Novell Nsure™ Identity Manager

2

August 3, 2004

www.novell.com

POLICY BUILDER AND DRIVER CUSTOMIZATION GUIDE

Novell®

Legal Notices

Novell, Inc. makes no representations or warranties with respect to the contents or use of this documentation, and specifically disclaims any express or implied warranties of merchantability or fitness for any particular purpose. Further, Novell, Inc. reserves the right to revise this publication and to make changes to its content, at any time, without obligation to notify any person or entity of such revisions or changes.

Further, Novell, Inc. makes no representations or warranties with respect to any software, and specifically disclaims any express or implied warranties of merchantability or fitness for any particular purpose. Further, Novell, Inc. reserves the right to make changes to any and all parts of Novell software, at any time, without any obligation to notify any person or entity of such changes.

You may not export or re-export this product in violation of any applicable laws or regulations including, without limitation, U.S. export regulations or the laws of the country in which you reside.

Copyright © 2000-2004 Novell, Inc. All rights reserved. No part of this publication may be reproduced, photocopied, stored on a retrieval system, or transmitted without the express written consent of the publisher.

U.S. Patent Nos. 5,349,642; 5,608,903; 5,671,414; 5,677,851; 5,758,344; 5,784,560; 5,818,936; 5,828,882; 5,832,275; 5,832,483; 5,832,487; 5,870,561; 5,870,739; 5,873,079; 5,878,415; 5,884,304; 5,919,257; 5,933,503; 5,933,826; 5,946,467; 5,956,718; 6,016,499; 6,065,017; 6,105,062; 6,105,132; 6,108,649; 6,167,393; 6,286,010; 6,308,181; 6,345,266; 6,424,976; 6,516,325; 6,519,610; 6,539,381; 6,578,035; 6,615,350; 6,629,132. Patents Pending.

Novell, Inc.
1800 South Novell Place
Provo, UT 84606
U.S.A.

www.novell.com

Policy Builder and Driver Customization Guide
August 3, 2004

Online Documentation: To access the online documentation for this and other Novell products, and to get updates, see www.novell.com/documentation.

Novell Trademarks

DirXML is a registered trademark of Novell, Inc. in the United States and other countries.
eDirectory is a trademark of Novell, Inc.
Novell is a registered trademark of Novell, Inc. in the United States and other countries.
Nsure is a trademark of Novell, Inc.

Third-Party Trademarks

All third-party trademarks are the property of their respective owners.

Contents

About This Guide 9

1 Policies and Filters 11

What Are Policies and Filters? 11
Terminology Changes from DirXML 1.x. 13
Introduction to Policies 13
Basic Policies 13
Transformation Policies 15
Defining Policies 15
Introduction to Filters 16

2 Defining Policies Using Policy Builder 19

Policy Builder Tasks 19
Opening Policy Builder 19
Creating a Policy 20
Defining Individual Rules within a Policy 20
Defining Individual Arguments within a Rule 21
Modifying a Policy 24
Deleting a Policy 24
Importing a Policy from an XML File 24
Exporting a Policy to an XML File 24
Creating a Policy Reference 25
Conditions 25
If Association 26
If Attribute 27
If Class Name 29
If Destination Attribute 30
If Destination DN 32
If Entitlement 34
If Global Configuration Value 36
If Local Variable 37
If Named Password 38
If Operation Attribute 39
If Operation 41
If Operation Property 42
If Password 43
If Source Attribute 44
If Source DN 46
If Xpath Expression 48
Actions 48
Add Association 50
Add Destination Attribute Value 51
Add Destination Object 52
Add Source Attribute Value 53
Add Source Object 54

Append XML Element . . . 55
Append XML Text . . . 56
Break . . . 57
Clear Destination Attribute Value . . . 58
Clear Operation Property . . . 59
Clear Source Attribute Value . . . 60
Clone Operation Attribute . . . 61
Clone by Xpath Expressions . . . 62
Delete Destination Object . . . 63
Delete Source Object . . . 64
Find Matching Object . . . 65
For Each . . . 67
Generate Event . . . 68
Move Destination Object . . . 70
Move Source Object . . . 71
Reformat Operation Attribute . . . 72
Remove Association . . . 73
Remove Destination Attribute Value . . . 74
Remove Source Attribute Value . . . 75
Rename Destination Object . . . 76
Rename Operation Attribute . . . 77
Rename Source Object . . . 78
Send Email . . . 79
Send Email From Template . . . 81
Set Default Attribute Value . . . 83
Set Destination Attribute Value . . . 84
Set Destination Password . . . 85
Set Local Variable . . . 86
Set Operation Association . . . 87
Set Operation Class Name . . . 88
Set Operation Destination DN . . . 89
Set Operation Property . . . 90
Set Operation Source DN . . . 91
Set Operation Template DN . . . 92
Set Source Attribute Value . . . 93
Set Source Password . . . 94
Set XML Attribute . . . 95
Status . . . 96
Strip Operation Attribute . . . 97
Strip Xpath . . . 98
Trace Message . . . 99
Veto . . . 100
Veto If Operation Attribute Not Available . . . 101
Nouns . . . 101
Added Entitlement . . . 102
Association . . . 103
Attribute . . . 104
Class Name . . . 105
Destination Attribute . . . 106
Destination DN . . . 107
Destination Name . . . 108
Entitlement . . . 109
Global Configuration Value . . . 110
Local Variable . . . 111
Named Password . . . 112

Operation . . . 113
Operation Attribute . . . 114
Operation Property . . . 115
Password . . . 116
Removed Attribute . . . 117
Removed Entitlement . . . 118
Source Attribute . . . 119
Source DN . . . 120
Source Name . . . 121
Text . . . 122
Unique Name . . . 123
Unmatched Source DN . . . 125
XPath . . . 126
Verbs . . . 126
Escape Destination DN . . . 127
Escape Source DN . . . 128
Lower Case . . . 129
Parse DN . . . 130
Replace All . . . 132
Replace First . . . 133
Substring . . . 134
Upper Case . . . 135
Values . . . 135
Comparison Modes . . . 135

3 Defining Policies using XSLT Style Sheets 137

Managing XSLT Style Sheets in iManager . . . 137
Adding an XSLT Policy . . . 137
Restrictions . . . 138
Matching Rule Restrictions . . . 138
Create Rule Restrictions . . . 139
Placement Rule Restrictions . . . 139
Starting with an Identity Transformation . . . 139
Using the Parameters that DirXML Passes . . . 139
Using Extension Functions . . . 142
Testing Style Sheets Outside of DirXML . . . 142
Creating a Password Example: Create Rule . . . 143
Creating an eDirectory User Example: Create Rule . . . 144

4 Defining Filters 149

Filter Tasks . . . 149
Managing Filters . . . 149
Viewing and Modifying Filters . . . 149

About This Guide

Novell® Nsure™ Identity Manager 2, which is powered by DirXML®, is a data sharing and synchronization service that enables applications, directories, and databases to share information. It links together scattered information and enables you to establish policies that govern automatic updates to designated systems when identity changes occur.

Identity Manager provides the foundation for account provisioning, security, single sign-on, user self-service, authentication, authorization, automated workflows and Web services. It allows you to integrate, manage and control your distributed identity information so you can securely deliver the right resources to the right people.

This guide provides detailed reference on Policy Builder and Driver Configuration in Identity Manager 2.

Additional Documentation

For documentation on using the DirXML drivers, see the DirXML Documentation Web site (http://www.novell.com/documentation/lg/dirxmldrivers/index.html)

For documentation on Identity Manager 2.0, see the DirXML Documentation Web site (http://www.novell.com/documentation/lg/dirxml20/index.html)

Documentation Updates

For the most recent version of this document, see the DirXML Documentation Web site (http://www.novell.com/documentation/lg/dirxml20/index.html)

Documentation Conventions

In this documentation, a greater-than symbol (>) is used to separate actions within a step and items within a cross-reference path.

A trademark symbol (®, ™, etc.) denotes a Novell trademark. An asterisk (*) denotes a third-party trademark.

User Comments

We want to hear your comments and suggestions about this manual and the other documentation included with this product. To contact us, send e-mail to proddoc@novell.com.

1 Policies and Filters

This section contains an overview of policies and filters, and their function in a DirXML® environment. The following topics are covered:

- "What Are Policies and Filters?" on page 11
- "Introduction to Policies" on page 13

What Are Policies and Filters?

At a high level, policies enable you to customize the way DirXML sends and receives updates.

To understand policies, it helps to understand some level of detail regarding what a driver shim is written to do.

When a driver shim is written, an attempt is made to include the ability to synchronize anything a company deploying the driver might use. The developer writes the driver shim to detect any relevant changes in the connected system, then pass this change to Novell® eDirectory™.

This change is contained in an XML document, formatted according to the DirXML specification. The following snippet contains one of these XML documents:

```
<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>
```

Now, depending on what you are trying to accomplish, you might not care that a user named Smith with a telephone number of 111-1111 was added to a system. However, someone else might.

Point is, drivers are designed to report any relevant changes, then enable you to filter or modify the change however you see fit. The logic of what changes are important and how to process these changes is handled in the engine, not in the driver shim.

If one company wasn't very concerned with users, they could implement a filter to block all operations regarding users in either eDirectory or the connected system. If users were all they cared about, they could implement a filter to do the reverse.

Defining filters to prevent the synchronization of objects that aren't interesting to you is the first step in driver customization.

The next step defines what DirXML does with the objects that aren't blocked by your filter. As an example, let's refer to the add operation in the XML document above. A user named Smith with a telephone number of 111-1111 was added to your connected system. Assuming you don't filter this operation, DirXML needs to decide what to do with this user.

To make this decision, DirXML applies a set of policies, in a specific order (for now we are going to ignore transformation policies, which occur before the filter is applied on the publisher channel, and as the last step on the subscriber).

The first policy, matching, answers the question, "Is this object already in the data store?" To answer this, you need to define the characteristics that are unique to an object. A common attribute to check might be an e-mail address, since these are generally unique to ensure we all receive our fair share of spam. You could define a policy that says "If two objects have the same e-mail address, they are the same object."

If a match is found, DirXML notes this find in an attribute called an association. An association is a unique value that enables DirXML to associate objects in connected systems.

In circumstances where a match is not found, the second policy, creation, is called on. The create policy tells DirXML under what conditions you would like objects created. You can make the existence of certain attributes mandatory in the creation rule. If these attributes do not exist, DirXML blocks the creation of the object until the required information is provided.

After the object is created, the third rule, placement, tells DirXML where to put it. You could specify that objects should be created in a hierarchical structure identical to the system they came from, or you could place them somewhere completely different based on an attribute value.

If you would like to place users in a hierarchy according to a location attribute on the object, and name them according to their Full Name, you could make these attributes required in the create policy. This way you can ensure that the attribute exists so your placement strategy works correctly.

There are many other things you can do with policies. Using Policy Builder, you can easily generate unique values, add and remove attributes, generate events, send e-mail, and a laundry list of other operations. Even more advanced transformations are available by using XSLT to transform the XML document directly (remember that changes are sent to and from eDirectory in XML documents).

The basic thing to keep in mind is policies enable you to control how DirXML handles updates.

Continue to "Introduction to Policies" on page 13 to learn more about the different types of policies, then move on to Chapter 2, "Defining Policies Using Policy Builder," on page 19 to get your hands dirty in Policy Builder.

A Note on Transformation Policies

Transformation policies act as a translation mechanism between DirXML and the connected system. They transform schema between systems, and make preliminary changes to operations coming in, and final changes going out.

In a basic sense, transformation policies are used to make the other rules discussed previously (matching, create, placement), work correctly. The default configuration for each driver contains all of the necessary transformation policies, so you don't need to worry about these at first (the only exception might be the schema mapping policy, which you can easily modify using a GUI in iManager).

After you have a grasp of the basic policy types, understanding transformation policies might enable you to perform some customization that isn't possible with the basic policies.

Terminology Changes from DirXML 1.x

If you have not used DirXML 1.x, you do not need to review this section.

In DirXML 1.x, the term rule was used to describe a set of rules, the individual rules in this set, and the conditions and actions within the individual rules, depending on the context. This overlap causes confusion in circumstances when the context is not clear.

In DirXML 2, the term policy is now used to replace the previous usage of the term rule, when describing the high level transformation that is occurring. You now define a set of policies, which consists of one or more policies, where each policy contains one or more rules. The term rule is now used to describe only an individual set of conditions and actions.

The following table shows this terminology change:

Item being described	DirXML 2 Terminology	DirXML 1.x Terminology
Set of transformations	Set of Policies	Rule
An individual transformation within a set	Policy	Rule
The conditions and actions within an individual transformation	Rule	Rule

Introduction to Policies

This section provides an introduction to the types of policies available, their roles in DirXML, and how to define your own policies. The following topics are covered:

- "Basic Policies" on page 13
- "Transformation Policies" on page 15
- "Defining Policies" on page 15

Basic Policies

There are several different types of policies you can define on both the Subscriber and Publisher channels. Each policy is applied at a different step in the data transformation, and some policies are only applied when a certain action occurs. For example, a creation policy is applied only when a new object is created.

Policy	Description
Subscriber Matching	The object containing the criteria used to find objects in the application that match objects in eDirectory, so those matching objects can be associated with each other.
Subscriber Create	The object containing the definition of the attributes required to create a new object in the application.
Subscriber Placement	The object containing the criteria that determine where new application objects should be created.
Publisher Matching	The object containing the criteria used to find objects in eDirectory that match objects in the application so those matching objects can be associated with each other.
Publisher Create	The object containing the definition of the attributes required to create a new object in eDirectory.
Publisher Placement	The object containing the criteria that determine where new eDirectory objects should be created.
Schema Mapping	The object that holds the definition of the schema mappings between eDirectory and the application

Create

Create policies define the minimum set of attributes that must be present to create a new object.

For example, you create a new user in eDirectory, but you only give the new User object a name and ID. This creation is mirrored in the eDirectory tree, but the addition is not immediately reflected in applications connected to eDirectory because you have a Create policy specifying that only User objects with a more complete definition are allowed.

A Create policy can be the same for both the Subscriber and the Publisher, or it can be different.

The create policy is represented in eDirectory as an object in the driver.

Matching

Matching policies define the minimum criteria that two objects must meet to be considered the same.

Placement

Placement policies determine where new objects are created in eDirectory and the connected application.

Each driver requires at least two Placement policies: one to specify where to place a new eDirectory object when the external application database creates a new object, and one to specify where to create an external application database object when a new object is created in eDirectory.

Because eDirectory is hierarchical, multiple policies are useful because they let you create objects in multiple containers, However, you might prefer to have all new objects created in the same container, then later move them to department containers.

Schema Mapping

Schema Mapping policies hold the definition of the schema mappings between eDirectory and the connected system.

The eDirectory schema is read from eDirectory. The DirXML driver for the connected system supplies the application's schema. After the two schemas have been identified, a simple mapping is created between eDirectory and the target application.

After a schema mapping is defined in the DirXML driver configuration, the corresponding data can be mapped.

Transformation Policies

The following policies are used to transform the event data format between eDirectory and the application:

Policy	Description
Output Transformation	The transform action that should be used as information is passed from eDirectory to the application.
Input Transformation	The transform action that should be used as information is passed from the application to eDirectory.

The following policies are used to transform the event action between eDirectory and the application:

Policy	Description
Subscriber Event Transformation	The transform action used to convert from one event to another.
Publisher Event Transformation	The transform action used to convert from one event to another.

The following policies are used to transform commands between eDirectory and the application:

Policy	Description
Subscriber Command Transformation	The transform actions used on commands sent to eDirectory by the DirXML engine.
Publisher Command Transformation	The transform actions used on commands sent by the driver to the DirXML engine.

Defining Policies

Policies are defined in one of two ways:

- Using the Policy Builder interface to generate DirXML Script. Existing, non-XSLT rules are converted to DirXML Script automatically upon import.
- Using XSLT style sheets.

Policy Builder and DirXML Script

The Policy Builder interface is used to define the majority of policies you might implement. The Policy Builder interface uses a graphical environment to enable you to easily define and manage policies.

The underlying functionality of rule creation within Policy Builder is provided by a custom scripting language, called DirXML Script.

DirXML Script contains a wide variety of conditions you can test, actions to perform, and dynamic values to add to your policies. Each of these options are presented using intelligent drop-down lists, providing only valid selections at each point, and quick links to common values.

Policy Builder makes working directly with DirXML script unnecessary.

See Chapter 2, "Defining Policies Using Policy Builder," on page 19, for more information on Policy Builder.

TIP: Although not necessary to use Policy Builder, a complete DirXML script reference is available with the DirXML Driver Developer Kit at http://developer.novell.com/ndk/doc/dirxml/dirxmlbk/ref/index.html (http://developer.novell.com/ndk/doc/dirxml/dirxmlbk/ref/index.html)

XSLT Style Sheets

To define more complex policies, XSLT style sheets are used to directly transform one XML document into another XML document containing the required changes.

Style sheets provide you a large amount of flexibility, and are used when the transformation doesn't fit into the predefined conditions and actions available using rule creation in Policy Builder.

To create an XSLT style sheet, you need a through understanding of XSLT the nds.dtd, and the commands and events transferred to and from the DirXML engine. For detailed nds.dtd reference, see the NDS DTD reference (http://developer.novell.com/ndk/doc/dirxml/dirxmlbk/ref/ndsdtd/DTD-TREE.html), and nds.dtd (http://developer.novell.com/ndk/doc/dirxml/dirxmlbk/ref/nds.dtd).

See Chapter 3, "Defining Policies using XSLT Style Sheets," on page 137 for more information on XSLT style sheets.

Introduction to Filters

Filters specify the object classes and the attributes for which the DirXML engine processes events.

Separate event filters are specified for the subscriber and publisher channels. Event filters only pass events occurring on objects whose base class matches one of those classes specified by the filter. Event filters do not pass events occurring on objects that are a subordinate class of a class specified in the filter unless the subordinate class is also specified.

NOTE: In eDirectory, a base class is the object class that is used to create an entry. You must specify that class in the filter, rather than a super class from which the base class inherits.

For example, if the User class is specified in the event filter with the Surname and Given Name attributes, the DirXML engine passes on any changes to these attributes. However, if the entry's Telephone Number attribute is modified, the DirXML engine drops this event because the Telephone Number attribute is not in the event filter.

Filters must be configured to include the following:

- Attributes required by the rules
- Attributes that are to be synchronized

See Chapter 4, “Defining Filters,” on page 149 for information on defining filters.

2 Defining Policies Using Policy Builder

Policy Builder is a complete, graphical interface for creating and managing the policies that define the exchange of data between connected systems.

This section covers the following topics on using Policy Builder:

- "Policy Builder Tasks" on page 19

This section also contains the following detailed reference sections:

- "Conditions" on page 26
- "Actions" on page 50
- "Nouns" on page 104
- "Verbs" on page 129

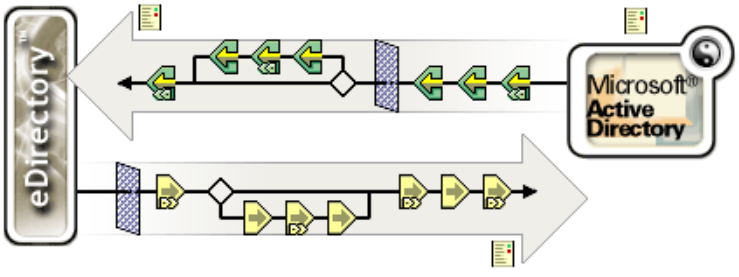
Policy Builder Tasks

This section contains instructions on performing common tasks in Policy Builder:

- "Opening Policy Builder" on page 19
- "Creating a Policy" on page 20
- "Modifying a Policy" on page 25
- "Defining Individual Rules within a Policy" on page 20
- "Defining Individual Arguments within a Rule" on page 22

Opening Policy Builder

1. In iManager, expand the DirXML® Management Role, then click Overview.
2. Specify a driver set.
3. Click the driver for which you want to manage policies. The DirXML Driver Overview opens:



4 Policies are managed from the DirXML Driver Overview.

Creating a Policy

1 Open the DirXML Driver Overview for the driver you want to manage.

2 Click the icon representing the policy you want to define.

represents an undefined policy.

represents a defined policy.

