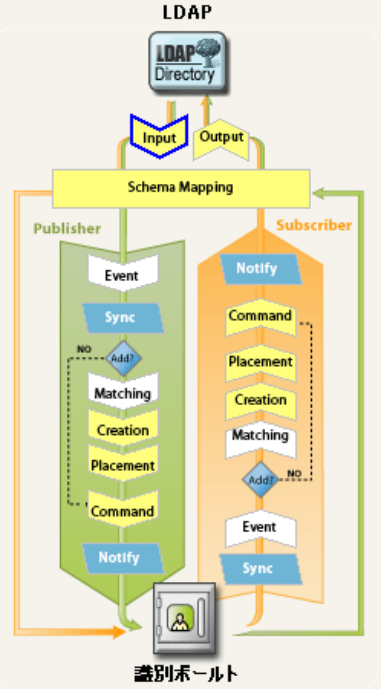
購読者チャネルと発行者チャネルの両方で定義できるポリシーにはさまざまなタイプがあります。各ポリシーは、データ変換のさまざまな段階で適用されます。また、一部のポリシーは特定のアクションが発生した場合にのみ適用されます。たとえば、作成ポリシーは新しいオブジェクトが作成される場合にのみ適用されます。

さまざまなポリシータイプとチャネル上でのそれらの実行順序は次のとおりです。


- 27 ページのセクション 3.4.1「イベント変換ポリシー」
- 30 ページのセクション 3.4.2「一致ポリシー」
- 31 ページのセクション 3.4.3「作成ポリシー」
- 35 ページのセクション 3.4.4「配置ポリシー」
- 38 ページのセクション 3.4.5「コマンド変換ポリシー」
- 41 ページのセクション 3.4.6「スキーママッピングポリシー」

- 43 ページのセクション 3.4.7「出力変換ポリシー」
- 46 ページのセクション 3.4.8「入力変換ポリシー」

*図 3-2 ポリシーの実行順序*




## 3.4.1 イベント変換ポリシー

イベント変換ポリシーは、識別ボールトまたは接続アプリケーションで発生したイベントをメタディレクトリエンジンに表示する方法を変更します。イベント変換ポリシーで実行される最も一般的なタスクは、スコープフィルタリングやイベントタイプフィルタリングなどのカスタムフィルタリングです。

スコープフィルタリングは、イベントの位置または属性値に基づいて、不要なイベントを削除します。たとえば、部署属性が特定の値と一致しないか、特定のグループのメンバではない場合に、イベントを削除します。

イベントタイプフィルタリングは、イベントタイプに基づいて、不要なイベントを削除します。たとえば、すべての削除イベントを削除します。

例 :

- 27 ページの 「スコープフィルタリング : 例 1」
- 28 ページの 「スコープフィルタリング : 例 2」
- 29 ページの 「タイプフィルタリング : 例 1」
- 30 ページの 「タイプフィルタリング : 例 2」

### スコープフィルタリング : 例 1

次の DirXML スクリプトポリシーの例では、Users サブツリー内に含まれ、無効になっておらず、役職属性に「Consultant」または「Manager」という語が含まれていないユーザに

対するイベントだけを許可します。操作がブロックされていることを示すステータスドキュメントも生成します。このポリシーを XML で表示するには、Event_Transformation_Scope1.xml (../samples/Event_Transformation_Scope1.xml) を参照してください。

```
<policy>
      <rule>
            <description>Scope Filtering</description>
            <conditions>
                     <or>
                        <if-class-name op="equal">User</if-class-name>
                     </or>
                     <or>
                           <if-src-dn op="not-in-subtree">Users</if-
src-dn>
                            <if-attr name="Login Disabled"
op="equal">True</if-attr>
                        <if-attr mode="regex" name="Title"
op="equal">.*Consultant.*</if-attr>
                        <if-attr mode="regex" name="Title"
op="equal">.*Manager.*</if-attr>
                  </or>
            </conditions>
            <actions>
                  <do-status level="error">
                        <arg-string>
                              <token-text>User doesn't meet required
conditions</token-text>
                        </arg-string>
                  </do-status>
                  <do-veto/>
               </actions>
      </rule>
</policy>
```

### スコープフィルタリング : 例 2

次の DirXML スクリプトポリシーは、ユーザオブジェクトに対する変更操作を拒否します。ただし、すでに関連付けられているオブジェクトの変更を除きます。このポリシーを XML で表示するには、Event_Transformation_Scope2.xml (../samples/Event_Transformation_Scope2.xml) を参照してください。

```
<policy>
      <rule>
             <description>Veto all operation on User except modifies
of already associated objects</description>
            <conditions>
                  <or>
                        <if-class-name op="equal">User</if-class-name>
                  </or>
                  <or>
                        <if-operation op="not-equal">modify</if-
operation>
                         <if-association op="not-associated"/>
                  </or>
```

```
            </conditions>
            <actions>
                        <do-veto/>
            </actions>
      </rule>
</policy>
```

### タイプフィルタリング : 例 1

次の DirXML スクリプトポリシーの例では、最初のルールで、Employee および Contractor コンテナ内のオブジェクトの同期だけを許可します。2 つ目のルールでは、すべての名前変更操作および移動操作をブロックします。このポリシーを XML で表示するには、Event_Transformation_Type1.xml (../samples/Event_Transformation_Type1.xml) を参照してください。

```
<policy>
       <rule>
            <description>Only synchronize the Employee and Contractor
subtrees</description>
            <conditions>
                        <and>
                               <if-src-dn op="not-in-
container">Employees</if-src-dn>
                               <if-src-dn op="not-in-
container">Contractors</if-src-dn>
                        </and>
            </conditions>
            <actions>
                        <do-status level="warning">
                           <arg-string>
                             <token-text>Change ignored: Out of
scope.</token-text>
                           </arg-string>
                         </do-status>
                          <do-veto/>
            </actions>
       </rule>
       <rule>
            <description>Don't synchronize moves or renames</
description>
            <conditions>
                        <or>
                            <if-operation op="equal">move</if-
operation>
                            <if-operation op="equal">rename</if-
operation>
                        </or>
           </conditions>
            <actions>
                        <do-status level="warning">
                                <arg-string>
                                          <token-text>Change ignored:
We don't like you to do that.</token-text>
                                </arg-string>
```

```
                    </do-status>
                    <do-veto/>
                </actions>
        </rule>
</policy>
```

#### タイプフィルタリング : 例 2

次の DirXML スクリプトポリシーは、すべての追加イベントをブロックします。このポリシーを XML で表示するには、Event_Transformation_Type2.xml (../samples/Event_Transformation_Type2.xml) を参照してください。

```
<policy>
      <rule>
            <description>Type Filtering</description>
            <conditions>
                  <and>
                        <if-operation op="equal">add</if-
operation>
                  </and>
            </conditions>
            <actions>
                   <do-status level="warning">
                         <arg-string>
                               <token-text>Change ignored:
Adds are not allowed.</token-text>
                          </arg-string>
                   </do-status>
                   <do-veto/>
             </actions>
        </rule>
</policy>
```

### 3.4.2 一致ポリシー

購読者一致および発行者一致などの一致ポリシーは、ソースデータストア内の関連付けられていないオブジェクトに対応するオブジェクトをターゲットデータストアで検索します。重要な点は、一致ポリシーが常に必要または望ましいとは限らないということです。

たとえば、既存のオブジェクトや対応するオブジェクトが存在する場合に初めて移行を実行するときは、一致ポリシーは望ましくない可能性があります。

一致ポリシーは、誤った一致が検出されないように十分に注意して作成する必要があります。

- 30 ページの 「インターネット電子メールアドレスによる一致 : 例」
- 31 ページの 「名前による一致 : 例」

#### インターネット電子メールアドレスによる一致 : 例

次の DirXML スクリプトポリシーの例は、インターネット電子メールアドレスに基づいてユーザを照合します。このポリシーを XML で表示するには、Matching1.xml (../samples/Matching1.xml) を参照してください。

```
<policy>
        <rule>
            <description>Match Users based on email address</
description>
            <conditions>
                      <and>
                        <if-class-name op="equal">User</if-class-name>
                      </and>
            </conditions>
            <actions>
                   <do-find-matching-object>
                         <arg-dn>
                           <token-text>ou=people,o=novell</token-text>
                         </arg-dn>
                  <arg-match-attr name="Internet EMail Address"/>
                    </do-find-matching-object>
            </actions>
       </rule>
</policy>
```

**名前による一致 : 例**

次の DirXML スクリプトポリシーの例では、その共通名属性に基づいて、グループオブジェクトを照合します。このポリシーを XML で表示するには、Matching2.xml (../samples/Matching2.xml) を参照してください。

```
<?xml version="1.0" encoding="UTF-8"?>
<policy>
        <rule>
                <description>Match Group by Common Name</description>
                <conditions>
                          <or>
                                  <if-class-name op="equal">Group</
if-class-name>
                           </or>
                </conditions>
                <actions>
                        <do-find-matching-object scope="subtree">
                               <arg-match-attr name="CN"/>
                         </do-find-matching-object>
                </actions>
         </rule>
</policy>
```

## 3.4.3 作成ポリシー

購読者作成ポリシーおよび発行者作成ポリシーなどの作成ポリシーは、新しいオブジェクトを作成するときに満たす必要がある条件を定義します。作成ポリシーがないことは、オブジェクトを作成できることを意味します。

