

# Novell Identity Manager

3

December 13, 2005

POLICY BUILDER AND DRIVER  
CUSTOMIZATION GUIDE

[www.novell.com](http://www.novell.com)

# N

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.

Any products or technical information provided under this Agreement may be subject to U.S. export controls and the trade laws of other countries. You agree to comply with all export control regulations and to obtain any required licenses or classification to export, re-export, or import deliverables. You agree not to export or re-export to entities on the current U.S. export exclusion lists or to any embargoed or terrorist countries as specified in the U.S. export laws. You agree to not use deliverables for prohibited nuclear, missile, or chemical biological weaponry end uses. Please refer to [www.novell.com/info/exports/](http://www.novell.com/info/exports/) for more information on exporting Novell software. Novell assumes no responsibility for your failure to obtain any necessary export approvals.

Copyright © 2005 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.

Novell, Inc.  
404 Wyman Street, Suite 500  
Waltham, MA 02451  
U.S.A.  
[www.novell.com](http://www.novell.com)

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

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



# Contents

<b>About This Guide</b>	<b>11</b>
<b>1 Policies and Filters</b>	<b>13</b>
1.1 What Are Policies and Filters? . . . . .	13
1.1.1 Terminology Changes from Earlier Versions. . . . .	14
1.1.2 DirXML Script . . . . .	15
1.2 Introduction to Policies. . . . .	15
1.2.1 Policies . . . . .	16
1.2.2 Defining Policies . . . . .	35
1.3 Filters . . . . .	36
<b>2 Defining Policies By Using The Policy Builder With Designer</b>	<b>37</b>
2.1 Policies . . . . .	37
2.1.1 Operation. . . . .	38
2.1.2 Current Operation . . . . .	38
2.1.3 Current Object. . . . .	38
2.2 Policy Builder Tasks in Designer . . . . .	38
2.2.1 Opening Policy Builder . . . . .	38
2.2.2 Creating a Policy . . . . .	40
2.2.3 Creating a Rule . . . . .	44
2.2.4 Creating an Argument . . . . .	50
2.2.5 Modifying a Policy . . . . .	58
2.2.6 Using Predefined Rules. . . . .	61
2.2.7 Testing Policies with the Policy Simulator. . . . .	90
2.2.8 Editing the DirXML Script . . . . .	97
2.3 Regular Expressions . . . . .	101
2.4 XPath 1.0 Expressions . . . . .	101
2.5 Conditions . . . . .	102
2.5.1 If Association. . . . .	102
2.5.2 If Attribute . . . . .	103
2.5.3 If Class Name . . . . .	104
2.5.4 If Destination Attribute. . . . .	105
2.5.5 If Destination DN . . . . .	107
2.5.6 If Entitlement . . . . .	108
2.5.7 If Global Configuration Value . . . . .	109
2.5.8 If Local Variable. . . . .	110
2.5.9 If Named Password . . . . .	112
2.5.10 If Operation . . . . .	112
2.5.11 If Operation Attribute. . . . .	113
2.5.12 If Operation Property. . . . .	115
2.5.13 If Password . . . . .	116
2.5.14 If Source Attribute . . . . .	116
2.5.15 If Source DN . . . . .	117
2.5.16 If XPath Expression. . . . .	118
2.6 Actions. . . . .	119
2.6.1 Add Association. . . . .	120
2.6.2 Add Destination Attribute Value . . . . .	120
2.6.3 Add Destination Object . . . . .	122
2.6.4 Add Source Attribute Value . . . . .	123