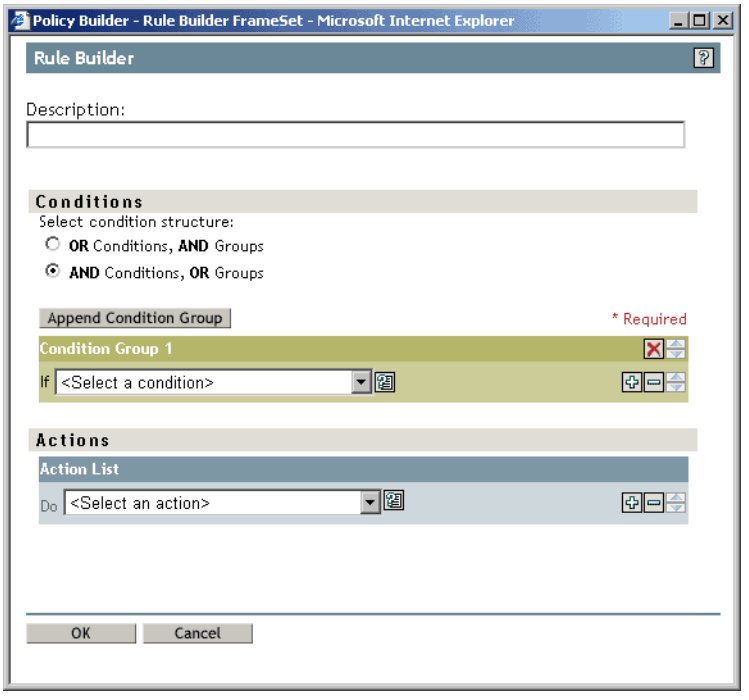
3 Click Insert.

4 Enter a name for the new policy, then select Policy Builder.

5 The policy is displayed. To define one or more rules for this policy, click Append New Rule, then follow the instructions in "Defining Individual Rules within a Policy" on page 20.

Defining Individual Rules within a Policy

Rules are defined in the Rule Builder window of Policy Builder:

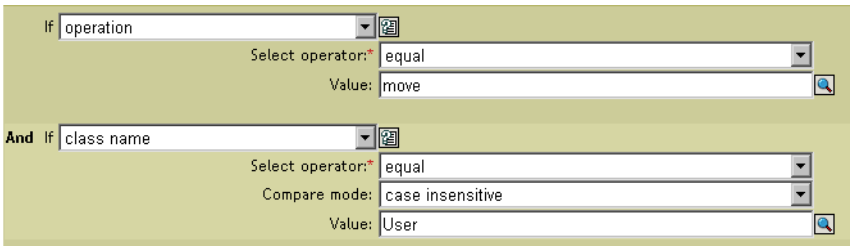


The Rule Builder interface enables you to quickly create and modify rules using intelligent drop-down menus.

In Rule Builder, you define a set of conditions that must be met before a defined action occurs.

For example, if you needed to create a rule that disallowed any new objects from being added to your environment, you might define this rule similar to the following: When an add operation occurs, veto the operation.

To implement this logic in Rule Builder, you could select the following condition:

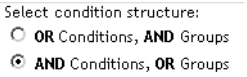


And the following action:

See "Conditions" on page 26 and "Actions" on page 50 or a detailed reference on the conditions and actions available in the Rule Builder.

Tips

To create more complex conditions, you can join conditions and groups of conditions together with and/or statements. You can modify the way these are joined by selecting the condition structure:

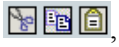


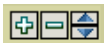
Click the icon to see a list of values for a field. In the example above, this icon opens a list of valid class names.

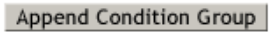
Click the icon to use the Argument Builder interface to construct an argument.

Click the icon to disable a policy, rule, condition, or action. Click the icon to re-enable it.

Click the icon to add a comment to a policy or rule. Comments are stored directly on the policy or rule, and can be as long as necessary.

Use the Cut/Copy/Paste icons,  , to use the Policy Builder clipboard. The Paste icon is disabled if the current content on the clipboard is invalid at that location.

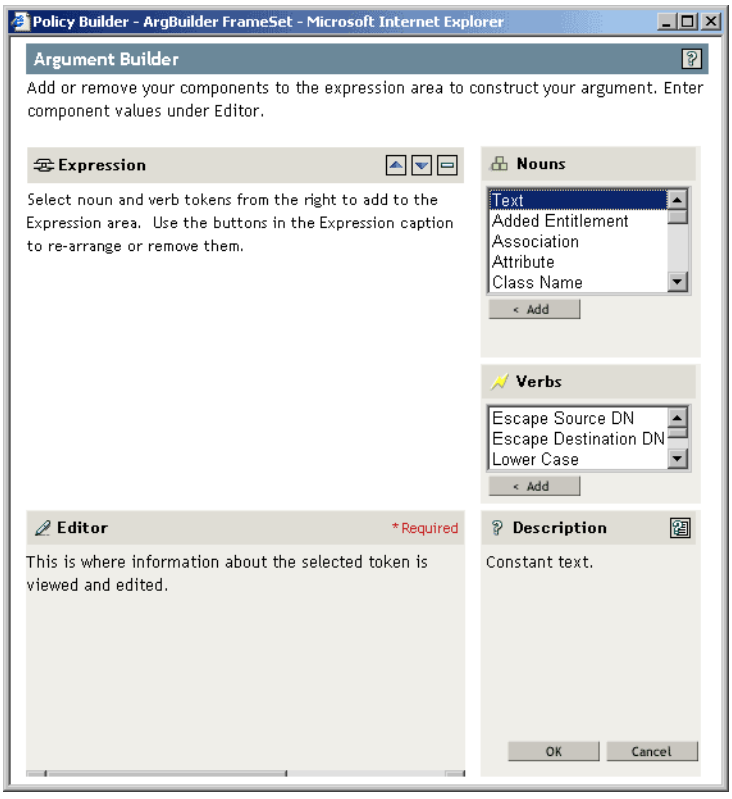
Use the  icons to add, remove, and position conditions.

Use the  button add condition groups.

Use the icons to remove, and position condition groups.

Defining Individual Arguments within a Rule

Argument Builder provides a dynamic, graphical interface which enables you to construct complex argument expressions for use within Rule Builder:

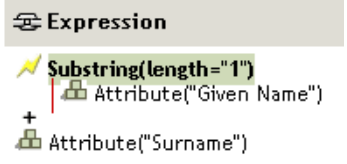


To define an expression, select one or more nouns (values, objects, variables, etc.), and combine them with verbs (substring, escape, upper and lower case) to construct arguments.

Multiple nouns, verbs, and expressions are combined to construct complex arguments.

For example, if you would like the argument set to an attribute value, you simply select the attribute noun, and enter or select the attribute name:

If you only want a portion of this attribute, you can combine the attribute noun with the substring verb:



See "Nouns" on page 104 and "Verbs" on page 129 for a detailed reference on the nouns and verbs available in the Argument Builder.

Tips

To create more complex conditions, you can join conditions or groups of conditions together with and/or statements.

Use the icons to move and delete nouns and verbs.

Click the icon to see a list of values for a field.

After you add a noun or verb, you can provide values in the editor then immediately add another noun or verb. You do not need to refresh the Expression pane to apply your changes, they appear when the next operation is performed.

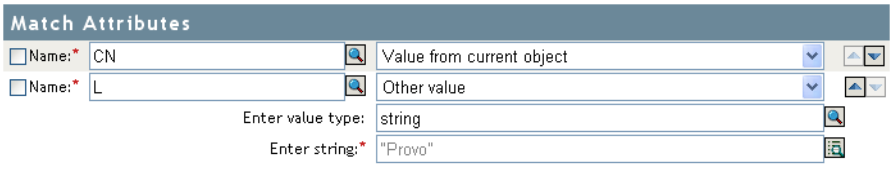
Although you define most arguments using this standard interface, there are a few other custom Argument Builder windows used to provide information in certain circumstances. Several of these windows launch the default Argument Builder to provide values.

The following sections contain an explanation of these additional Argument Builder interfaces and the conditions and actions which use them:

- “Matching Attribute Builder” on page 24
- “Argument Actions Builder” on page 24
- “Named String Builder” on page 25
- “Argument Value List Builder” on page 25

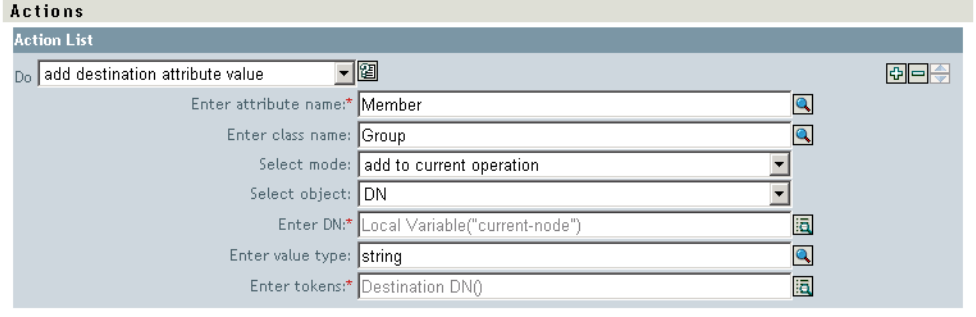
Matching Attribute Builder

The Matching Attribute Builder is used to construct conditions to satisfy a Find Matching Object (page 67) action.



Argument Actions Builder

The Argument Actions Builder is used to construct a list of actions to take in actions such as For Each (page 69).



Named String Builder

The Named String Builder is used to create name/value pairs for use in Actions such as Generate Event (page 70) and Send Email (page 82).

Argument Value List Builder

The Argument Value List Builder is used to create arguments for actions such as Set Default Attribute Value (page 86). In this example, a string argument with the value unknown has been created to set the default location.

Modifying a Policy

1. Open the DirXML Driver Overview for the driver you want to manage.
2. Click the icon representing the policy you want to modify.
3. Select the policy you want to modify, then click Edit.

Deleting a Policy

1. Open the DirXML Driver Overview for the driver you want to manage.
2. Click the icon representing the policy you want to delete.
3. Select the policy you want to delete, then click Remove.

Importing a Policy from an XML File

1. Open the DirXML Driver Overview for the driver you want to manage.
2. Click the icon representing the policy you want to delete.
3. Edit an existing policy, or create a new policy.
4. Click the Insert button, and select Import an XML file containing DirXML Script.
5. Browse to the policy file to import, then click OK.

Exporting a Policy to an XML File

1. Open the DirXML Driver Overview for the driver you want to manage.
2. Click the icon representing the policy you want to delete.
3. Edit an existing policy, or create a new policy.
4. Click the Save As button, then select a location to save the DirXML Script XML file.

Creating a Policy Reference

A policy reference enables you to create a single policy, and reference it in multiple locations. If you have a policy that is used by more than one driver or policy, creating a reference simplifies management of this policy.

1. Open the DirXML Driver Overview for the driver you want to manage.
2. Click the icon representing the policy you want to delete.
3. Edit an existing policy, or create a new policy.
4. Click the insert button, and select Append a reference to a policy containing DirXML Script.
5. Browse to the policy object to reference, then click OK.

Conditions

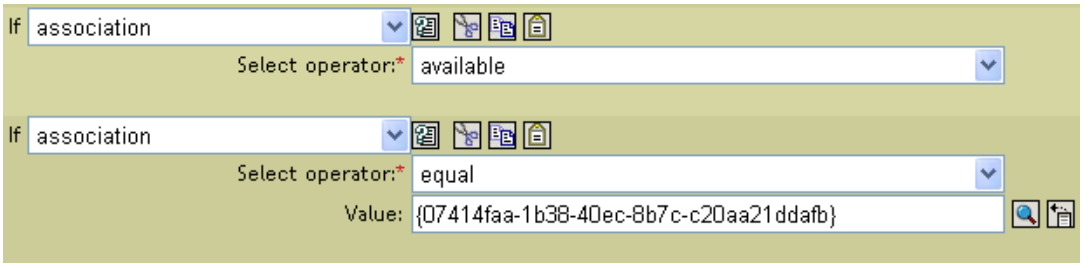
This section contains detailed reference to all conditions available using the Policy Builder interface.

If Association (page 28)
If Attribute (page 29)
If Class Name (page 31)
If Destination Attribute (page 32)
If Destination DN (page 34)
If Entitlement (page 36)
If Global Configuration Value (page 38)
If Local Variable (page 39)
If Named Password (page 40)
If Operation Attribute (page 41)
If Operation (page 43)
If Operation Property (page 44)
If Password (page 45)
If Source Attribute (page 46)
If Source DN (page 48)
If Xpath Expression (page 50)

If Association

If Association performs a test on the association value of current operation or the current object.

Example



Condition

Operator	Condition is met when...
associated	There is an established association for the current object.
available	There is a non-empty association value specified by the current operation.
equal	The association value specified by the current operation is exactly equal to the content of if association.
not-associated	Associated would return False.
not available	Available would return False.
not-equal	Equal would return False.

Fields

Operator

Select the condition test type.

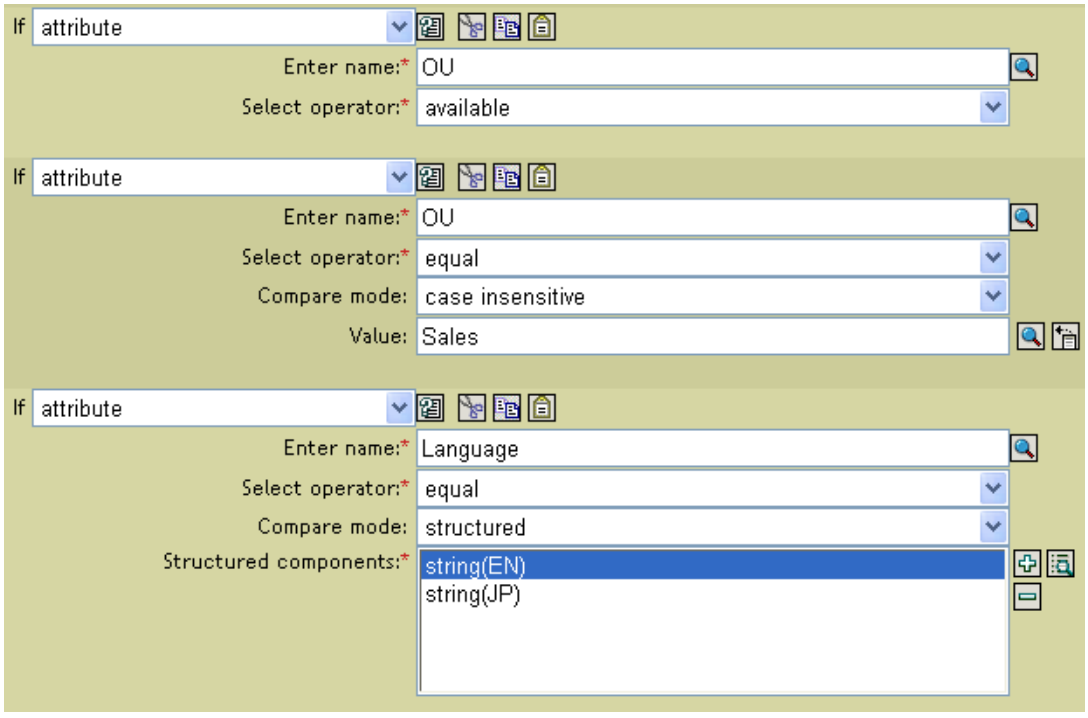
Compare Mode

Select the comparison mode. See “Comparison Modes” on page 138.

If Attribute

If Attribute performs a test on attribute values of the current object in either the current operation or the source data store.

Example



Condition

Operator	Condition is met when...
available	There is a value available in either the current operation or the source data store for the specified attribute.
equal	There is a value available in either the current operation or the source data store for the specified attribute, that equals the specified value when compared using the specified comparison mode.
not available	Available would return False.
not-equal	Equal would return False.

Fields

Name

Specify the name of the attribute to test for the selected condition.

Operator

Select the condition test type.

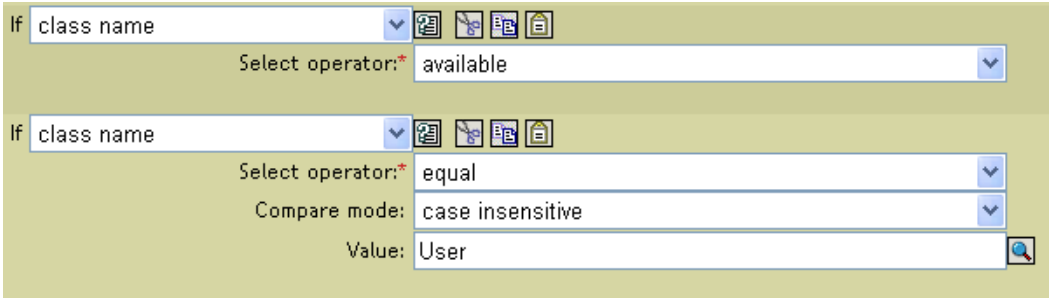
Compare Mode

Select the comparison mode. See “Comparison Modes” on page 138.

If Class Name

If Class Name performs a test on the object class name in the current operation.

Example



Condition

Operator	Condition is met when...
available	there is an object class name available in the current operation.
equal	there is an object class name available in the current operation, and it equals the specified value when compared using the specified comparison mode.
not available	Available would return False.
not-equal	Equal would return False.

Fields

Operator

Select the condition test type.

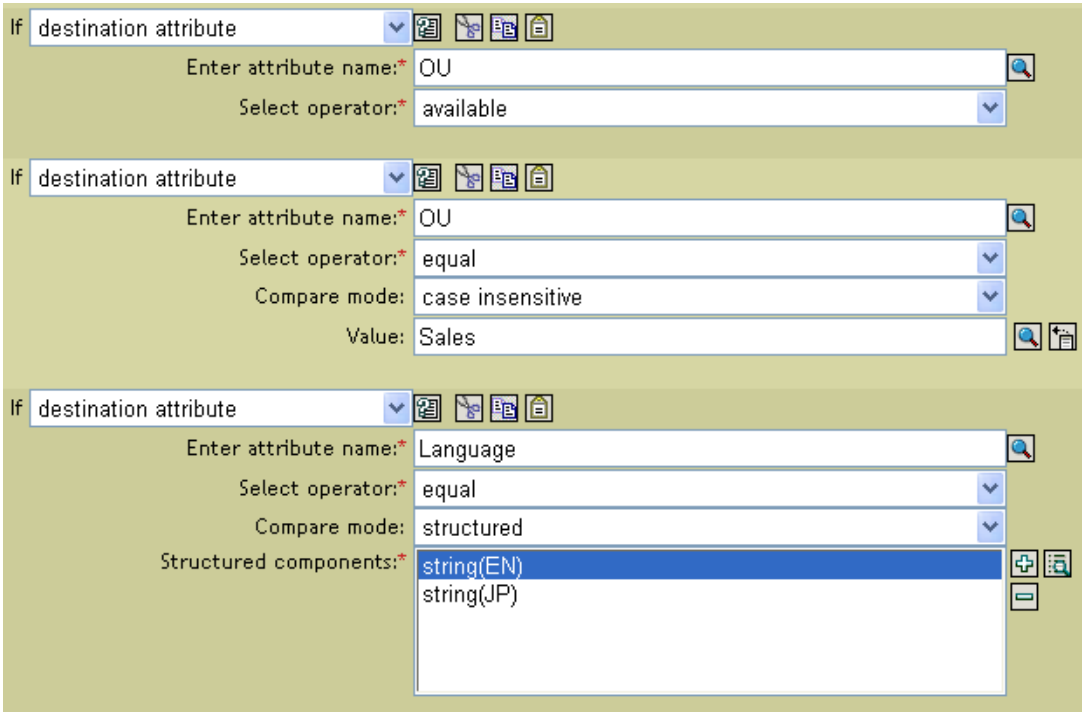
Compare Mode

Select the comparison mode. See "Comparison Modes" on page 138.