たとえば、識別ボールトに新しいユーザを作成する一方で、新しいユーザオブジェクトには名前と ID だけを指定するとします。この作成は eDirectory ツリーでミラーリングされますが、追加は識別ボールトに接続されているアプリケーションでは直ちに反映されませ

ん。これは、より完全な定義が指定されたユーザオブジェクトだけを許可するように、作成ポリシーで指定されているためです。

作成ポリシーは、発行者と購読者の両方で同じにすることも別にすることもできます。

テンプレートオブジェクトを指定して、オブジェクトが eDirectory で作成される場合に作成プロセスで使用することができます。

作成ポリシーは、通常、次の目的で使用されます。

- 属性がないなどの理由で条件を満たさないオブジェクトの作成を拒否する。
- デフォルト属性値を設定する。
- デフォルトパスワードを設定する。

例 :


- 32 ページの 「必須属性 : 例」
- 33 ページの 「デフォルト属性値 : 例」
- 34 ページの 「デフォルトパスワード : 例」
- 34 ページの 「テンプレートの指定 : 例」


### 必須属性 : 例

次に示す DirXML スクリプトポリシー例の最初のルールでは、ユーザを作成するにはユーザオブジェクトに CN、名前、名字、インターネット電子メールアドレスの各属性があらかじめ含まれている必要があります。2 つ目のルールでは、すべての部門オブジェクトで OU 属性が必要になります。最後のルールでは、名前が「Fred」であるすべてのユーザオブジェクトが拒否されます。このポリシーを XML で表示するには、Create1.xml (../samples/Create1.xml) を参照してください。

```
<policy>
      <rule>
            <description>Veto if required attributes CN, Given Name,
Surname and Internet EMail Address not available</description>
            <conditions>
                  <or>
                        <if-class-name op="equal">User</if-class-
name>
                  </or>
            </conditions>
            <actions>
             <do-veto-if-op-attr-not-available name="CN"/>
             <do-veto-if-op-attr-not-available name="Given Name"/>
                  <do-veto-if-op-attr-not-available name="Surname"/>
                  <do-veto-if-op-attr-not-available name="Internet
EMail Address"/>
            </actions>
      </rule>
    <rule>
            <description>Organizational Unit Required Attributes</
description>
            <conditions>
                <or>
```

```
                        <if-class-name op="equal">Organizational
Unit</if-class-name>
                  </or>
            </conditions>
            <actions>
                  <do-veto-if-op-attr-not-available name="OU"/>
            </actions>
      </rule>
      <rule><description>Conditionally veto guys named "Fred"</
description>
            <conditions>
                  <and>
                        <if-global-variable name="no-fred"
op="equal">true</if-global-variable>
                        <if-op-attr name="Given Name"
op="equal">Fred</if-op-attr>
                  </and>
            </conditions>
            <actions>
                  <do-status level="warning">
                        <arg-string>
                              <token-text xml:space="preserve"
xmlns:xml="http://www.w3.org/XML/1998/namespace">Vetoed "Fred"</token-
text>
                        </arg-string>
                  </do-status>
                <do-veto/>
            </actions>
       </rule>
</policy>
```

### デフォルト属性値 : 例

次の DirXML スクリプトポリシーの例では、ユーザの説明属性のデフォルト値を追加します。このポリシーを XML で表示するには、Create2.xml (../samples/Create2.xml) を参照してください。

```
<policy>
      <rule>
            <description>Default Description of New Employee</
description>
            <conditions>
                  <or>
                    <if-class-name op="equal">User</if-class-name>
                  </or>
            </conditions>
            <actions>
                 <do-set-default-attr-value name="Description">
                    <arg-value type="string">
                          <token-text>New Employee</token-text>
                    </arg-value>
                 </do-set-default-attr-value>
            </actions>
```

```
      </rule>
</policy>
```

### デフォルトパスワード : 例

次の DirXML スクリプトポリシーの例では、名前の最初の 2 文字と名字の最初の 6 文字 ( すべて小文字 ) で構成したパスワード値を作成します。このポリシーを XML で表示するには、Create3.xml (../samples/Create3.xml) を参照してください。

```
<policy>
      <rule>
                  <description>Default Password of [2]FN+[6]LN</
description>
            <conditions>
                  <and>
                  <if-class-name op="equal">User</if-class-name>
                   <if-password op="not-available"/>
                  </and>
            </conditions>
            <actions>
                  <do-set-dest-password>
                  <arg-string>
                        <token-lower-case>
                              <token-substring length="2">
                                    <token-op-attr name="Given
Name"/>
                              </token-substring>
                          <token-substring length="6">
                                    <token-op-attr
name="Surname"/>
                              </token-substring>
                               </token-lower-case>
                     </arg-string>
                </do-set-dest-password>
             </actions>
     </rule>
</policy>
```

### テンプレートの指定 : 例

次の DirXML スクリプトポリシーの例では、ユーザの役職属性で、そのユーザがマネージャであることが示されている (「Manager」を含んでいる ) 場合のテンプレートオブジェクトを指定します。このポリシーを XML で表示するには、Create4.xml (../samples/Create4.xml) を参照してください。

```
<policy>
        <rule>
                  <description>Assign Manager Template if Title
contains Manager</description>
                  <conditions>
                        <and>
                           <if-class-name op="equal">User</if-class-
name>
                           <if-op-attr name="Title" op="available"/>
                             <if-op-attr mode="regex" name="Title"
```

```
op="equal">.*Manager.*</if-op-attr>
                          </and>
                    </conditions>
                    <actions>
                          <do-set-op-template-dn>
                                <arg-dn>
                                      <token-text>Users\Manager
Template</token-text>
                                </arg-dn>
                          </do-set-op-template-dn>
                    </actions>
          </rule>
</policy>
```

## 3.4.4 配置ポリシー

配置ポリシーは、新しいオブジェクトを配置する場所と、識別ボールトおよび接続されたアプリケーションでのオブジェクトの名前を決定します。

配置ポリシーは識別ボールトでオブジェクト作成を行う場合に、発行者チャネルにおいて必要です。配置ポリシーは、購読者チャネルでは必須でない場合もあります。これはターゲットデータストアの特性に応じて、接続されたアプリケーションでオブジェクトの作成を行う場合も同じですたとえば、リレーショナルデータベースへの同期では、リレーショナルデータベースの行には位置も名前もないので配置ポリシーは不要です。


- 35 ページの 「属性値による配置 : 例 1」
- 36 ページの 「属性値による配置 : 例 2」
- 37 ページの 「名前による配置 : 例」


### 属性値による配置 : 例 1

次の DirXML スクリプトポリシーの例では、部署属性の値に基づいて、特定のコンテナにユーザを作成します。このポリシーを XML で表示するには、Placement1.xml (../samples/Placement1.xml) を参照してください。

```
<policy>
        <rule>
             <description>Department Engineering</description>
             <conditions>
                  <and>
                        <if-class-name op="equal">User</if-class-name>
                        <if-op-attr mode="regex" name="Department"
op="equal">.*Engineering.*</if-op-attr>
                  </and>
             </conditions>
             <actions>
                  <do-set-op-dest-dn>
                        <arg-dn>
                              <token-text>Eng</token-text>
                          <token-text>\</token-text>
                                    <token-op-attr name="CN"/>
                        </arg-dn>
                  </do-set-op-dest-dn>
```

```
            </actions>
        </rule>
        <rule>
            <description>Department HR</description>
            <conditions>
                  <and>
                        <if-class-name op="equal">User</if-class-
name>
                       <if-op-attr mode="regex" name="Department"
op="equal">.*HR.*</if-op-attr>
                  </and>
            </conditions>
            <actions>
                  <do-set-op-dest-dn>
                       <arg-dn>
                             <token-text>HR</token-text>
                         <token-text>\</token-text>
                          <token-op-attr name="CN"/>
                       </arg-dn>
                  </do-set-op-dest-dn>
            </actions>
      </rule>
</policy>
```

### 属性値による配置 : 例 2

次の DirXML スクリプトポリシーでは、入力ドキュメントの src-dn によって、ユーザまたは部門の配置を決定します。このポリシーを XML で表示するには、Placement2.xml (../samples/Placement2.xml) を参照してください。

```
<policy>
        <rule>
              <description>PublisherPlacementRule</description>
              <conditions>
                        <or>
                             <if-class-name op="equal">User</if-class-
name>
                              <if-class-name op="equal">Organizational
Unit</if-class-name>
                        </or>
                        <or>
                              <if-src-dn op="in-subtree">o=people,
o=novell</if-src-dn>
                       </or>
                </conditions>
                <actions>
                     <do-set-op-dest-dn>
                         <arg-dn>
                               <token-text>People</token-text>
                               <token-text>\</token-text>
                               <token-unmatched-src-dn convert="true"/>
                         </arg-dn>
                      </do-set-op-dest-dn>
                   </actions>
```

```
        </rule>
</policy>
```