2.6.5	Add Source Object	124
2.6.6	Append XML Element	125
2.6.7	Append XML Text	125
2.6.8	Break	126
2.6.9	Clear Destination Attribute Value	126
2.6.10	Clear Operation Property	127
2.6.11	Clear Source Attribute Value	127
2.6.12	Clone By XPath Expressions	128
2.6.13	Clone Operation Attribute	128
2.6.14	Delete Destination Object	129
2.6.15	Delete Source Object	130
2.6.16	Find Matching Object	130
2.6.17	For Each	132
2.6.18	Generate Event	133
2.6.19	Implement Entitlement	135
2.6.20	Move Destination Object	136
2.6.21	Move Source Object	137
2.6.22	Reformat Operation Attribute	138
2.6.23	Remove Association	139
2.6.24	Remove Destination Attribute Value	139
2.6.25	Remove Source Attribute Value	140
2.6.26	Rename Destination Object	141
2.6.27	Rename Operation Attribute	142
2.6.28	Rename Source Object	142
2.6.29	Send Email	143
2.6.30	Send Email From Template	144
2.6.31	Set Default Attribute Value	145
2.6.32	Set Destination Attribute Value	146
2.6.33	Set Destination Password	148
2.6.34	Set Local Variable	148
2.6.35	Set Operation Association	149
2.6.36	Set Operation Class Name	150
2.6.37	Set Operation Destination DN	150
2.6.38	Set Operation Property	151
2.6.39	Set Operation Source DN	151
2.6.40	Set Operation Template DN	152
2.6.41	Set Source Attribute Value	153
2.6.42	Set Source Password	154
2.6.43	Set XML Attribute	154
2.6.44	Status	155
2.6.45	Strip Operation Attribute	156
2.6.46	Strip XPath	156
2.6.47	Trace Message	157
2.6.48	Veto	158
2.6.49	Veto If Operational Attribute Not Available	159
2.7	Noun Tokens	159
2.7.1	Added Entitlement	160
2.7.2	Association	160
2.7.3	Attribute	161
2.7.4	Class Name	162
2.7.5	Destination Attribute	162
2.7.6	Destination DN	163
2.7.7	Destination Name	164
2.7.8	Entitlement	164
2.7.9	Global Configuration Value	165
2.7.10	Local Variable	165
2.7.11	Named Password	166
2.7.12	Operation	166

2.7.13	Operation Attribute	167
2.7.14	Operation Property	167
2.7.15	Password	168
2.7.16	Removed Attribute	168
2.7.17	Removed Entitlement	168
2.7.18	Source Attribute	168
2.7.19	Source DN	169
2.7.20	Source Name	169
2.7.21	Text	170
2.7.22	Unique Name	171
2.7.23	Unmatched Source DN	172
2.7.24	XPath	173
2.8	Verb Tokens	173
2.8.1	Escape Destination DN	174
2.8.2	Escape Source DN	174
2.8.3	Lower Case	174
2.8.4	Parse DN	175
2.8.5	Replace All	177
2.8.6	Replace First	177
2.8.7	Substring	178
2.8.8	Upper Case	179
2.9	Values	180
2.9.1	Comparison Modes	180

### **3 Defining Policies By Using The Policy Builder In iManager 183**

3.1	Policies	183
3.1.1	Operation	184
3.1.2	Current Operation	184
3.1.3	Current Object	184
3.2	Policy Builder Tasks in iManager	184
3.2.1	Opening The Policy Builder	184
3.2.2	Creating a Policy	185
3.2.3	Defining Individual Rules within a Policy	185
3.2.4	Defining Individual Arguments within a Rule	187
3.2.5	Modifying a Policy	194
3.2.6	Removing a Policy	194
3.2.7	Renaming a Policy	195
3.2.8	Deleting a Policy	195
3.2.9	Importing a Policy from an XML File	195
3.2.10	Exporting a Policy to an XML File	195
3.2.11	Creating a Policy Reference	196
3.2.12	Using Predefined Rules	196
3.3	Regular Expressions	216
3.4	XPath 1.0 Expressions	217
3.5	Conditions	218
3.5.1	If Association	218
3.5.2	If Attribute	219
3.5.3	If Class Name	220
3.5.4	If Destination Attribute	221
3.5.5	If Destination DN	222
3.5.6	If Entitlement	223
3.5.7	If Global Configuration Value	225
3.5.8	If Local Variable	226
3.5.9	If Named Password	228
3.5.10	If Operation	228
3.5.11	If Operation Attribute	229