If Destination Attribute

If Destination Attribute performs a test on attribute values of the current object in the destination data store.

Example



Condition

Operator	Condition is met when...
available	There is a value available in the destination data store for the specified attribute.
equal	There is a value available for the specified attribute in the destination data store that equals the specified value when compared using the specified comparison mode.
not available	Available would return False.
not-equal	Equal would return False.

Fields

Name

Specify the name of the attribute to test for the selected condition.

Operator

Select the condition test type.

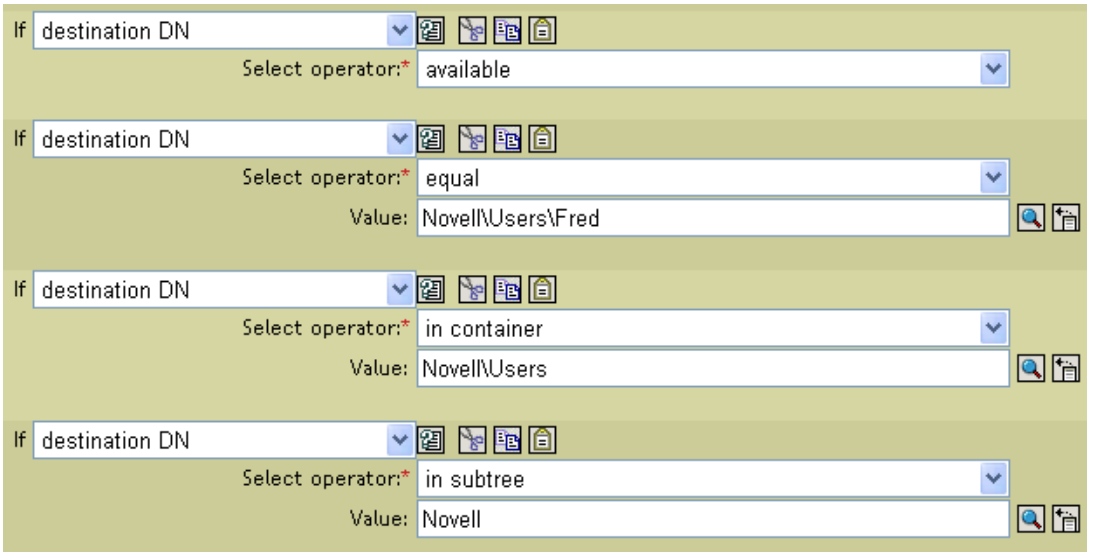
Compare Mode

Select the comparison mode. See “Comparison Modes” on page 138.

If Destination DN

If Destination DN performs a test on the destination DN in the current operation.

Example



Condition

Operator	Condition is met when...
available	There is a destination DN available.
equal	There is a destination DN available, and it equals the specified value when compared using semantics appropriate to the DN format of the destination data store.
in-container	There is a destination DN available, and it represents an object in the container, specified by value, when compared using semantics appropriate to the DN format of the destination data store.
in-subtree	There is a destination DN available, and it represents an object in the subtree, specified by value, when compared using semantics appropriate to the DN format of the destination data store.
not available	Available would return False.
not-equal	Equal would return False.
not-in-container	In-container would return False.
not-in-subtree	In-subtree would return False.

Fields

Operator

Select the condition test type.

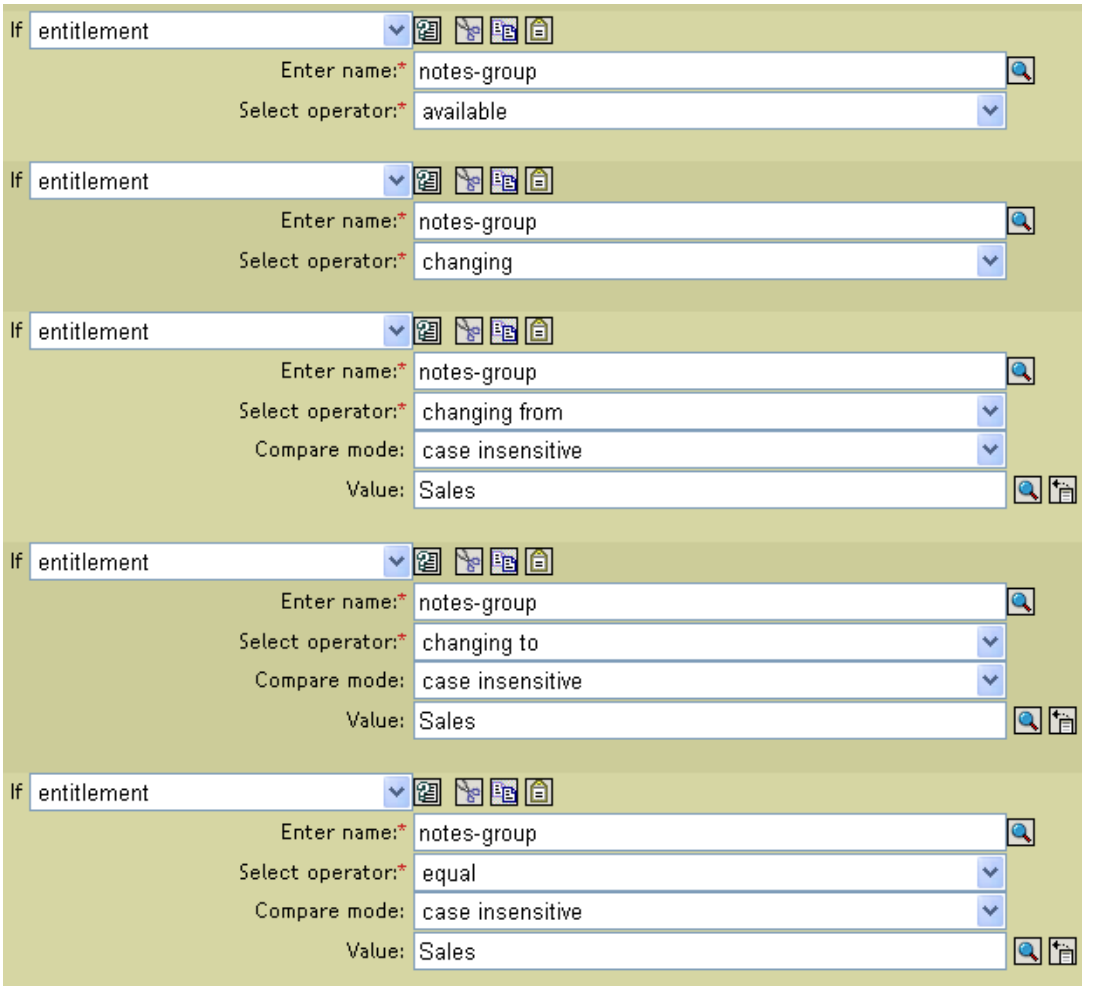
Compare Mode

Select the comparison mode. See “Comparison Modes” on page 138.

If Entitlement

If Entitlement performs a test on entitlements of the current object, in either the current operation or eDirectory.

Example



Condition

Operator	Condition is met when...
available	The named entitlement is available in either the current operation or the eDirectory™ data store.
changing	The current operation contains a change (modify attribute or add attribute) of the named entitlement.
changing-from	The current operation contains a change that removes a value (remove value) of the named entitlement, that has a value which equals the specified value, when compared using the specified comparison mode.

Operator	Condition is met when...
changing-to	The current operation contains a change that adds a value (add value or add attribute) to the named entitlement, that has a value which equals the specified value, when compared using the specified comparison mode.
equal	There is a value available for the specified attribute in the destination data store that equals the specified value when compared using the specified comparison mode.
not available	Available would return False.
not-changing	Changing would return False.
not-changing-from	Changing-from would return False.
not-changing-to	Changing-to would return False.
not-equal	Equal would return False.

Fields

Name

Specify the name of the entitlement to test for the selected condition.

Operator

Select the condition test type.

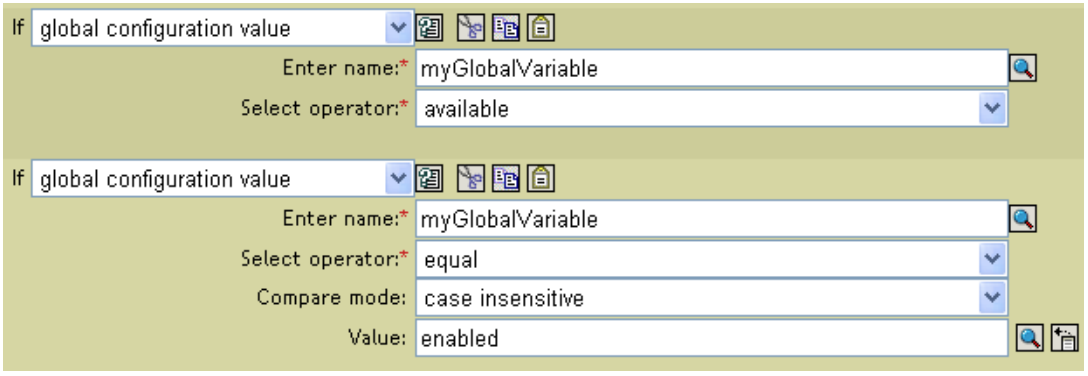
Compare Mode

Select the comparison mode. See “Comparison Modes” on page 138.

If Global Configuration Value

If Global Configuration Value performs a test on a global configuration variable.

Example



Condition

Operator	Condition is met when...
available	There is a global configuration variable with the specified name.
equal	There is a global configuration variable with the specified name and its value equals the specified value when compared using the specified comparison mode.
not available	Available would return False.
not-equal	Equal would return False.

Fields

Name

Specify the name of the global variable to test for the selected condition.

Operator

Select the condition test type.

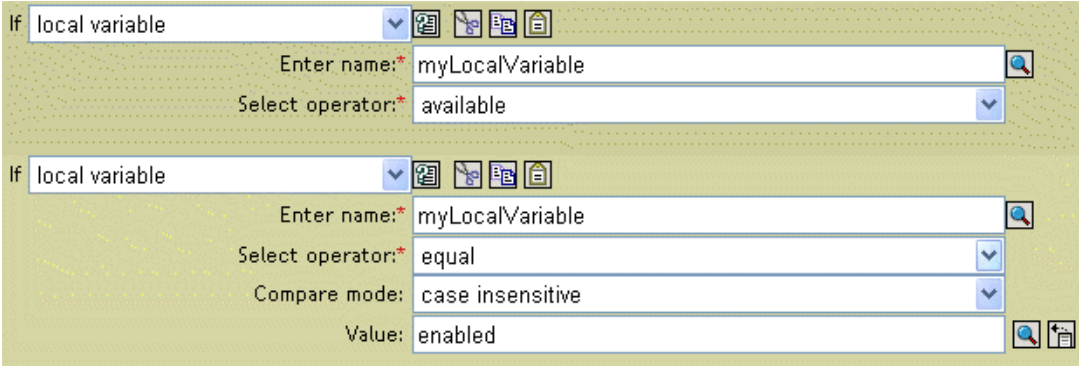
Compare Mode

Select the comparison mode. See “Comparison Modes” on page 138.

If Local Variable

If Local Variable performs a test on a local variable.

Example



Condition

Operator	Condition is met when...
available	There is a local variable with the specified name that has been defined by an action of a earlier rule within the policy.
equal	There is a local variable with the specified name, and its value equals the specified value when compared using the specified comparison mode.
not available	Available would return False.
not-equal	Equal would return False.

Fields

Name

Specify the name of the local variable to test for the selected condition.

Operator

Select the condition test type.

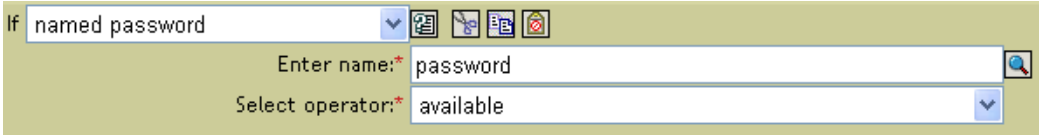
Compare Mode

Select the comparison mode. See “Comparison Modes” on page 138.

If Named Password

If Named Password performs a test on a password in the current operation with the specified name. The type of test performed depends on the selected operator. The following table shows the type of test performed by each operator.

Example



Condition

Operator	Condition is met when...
available	There is password with the specified name available.
not available	Available would return False.

Fields

Name

Specify the name of the named password to test for the selected condition.

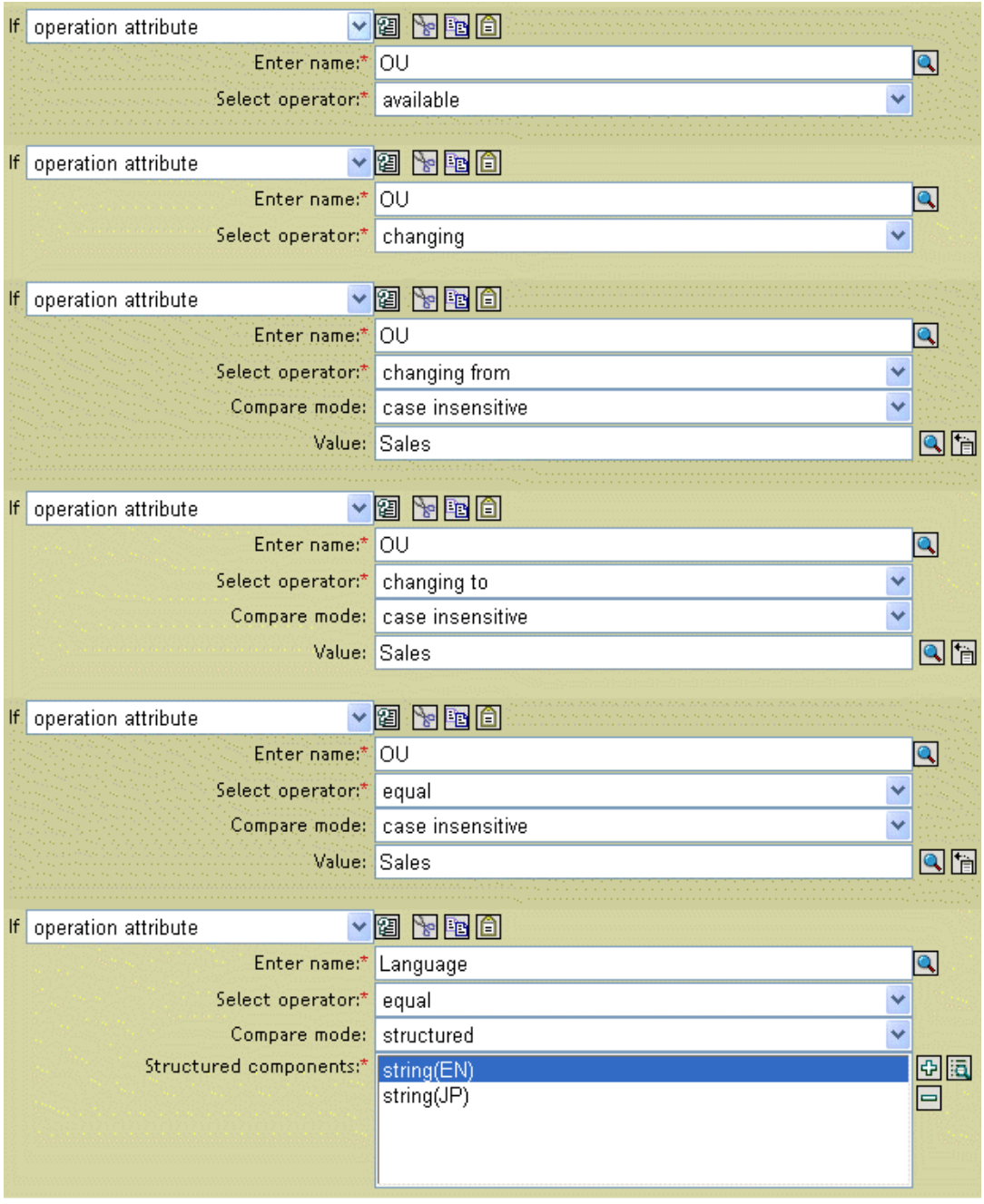
Operator

Select the condition test type.

If Operation Attribute

If Operation Attribute performs a test on attribute values in the current operation.

Example



Condition

Operator	Condition is met when...
available	There is a value available in the current operation (add attribute, add value, attribute) for the specified attribute.
changing	The current operation contains a change (modify attribute or add attribute) of the specified attribute.
changing-from	The current operation contains a change that removes a value (remove value) of the specified attribute, that equals the specified value when compared using the specified comparison mode.
changing-to	The current operation contains a change that adds a value (add value or add attribute) to the specified attribute, that equals the specified value when compared using the specified comparison mode.
equal	There is a value available in the current operation (other than a remove value) for the specified attribute, that equals the specified value when compared using the specified comparison mode.
not available	Available would return False.
not-changing	Changing would return False.
not-changing-from	Changing-from would return False.
not-changing-to	Changing-to would return False.
not-equal	Equal would return False.

Fields

Name

Specify the name of the attribute to test for the selected condition.

Operator

Select the condition test type.

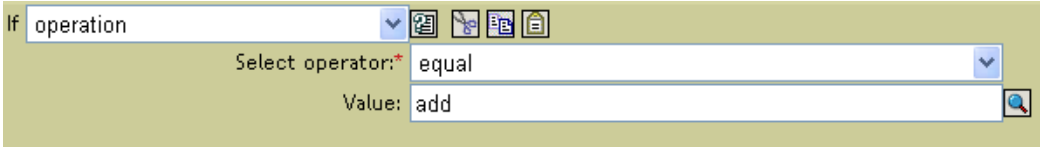
Compare Mode

Select the comparison mode. See “Comparison Modes” on page 138.

If Operation

If Operation performs a test on the name of the current operation.

Example



Condition

Operator	Condition is met when...
equal	The name of the current operation is exactly equal to content of If Operation.
not-equal	Equal would return False.

Fields

Operator

Select the condition test type.

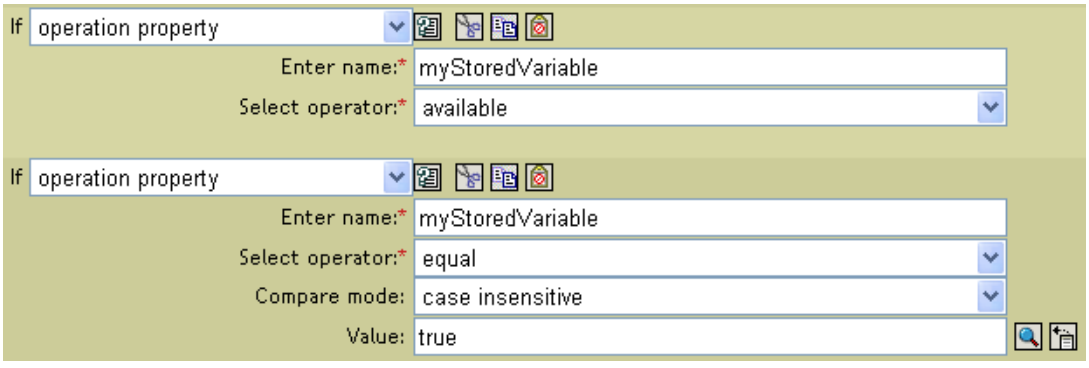
Compare Mode

Select the comparison mode. See “Comparison Modes” on page 138.