### 名前による配置 : 例

次の DirXML スクリプトポリシーの例では、ユーザの名字の最初の文字に基づいて、特定のコンテナにユーザを配置します。名字が A ～ I で始まるユーザは Users1 コンテナ、J ～ R のユーザは Users2 コンテナ、S ～ Z は Users3 コンテナに配置されます。このポリシーを XML で表示するには、Placement3.xml (../samples/Placement3.xml) を参照してください。

```
<policy>
       <rule>
            <description>Surname - A to I in Users1</description>
            <conditions>
                        <and>
                              <if-class-name op="equal">User</if-
class-name>
                              <if-op-attr mode="regex" name="Surname"
op="equal">[A-I].*</if-op-attr>
                         </and>
            </conditions>
            <actions>
                <do-set-op-dest-dn>
                        <arg-dn>
                              <token-text>Users1</token-text>
                        <token-text>\</token-text>
                              <token-op-attr name="CN"/>
                        </arg-dn>
                  </do-set-op-dest-dn>
            </actions>
       </rule>
       <rule>
            <description>Surname - J to R in Users2</description>
            <conditions>
                     <and>
                             <if-class-name op="equal">User</if-class-
name>
                              <if-op-attr mode="regex" name="Surname"
op="equal">[J-R].*</if-op-attr>
                        </and>
            </conditions>
            <actions>
               <do-set-op-dest-dn>
                     <arg-dn>
                              <token-text>Users2</token-text>
                        <token-text>\</token-text>
                         <token-op-attr name="CN"/>
                        </arg-dn>
                  </do-set-op-dest-dn>
            </actions>
        </rule>
        <rule>
             <description>Surname - S to Z in Users3</description>
```

```
            <conditions>
                        <and>
                              <if-class-name op="equal">User</if-class-
name>
                               <if-op-attr mode="regex" name="Surname"
op="equal">[S-Z].*</if-op-attr>
                         </and>
            </conditions>
            <actions>
                  <do-set-op-dest-dn>
                     <arg-dn>
                              <token-text>Users3</token-text>
                              <token-text>\</token-text>
                              <token-op-attr name="CN"/>
                         </arg-dn>
                  </do-set-op-dest-dn>
            </actions>
      </rule>
</policy>
```

## 3.4.5 コマンド変換ポリシー

コマンド変換ポリシーは、コマンドを置き換えるか追加することによって、Identity Manager がターゲットデータストアに送信するコマンドを変更します。削除コマンドを行わずに、それを変更、移動、または無効化コマンドに置き換えるのは、コマンド変換ポリシーのコマンド置き換えの例です。コマンド変換ポリシーでのコマンドの追加の一般的な例として、追加コマンドの属性値に基づいた変更コマンドの作成があります。

基本的には、コマンド変換ポリシーは、メタディレクトリエンジンに送信されたイベントのデフォルト処理の結果として Identity Manager が実行するコマンドを変更するために使用されます。

また、他のポリシーの説明に合致しないポリシーをここで含めるのも一般的な使用法です。


- 38 ページの 「削除から変更への変換 : 例」
- 39 ページの 「追加の操作の作成 : 例」
- 40 ページの 「パスワード期限の時刻の設定 : 例」


### 削除から変更への変換 : 例

次の DirXML スクリプトポリシーでは、「ログインの無効化」属性の削除操作を変更操作に変換します。このポリシーを XML で表示するには、Comannd1.xml (../samples/Command1.xml) を参照してください。

```
<policy>
      <rule>
            <description>Convert User Delete to Modify</description>
            <conditions>
                  <and>
                              <if-operation op="equal">delete</if-
operation>
                        <if-class-name op="equal">User</if-class-name>
```

```
                  </and>
            </conditions>
            <actions>
                  <do-set-dest-attr-value name="Login Disabled">
                        <arg-value type="state">
                              <token-text>true</token-text>
                        </arg-value>
                  </do-set-dest-attr-value>
                  <do-veto/>
            </actions>
      </rule>
</policy>
```

### 追加の操作の作成：例

次の DirXML スクリプトポリシーでは、ユーザのターゲットコンテナがすでに存在するかどうかを判断します。コンテナがない場合、このポリシーは、コンテナオブジェクトを作成する追加操作を作成します。このポリシーを XML で表示するには、Command2.xml (../samples/Command2.xml) を参照してください。

```
<policy>
      <rule>
            <description>Check if destination container already
exists</description>
            <conditions>
                  <and>
                        <if-operation op="equal">add</if-operation>
                  </and>
            </conditions>
    <actions>
                  <do-set-local-variable name="target-container">
                        <arg-string>
                              <token-dest-dn length="-2"/>
                        </arg-string>
                  </do-set-local-variable>
                  <do-set-local-variable name="does-target-exist">
                        <arg-string>
                              <token-dest-attr class-
name="OrganizationalUnit" name="objectclass">
                                    <arg-dn>
                                          <token-local-variable
name="target-container"/>
                                    </arg-dn>
                              </token-dest-attr>
                        </arg-string>
                  </do-set-local-variable>
            </actions>
      </rule>
      <rule>
            <description>Create the target container if necessary</
description>
            <conditions>
                  <and>
                        <if-local-variable name="does-target-exist"
```

```
op="available"/>
                        <if-local-variable name="does-target-exist"
op="equal"/>
                  </and>
                </conditions>
            <actions>
                  <do-add-dest-object class-name="organizationalUnit"
direct="true">
                        <arg-dn>
                              <token-local-variable name="target-
container"/>
                        </arg-dn>
                  </do-add-dest-object>
                  <do-add-dest-attr-value direct="true" name="ou">
                        <arg-dn>
                              <token-local-variable name="target-
container"/>
                        </arg-dn>
                        <arg-value type="string">
                              <token-parse-dn dest-dn-format="dot"
length="1" src-dn-format="dest-dn" start="-1">
                                    <token-local-variable
name="target-container"/>
                              </token-parse-dn>
                                    </arg-value>
                  </do-add-dest-attr-value>
            </actions>
       </rule>
</policy>
```

### パスワード期限の時刻の設定：例

次の DirXML スクリプトポリシーでは、eDirectory ユーザの「パスワード期限の時刻」属性を変更します。このポリシーを XML で表示するには、Command3.xml (../samples/Command3.xml) を参照してください。

```
<?xml version="1.0" encoding="UTF-8"?>
<policy xmlns:jsystem="http://www.novell.com/nxsl/java/
java.lang.System">
    <rule>
            <description>Set password expiration time for a given
interval from current day</description>
            <conditions>
                        <and>
                        <if-operation op="equal">modify-password</if-
operation>
                  </and>
            </conditions>
            <actions>
                   <do-set-local-variable name="interval">
                        <arg-string>
                              <token-text>30</token-text>
                        </arg-string>
                  </do-set-local-variable>
```

```
                  <do-set-dest-attr-value class-name="User"
name="Password Expiration Time" when="after">
                                                  <arg-association>
                              <token-association/>
                        </arg-association>
                        <arg-value type="string">
                                                                        <token-
xpath expression="round(jsystem:currentTimeMillis() div 1000 +
(86400*$interval))"/>
                        </arg-value>
                  </do-set-dest-attr-value>
            </actions>
      </rule>
</policy>
```

## 3.4.6 スキーママッピングポリシー

スキーママッピングポリシーは、識別ボールトと接続システム間のスキーママッピングの定義を保持します。

識別ボールトスキーマは、eDirectory から読み込まれます。接続されているアプリケーションのスキーマは、接続システムの Identity Manager ドライバから提供されます。2 つのスキーマが特定された後、識別ボールトとターゲットアプリケーション間で単純なマッピングが作成されます。

スキーママッピングポリシーを Identity Manager ドライバ環境設定に定義した後は、対応するデータをマップできます。

次の点に十分注意してください。

- 同じポリシーが両方向で適用されます。
- メタディレクトリエンジンとアプリケーションシム間で渡されるドキュメントは、いずれのチャネルでも、またいずれの方向でも、すべてのドキュメントがスキーママッピングポリシーを通過します。

管理の詳細については、『*Designer 2.1 のポリシー*』の「“ スキーママッピングポリシーの定義 ”」、または『*iManager for Identity Manager 3.5.1 のポリシー*』の「“ スキーママッピングポリシーの定義 ”」を参照してください。


- 41 ページの 「基本的なスキーママッピングポリシー : 例」
- 42 ページの 「カスタムスキーママッピングポリシー : 例」


### 基本的なスキーママッピングポリシー : 例

次の DirXML スクリプトポリシーの例では、基本的なスキーママッピングポリシーの XML ソースをそのまま表示しています。ただし、Identity Manager の Designer を使用してポリシーを編集する場合、デフォルトのスキーママッピングエディタで、ポリシーをグラフィカルに表示および編集できます。このポリシーを XML で表示するには、SchemaMap1.xml (../samples/SchemaMap1.xml) を参照してください。

```
<?xml version="1.0" encoding="UTF-8"?><attr-name-map>
      <class-name>
            <app-name>WorkOrder</app-name>
            <nds-name>DirXML-nwoWorkOrder</nds-name>
```

```
</class-name>
<class-name>
         <app-name>PbxSite</app-name>
         <nds-name>DirXML-pbxSite</nds-name>
</class-name>
<attr-name class-name="DirXML-pbxSite">
         <app-name>PBXName</app-name>
         <nds-name>DirXML-pbxName</nds-name>
</attr-name>
<attr-name class-name="DirXML-pbxSite">
            <app-name>TelephoneNumber</app-name>
           <nds-name>Telephone Number</nds-name>
</attr-name>
<attr-name class-name="DirXML-pbxSite">
         <app-name>LoginName</app-name>
         <nds-name>DirXML-pbxLoginName</nds-name>
 </attr-name>
 <attr-name class-name="DirXML-pbxSite">
         <app-name>Password</app-name>
         <nds-name>DirXML-pbxPassword</nds-name>
 </attr-name>
 <attr-name class-name="DirXML-pbxSite">
         <app-name>Nodes</app-name>
         <nds-name>DirXML-pbxNodesNew</nds-name>
 </attr-name>
</attr-name-map>
```

### カスタムスキーママッピングポリシー : 例

次の DirXML スクリプトポリシーの例は、DirXML スクリプトを使用してカスタムスキーママッピングを実行します。このポリシーを XML で表示するには、SchemaMap2.xml (../samples/SchemaMap2.xml) を参照してください。

```
<?xml version="1.0" encoding="UTF-8"?><policy>
      <rule>
            <!--
            The Schema Mapping Policy can only handle one-to-one
mappings.
            That Mapping Policy maps StudentPersonal addresses.
            This rule maps StaffPersonal addresses.
      -->
            <description>Publisher Staff Address Mappings</
description>
            <conditions>
                  <and>
                        <if-local-variable name="fromNds"
op="equal">false</if-local-variable>
                        <if-xpath op="true">@original-class-name = '
StaffPersonal' </if-xpath>
                  </and>
            </conditions>
            <actions>
                  <do-rename-op-attr dest-name="SA" src-name="Address/
Street/Line1"/>
```

```
                  <do-rename-op-attr dest-name="Postal Office Box"
src-name="Address/Street/Line2"/>
                  <do-rename-op-attr dest-name="Physical Delivery
Office Name" src-name="Address/City"/>
                  <do-rename-op-attr dest-name="S" src-name="Address/
StatePr"/>
                  <do-rename-op-attr dest-name="Postal Code" src-
name="Address/PostalCode"/>
            </actions>
      </rule>
      <rule>
            <description>Subscriber Staff Address Mappings</
description>
            <!--
            The Schema Mapping Policy has already mapped addresses to
StudentPersonal.
            This rule maps StudentPersonal to StaffPersonal.
      -->
            <conditions>
                  <and>
                        <if-local-variable name="fromNds"
op="equal">true</if-local-variable>
                        <if-op-attr name="DirXML-sifIsStaff"
op="equal">true</if-op-attr>
                  </and>
            </conditions>
            <actions>
                  <do-rename-op-attr dest-name="Address/Street/Line1"
src-name="StudentAddress/Address/Street/Line1"/>
                  <do-rename-op-attr dest-name="Address/Street/Line2"
src-name="StudentAddress/Address/Street/Line2"/>
                  <do-rename-op-attr dest-name="Address/City" src-
name="StudentAddress/Address/City"/>
                  <do-rename-op-attr dest-name="Address/StatePr" src-
name="StudentAddress/Address/StatePr"/>
                  <do-rename-op-attr dest-name="Address/PostalCode"
src-name="StudentAddress/Address/PostalCode"/>
            </actions>
      </rule>
</policy>
```

## 3.4.7 出力変換ポリシー

出力変換ポリシーは主に、メタディレクトリエンジンが提供するデータからアプリケーションシムで求められるデータへのデータ形式の変換を行います。次に変換の例を示します。

- 属性値の形式の変換
- XML ボキャブラリの変換
- メタディレクトリエンジンからアプリケーションシムに返されるステータスメッセージのカスタム処理

いずれかのチャネルでメタディレクトリエンジンがアプリケーションシムに提供するドキュメントはすべて出力変換ポリシーを通過します。出力変換はスキーママッピングの後に実行されるので、すべてのスキーマ名はアプリケーションネームスペース内にあります。


- 44 ページの 「属性値の変換 : 例」
- 44 ページの 「ステータスメッセージのカスタム処理」


### 属性値の変換 : 例

次の DirXML スクリプトポリシーの例は、電話番号を「(nnn) nnn-nnnn」形式から「nnn.nnn.nnnn」形式に再フォーマットします。逆の変換については、入力変換ポリシーの例を参照してください。このポリシーを XM で表示するには、Output_Transformation1.xml (../samples/Output_Transformation1.xml) を参照してください。

```
<policy>
      <rule>
            <description>Reformat all telephone numbers from (nnn)
nnn-nnnn to nnn.nnn.nnnn</description>
            <conditions/>
            <actions>
                        <do-reformat-op-attr name="telephoneNumber">
                            <arg-value type="string">
                                        <token-replace-first
regex="^\((\d\d\d)\) *(\d\d\d)-(\d\d\d\d)$" replace-with="$1.$2.$3">
                                                    <token-local-
variable name="current-value"/>
                                    </token-replace-first>
                            </arg-value>
                         </do-reformat-op-attr>
            </actions>
     </rule>
</policy>
```

### ステータスメッセージのカスタム処理

次の DirXML スクリプトポリシーの例では、レベルが「success」と等しくなく、また操作データ内に子の password-publish-status 要素を含むステータスドキュメントを検出し、DoSendEmailFromTemplate ( テンプレートから電子メールを送信 ) アクションを使用して電子メールを生成します。このポリシーを XML で表示するには、Output_Transformation2.xml (../samples/Output_Transformation2.xml) を参照してください。

```
<?xml version="1.0" encoding="UTF-8"?>
      <policy>
                  <description>Email notifications for failed password
publications</description>
            <rule>
                        <description>Send e-mail for a failed publish
password operation</description>
                     <conditions>
                            <and>
                                        <if-global-variable
mode="nocase" name="notify-user-on-password-dist-failure"
op="equal">true</if-global-variable>
                                        <if-operation
```

```
op="equal">status</if-operation>
                                        <if-xpath
op="true">self::status[@level != ’ success’]/operation-data/password-
publish-status</if-xpath>
                                    </and>
                        </conditions>
                        <actions>
                        <!-- generate email notification -->
                         <do-send-email-from-template notification-
dn="\cn=security\cn=Default Notification Collection" template-
dn="\cn=security\cn=Default Notification Collection\cn=Password Sync
Fail">
                               <arg-string name="UserFullName">
                                     <token-src-attr name="Full Name">
                                         <arg-association>
                                                      <token-xpath
expression="self::status/operation-data/password-publish-status/
association"/>
                                         </arg-association>
                                     </token-src-attr>
                                </arg-string>
                                <arg-string name="UserGivenName">
                                      <token-src-attr name="Given Name">
                                             <arg-association>
                                                       <token-xpath
expression="self::status/operation-data/password-publish-status/
association"/>
                                             </arg-association>
                                    </token-src-attr>
                               </arg-string>
                               <arg-string name="UserLastName">
                                    <token-src-attr name="Surname">
                                                      <arg-association>
                                                 <token-xpath
expression="self::status/operation-data/password-publish-status/
association"/>
                                             </arg-association>
                                    </token-src-attr>
                                </arg-string>
                                <arg-string name="ConnectedSystemName">
                                        <token-global-variable
name="ConnectedSystemName"/>
                                </arg-string>
                                <arg-string name="to">
                                       <token-src-attr name="Internet Email
Address">
                                        <arg-association>
                                                <token-xpath
expression="self::status/operation-data/password-publish-status/
association"/>
                                          </arg-association>
                                     </token-src-attr>
                                </arg-string>
                                <arg-string name="FailureReason">
```

```
                                        <token-text/>
                                           <token-xpath
expression="self::status/child::text()"/>
                                </arg-string>
                     </do-send-email-from-template>
               </actions>
         </rule>
</policy>
```

## 3.4.8 入力変換ポリシー

入力変換ポリシーは主に、アプリケーションシムが提供するデータからメタディレクトリエンジンで求められるデータへのデータ形式の変換を行います。次に変換の例を示します。

- 属性値の形式の変換
- XML ボキャブラリの変換
- ドライバハートビート
- アプリケーションシムからメタディレクトリエンジンに返されるステータスメッセージのカスタム処理

いずれかのチャネルでアプリケーションシムがメタディレクトリエンジンに提供するドキュメントはすべて入力変換ポリシーを通過します。


- 46 ページの 「属性値の形式の変換 : 例」
- 47 ページの 「ドライバハートビート : 例」


### 属性値の形式の変換 : 例

次の DirXML スクリプトポリシーの例では、電話番号を「nnn.nnn.nnnn」形式から「(nnn) nnn-nnnn」形式に再フォーマットします。逆の変換については、43 ページのセクション 3.4.7「出力変換ポリシー」の例を参照してください。このポリシーを XML で表示するには、Input_Transformation1.xml (../samples/Input_Transformation1.xml) を参照してください。

```
<policy>
      <rule>
            <description>Reformat all telephone numbers from
nnn.nnn.nnnn to (nnn) nnn-nnnn</description>
            <conditions/>
            <actions>
                  <do-reformat-op-attr name="telephoneNumber">
                        <arg-value type="string">
                                          <token-replace-first
regex="^(\d\d\d)\.(\d\d\d)\.(\d\d\d\d)$" replace-with="($1) $2-$3">

<token-local-variable name="current-value"/>
                                          </token-replace-first>
                        </arg-value>
                  </do-reformat-op-attr>
            </actions>
      </rule>
</policy>
```