If Operation Property

If Operation Property performs a test on an operation property on the current operation. The type of test performed depends on the selected operator. The following table shows the type of test performed by each operator.

Example



Condition

Operator	Condition is met when...
available	There is an operation property with the specified name on the current operation.
equal	There is a an operation property with the specified name on the current operation and its value equals the provided content when compared using the specified comparison mode.
not available	Available would return False.
not-equal	Equal would return False.

Fields

Name

Specify the name of the operation property to test for the selected condition.

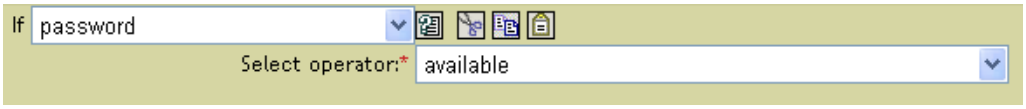
Operator

Select the condition test type.

If Password

If Password performs a test on a password in the current operation.

Example



Condition

Operator	Condition is met when...
available	There is password available in the current operation.
not available	Available would return False.

Fields

Operator

Select the condition test type.

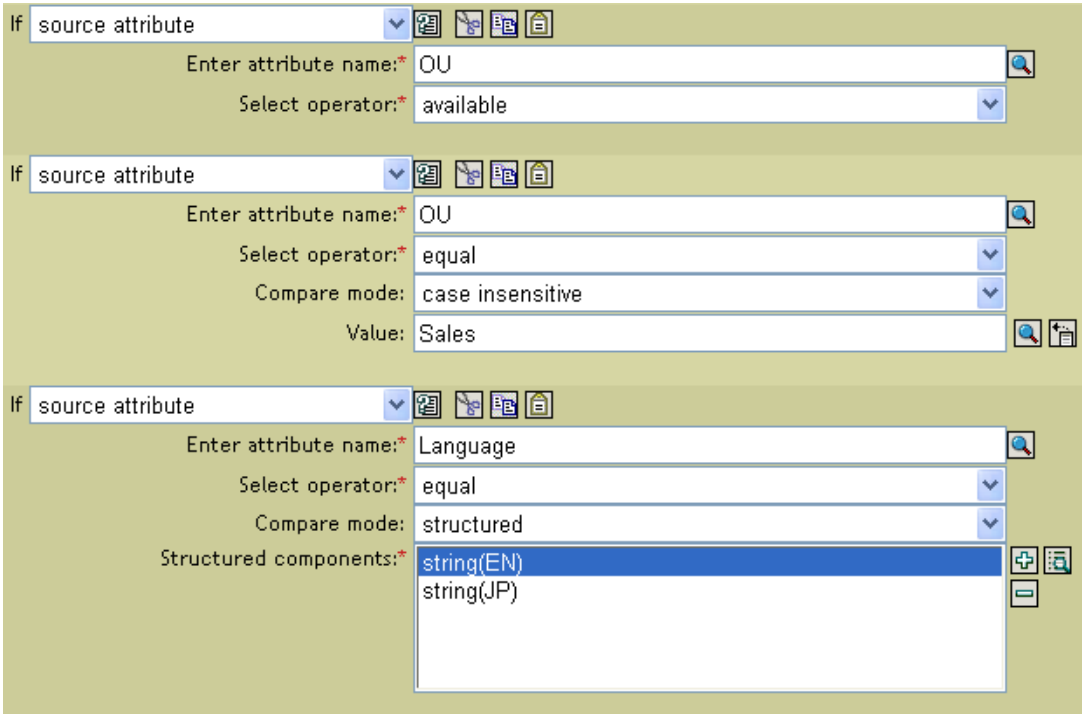
Compare Mode

Select the comparison mode. See “Comparison Modes” on page 138.

If Source Attribute

If Source Attribute performs a test on attribute values of the current object in the source data store.

Example



Condition

Operator	Condition is met when...
available	There is a value available in the source data store for the specified attribute.
equal	There is a value available in the source data store for the specified attribute, that equals the specified value when compared using the specified comparison mode.
not available	Available would return False.
not-equal	Equal would return False.

Fields

Name

Specify the name of the source attribute to test for the selected condition.

Operator

Select the condition test type.

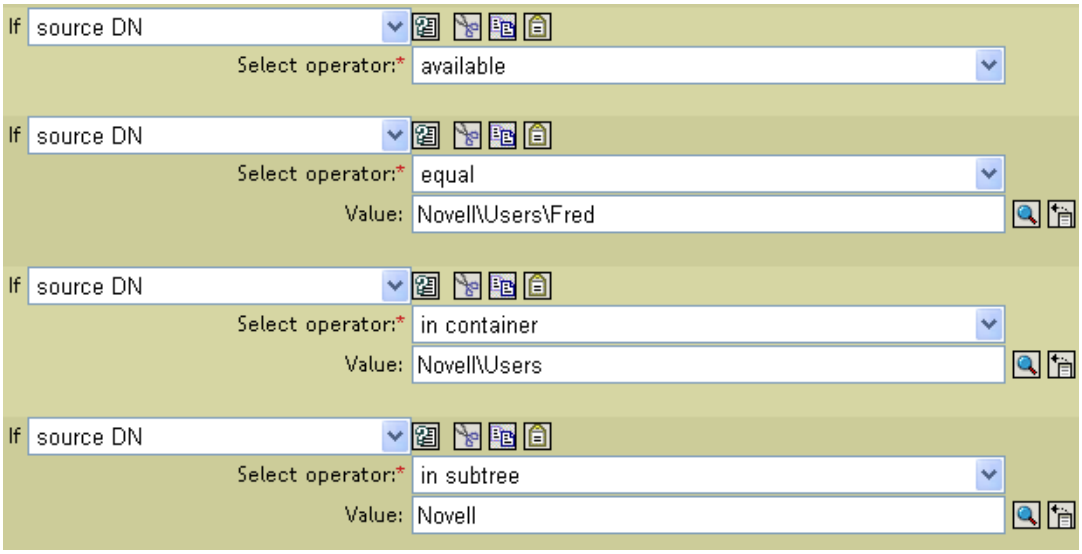
Compare Mode

Select the comparison mode. See “Comparison Modes” on page 138.

If Source DN

If Source DN performs a test on the source DN in the current operation.

Example



Condition

Operator	Condition is met when...
available	There is a source DN available.
equal	There is a source DN available, and it equals the content of the specified value when compared using semantics appropriate to the DN format of the source data store.
in-container	There is a source DN available, and it represents an object in the container specified by value, when compared using semantics appropriate to the DN format of the source data store.
in-subtree	There is a source DN available, and it represents an object in the subtree specified by value, when compared using semantics appropriate to the DN format of the source data store.
not available	Available would return False.
not-equal	Equal would return False.
not-in-container	In-container would return False.
not-in-subtree	In-subtree would return False.

Fields

Operator

Select the condition test type.

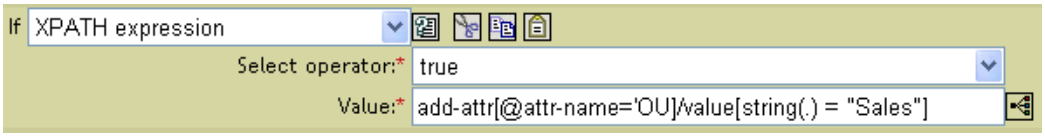
Compare Mode

Select the comparison mode. See “Comparison Modes” on page 138.

If Xpath Expression

If Xpath Expression performs a test on the results of evaluating an XPATH 1.0 expression.

Example



Condition

Operator	Condition is met when...
true	The XPATH expression evaluates to True.
false	True would return False.

Fields

Operator

Select the condition test type.

Actions

This section contains detailed reference to all actions available using the Policy Builder interface.

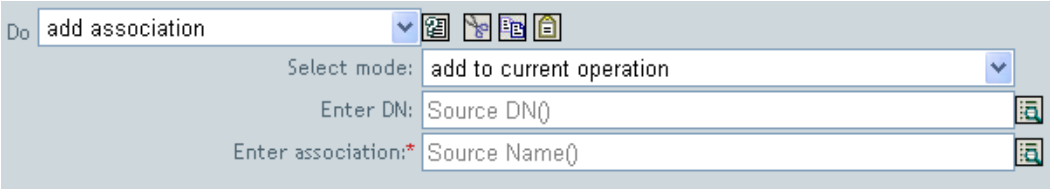
Add Association (page 52)
Add Destination Attribute Value (page 53)
Add Destination Object (page 54)
Add Source Attribute Value (page 55)
Add Source Object (page 56)
Append XML Element (page 57)
Append XML Text (page 58)
Break (page 59)
Clear Destination Attribute Value (page 60)
Clear Operation Property (page 61)
Clear Source Attribute Value (page 62)
Clone Operation Attribute (page 63)
Clone by Xpath Expressions (page 64)
Delete Destination Object (page 65)
Delete Source Object (page 66)
Find Matching Object (page 67)
Find Matching Object (page 67)
For Each (page 69)

Generate Event (page 70)
Move Destination Object (page 73)
Move Source Object (page 74)
Reformat Operation Attribute (page 75)
Remove Association (page 76)
Remove Destination Attribute Value (page 77)
Rename Destination Object (page 79)
Rename Operation Attribute (page 80)
Rename Source Object (page 81)
Send Email (page 82)
Send Email From Template (page 84)
Set Default Attribute Value (page 86)
Set Destination Password (page 88)
Set Local Variable (page 89)
Set Operation Association (page 90)
Set Operation Class Name (page 91)
Set Operation Destination DN (page 92)
Set Operation Property (page 93)
Set Operation Source DN (page 94)
Set Operation Template DN (page 95)
Set Source Attribute Value (page 96)
Set Source Password (page 97)
Set XML Attribute (page 98)
Status (page 99)
Strip Operation Attribute (page 100)
Strip Xpath (page 101)
Trace Message (page 102)
Veto (page 103)
Veto If Operation Attribute Not Available (page 104)

Add Association

This action causes an add association command to be sent to eDirectory.

Example



Fields

Mode

Select whether this actions should be added to the current operation, or written directly to the destination data store.

DN

Provide the DN of the object to receive the association using the Argument Builder.

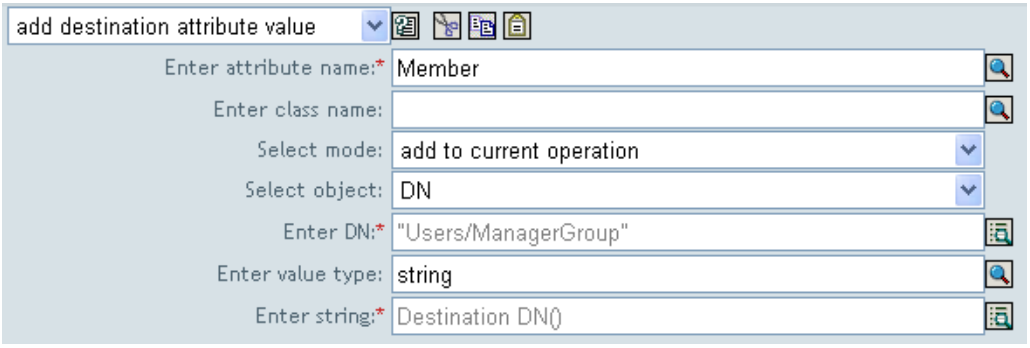
Association

Provide the value of the association using the Argument Builder.

Add Destination Attribute Value

This action causes the specified value to be added to the named attribute on an object in the destination data store. The target object is the current object, a DN, or an association.

Example



Fields

Attribute Name

Specify the name of the attribute to add to the target object in the destination data store.

Class Name

(Optional) Specify the class name of the target object in the destination data store. This value might be required if object is other than current object, for schema mapping purposes.

Select Mode

Select whether this action should be added to, before, or after the current operation, or written directly to the destination data store.

Select Object

Select the object in the destination data store to receive the attribute. This object can be the current object, or specified by a DN or an association.

Value Type

Select the syntax of the new attribute value.

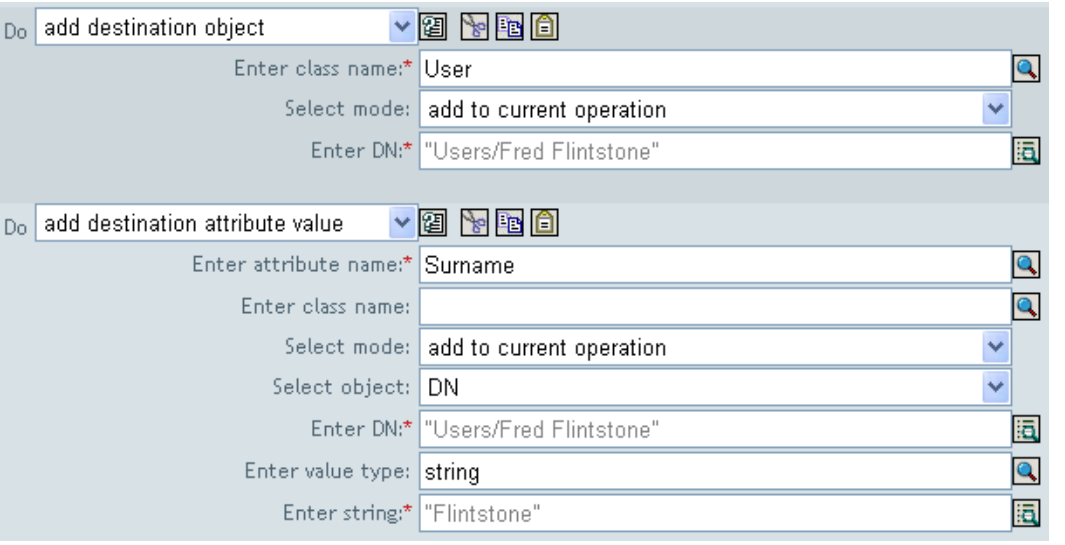
Tokens

Provide the value of the new attribute using the Argument Builder.

Add Destination Object

This action causes an object of the specified type to be created in the destination data store, with the name and location specified in the Enter DN field. Any attribute values to be added as part of the object creation must be done in subsequent Add Destination Attribute Value (page 53) actions using the same DN.

Example



Fields

Class Name

Specify the class name of the object to add to the destination data store.

Mode

Select whether this action should be added to, before, or after the current operation, or written directly to the destination data store.

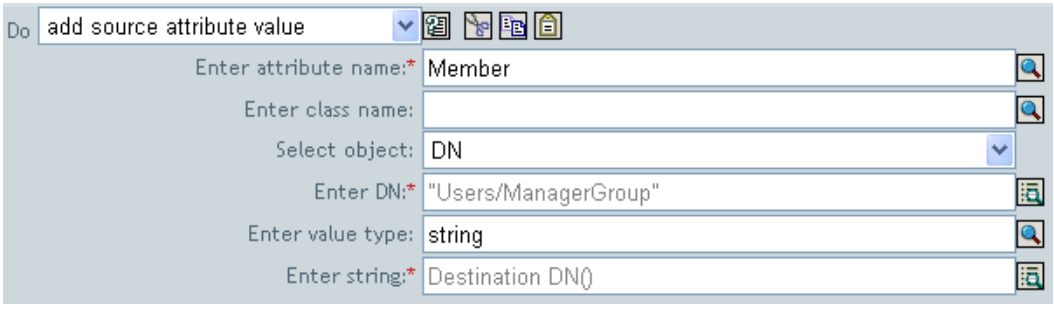
DN

Specify the DN of the new object to add to the destination data store.

Add Source Attribute Value

This action causes the specified value to be added to the specified attribute on an object in the source data store. The target object is the current object, a DN, or an association.

Example



Fields

Attribute Name

Specify the name of the attribute to add to the target object in the source data store.

Class Name

(Optional) Specify the class name of the target object in the source data store. This value might be required if object is other than current object, for schema mapping purposes.

Object

Select the target object in the source data store to receive the attribute. This object can be the current object, or specified by a DN or an association.

Value Type

Select the syntax of the new attribute value.

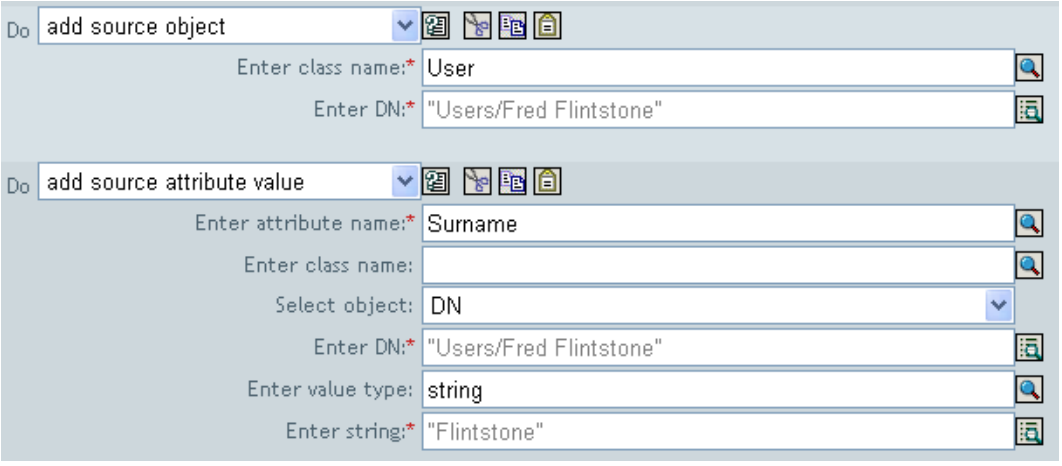
Tokens

Provide the value of the new attribute using the Argument Builder.

Add Source Object

This action causes an object of the specified type to be created in the source data store. Any attribute values to be added as part of the object creation must be done in subsequent Add Source Attribute Value (page 55) actions using the same DN.

Example



Fields

Class Name

Specify the class name of the object to add to the source data store.

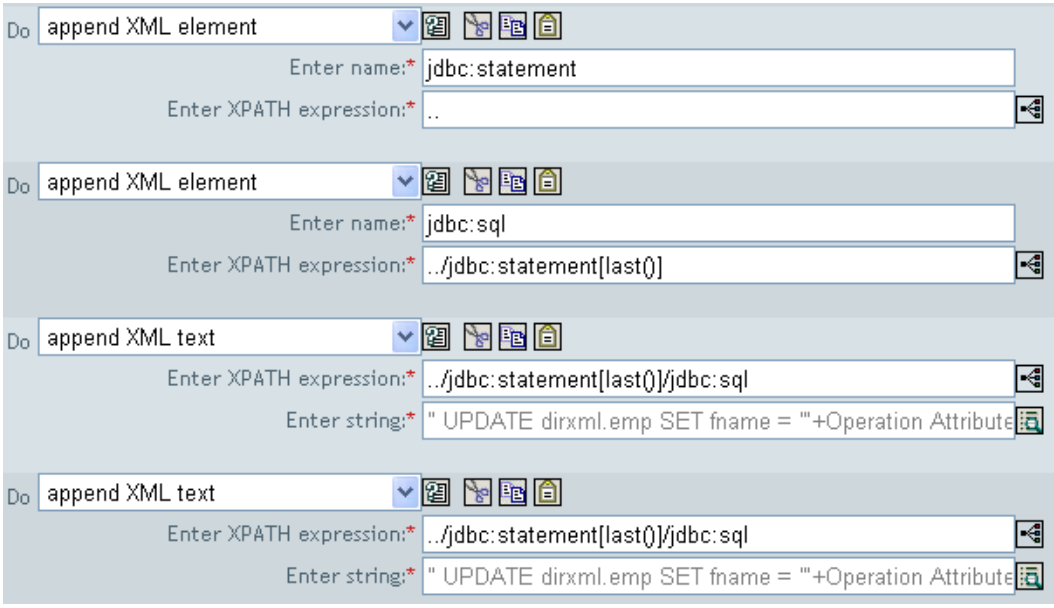
DN

Specify the DN of the new object to add to the source data store.