### ドライバハートビート : 例

次の DirXML スクリプトポリシーでは、ステータスハートビートイベントを作成します。ドライバのハートビート機能は、各ハートビート間隔ごとに成功メッセージ (HEARTBEAT: $driver) を送信するために使用されます。メッセージは、Novell® Audit によって監視されます。Identity Manager ドライバは、ハートビートをサポートしている必要があります。また、ハートビートがドライバ環境設定ページで有効になっている必要があります。このポリシーを XML で表示するには、Input_Transformation2.xml (../samples/Input_Transformation2.xml) を参照してください。

```
<?xml version="1.0" encoding="UTF-8" ?>
 <policy>
      <rule>
      <description>Heartbeat Rule, v1.01, 040126, by Holger Dopp</
description>
            <conditions>
                  <and>
                        <if-operation op="equal">status</if-operation>
                        <if-xpath op="true">@type="heartbeat"</if-
xpath>
                  </and>
            </conditions>
            <actions>
                  <do-set-xml-attr expression="." name="text1">
                        <arg-string>
                              <token-global-variable
name="dirxml.auto.driverdn" />
             </arg-string>
                  </do-set-xml-attr>
                  <do-set-xml-attr expression="." name="text2">
                        <arg-string>
                              <token-text>HEARTBEAT</token-text>
                        </arg-string>
                  </do-set-xml-attr>
            </actions>
  </rule>
</policy>
```

## 3.5 ポリシーの定義

すべてのポリシーは、次の 2 つの方法のいずれかで定義されます。

- ポリシービルダインタフェースを使用して DirXML スクリプトを作成する。既存の XSLT 以外のルールは、インポート時に自動的に DirXML スクリプトに変換されます。
- XSLT スタイルシートを使用する。

スキーママッピングポリシーは、スキーママッピングテーブルを使用して定義することもできます ( 通常はこちらが使用されます )。

### 3.5.1 ポリシービルダおよび DirXML スクリプト

実装するポリシーのほとんどの定義には、ポリシービルダインタフェースを使用します。ポリシービルダインタフェースでは、グラフィック環境を使用して、ポリシーを簡単に定義および管理できるようになっています。

ポリシービルダ内でルールを作成する際の基礎となる機能は、DirXML スクリプトによって提供されています。ただし、DirXML スクリプトを直接操作する必要はありません。

その代わりに、テスト可能な多岐にわたる条件、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を利用できます。各オプションは、それぞれの場所で有効な選択肢のみを提示するインテリジェントドロップダウンリスト、および共通の値へのクイックリンクを使用して表示されます。

ポリシービルダの詳細については、『*Designer 2.1 のポリシー*』および『*iManager for Identity Manager 3.5.1 のポリシー*』を参照してください。DirXML スクリプトの詳細については、51 ページのセクション 4.1「DirXML スクリプト」を参照してください。

---

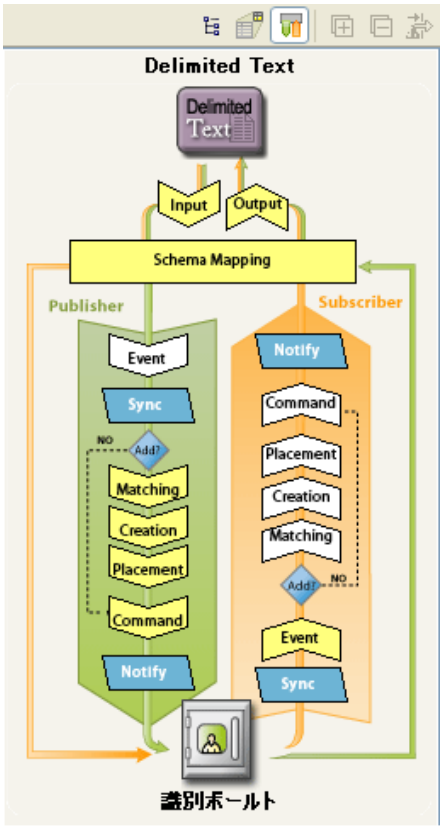
**ヒント :** ポリシービルダを使用する際に必要ではありませんが、『*Identity Manager 3.5.1 DTD リファレンス*』には “DirXML スクリプト DTD ” が収録されています。

---

## 3.6 ポリシーフローの理解

Designer の［ポリシーフロー］領域では、ドライバクラスや属性フィルタのほかに、さまざまなポリシーセットに含まれるポリシーも表示および管理することができます。ポリシーセット ( 例 : 発行者一致ポリシーセット ) を選択すると、ポリシーセットにはそのセットに含まれるポリシーが表示され、ポリシービルダには選択したポリシー内のルールが表示されます。Designer の詳細については、『*Designer 2.1 のポリシー*』を参照してください。

*図 3-3 Designer でのポリシーフロー*




システム内でのドキュメントのフローは、ポリシーフローイメージでも表示されます。たとえば、アプリケーションが生成した入力ドキュメントは、入力ポリシーセットから始まる緑色の矢印で示されている発行者パスに沿って、識別ボールトに移動します。入力ドキュメントに対してメタディレクトリエンジンが生成した結果ドキュメントは、引き続き緑色の矢印に沿って、スキーママッピングおよび出力ポリシーの各セットを経由して接続システムに戻ります。

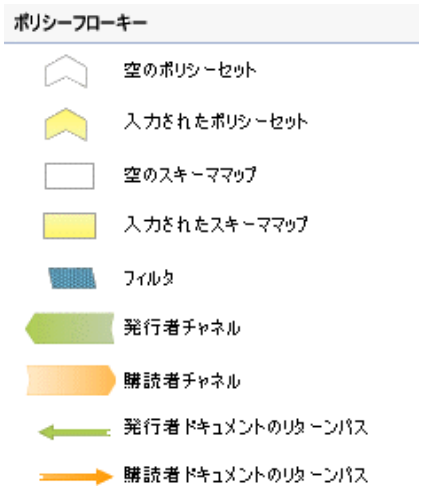
発行者チャネルで生成された入力ドキュメントも同様に動作し、識別ボールトから始まるオレンジ色の矢印に沿って、接続されたアプリケーションへ流れます。アプリケーションと Identity Manager ドライバが生成した結果ドキュメントは、引き続きオレンジ色の矢印に沿って、入力およびスキーママッピングの各ポリシーセットを経由して識別ボールトに戻ります。

追加操作だけは、発行者チャネルおよび購読者チャネルの両方で、一致、作成、および配置の各ポリシーセットを経由して流れます。その他の操作はすべて、点線に沿って、コマンドポリシーセットに至るまでこれらのポリシーセットをバイパスします。

## 3.6.1 ポリシーフローキー

ポリシーフローキーは、ポリシーフローイメージのさまざまな側面を記述します。たとえば、空のポリシーセットや入力済みのポリシーセットのほかに、発行者および購読者の各チャネル上でドキュメントがたどるパスなどです。購読者チャネル上のドキュメントは識別ボールトから始まる一方で、発行者チャネル上のドキュメントは接続されたアプリケーションから始まります。

*図 3-4 ポリシーフローキー*




## 3.6.2 ポリシーフローのキーボードサポート

ポリシーフロービューではキー操作がサポートされています。サポートされているキー操作については、50 ページの 表 3-1 を参照してください。

*表 3-1 ポリシーフローのキーボードサポート*

| ショートカットキー | 説明 |
|---|---|
| < 上矢印 > | 上へ移動 |
| < 下矢印 > | 下へ移動 |
| <Tab> | 下へ移動 |
| <Shift>+<Tab> | 上へ移動 |
| < 左矢印 > | チャネルの切り替え |
| < 右矢印 > | チャネルの切り替え |

# ポリシーコンポーネントの理解

# 4



## 4.1 DirXML スクリプト

DirXML® スクリプトは、Identity Manager ポリシーを実装する主要な方法です。このスクリプトでは、順序が指定された一連のルールによって実装されるポリシーを記述します。ルールにはテストする一連の条件と、その条件を満たしたときに順次実行される一連のアクションが含まれています。

DirXML スクリプトの作成には、使いやすい GUI を備えたポリシービルダを使用します。

Identity Manager は XML ベースのアプリケーションなので、DirXML スクリプトでは、XML ドキュメントを使用して、識別ボールトと外部データストア間で送信されるデータを変更および操作します。DirXML スクリプトを理解するには、XML を理解する必要があります。XML の詳細については、W3C Extensible Markup Language (XML) (http://www.w3.org/XML/) の Web サイトを参照してください。

DirXML スクリプトには、DirXML スクリプトの動作方法を定義したドキュメントタイプ定義 (DTD) があります。Identity Manager が使用する DTD を理解するには、『*Identity Manager 3.5.1 DTD リファレンス*』を参照してください。ここには次の DTD が収録されています。

- “ フィルタ DTD”
- “NDS DTD”
- “ マップ DTD”
- “DirXML スクリプト DTD”
- “DirXML エンタイトルメント DTD”

## 4.2 変数

DirXML スクリプトでは、グローバルおよびローカルの 2 種類の変数がサポートされています。グローバル変数は、ドライバまたはドライバセットのグローバル構成値で定義される変数です。グローバル変数は本質的に読み込み専用です。ローカル変数は、ポリシーで設定される変数です。ローカル変数は、ポリシーまたはドライバという 2 つの異なるスコープの一方に存在できます。ポリシースコープ変数は、変数を設定したポリシーによって現在の操作の処理中にのみ表示できます。ドライバスコープ変数は、ドライバを停止するまで、同じドライバ内で実行されるすべての DirXML スクリプトポリシーから表示できます。

変数名は、有効な XML 名でなければなりません。有効な XML 名の詳細については、W3C Extensible Markup Language (XML) (http://www.w3.org/TR/2006/REC-xml11-20060816/#sec-suggested-names) を参照してください。

自動的に定義されるグローバル変数やローカル変数も数多くあります。

***表 4-1*** *定義済みグローバル変数とローカル変数*

| 名前 | タイプ | 説明 |
|---|---|---|
| dirxml.auto.driverdn | グローバル / 文字列 | 現在のドライバのスラッシュ形式の DN。 |
| dirxml.auto.driverguid | グローバル / 文字列 | 現在のドライバの GUID。 |
| dirxml.auto.treename | グローバル / 文字列 | ローカル eDirectory™ インスタンスのツリー名。 |
| fromNDS | ポリシーローカル / ブール | ソースデータストアが eDirectory の場合、True。ソースデータストアが接続されたアプリケーションの場合、False。 |
| destQueryProcessor | ポリシーローカル /java オブジェクト | ソースデータストアにクエリを実行する際に使用される XdsQueryProcessor のインスタンス。 |
| srcQueryProcessor | ポリシーローカル /java オブジェクト | ターゲットデータストアにクエリを実行する際に使用される XdsQueryProcessor のインスタンス。 |
| destCommandProcessor | ポリシーローカル /java オブジェクト | ターゲットデータストアにクエリを実行する際に使用される XdsCommandProcessor のインスタンス。 |
| srcCommandProcessor | ポリシーローカル /java オブジェクト | ソースデータストアにクエリを実行する際に使用される XdsCommandProcessor のインスタンス。 |
| dnConverter | ポリシーローカル /java オブジェクト | DNConverter のインスタンス。 |
| current-node | ポリシーローカル / ノードセット | 各要素の各繰り返し用のループ変数。 |
| current-value | ポリシーローカル / ノードセット | 再フォーマット操作属性の各繰り返し用のループ変数。 |
| current-op | ポリシーローカル / ノードセット | 現在の操作。**<do-set-local-variable> 要素を使用してこの変数を設定すると、**<arg-node-set> で指定されている最初の操作が、現在のポリシー実行の残りの間、またはこの変数が別の値に設定されるまで、現在の操作になります。新しい現在の操作は、元の現在の操作の要素の兄弟でなければならず、現在のポリシーによって追加されている必要があります。 |

## 4.3 変数の拡張

多くの条件、アクション、およびトークンでは、属性またはコンテンツで動的な変数の拡張がサポートされています。動的な変数の拡張がサポートされている場合、フォーム $<variable-name>$ の埋め込み参照は、指定した名前を持つローカル変数またはグローバル変数に置き換えられます。$<variable-name>$ は有効な変数名である必要があります。有効な XML 名の詳細については、W3C Extensible Markup Language (XML) (http://www.w3.org/TR/2004/REC-xml-20040204/#NT-Name) を参照してください。

指定した変数が存在しない場合、参照は空の文字列に置き換えられます。単一の $ を使用する際に、変数参照として解釈されないようにしたい場合は、$ を追加してエスケープ処理する必要があります ( 例 : 「$$100.00 借りている」 )。変数の拡張をサポートしているコンテンツ属性については、『*Identity Manager 3.5.1 DTD リファレンス*』を参照してください。

## 4.4 日付 / 時刻パラメータ

日付と時刻を扱うトークンには、日付と時刻の表し方のフォーマット、言語、およびタイムゾーンを処理する引数があります。日付フォーマットの引数は、「!」を使用するか、または「!」文字を使用しないことで指定できます。フォーマットが「!」文字で始まっている場合、そのフォーマットは名前付きフォーマットです。有効な名前は、53 ページの 表 4-2 で定義されています。

***表 4-2*** *有効な日付 / 時刻パラメータ*

| 名前 | 説明 |
|---|---|
| !CTIME | 1970 年 1 月 1 日深夜 12 時からの秒数 (eDirectory の時刻構文と互換性があります )。 |
| !JTIME | 1970 年 1 月 1 日深夜 12 時からのミリ秒数 (Java* の時刻と互換性があります )。 |
| !FILETIME | 1601 年 1 月 1 日からの 100 ナノ秒間隔数 (Win32 の FILETIME と互換性があります )。 |
| !FULL.TIME | 言語固有の完全な時刻フォーマット。 |
| !LONG.TIME | 言語固有の長い時刻フォーマット。 |
| !MEDIUM.TIME | 言語固有の中間の長さの時刻フォーマット。 |
| !SHORT.TIME | 言語固有の短い時刻フォーマット。 |
| !FULL.DATE | 言語固有の完全な日付フォーマット。 |
| !LONG.DATE | 言語固有の長い日付フォーマット。 |
| !MEDIUM.DATE | 言語固有の中間の長さの日付フォーマット。 |
| !SHORT.DATE | 言語固有の短い日付フォーマット。 |
| !FULL.DATETIME | 言語固有の完全な日付 / 時刻フォーマット。 |
| !LONG.DATETIME | 言語固有の長い日付 / 時刻フォーマット。 |
| !MEDIUM.DATETIME | 言語固有の中間の長さの日付 / 時刻フォーマット。 |

| 名前 | 説明 |
|---|---|
| !SHORT.DATETIME | 言語固有の短い日付 / 時刻フォーマット。 |

フォーマットが「!」で始まっている場合は、Java クラス java.text.SimpleDateFormat で認識できるパターンに準拠したカスタム日付 / 時刻フォーマットと解釈されます。

言語の引数は、IETF RFC 3066 に準拠した識別子で指定できます。システムが理解できる識別子のリストを取得するには、Java クラス java.util.Locale.getAvailableLocales() をコールし、結果に含まれる下線文字をすべてハイフンに置き換えます。言語の引数を省略するか、空白にすると、システムのデフォルトの言語が使用されます。

タイムゾーンの引数は、Java クラス java.util.TimeZone.getTimeZone() が認識可能な識別子で指定できます。システムが理解できる識別子のリストを取得するには、Java クラス java.util.TimeZone.getAvailableIDs() をコールします。タイムゾーンの引数を省略するか、空白にすると、システムのデフォルトのタイムゾーンが使用されます。

## 4.5 正規表現

正規表現とは、あるパターンに従ったテキスト文字列を照合するための式です。正規表現は、標準文字とメタ文字から構成されます。標準文字には、大文字と小文字、数字があります。メタ文字には特別な意味があります。次の表に、一般的なメタ文字とその意味を示します。

***表 4-3*** *一般的な正規表現*

| メタ文字 | 説明 |
|---|---|
| . | 任意の 1 文字を意味します。 |
| $ | 行の終わりを意味します。 |
| ^ | 行の先頭を意味します。 |
| * | 直前の文字が 0 個以上含まれることを意味します。 |
| \ | リテラルのエスケープ文字です。この文字を使用することで、検索対象に任意のメタ文字を指定できます。たとえば、「\$」と指定した場合、行の終わりではなく、$1000 が検索結果になります。 |
| [ ] | 角括弧で囲まれた文字のいずれかを意味します。 |
| [0-9] | ハイフンの前後の文字範囲が対象になります。この例では、すべての数字を意味します。 |
| [A-Za-z] | 複数の範囲が対象になります。この例では、すべての大文字と小文字が対象になります。 |
| (?u) | Unicode* 対応のケースフォールディングを有効にします。このフラグはパフォーマンスに影響を及ぼす可能性があります。 |
| (?i) | 大文字と小文字を区別した一致を有効にします。 |

引数ビルダは、Java で定義されている正規表現を使用するように設計されています。Java Web サイト (http://java.sun.com/j2se/1.4.2/docs/api/java/util/regex/Pattern.html) で詳細情報を参照できます。

## 4.6 XPath 1.0 の式

条件、アクション、およびトークンの中には、引数で XPath 1.0 の式を使用するものがあります。XPath は、XSLT および XPointer とで共有される機能に対し、共通の構文とセマンティックを提供するために開発された言語です。主に XML ドキュメントのアドレス指定で使用されますが、文字列、数値およびブールなどのデータ操作を行う基本的な機能も備わっています。

XPath の仕様では、埋め込みアプリケーションが、情報を定義された複数のアプリケーションにコンテキストを提供する必要があります。DirXML スクリプト (51 ページのセクション 4.1「DirXML スクリプト」を参照 ) では、XPath は次のコンテキストで評価されます。

- 式の記述で特に指定されていない限り、コンテキストノードが現在の操作になる。
- コンテキストの位置とサイズが 1。
- 使用可能な変数が次のように複数ある。
  - Identity Manager 内でスタイルシートのパラメータとして使用可能なもの ( 現在のところ、fromNDS、srcQueryProcessor、destQueryProcessor、srcCommandProcessor、destCommandProcessor および dnConverter)。
  - グローバル設定変数。
  - ローカルポリシーの変数。
  - 複数の変数ソース間で名前が競合する場合、優先順位はローカル ( ポリシースコープ )、ローカル ( ドライバスコープ )、グローバルの順になる。
  - XPath の構文のため、名前にコロン文字が含まれる変数には XPath からはアクセスできない。
- 利用できるネームスペース定義が複数ある。
  - XMNS:prefix を使用して <policy> 要素で明示的に宣言されたネームスペース。
  - 暗黙的に定義された次のネームスペース ( 同じプレフィックスが明示的に定義されている場合を除く )。
    - xmlns:js=“http://www.novell.com/nxsl/ecmascript”
    - xmlns:es=“http://www.novell.com/nxsl/ecmascript”
    - xmlns:query=“http://www.novell.com/nxsl/java/com.novell.nds.dirxml.driver.XdsQueryProcessor”
    - xmlns:cmd=“http://www.novell.com/nxsl/java/com.novell.nds.dirxml.driver.XdsCommandProcessor”
    - xmlns:jdbc=“urn:dirxml:jdbc”
  - 別途マップされていないネームスペースプレフィックスは自動的に http://www.novell.com/nxsl/java/ <*prefix*> にマップされる ( プレフィックスが、イントロスペクションを使用して利用可能な Java クラスに解決できる Java クラスの完全修飾クラス名である場合のみ )。
  - プレフィックスを Java クラスに関連付けるためのネームスペース宣言は、ポリシー要素で宣言する必要がある。