Append XML Element

This action causes a custom element to be appended to the set of elements selected by the XPATH expression.

Example



Fields

Name

Tag name of the XML element. This name can contain a namespace prefix if the prefix has been previously defined on this policy.

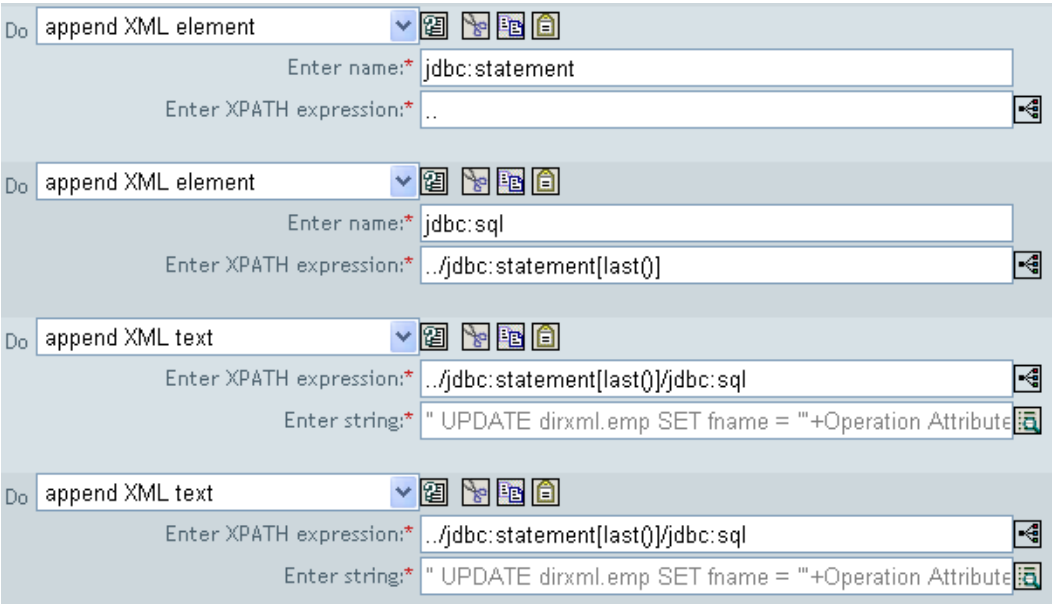
XPATH Expression

XPATH 1.0 expression that returns a nodeset containing the element(s) to which the new element(s) should be appended.

Append XML Text

This action causes the specified text to be appended to the set of elements selected by the XPATH expression.

Example



Fields

XPATH Expression

XPATH 1.0 expression that returns a nodeset containing the element(s) to which the new element(s) should be appended.

String

Text to be appended to the set of element(s) selected by expression.

Break

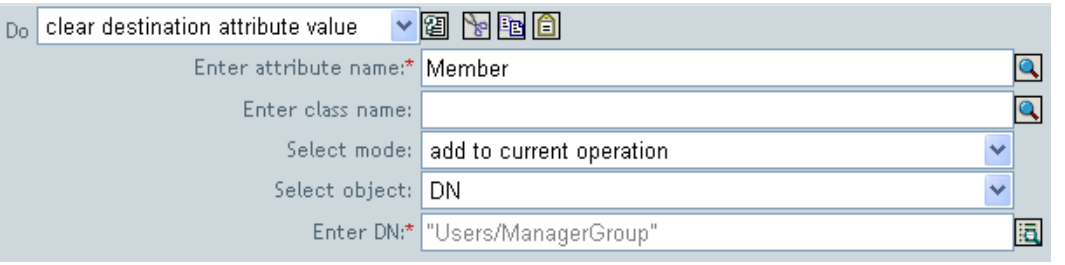
This action causes the current operation to not be processed by any more actions or rules within the current policy.

Example

Clear Destination Attribute Value

This action causes the all values for the named attribute to be removed from an object in the destination data store. The target object is the current object, a DN, or an association.

Example



Fields

Attribute Name

Specify the name of the attribute to add to the target object in the destination data store.

Class Name

(Optional) Specify the class name of the target object in the destination data store. This value might be required if object is other than current object, for schema mapping purposes.

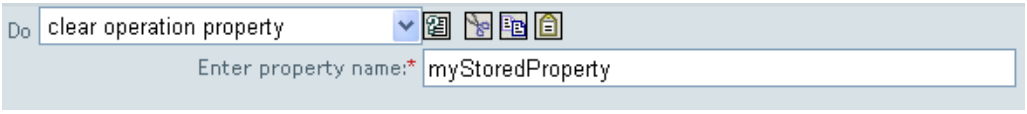
Mode

Select whether this action should be added to, before, or after the current operation, or written directly to the destination data store.

Clear Operation Property

This action causes any operation property with the provided name to be cleared from the current operation.

Example



Fields

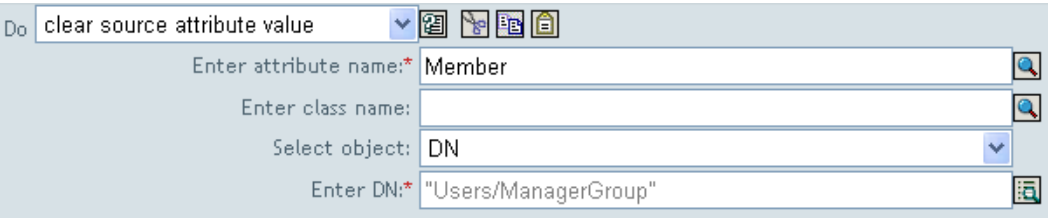
Property Name

Specify the name of the operation property to clear.

Clear Source Attribute Value

This action causes the all values for the named attribute to be removed from an object in the source data store. The target object is the current object, a DN, or an association.

Example



Fields

Attribute Name

Specify the name of the attribute to add to the target object in the source data store.

Class Name

(Optional) Specify the class name of the target object in the source data store. This value might be required if object is other than current object, for schema mapping purposes.

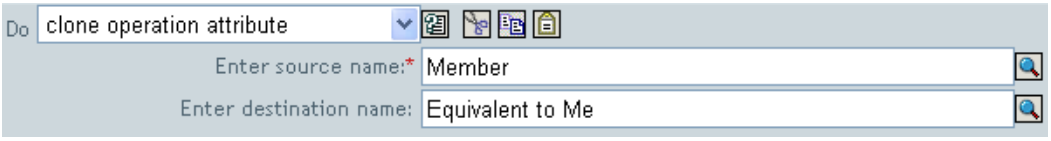
Select Object

Select the object in the source data store to receive the attribute. This object can be the current object, or specified by a DN or an association.

Clone Operation Attribute

This action causes all elements that are children of the current operation with an attribute name equal to the specified source name, to be duplicated within the operation, with the attribute name set to the specified destination name.

Example



Fields

Source Name

Specify the attribute name to clone.

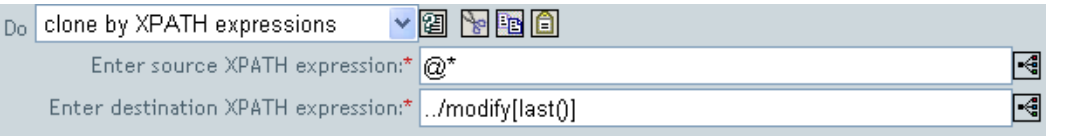
Destination Name

Specify the attribute name to give to the clone.

Clone by Xpath Expressions

This action causes causes deep copies of the nodes specified by the source field to be appended to the set of elements specified by the destination field.

Example



Fields

Source XPATH Expression

XPATH 1.0 expression that returns a nodeset containing the element(s) to which the new element(s) should be appended.

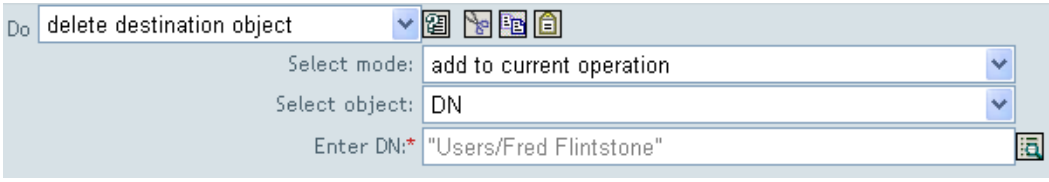
Destination XPATH Expression

XPATH 1.0 expression that returns a nodeset containing the node(s) that are to be cloned.

Delete Destination Object

This action causes an object in the destination data store to be deleted. The target object is the current object, a DN, or an association.

Example



Fields

Mode

Select whether this action should be added to, before, or after the current operation, or written directly to the destination data store.

Object

Select the target object in the destination data store to delete. This object can be the current object, or specified by a DN or an association.

Delete Source Object

This action causes the object in the source data store to be deleted. The target object is either the current object, a DN, or an association.

Example

Fields

Mode

Select whether this action should be added to, before, or after the current operation, or written directly to the destination data store.

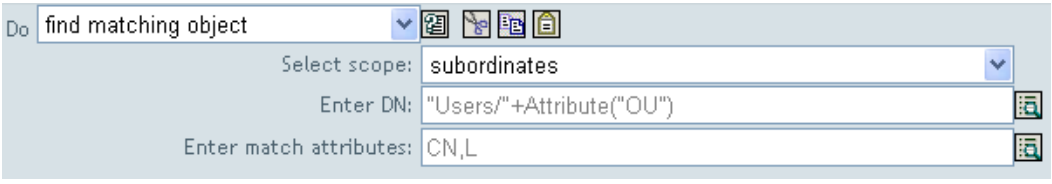
Object

Select the target object in the source data store to delete. This object can be the current object, or specified by a DN or an association.

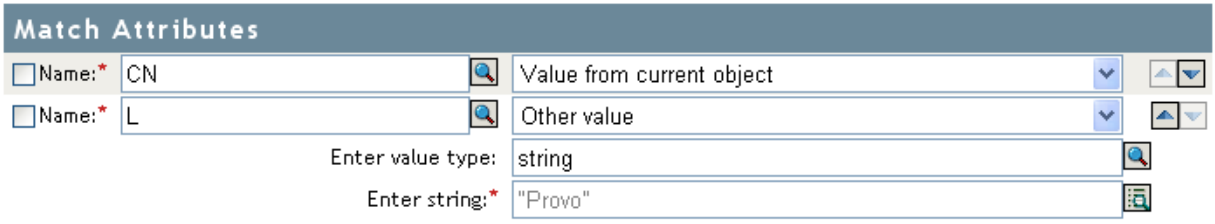
Find Matching Object

This action causes a query to be performed in the destination data store, and an appropriate destination DN, or an appropriate destination association, to be added to the current operation.

Example



The following is an example of the Argument Builder used to provide the match attributes:



Fields

Scope

Scope of the operation. Select entry, subordinates, or subtree.

DN

DN of the location to search using the selected scope.

Match Attributes

Provide the attributes which must match to consider the search successful.

Remarks

A DN argument is required when scope="entry", and is optional otherwise. At least one match attribute is required when scope= "subtree" or scope="subordinates".

Note that since it is undefined what a query does with the search attribute when scope="entry", it is also undefined what Find Matching Object will do.

The query generated has a scope attribute based on the selected scope, a destination DN attribute set to the content of the Enter DN field, if specified. It also has a class name attribute and search class based on the class name of the current object.

If the destination data store is the application, then an association will be added to the current operation for each successful match that is returned. No query will be performed if the current

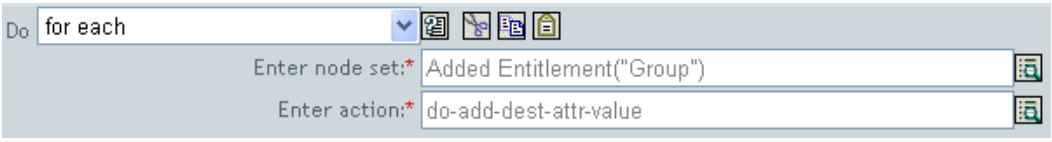
operation already has a non-empty association, thus allowing multiple find matching object actions to be strung together in the same rule.

If the destination data store is eDirectory, then the destination DN attribute for the current operation is set. No query is performed if the current operation already has a non-empty destination DN attribute, thus allowing multiple find matching object actions to be strung together in the same rule. If only a single result is returned and it is not already associated, then the destination DN of the current operation is set to the source DN of the matching object. If only a single result is returned and it is already associated, then the destination DN of the current operation is set to the single character ￼. If multiple results are returned then the destination DN of the current operation is set to the single character &#xFFFD; .

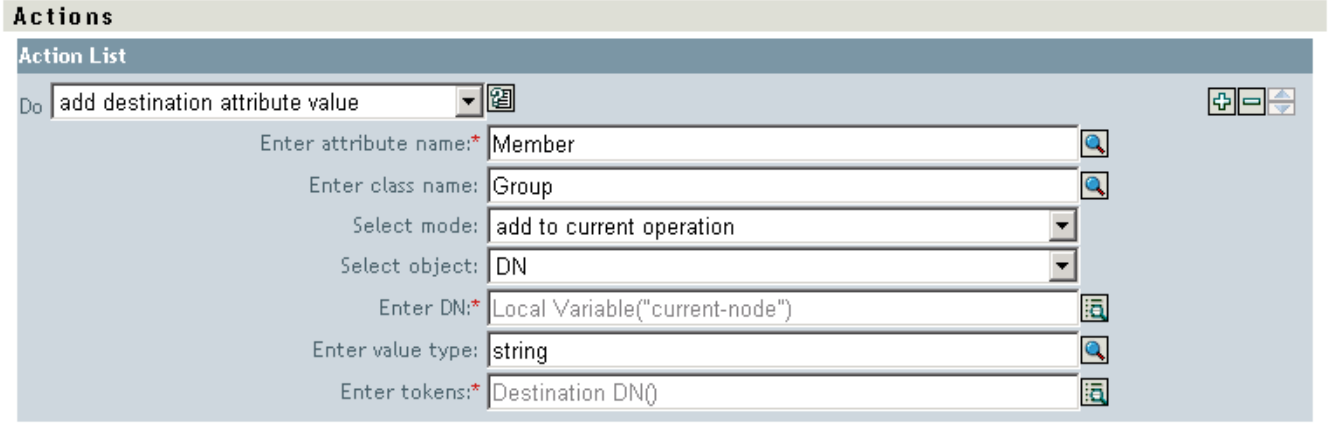
For Each

This action causes the specified action to be repeated once for each node in the specified node set.

Example



The following is an example of the Argument Actions Builder, used to provide the action argument:



Fields

Node Set

Node set on which the specified action is repeated.

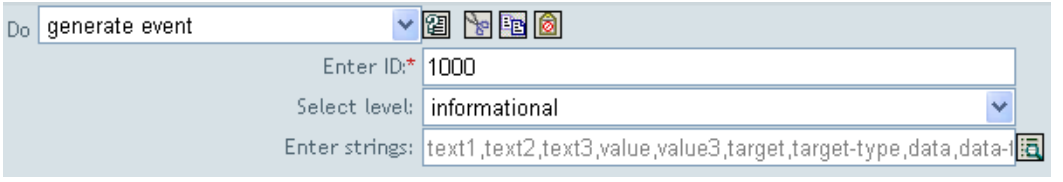
Action

Action to perform on each node in the node set.

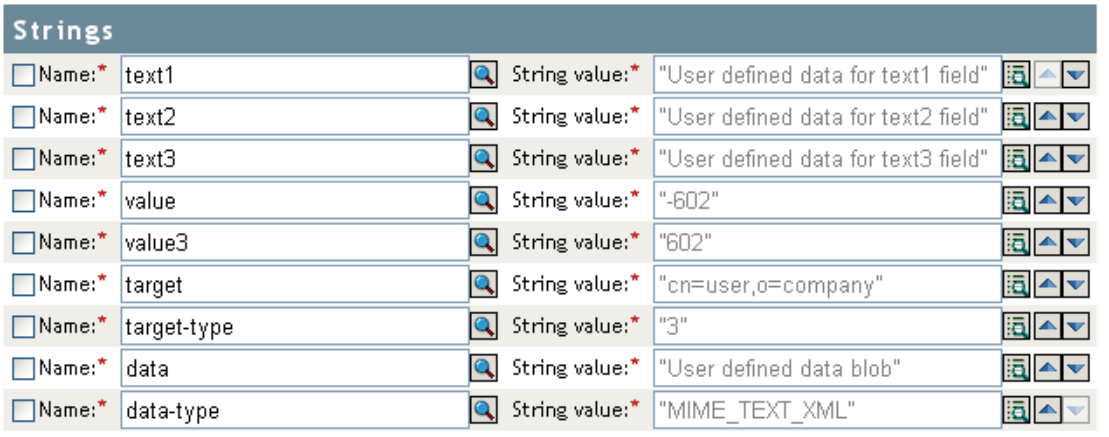
Generate Event

This action causes a DirXML user-defined event to be sent to Nsure™ Audit.

Example



The following is an example of the Named String Builder, used to provide the strings argument:



Fields

ID

ID of the event. The provided value must result in an integer in the range of 1000-1999 when parsed using the parseInt method of java.lang.Integer.

Level

Level of the event.

Level	Description
log-emergency	Events that cause the DirXML engine or driver to shutdown.
log-alert	Events that require immediate attention.
log-critical	Events that can cause parts of the DirXML engine or driver to malfunction.
log-error	Events describing errors which can be handled by the DirXML engine or driver.
log-warning	Negative events not representing a problem.

Level	Description
log-notice	Events (positive or negative) an administrator can use to understand or improve use and operation.
log-info	Positive events of any importance.
log-debug	Events of relevance for support or engineers to debug operation of the DirXML engine or driver.

Strings

User-defined string, integer, and binary values to include with the event. These values are provided using the Named String Builder.

Tag	Description
target	The object being acted upon.
target-type	Integer specifiying a predefined format for the target. Predefined values for target-type are currently: ◆ 0 = None ◆ 1 = Slash Notation ◆ 2 = Dot Notation ◆ 3 = LDAP Notation
subTarget	The sub-component of the target being acted upon.
text1	Text entered here will be stored in the text1 event field.
text2	Text entered here will be stored in the text2 event field.
text3	Text entered here is stored in the text3 field.
value	Any number entered here is stored in the value event field.
value3	Any number entered here will be stored in the value3 event field.
data	Data entered here will be stored in the blob event field.
data-type	MIME-type of data. See logevents.h, for a complete list.

Remarks

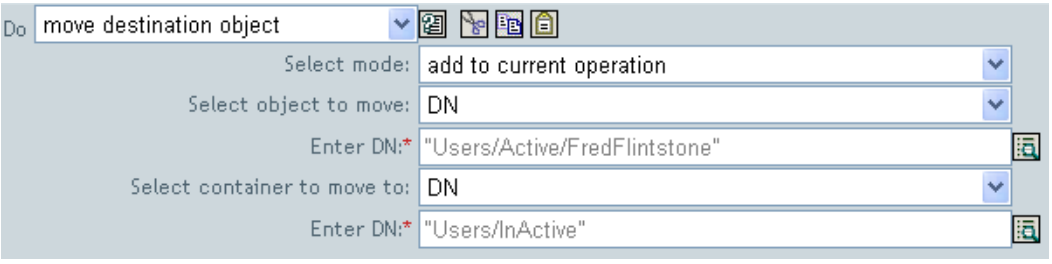
DirXML user-defined event IDs must be between the range of 1000 to 1999. Valid event levels are definend in the following table. The remaining event data fields are provided by four string elements with name attributes. The Nsure Audit event structure contains a target, a subTarget, three strings (text1, text2, text3), two integers (value, value3), and generic field (data). The text fields are limited to 256 bytes, while the data field may contain up to 3KB of information, unless a larger data field is enabled in your environment.

A detailed discussion of generating events using Policy Builder is contained in the *Identity Manager 2 Administration Guide* in the "Logging and Reporting Using Nsure Audit" section.

Move Destination Object

This action causes an object in the destination data store to be moved. Select the current object, a DN, or an association to move to another location specified by a DN, or an association.

Example



Fields

Mode

Select whether this action should be added to, before, or after the current operation, or written directly to the destination data store.

Object to Move

Select the object to be moved in the destination data store. This object can be the current object, or specified by a DN or an association.

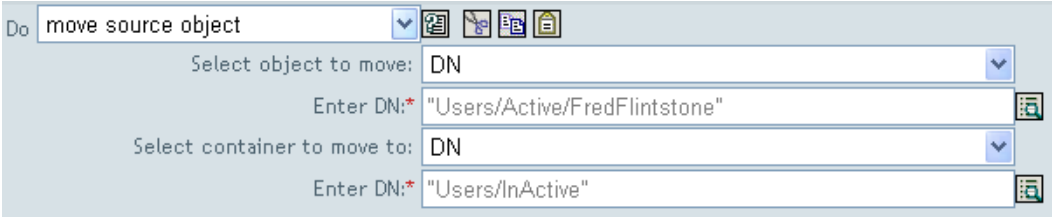
Container

Select the container to receive the object. This container is specified by a DN or an association.

Move Source Object

This action causes an object in the source data store to be moved. Select the current object, a DN, or an association to move to another location specified by a DN, or an association.

Example



Fields

Object to Move

Select the object to be moved in the source data store. This object can be the current object, or specified by a DN or an association.

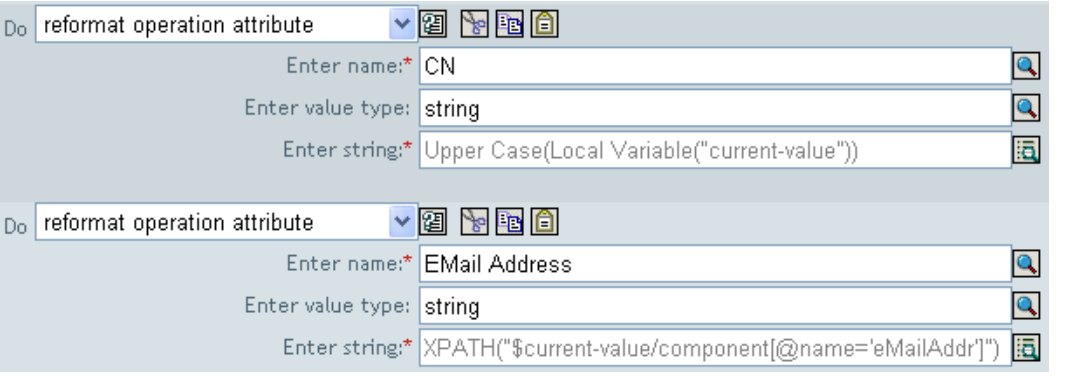
Select Container

Select the container to receive the object. This container is specified by a DN or an association.

Reformat Operation Attribute

This action causes all values for the named attribute within the current operation to be replaced with the specified value. The specified value is evaluated once for each value being replaced with the local variable current-value set to the original value.

Example



Fields

Name

Specify the name of the attribute to reformat.

Value Type

Specify the syntax of the new attribute value.

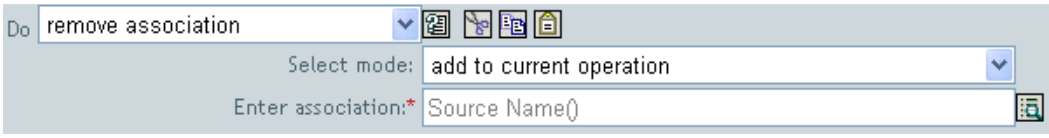
Tokens

Provide the new format of the attribute using the Argument Builder.

Remove Association

This action causes a remove association command to be sent to eDirectory.

Example



Fields

Mode

Select whether this action should be added to, before, or after the current operation, or written directly to the destination data store.

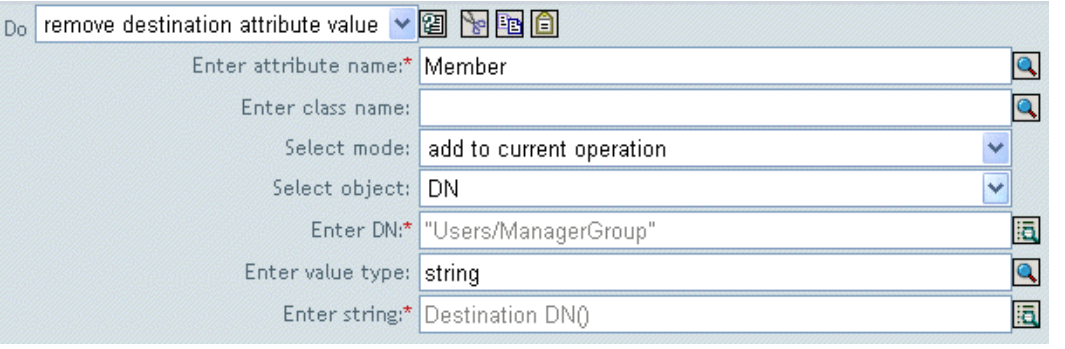
Association

Provide the value of the association using the Argument Builder.

Remove Destination Attribute Value

This action causes the specified value to be removed from the named attribute on an object in the destination data store. The target object is the current object, a DN, or an association.

Example



Fields

Attribute Name

Specify the name of the attribute to add to the target object in the destination data store.

Class Name

(Optional) Specify the class name of the target object in the destination data store. This value might be required if object is other than current object, for schema mapping purposes.

Mode

Select whether this action should be added to, before, or after the current operation, or written directly to the destination data store.

Select Object

Select the target object in the destination data store. This object can be the current object, or specified by a DN or an association.

Value Type

Specify the syntax of the new attribute value.

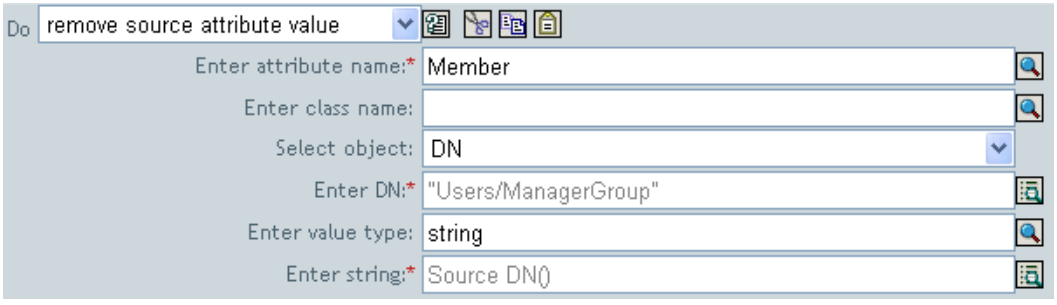
Tokens

Provide the value of the new attribute using the Argument Builder.

Remove Source Attribute Value

This action causes the specified value to be removed from the named attribute on an object in the source data store. The target object is the current object, a DN, or an association.

Example



Fields

Attribute Name

Specify the name of the attribute to add to the target object in the source data store.

Class Name

(Optional) Specify the class name of the target object in the source data store. This value might be required if object is other than current object, for schema mapping purposes.

Select Object

Select the object in the destination data store to receive the attribute. This object can be the current object, or specified by a DN or an association.

Value Type

Specify the syntax of the new attribute value.

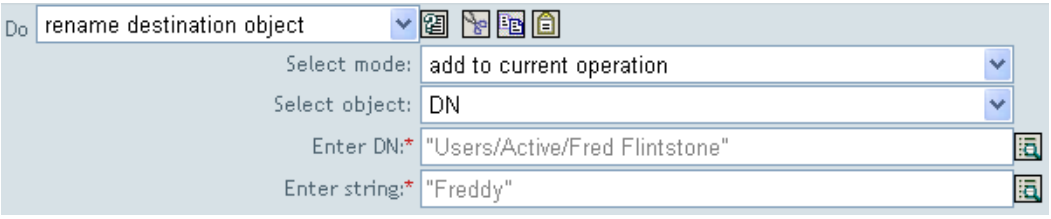
Tokens

Provide the value of the new attribute using the Argument Builder.

Rename Destination Object

This action causes an object in the destination data store to be renamed. The target object is the current object, a DN, or an association.

Example



Fields

Mode

Select whether this action should be added to, before, or after the current operation, or written directly to the destination data store.

Object

Select the target object in the destination data store. This object can be the current object, or specified by a DN or an association.

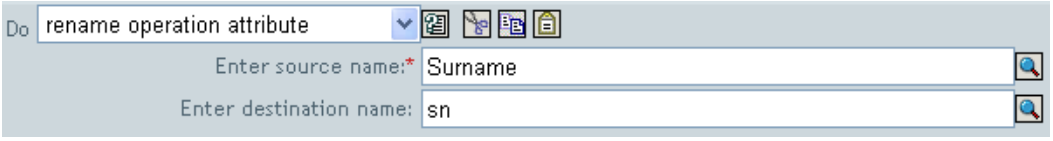
String

Provide the new name of the object in the destination data store using the Argument Builder.

Rename Operation Attribute

This action causes all elements that are children of the current operation with the specified attribute equal to the name specified, to have the specified attribute set to the destination attribute name.

Example



Fields

Source Name

Specify the name of the attribute in the source data store.

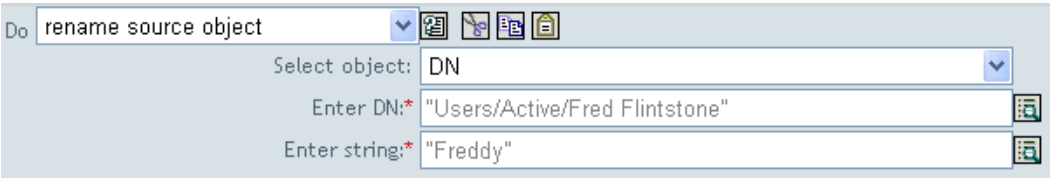
Destination Name

Specify the name of the attribute in the destination data store.

Rename Source Object

This action causes an object in the source data store to be renamed to the specified name. The target object is the current object, a DN, or an association.

Example



Fields

Select Object

Select the target object in the source data store. This object can be the current object, or specified by a DN or an association.

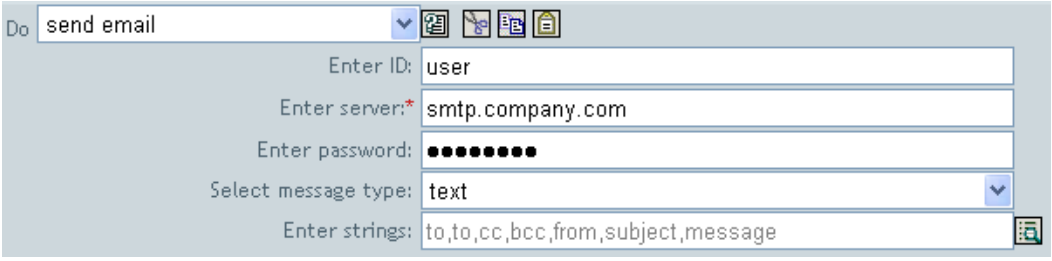
String

Provide the new name of the object in the source data store using the Argument Builder.

Send Email

This action causes an e-mail notification to be sent to the specified server. Optional credentials for authentication to the SMTP server are provided in the id and password.

Example



The following is an example of the Named String Builder, used to provide the strings argument:

Fields

ID

(Optional) User ID in the SMTP system sending the message.

Server

SMTP server name.

Password

(Optional) SMTP server account password.

WARNING: The value of the password attribute is stored in clear text.

Type

Select the e-mail message type.

Strings

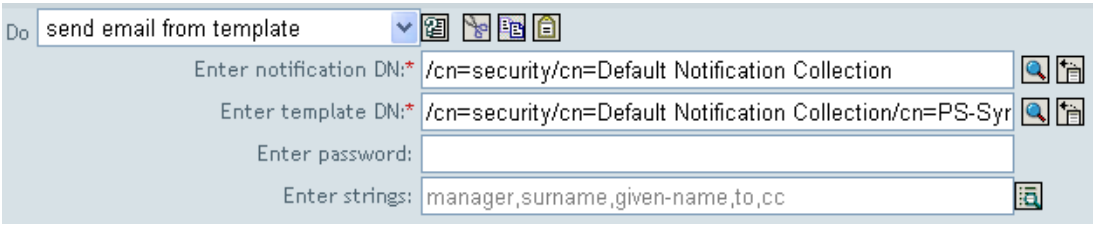
These values contain the various e-mail addresses, subject and message. The following table lists valid named string arguments:

String Name	Description
to	Adds the address to the list of e-mail recipients, multiple instances are allowed.
cc	Adds the address to the list of CC e-mail recipients, multiple instances are allowed.
bcc	Adds the address to the list of BCC e-mail recipients, multiple instances are allowed.
from	Specifies the address to be used as the originating e-mail address.
reply-to	Specifies the address to be used as the e-mail message reply address.
subject	Specifies the e-mail subject.
message	Specifies the content of the e-mail message.
encoding	Specifies the character encoding to use for the e-mail message.

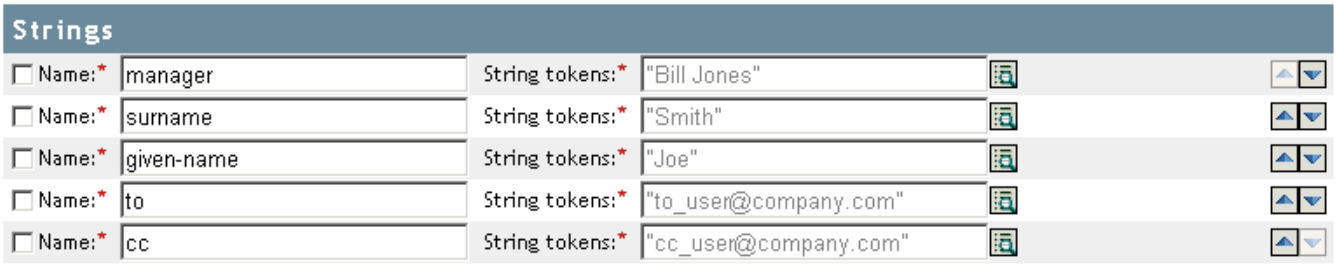
Send Email From Template

This action causes an e-mail notification to be generated using a SMTP notification configuration object, e-mail template object and replacement tokens.

Example



The following is an example of the Named String Builder, used to provide the strings argument:



Fields

Notification DN

Slash form DN of SMTP notification configuration object.

Template DN

Slash form DN of e-mail template object.

Password

(Optional) SMTP server account password.

WARNING: The value of the password attribute is stored in clear text.

Strings

Replacement tokens for the e-mail message. The following table contains reserved replacement tokens, which specify the various e-mail addresses:

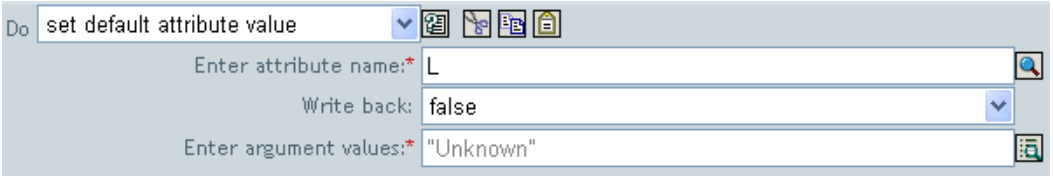
String Name	Description
to	Adds the address to the list of e-mail recipients, multiple instances are allowed.
cc	Adds the address to the list of CC e-mail recipients, multiple instances are allowed.

String Name	Description
bcc	Adds the address to the list of BCC e-mail recipients, multiple instances are allowed.
reply-to	Specifies the address to be used as the e-mail message reply address.
encoding	Specifies the character encoding to use for the e-mail message.

Set Default Attribute Value

This action causes the values specified to be added to the current operation, for the named attribute, if no values for that attribute already exist. It is only valid when the current operation is add. If write-back="true", default values are also written back to the source object.

Example



Fields

Attribute Name

Specify the name of the attribute to add to the target object in the destination data store.

Write Back

If write back is set to true, default values are also written back to the source object.

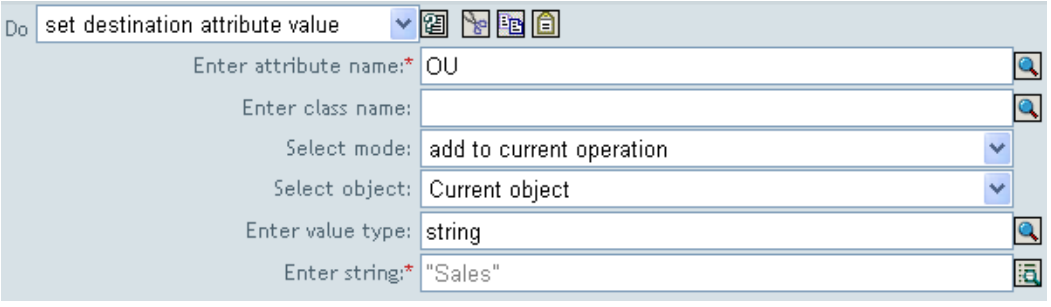
Values

Provide the default value(s) of the attribute using the Argument Builder.

Set Destination Attribute Value

This action causes the specified value to be added to the named attribute on an object in the destination datastore, and all other values for that attribute to be removed. The target object is the current object, a DN, or an association.

Example



Fields

Attribute Name

Specify the name of the attribute to add to the target object in the destination data store.

Class Name

(Optional) Specify the class name of the target object in the destination data store. This value might be required if object is other than current object, for schema mapping purposes.

Mode

Select whether this action should be added to, before, or after the current operation, or written directly to the destination data store.

Object

Select the target object in the source data store to receive the attribute. This object can be the current object, or specified by a DN or an association.

Value Type

Select the syntax of the attribute value.

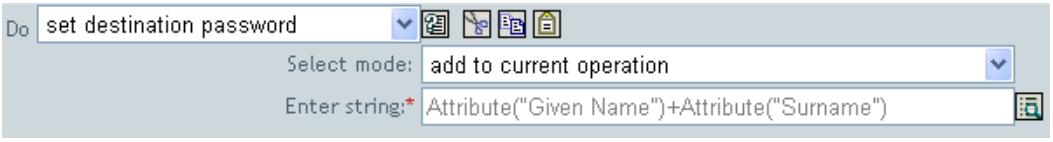
Tokens

Provide the value of the attribute using the Argument Builder.

Set Destination Password

This action causes the specified value to be set as the password for the current object in the destination data store.

Example



Fields

Mode

Select whether this action should be added to, before, or after the current operation, or written directly to the destination data store.

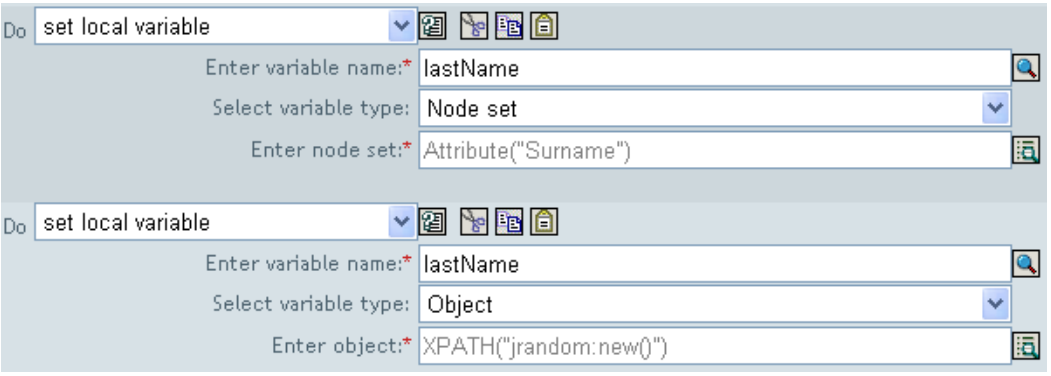
String

Provide the value of the password using the Argument Builder.