- 使用可能な関数が次のように複数ある。
  - 組み込まれている XPath 1.0 のすべての関数。
  - NXSL で提供されている Java 拡張関数。
    - Java 拡張関数にアクセスするには、http://www.novell.com/nxsl/java/ <*fully-qualified-class-name*> というフォームの URI にマップされているネームスペースプレフィックスを使用します。
    - 利便性のため、別途マップされていないプレフィックスはすべて http://www.novell.com/nxsl/java/ <*prefix*> にマップされます ( プレフィックスが、イントロスペクションによって検出できる Java クラスの完全修飾クラス名である場合 )。
  - NXSL で提供されている ECMAScript 拡張関数
    - ECMAScript 拡張関数定義は、ドライバに関連付けられている ECMAScript リソースのセットから利用します。
    - ECMAScript 拡張関数にアクセスするには、URI http://www.novell.com/nxsl/ecmascript にマップされているネームスペースプレフィックスを使用します。
    - 利便性のため、プレフィックス js および es は、別途明示的に定義しない限り、どちらも暗黙的に http://www.novell.com/nxsl/ecmascript にマップされます。

XPath の詳細情報は、W3 Web サイト (http://www.w3.org/TR/1999/REC-xpath-19991116) で参照できます。

# Identity Manager ポリシーのダウンロード

5

Novell® では、ダウンロードして自分の環境で使用できるサンプルポリシーを提供しています。これらのポリシーは、Novell のサポート Web サイト (http://support.novell.com/patches.html) で入手できます。ポリシーをダウンロードするには :

1 Novell のサポート Web サイト (http://support.novell.com/patches.html) で、［*View the minimum patch list ( 最小限のパッチリストを表示)*］を選択します。
2 ［製品または技術］フィールドで［Identity Manager］を参照して選択し、［*検索*］をクリックします。
3 目的のポリシーを参照して選択します。

   ダウンロードできるポリシーのリストは、表 5-1 に記載されています。
4 ［*ダウンロードに進む*］を選択して、ポリシーをダウンロードします。
5 ファイル名の横にある［*ダウンロード*］をクリックします。
6 ［*保存*］をクリックして、ファイルを保存する場所を参照して選択します。
7 ［*保存*］をクリックして、［*閉じる*］をクリックします。
8 ファイルを抽出して How_To_Install.rtf ファイルを読み、インストール方法を確認します。

***表 5-1*** *ダウンロード可能なポリシー*

| 名前 | ファイル名 |
|---|---|
| 名字で配置するポリシー | placebyname.tgz |
| ポリシー : 電子メール属性の値のリセット | pushback.tgz |
| 属性を強制的に存在させるためのポリシー | requiredattrs.tgz |
| ポリシー : GivenName および Surname からの電子メールの作成 | setemailname.tgz |
| ポリシー : GivenName および Surname からの FullName の作成 | synthfullname.tgz |
| ポリシー : 姓 / 名の大文字への変換 | uppercasenames.tgz |
| 役職に基づいてグループへユーザを追加するためのポリシー | addcreategroups.tgz |
| ポリシー : 役職に基づくユーザへのテンプレートの割り当て | assigntemplate.tgz |
| 退職時のユーザアカウントの無効化と移動 | dismvonterm.tgz |
| イベントをフィルタ処理するためのポリシー | filterby.tgz |
| 役職属性に基づくユーザのグループの制御 | groupchange.tgz |

Designer を使用してファイルをインポートするには、『*Designer 2.1 のポリシー*』の「“XML ファイルからのポリシーのインポート”」を参照してください。iManager を使用してファイルをインポートするには、『*iManager for Identity Manager 3.5.1 のポリシー*』の「“XML ファイルからのポリシーのインポート”」を参照してください。

# 6 XSLT スタイルシートを使用したポリシーの定義

XSLT は XML ドキュメントを変換するための標準言語で、ポリシーを XSLT スタイルシートとして実装するために使用できます。メタディレクトリエンジン内の XSLT プロセッサは、1999 年 11 月の W3C 勧告に準拠しています。関連の仕様については、次を参照してください。

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

次の項では、Identity Manager で XSLT スタイルシートを使用する具体的な方法について説明します。



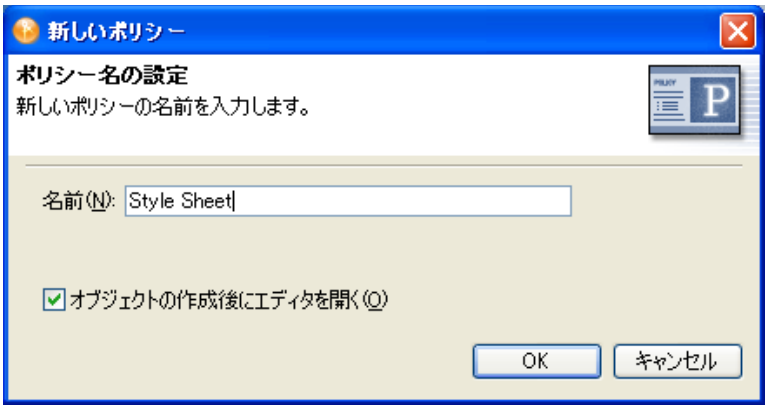
## 6.1 Designer による XSLT スタイルシートの管理

XSLT ポリシーのスタイルシートは、Designer を使用して追加、変更、および削除できます。次の項では、詳細について説明します。

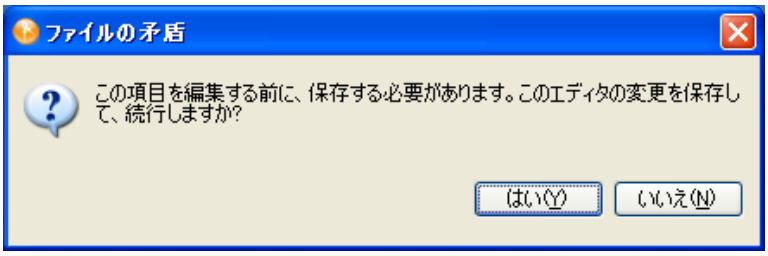


### 6.1.1 Designer による XSLT スタイルシートの追加

1. Designer でプロジェクトを開き、［*アウトライン*］タブを選択します。
2. スタイルシートを指定するドライバと場所を選択します。
3. 右クリックして、*［新規作成］*>*［XSLT］*の順に選択します。
4. スタイルシートの名前を指定します。
5. ［*Open Editor after creating policy ( ポリシーの作成後にエディタを開く )*］を選択し、［*OK*］をクリックします。