Set Local Variable

This action causes a local variable with the given name to be set to the string value specified, the XPATH 1.0 Node Set specified, or the Java* object specified.

Example



Fields

Variable Name

Specify the name of the new local variable.

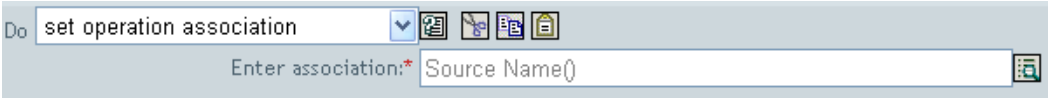
Variable Type

Select the type of local variable to add. This can be a string, an XPATH 1.0 Node Set, or a Java object.

Set Operation Association

This action causes the association value for the current operation to be set to the specified value.

Example



Fields

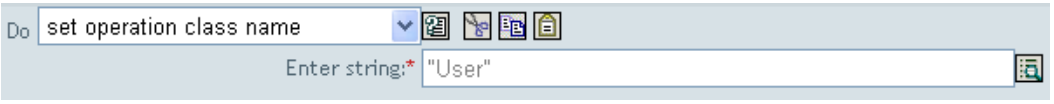
Association

Provide the new association value.

Set Operation Class Name

This action causes the object class name for the current operation to be set to the specified value.

Example



Fields

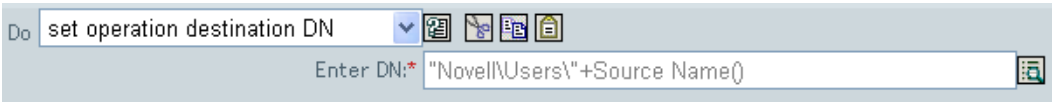
String

Provide the new class name.

Set Operation Destination DN

This action causes the destination DN for the current operation to be set to the specified value.

Example



Fields

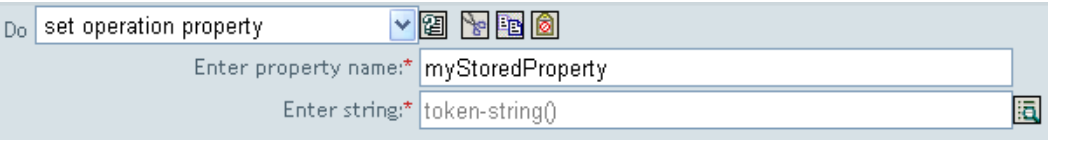
DN

Provide the new destination DN.

Set Operation Property

This action creates an operation property with the specified name and value on the current operation. An operation property is a named value that is stored within an operation, and is typically used to supply additional context that may be needed by the policy that handles the results of an operation.

Example



Fields

Property Name

Provide the name of the new operation property.

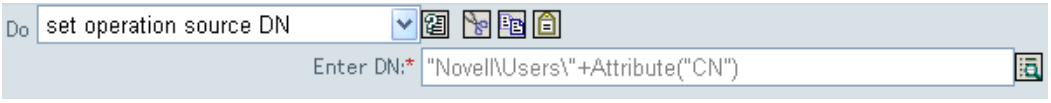
String

Provide the value of the new operation property.

Set Operation Source DN

This action causes the source DN for the current operation to be set to the specified value.

Example



Fields

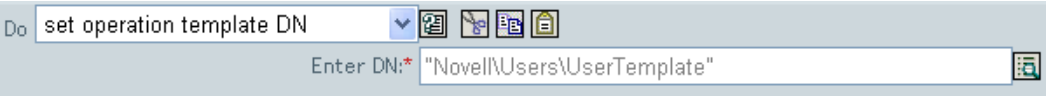
DN

Provide the new source DN.

Set Operation Template DN

This action causes the template DN for the current operation to be set to the specified value. This action is only valid when the current operation is add.

Example



Fields

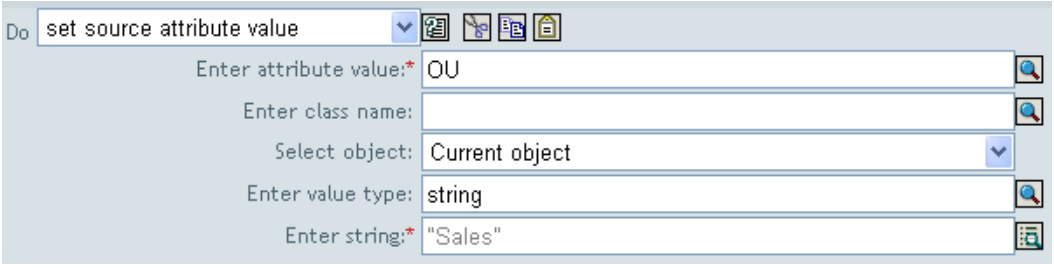
DN

Provide the new template DN.

Set Source Attribute Value

This action causes the specified value to be added to the named attribute on an object in the source data store, and all other values for that attribute to be removed. The target object is the current object, a DN, or association.

Example



Fields

Attribute Name

Specify the name of the attribute to add to the target object in the source data store.

Class Name

(Optional) Specify the class name of the target object in the source data store. This value might be required if object is other than current object, for schema mapping purposes.

Object

Select the target object in the source data store to receive the attribute. This object can be the current object, or specified by a DN or an association.

Value Type

Select the syntax of the attribute value.

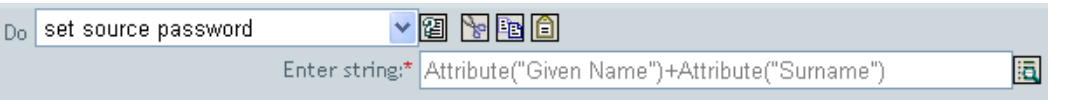
Tokens

Provide the value of the attribute using the Argument Builder.

Set Source Password

This action causes the specified value to be set as the password for the current object in the source data store.

Example



Fields

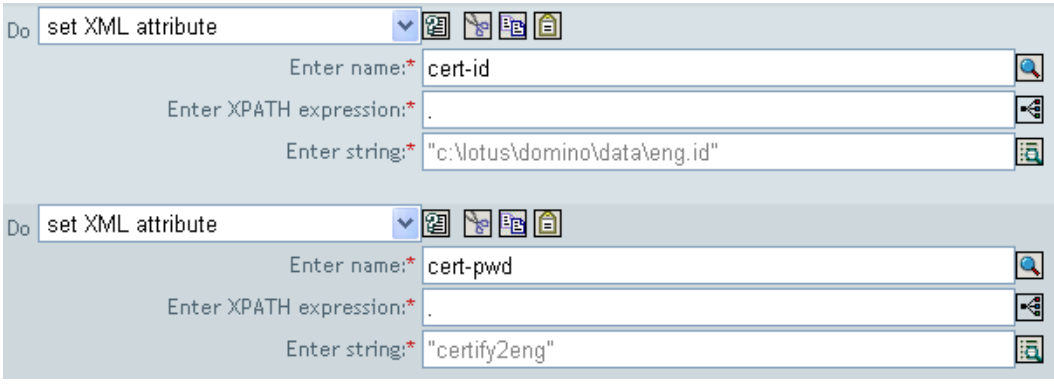
String

Provide the value of the source password using the Argument Builder.

Set XML Attribute

This action causes a custom XML attribute named by the name attribute to be set on the set of elements selected by the XPATH expression.

Example



Fields

Name

Tag name of the XML attribute. This name can contain a namespace prefix if the prefix has been previously defined on this policy.

XPATH Expression

XPATH 1.0 expression that returns a nodeset containing the element(s) on which the XML attribute should be set.

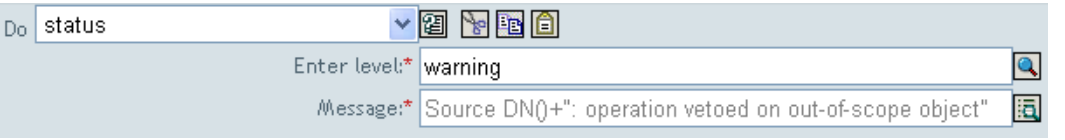
String

Provide the value of the XML attribute using the Argument Builder.

Status

This action causes a status notification to be generated with the specified level and message.

Example



Fields

Level

Specify the status level of the notification.

Message

Provide the status message using the Argument Builder.

Remarks

If level is retry then the policy will immediately halt processing of the input document and schedule a retry of the event currently being processed.

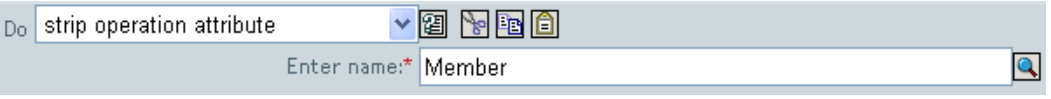
If level is fatal then the policy will immediately halt processing of the input document and initiate a shutdown of the driver.

If a the current operation has an event-id, then that event-id will by used for the status notification, otherwise there will be no event-id reported.

Strip Operation Attribute

This action causes all elements that are children of the current operation with the specified attribute name equal to the name specified to be stripped from the current operation.

Example



Fields

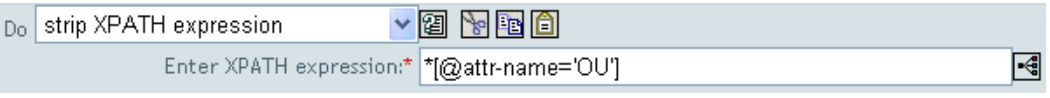
Name

Specify the name of the attribute to strip from the operation.

Strip Xpath

This action causes nodes selected by the XPATH 1.0 expression to be removed from the current operation. The expression must evaluate to a node-set.

Example



Fields

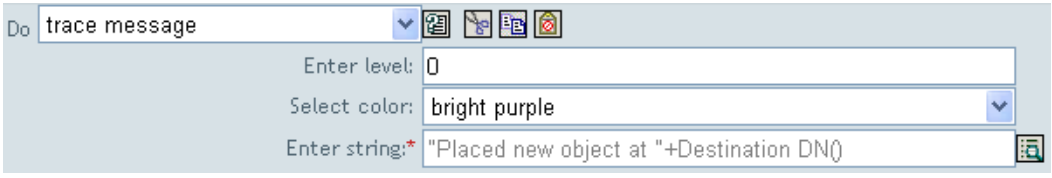
XPATH Expression

XPATH 1.0 expression that returns a nodeset containing the element(s) to be removed from the current operation.

Trace Message

This action sends the specified string to DSTRACE in the selected color. In order for the message to appear, the specified trace level must be less than or equal to the currently selected level in DSTRACE.

Example



Fields

Level

Enter the trace level of the message. The default level is 0.

Color

Select the trace message color.

String

Provide the value of the trace message.

Veto

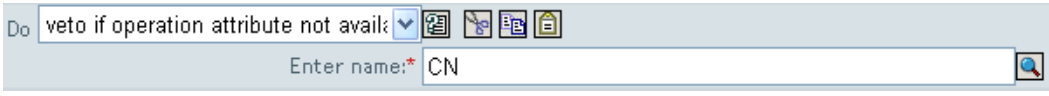
This action causes the current operation to be cancelled.

Example

Veto If Operation Attribute Not Available

This action causes the current operation to be cancelled if the specified attribute is not available in the current operation.

Example



Fields

Name

Specify the name of the attribute to check for availability before a veto is performed.

Nouns

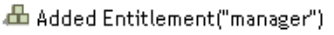
This section contains detailed reference to all nouns available using the Policy Builder interface.

Added Entitlement (page 105)
Association (page 106)
Attribute (page 107)
Class Name (page 108)
Destination Attribute (page 109)
Destination DN (page 110)
Destination Name (page 111)
Entitlement (page 112)
Global Configuration Value (page 113)
Local Variable (page 114)
Named Password (page 115)
Operation (page 116)
Operation Attribute (page 117)
Operation Property (page 118)
Password (page 119)
Removed Attribute (page 120)
Removed Entitlement (page 121)
Source Attribute (page 122)
Source DN (page 123)
Source Name (page 124)
Text (page 125)
Unique Name (page 126)
Unmatched Source DN (page 128)
XPath (page 129)

Added Entitlement

This noun expands to the value(s) of the named entitlement added in the current operation.

Example



Fields

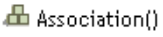
Name

Name of the entitlement.

Association

This noun expands to the association value specified in the current operation.

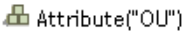
Example



Attribute

This noun expands to the value of the specified attribute in the current operation.

Example



Fields

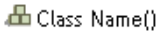
Name

Name of the attribute.

Class Name

This noun expands to the object class name specified in the current operation.

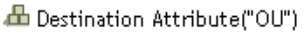
Example



Destination Attribute

This noun expands to the specified attribute value.

Example



Fields

Class Name

Class name of object in the destination data store to read. This might be required if object is other than the current object.

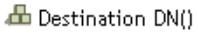
Name

Name of the attribute.

Destination DN

This noun expands to the destination DN specified in the current operation or a portion thereof.

Example



Fields

Convert

True converts to the DN format of the source data store.

Start

Segment index to start with:

- 0 is the rootmost segment
- >0 is an offset from the rootmost segment
- -1 is the leafmost segment
- <-1 is an offset from the leafmost segment towards the rootmost segment

Length

Number of DN segments to include. Negative numbers are interpreted as (total # of segments + length) + 1 (e.g for a DN with 5 segments a length of -1 = (5 + (-1)) + 1 = 5, -2 = (5 + (-2)) + 1 = 4, etc.)

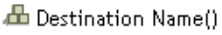
Remarks

If start and length are set to the default values {0,-1}, then the entire DN is used, otherwise only the portion of the DN specified by start and length is used. The format of the DN is automatically converted to the format of the source data store if convert to source DN format is set to True.

Destination Name

This expands to the unqualified Relative Distinguished Name (RDN) of the destination DN specified in the current operation.

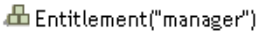
Example



Entitlement

This noun expands to the value(s) of the named entitlement for current object.

Example



Fields

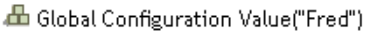
Name

Name of the entitlement.

Global Configuration Value

This noun expands to the value of the specified global configuration variable.

Example



Fields

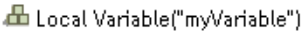
Name

Name of the global configuration value.

Local Variable

This noun expands to the value of the named local variable.

Example



Fields

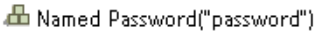
Name

Name of the local variable.

Named Password

This noun expands to the named password from the driver.

Example



Fields

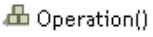
Name

Name of the password.

Operation

This noun expands to the name of the current operation.

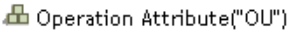
Example



Operation Attribute

This noun expands to the value of the specified attribute from the current operation (add attribute, add value, or attribute).

Example



Fields

Name

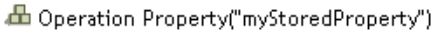
Name of the attribute from the current operation.

.

Operation Property

This noun expands to the value of the specified operation property on the current operation.

Example



Fields

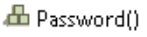
Name

Name of the property.

Password

This noun expands to the password specified in the current operation.

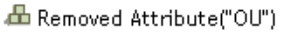
Example



Removed Attribute

This noun expands to the specified attribute value being removed in the current operation (remove attribute).

Example



Fields

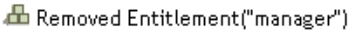
Name

Name of the removed attribute.

Removed Entitlement

This noun expands to the value(s) of the named entitlement removed in the current operation.

Example



Fields

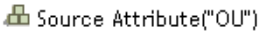
Name

Name of the entitlement.

Source Attribute

This noun expands to the specified attribute values from the current object, a DN, or association, in the source data store.

Example



Fields

Class Name

Class name of object in the source data store to read. This might be required if object is other than the current object.

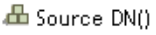
Name

Name of the attribute.

Source DN

This noun expands to the source DN specified in the current operation, or a portion thereof.

Example



Fields

Convert

True converts to the DN format of the destination data store.

Start

Segment index to start with:

- 0 is the rootmost segment
- >0 is an offset from the rootmost segment
- -1 is the leafmost segment
- <-1 is an offset from the leafmost segment towards the rootmost segment

Length

Number of DN segments to include. Negative numbers are interpreted as (total # of segments + length) + 1 (e.g for a DN with 5 segments a length of -1 = (5 + (-1)) + 1 = 5, -2 = (5 + (-2)) + 1 = 4, etc.)

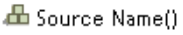
Remarks

If start and length are set to the default values {0,-1}, then the entire DN is used, otherwise only the portion of the DN specified by start and length is used. The format of the DN is converted to the format of the destination data store if the convert attribute is set to True.

Source Name

This expands to the unqualified Relative Distinguished Name (RDN) of the source DN specified in the current operation.

Example



Text

This noun expands to the specified text.

Example

Fields

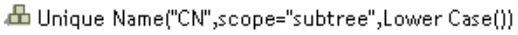
Text

Specify the text value.

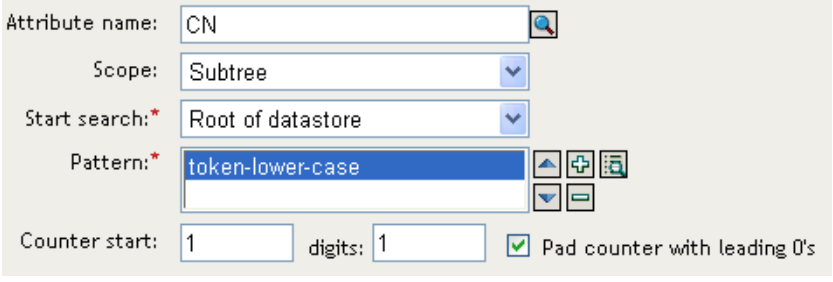
Unique Name

This noun expands to a pattern-based name that is unique in the destination data store according to the criteria specified.

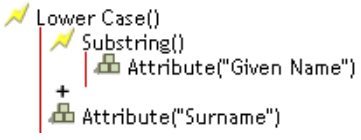
Example



The following is an example of the Editor pane when constructing the unique name argument:



The following pattern was constructed to provide unique names:



If this pattern does not generate a unique name, a digit is appended starting with counter start, incrementing up to the specified number of digits. In this example,9 additional unique names would be generated by the appended digit before an error occurs (pattern1 - pattern9).

Fields

Name

Name of attribute to check for uniqueness.

Scope

Scope in which to check uniqueness. The default scope is subtree.

Start Search

Select a starting point for the search. The starting point can be the root of the data store, or specified by a DN or association.

Pattern

Provide a pattern to use to generate unique values using the Argument Builder.

Counter Start

Number to start counter, default is 1.

Digits

Width in digits of counter, default is 1. The Pad counter with leading 0's checkbox prepends 0 to match the digit length. For example, with a digit width of 3, the initial unique value would be appended with 001, then 002, and so on.

Remarks

For each provided pattern, a query is performed for that value in the name attribute against the destination data store, using a DN, an association, or the root of the data store as the base of the query, and the selected scope.

Each provided pattern is tried in order until a value is found that does not return any instances.