6 ［*はい*］を選択して、新しいスタイルシートを編集する前にプロジェクトを保存します。




7 「*ここにカスタムテンプレートを追加します。*」という行の下に、スタイルシート情報を追加します。


8 ［*ファイル*］>［*Save to ( 形式を指定して保存)*］の順にクリックして、スタイルシートを保存します。

### 6.1.2 Designer による XSLT スタイルシートの変更

1 Designer でプロジェクトを開き、［*アウトライン*］タブを選択します。
2 変更するスタイルシートを選択します。
3 右クリックし、［*編集*］を選択します。

### 6.1.3 Designer による XSLT スタイルシートの削除

1 Designer でプロジェクトを開き、［*アウトライン*］タブを選択します。
2 削除する XSLT スタイルシートを選択して右クリックし、*［削除］*を選択します。

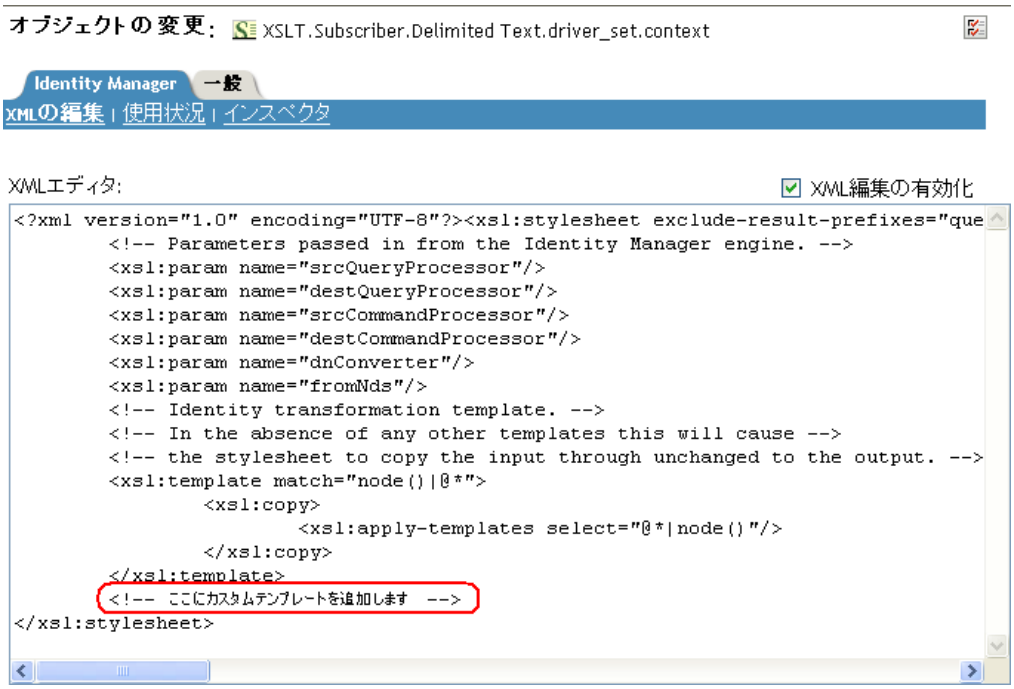
## 6.2 iManager による XSLT スタイルシートの管理

XSLT ポリシーのスタイルシートは、iManager を使用して追加、変更、および削除します。次の項では、詳細について説明します。

- 61 ページのセクション 6.2.1「iManager による XSLT ポリシーの追加」
- 62 ページのセクション 6.2.2「iManager による XSLT スタイルシートの変更」
- 62 ページのセクション 6.2.3「iManager による XSLT スタイルシートの削除」

### 6.2.1 iManager による XSLT ポリシーの追加

1 管理するドライバの［Identity Manager ドライバの概要］を開きます。
2 XSLT スタイルシートを追加するポリシーを表すアイコンをクリックします。
3 ［*挿入*］をクリックします。
4 新しい XSLT スタイルシートの名前を指定して、［*XSLT*］を選択し、［*OK*］をクリックします。
5 ［*XML 編集の有効化*］を選択して、XSLT スタイルシートを編集します。
6 「*ここにカスタムテンプレートを追加します。*」という行の下に、スタイルシート情報を追加します。




7 ［*OK*］をクリックして XSLT スタイルシートを保存します。

### 6.2.2 iManager による XSLT スタイルシートの変更

1 管理するドライバの［Identity Manager ドライバの概要］を開きます。
2 変更する XSLT スタイルシートが保存されているポリシーを表すアイコンをクリックします。
3 変更する XSLT スタイルシートをポリシーのリストから選択し、［編集］をクリックします。
4 XSLT スタイルシートを編集して、［*OK*］をクリックします。

### 6.2.3 iManager による XSLT スタイルシートの削除

1 管理するドライバの［Identity Manager ドライバの概要］を開きます。
2 削除する XSLT スタイルシートが保存されているポリシーを表すアイコンをクリックします。
3 削除する XSLT スタイルシートをポリシーのリストから選択し、［削除］をクリックします。
4 ［*OK*］をクリックして、XSLT スタイルシートを削除することを確認します。

## 6.3 XSLT スタイルシートに事前入力される情報

iManager または Designer で作成した新しいスタイルシートには、識別情報変換を実装するスタイルシートが事前入力されています。他にテンプレートがない場合は、識別情報の変換によって、入力 XML ドキュメントを通過させることができます。その際、スタイルシートは変更されません。ポリシーは通常、テンプレートを追加することで、変更する

XML に対してのみ動作するように実装します。スタイルシートを使用して、XDS とは異なる XML ボキャブラリ (SOAP 用入出力変換ポリシーや区切りテキストドライバなど ) 間でドキュメントを変換する場合、識別情報テンプレートを削除しなければならない場合があります。

## 6.4 Identity Manager から受け取るパラメータの使用

メタディレクトリエンジンは、ポリシーのスタイルシートに、次のスタイルシートパラメータを渡します。

***表 6-1*** *スタイルシートのパラメータ*

| パラメータ | 説明 |
|---|---|
| srcQueryProcessor | XdsQueryProcessor インタフェースを実装する Java オブジェクト。これにより、スタイルシートがソースデータストアに対し、より多くの情報を照会できるようになります。 |
| destQueryProcessor | XdsQueryProcessor インタフェースを実装する Java オブジェクト。これにより、スタイルシートがターゲットデータストアに対し、より多くの情報を照会できるようになります。 |
| srcCommandProcessor | XdsCommandProcessor インタフェースを実装する Java オブジェクト。これにより、スタイルシートがイベントソースに対し、コマンドをライトバックできるようになります。 |
| destCommandProcessor | XdsCommandProcessor インタフェースを実装する Java オブジェクト。これにより、スタイルシートがコマンドを発行して、ターゲットデータストアにコマンドを直接送信できるようになります。 |
| dnConverter | XdsCommandProcessor インタフェースを実装する Java オブジェクト。これにより、スタイルシートが識別ボールトオブジェクトの DN をあるフォーマットから別のフォーマットに変換できるようになります。詳細については、「Interface DNCoverter (http://developer.novell.com/ndk/doc/dirxml/dirxmlbk/api/com/novell/nds/dirxml/driver/DNConverter.html)」を参照してください。 |
| fromNds | ソースデータストアが識別ボールトである場合には「True」、接続システムである場合には「False」であるブール値です。 |

iManager または Designer で作成した新しいスタイルシートには、これらのパラメータの宣言が含まれるスタイルシートが事前入力されています。

スキーママッピングポリシー、入力変換ポリシー、および出力変換ポリシーでクエリやコマンドパラメータを使用する場合、次の制限が適用されます。

- アプリケーションシムに発行されたクエリは、そのアプリケーションシムによって予測された形式である必要があります。つまり、スキーマ名はアプリケーションのネームスペース内にあり、シムによってどのような XML ボキャブラリがネイティブで使用されているかをクエリで確認する必要があります。クエリには関連付けの参照は追加されません。
- アプリケーションシムからの応答は、そのシムの形式で返されます。変更やスキーママッピング、関連付けの参照の解決は行われません。

- eDirectory™ に発行されたクエリは、eDirectory によって予測された形式である必要があります。つまり、スキーマ名は eDirectory ネームスペース内にあり、クエリは XDS である必要があります。関連付けの参照は解決されません。
- アプリケーションシムからの応答は、そのシムの形式で返されます。変更やスキーママッピングは行われません。

### 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 を実装するかどうかによって決まります。クエリを実行するには、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。この作業は、次の内容をスタイルシートの <xsl:stylesheet> または <xsl:transform> 要素に追加することで行います。

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

iManager または Designer で作成したスタイルシートには、このネームスペース宣言が事前入力されています。クエリプロセッサについての詳細は、「Class XdsQueryProcessor (http://developer.novell.com/ndk/doc/dirxml/dirxmlbk/api/com/novell/nds/dirxml/driver/XdsQueryProcessor.html)」を参照してください。

次の例では、クエリプロセッサの 1 つを使用しています ( 長い行は折り返されているため、< で始まっていません )。このスタイルシートを表示するには、Query_Processors.xsl (../samples/Query_Processors.xsl) を参照してください。

```
<!-- Query object name queries NDS for the passed object name -->

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

<!-- build an xds query as a result tree fragment -->
   <xsl:variable name="query">
            <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>
   </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>
```

他の例です。

```
<?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>
```

# 6.5 拡張関数の使用

XSLT は、ある種の変換を実行するには優れたツールですが、ゼロでない文字列操作や反復処理などの変換では、十分な機能が備わっているとはいえません。ただし、Novell XSLT プロセッサには、スタイルシートから Java ( 広い意味では JNI でアクセス可能な他の言語 ) で実装された関数を呼び出すことができる拡張関数が実装されています。

具体的な例については、クエリプロセッサを使用する 64 ページの 「クエリプロセッサ」、および Java を使用した文字列操作を示した次の例を参照してください。長い行は折り返されているため、< で始まっていません。このスタイルシートを表示するには、Extension_Functions.xsl (../samples/Extension_Functions.xsl) を参照してください。

```
<!-- 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>
```

## 6.6 パスワードの変更：作成ポリシーの例

次のスタイルシートは、作成ポリシーで使用できます。これはユーザを作成し、そのユーザの Surname と CN の属性からパスワードを生成して、中断と変換を試行しているイベントを除き、ドキュメント内のすべてを通過する識別情報の変換を実行します。このスタイルシートを表示するには、Create_Password.xsl (../samples/Create_Password.xsl) を参照してください。

```
<?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>
```

# 6.7 eDirectory ユーザの作成：作成ポリシーの例

次のスタイルシートは、作成ポリシーで使用できます。ここでは、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、人事部のデータベースに新入社員を作成し、これをネットワーク上に移すという作業が基になっています。このスタイルシートは、ユーザの名前と名字を取得して、一意の CN を eDirectory ツリー内に生成します。eDirectory では、一意の CN を用いる必要があるのは CN が属すコンテナ内に限られますが、このスタイルシートでは eDirectory ツリー内のすべてのコンテナで一意になるようにします。このスタイルシートを表示するには、Create_User.xsl (../samples/Create_User.xsl) を参照してください。

```
<?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 Identity Manager so
things
     are pretty to look at -->
<xsl:strip-space elements="*"/>
<xsl:preserve-space elements="value,component"/>
<xsl:output method="xml" indent="yes"/>

<!-- Identity Manager 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>
```