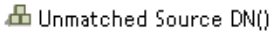
If all of the provided values are exhausted, then the final value will have a counter appended to it and the value will be tried repeatedly (increasing the counter each time) until the query does not return any instances. By default, the counter starts at 1 and is not padded. The counter can be set to start at a different number using the counter start field. The counter will use the number of digits specified by the digits field (default 1). If the number of digits is less than those specified, then the counter will be right padded with zeros. If/when the number of digits exceeds those specified, then no unique name will be generated and the enclosing rule will return an error status.

If the destination data store is eDirectory and name is omitted, then a search is performed against the pseudo-attribute "[Entry].rdn", which represents the RDN of an object without respect to what the naming attribute might be. If the destination data store is the application, then name is required.

Unmatched Source DN

This noun expands to the portion of the source DN in the current operation that corresponds to the part of the DN that was not matched by the most recent match of an If Source DN condition, in the conditions for this rule (taking into account short circuit evaluation).

Example



Fields

Convert

True converts to DN format of destination data store.

Remarks

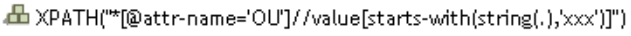
If there were no matches then the entire DN is used. The format of the DN is converted to the format of the destination data store if the convert attribute is set to True.

This token is equivalent to <copy-path-prefix> in DirXML 1.x and exists primarily for backward compatibility purposes.

XPath

This noun expands to results of evaluating an XPATH 1.0 expression.

Example



Fields

Expression

XPATH 1.0 expression to evaluate.

Verbs

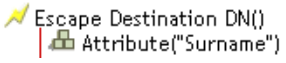
This section contains detailed reference to all verbs available using the Policy Builder interface.

Escape Destination DN (page 130)
Escape Source DN (page 131)
Lower Case (page 132)
Parse DN (page 133)
Replace All (page 135)
Replace First (page 136)
Substring (page 137)
Upper Case (page 138)

Escape Destination DN

This verb escapes the enclosed values according to the rules of the destination DN format.

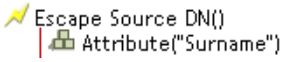
Example



Escape Source DN

This verb escapes the enclosed values according to the rules of the source DN format.

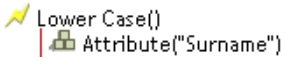
Example



Lower Case

This verb converts enclosed nouns and verbs to lower case.

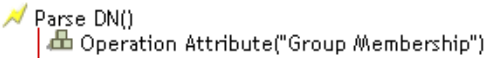
Example



Parse DN

This noun expands to a version of the DN specified by expansion of the concatenation of the enclosed tokens.

Example



Fields

Destination DN Delimiter

Specifies the custom destination DN delimiter.

Destination DN Format

Specifies the format used to ouput the parsed DN.

Length

Number of DN segments to include. Negative numbers are interpreted as (total # of segments + length) + 1 (e.g for a DN with 5 segments a length of -1 = (5 + (-1)) + 1 = 5, -2 = (5 + (-2)) + 1 = 4, etc.).

Source DN Delimiter

Specifies the custom source DN delimiter.

Source DN Format

Specifies the format used to parse the source DN.

Start

Segment index to start with.

- 0 is the rootmost segment
- >0 is an offset from the rootmost segment
- -1 is the leafmost segment
- <-1 is an offset from the leafmost segment towards the rootmost segment

Remarks

The DN is parsed according the format specified by src-dn-format. The portion of the DN specified by start and length is then converted to the format specified by dest-dn-format.

The parameters are used to specify custom DN formats. The 8 characters which make up the delimiter set are defined as follows:

1. Typed Name Boolean Flag: '0' means names are NOT typed, '1' means names are typed

2. Unicode No-Map Character Boolean Flag: '0' means don't output or interpret unmappable unicode characters as escaped hex digit strings, e.g.,\FEFF. The following unicode characters are not accepted by eDirectory: 0xfeff, 0xfffe, 0xfffd, and 0xffff.

3. Relative RDN Delimiter

4. RDN Delimiter

5. Name Divider

6. Name Value Delimiter

7. Wildcard Character

8. Escape Character

If RDN Delimiter and Relative RDN Delimiter are the same character, then the orientation of the name is root right, otherwise the orientation is root left.

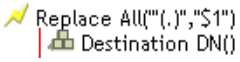
If there are more than 8 characters in the delimiter set, then the extra characters will be considered as characters that need to be escaped, but will have no other special meaning.

If start and length are set to the default values {0,-1}, then the entire DN is used, otherwise only the portion of the DN specified by start and length is used.

Replace All

This verb replaces all occurrences of the specified regular expression on all enclosed nouns and verbs.

Example



Fields

Regular Expression

Regular Expression that matches the substring to replace.

Replace With

Regular expression that specifies the replacement string.

Remarks

Each matching instance is replaced the string specified by the value specified in the Replace with field.

For details on creating regular expressions, see:

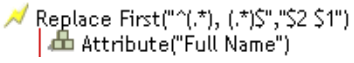
- http://java.sun.com/j2se/1.4/docs/api/java/util/regex/Pattern.html (http://java.sun.com/j2se/1.4/docs/api/java/util/regex/Pattern.html)
- http://java.sun.com/j2se/1.4/docs/api/java/util/regex/Matcher.html#replaceAll(java.lang.String) (http://java.sun.com/j2se/1.4/docs/api/java/util/regex/Matcher.html#replaceAll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

The pattern option CASE_INSENSITIVE, DOTALL, and UNICODE_CASE are used but can be reversed using the appropriate embedded escapes.

Replace First

This verb replaces the first occurrence of the specified regular expression.

Example



Fields

Regular Expression

Regular Expression that matches the substring to replace.

Replace With

Regular expression that specifies the replacement string.

Remarks

The matching instance is replaced the string specified by the value specified in the Replace with field.

For details on creating regular expressions, see:

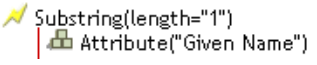
- http://java.sun.com/j2se/1.4/docs/api/java/util/regex/Pattern.html (http://java.sun.com/j2se/1.4/docs/api/java/util/regex/Pattern.html)
- http://java.sun.com/j2se/1.4/docs/api/java/util/regex/Matcher.html#replaceAll(java.lang.String) (http://java.sun.com/j2se/1.4/docs/api/java/util/regex/Matcher.html#replaceAll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

The pattern option CASE_INSENSITIVE, DOTALL, and UNICODE_CASE are used but can be reversed using the appropriate embedded escapes.

Substring

This verb expands to string containing the number of characters specified in the Length field. Enclosed nouns and verbs are concatenated before the substring verb is applied.

Example



Fields

Start

Starting location for the concatenation:

- 0 is the first character.
- >0 is an offset from the start of the string
- -1 is the last character.
- <-1 is an offset from the last character towards the start of the string.

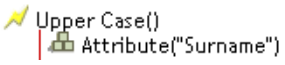
Length

Number of characters from start to include in the substring. Negative numbers are interpreted as (total # of characters + length) + 1 (e.g. for a string with 5 characters a length of -1 = (5 + (-1)) + 1 = 5, -2 = (5 + (-2)) + 1 = 4, etc.).

Upper Case

This verb converts enclosed nouns and verbs to upper case.

Example



Values

This section contains a list of common policy builder values.

Comparison Modes

Mode	Description
case	Character by character case sensitive comparison.
nocase	Character by character case insensitive comparison.
regex	Regular expression match of entire string. Case insensitive by default, but may be changed by an escape in the expression. See 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) and 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()). Note that pattern option CASE_INSENSITIVE, DOTALL, and UNICODE_CASE are used but can be reversed using the appropriate embedded escapes.
src-dn	Compare using semantics appropriate to the DN format for the source datastore.
dest-dn	Compare using semantics appropriate to the DN format for the destination datastore.
numeric	Compare numerically.
octet	Compare octet (Base64 encoded) values.
structured	Compare structured attribute according to the comparison rules for the structured syntax of the attribute.

3 Defining Policies using XSLT Style Sheets

Style sheets define XSLT transformation rules. The XSLT processor in the DirXML® engine is compliant with the 16 November 1999 W3C Recommendation. For the specifications, see the following:

- 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)

Style sheets can be used in the following places:

- Input transformation rules
- Output transformation rules
- Event transformation rules
- Matching, create, or placement rules
- Mapping rules

The following sections describe the implementation specifics of using style sheets with DirXML.

- "Restrictions" on page 138
- "Starting with an Identity Transformation" on page 139
- "Using the Parameters that DirXML Passes" on page 139
- "Using Extension Functions" on page 142
- "Testing Style Sheets Outside of DirXML" on page 142
- "Creating a Password Example: Create Rule" on page 143
- "Creating an eDirectory User Example: Create Rule" on page 144

Managing XSLT Style Sheets in iManager

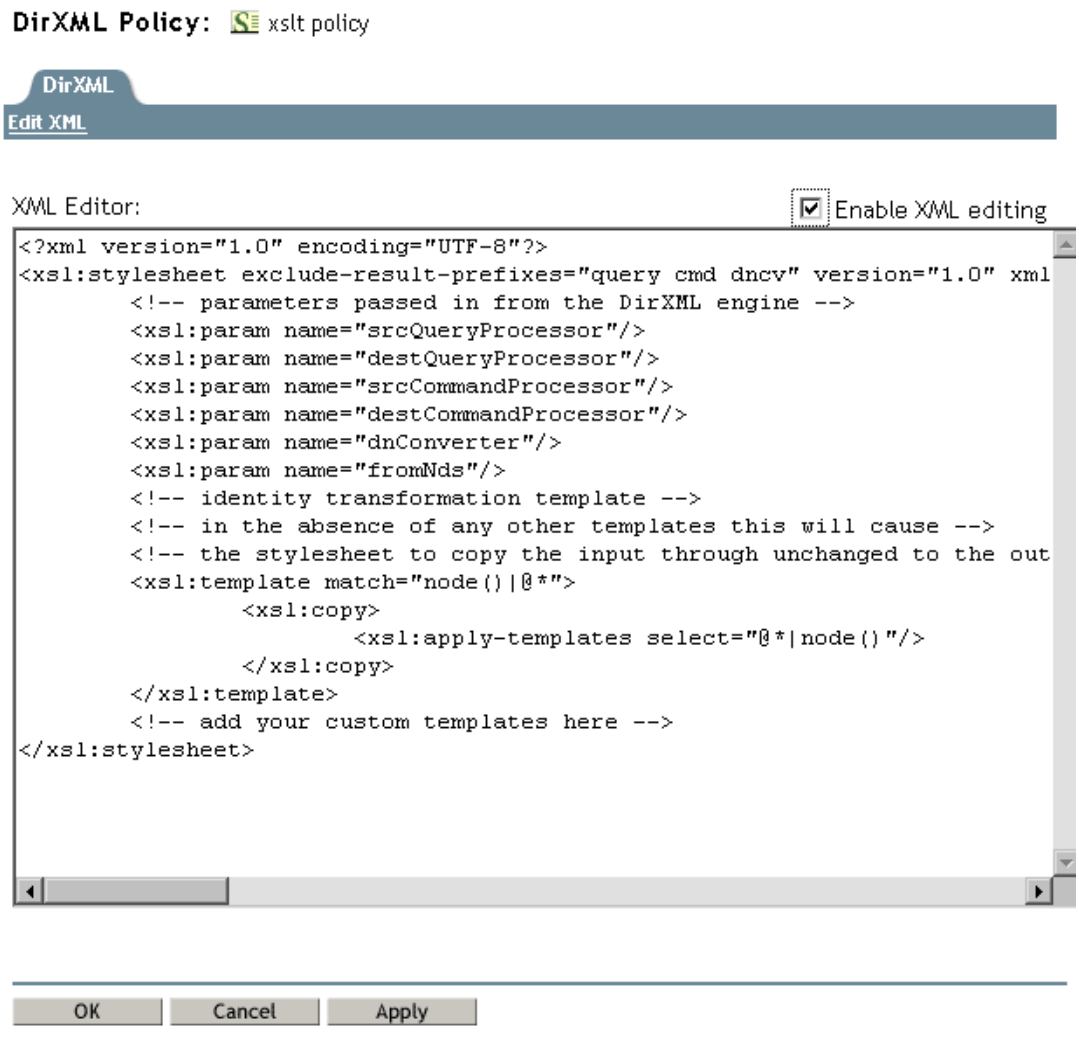
XSLT policy style sheets are added, modified, and deleted using iManager. The following sections provide details on using XSLT style sheets in iManager:

- "Adding an XSLT Policy" on page 137

Adding an XSLT Policy

1. Open the DirXML Driver Overview for the driver you want to manage.
2. Click the icon representing the policy you want to define.
3. Click Insert.
4. Enter a name for the new policy, select XSLT, then click enter.

5 Define your XSLT policy, then click OK:



Restrictions

Three of the rule types (matching, create, and placement) can be also be XML documents. When these rules are written as style sheets, they are subject to the following restrictions.

Matching Rule Restrictions

When matching rules are written as an XSLT style sheet, they are subject to the following restrictions:

- Use the special value of a single Unicode character 0xFFFD to signal that multiple matches were found.
- Can operate only on add events.
  - On the subscriber channel, the DirXML driver must add an <association> element for any matches that are found in the application.

- On the publisher channel, the DirXML driver must fill in the dest-dn attribute of the <add> element if a match is found in eDirectory™.

- Can remove events
- Cannot generate extra events
- Cannot change event types

The names of the attributes and classes are in the eDirectory name space.

Create Rule Restrictions

When create rules are written as an XSLT style sheet, they are subject to the following restrictions:

- Can operate only on add events.
- Can add attributes and values to the <add> element.
- Can remove events (this is how an add event is vetoed).

The names of the attributes and classes are in the eDirectory name space.

Placement Rule Restrictions

When placement rules are written as an XSLT style sheet, they are subject to the following restrictions:

- Can operate only on add events.
- Must fill in the dest-dn attribute of the <add> element.
- Can remove events.

The names of the attributes and classes are in the eDirectory name space.

Starting with an Identity Transformation

Unless you are translating to or from an XML format that is completely different from the DirXML format, you will want to start your style sheet with templates that implement the identity transformation. These templates allow the events in the document that you don't specifically try to intercept and change to pass through without any modifications.

The following two templates together implement the identity transformation:

```
<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>
```

Using the Parameters that DirXML Passes

The DirXML engine passes the rule style sheets the following parameters that the style sheet can use. Note that with DirXML 1.1, the query processor parameters are now passed to the schema

mapping rules and the input and output transformation rules. The command processor parameters are passed to all rules.

- fromNds—This is a boolean value that is true if the rule is being processed by the subscriber channel and false if the rule is being processed by the publisher channel.
- srcQueryProcessor—This is a Java object that implements the XdsQueryProcessor interface. This allows the style sheet to query the event source for more information.
- destQueryProcessor—This is a Java object that implements the XdsQueryProcessor interface. This allows the style sheet to query the event target for more information.
- srcCommandProcessor—This is a java object that implements the XdsCommandProcessor interface. This allows the style sheet to "write-back" a command to the event source. Not available in DirXML 1.0.
- destCommandProcessor—This is a java object that implements the XdsCommandProcessor interface.This allows the style sheet to issue a command to the command destination directly, bypassing most other rules. Not available in DirXML 1.0.

To use these parameters include the following in your style sheet:

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

With DirXML 1.1, processors will accept a query or command element as the top level element and will wrap it in <input> and <nds> if necessary.

When using the query and command parameters with the schema mapping rules, input transformation rules, and output transformation rules the following limitations apply:

1. Queries issued to the application shim must be in the form expected by the application shim. In other words, schema names must be in the application namespace and the query must conform to whatever XML vocabulary is used natively by the shim. No association refs will be added to the query.
2. Responses from the application shim will be in the form returned by the shim with no modification or schema mapping performed and no resolution of association refs.
3. Queries issued to NDS must be in the form expect by NDS. In other words schema names must be in the NDS namespace and the query must be XDS. Association refs will not be resolved.
4. Responses from the application shim will be in the form returned by the shim with no modification or schema mapping performed.

Query Processors

Use of the query processors depends on the Novell XSLT implementation of extension functions. To make a query, you need to declare a name space for the XdsQueryProcessor interface. This is done by adding the following to the <xsl:stylesheet> or <xsl:transform> element of the style sheet.

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

The following example uses one of the query processors (the extra long lines are wrapped and do not begin with a <):

```
<!-- 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>
```

Command Parameters

In order to allow channel write-back for default attributes added by a create rule, a new XML attribute called write-back was added to the <required-attr> element of the Create Rule. If present and set to true, the create rule will call the srcCommandProcessor with a modify command to write the default value back to the source.

The following example uses command parameters to perform a write back operation.

```
<?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>
```

Using Extension Functions

XSLT is an excellent tool for performing some kinds of transformations and a rather poor tool for other types of transformations such as non-trivial string manipulation and iterative processes. Fortunately the Novell XSLT processor implements extension functions which allow the style sheet to call a function implemented in Java, and by extension, any other language that can be accessed through JNI.

For specific examples, see the above example using the query processor, and the following example that illustrates using Java for string manipulation (the extra long lines are wrapped and do not begin with a <).

```
<!-- 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>
```

Testing Style Sheets Outside of DirXML

The XSLT process in the DirXML engine may be invoked from the command line and can be used to test style sheets in a more controlled environment before installing them into DirXML.

The following batch file may be used to invoke the XSLT processor on NT or Windows 2000.

```
@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
```

Invoking the processor without any arguments prints out the latest information on the command syntax for the processor.

Since you are running outside of DirXML, the srcQueryProcessor and destQueryProcessor will not be available. To get around this limitation, you can temporarily comment out code that uses the query processor and replace it with an explicit assignment of the reply you might expect from the query. For example:

```
<!-- 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>
```

Creating a Password Example: Create Rule

The following style sheet can be used for a create rule. It creates a user, generates a password for the user from the user's Surname and CN attributes, and performs an identity transform (which passes through everything in the document except the events you are trying to intercept and transform).

```
<?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>
```

Creating an eDirectory User Example: Create Rule

This style sheet can be used for a create rule. It shows how to create an eDirectory user from an entry created in an external application. This example is based on the idea that a newly hired person is first created in the Human Resources database and then on the network. It takes the user's first name and last name and generates a unique CN in the eDirectory tree. Although eDirectory requires the CN to be unique in only the container, this style sheet ensures that it is unique across all containers in the eDirectory tree.

```
<?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 Defining Filters

Filters enable you to specify the objects and attributes synchronized by Nsure™ Identity Manager.

This section covers the following filter-related topics:

- “Filter Tasks” on page 149

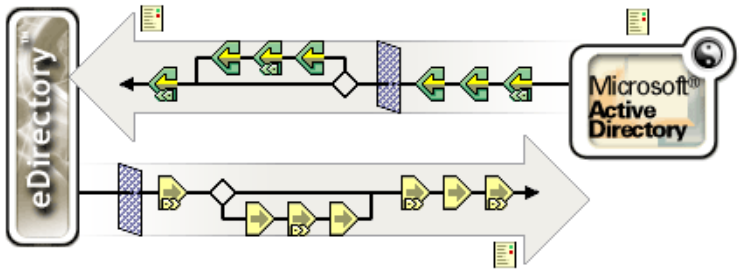
Filter Tasks

This section contains instructions on performing common filter-related tasks in iManager:

- “Managing Filters” on page 149
- “Viewing and Modifying Filters” on page 149

Managing Filters

1. In iManager, expand the DirXML Management Role, then click Overview.
2. Specify a driver set.
3. Click the driver for which you want to manage filters. The DirXML Driver Overview opens:



4. Filters are managed from the DirXML Driver Overview.

Viewing and Modifying Filters

1. Open the DirXML Driver Overview for the driver you want to manage.
2. Click the icon representing the filter you want to define on the publisher or subscriber channel.

3 The Filter window opens, displaying the currently defined filter. Use the Filter window to modify the filter. Click the help icon in the filter window for additional information.