3.5.12	If Operation Property	231
3.5.13	If Password	232
3.5.14	If Source Attribute	232
3.5.15	If Source DN	234
3.5.16	If XPath Expression	235
3.6	Actions	236
3.6.1	Add Association	237
3.6.2	Add Destination Attribute Value	238
3.6.3	Add Destination Object	239
3.6.4	Add Source Attribute Value	240
3.6.5	Add Source Object	241
3.6.6	Append XML Element	242
3.6.7	Append XML Text	243
3.6.8	Break	244
3.6.9	Clear Destination Attribute Value	244
3.6.10	Clear Operation Property	245
3.6.11	Clear Source Attribute Value	245
3.6.12	Clone By XPath Expression	246
3.6.13	Clone Operation Attribute	246
3.6.14	Delete Destination Object	247
3.6.15	Delete Source Object	247
3.6.16	Find Matching Object	248
3.6.17	For Each	249
3.6.18	Generate Event	250
3.6.19	Implement Entitlement	252
3.6.20	Move Destination Object	253
3.6.21	Move Source Object	254
3.6.22	Reformat Operation Attribute	254
3.6.23	Remove Association	255
3.6.24	Remove Destination Attribute Value	256
3.6.25	Remove Source Attribute Value	257
3.6.26	Rename Destination Object	258
3.6.27	Rename Operation Attribute	258
3.6.28	Rename Source Object	258
3.6.29	Send Email	259
3.6.30	Send Email from Template	260
3.6.31	Set Default Attribute Value	261
3.6.32	Set Destination Attribute Value	262
3.6.33	Set Destination Password	263
3.6.34	Set Local Variable	264
3.6.35	Set Operation Association	265
3.6.36	Set Operation Class Name	265
3.6.37	Set Operation Destination DN	266
3.6.38	Set Operation Property	266
3.6.39	Set Operation Source DN	267
3.6.40	Set Operation Template DN	267
3.6.41	Set Source Attribute Value	268
3.6.42	Set Source Password	269
3.6.43	Set XML Attribute	269
3.6.44	Status	270
3.6.45	Strip Operation Attribute	270
3.6.46	Strip XPath	271
3.6.47	Trace Message	271
3.6.48	Veto	272
3.6.49	Veto if Operation Attribute Not Available	273
3.7	Noun Tokens	274
3.7.1	Added Entitlement	274
3.7.2	Association	275



3.7.3	Attribute	275
3.7.4	Class Name	276
3.7.5	Destination Attribute	276
3.7.6	Destination DN	277
3.7.7	Destination Name	278
3.7.8	Entitlement	278
3.7.9	Global Configuration Value	279
3.7.10	Local Variable	279
3.7.11	Named Password	280
3.7.12	Operation	280
3.7.13	Operation Attribute	280
3.7.14	Operation Property	281
3.7.15	Password	281
3.7.16	Removed Attribute	282
3.7.17	Removed Entitlements	282
3.7.18	Source Attribute	282
3.7.19	Source DN	283
3.7.20	Source Name	283
3.7.21	Text	283
3.7.22	Unique Name	284
3.7.23	Unmatched Source DN	286
3.7.24	XPath	286
3.8	Verb Tokens	287
3.8.1	Escape Destination DN	287
3.8.2	Escape Source DN	288
3.8.3	Lower Case	288
3.8.4	Parse DN	289
3.8.5	Replace All	290
3.8.6	Replace First	291
3.8.7	Substring	292
3.8.8	Upper Case	293
3.9	Values	294
3.9.1	Comparison Modes	294

## **4 Defining Policies using XSLT Style Sheets 295**

4.1	Managing XSLT Style Sheets in Designer	295
4.1.1	Adding an XSLT Policy in Designer	295
4.2	Managing XSLT Style Sheets in iManager	297
4.2.1	Adding an XSLT Policy in iManager	297
4.3	Starting with an Identity Transformation	298
4.4	Using the Parameters that Identity Manager Passes	298
4.5	Using Extension Functions	301
4.6	Creating a Password Example: Creation Policy	302
4.7	Creating an eDirectory User Example: Creation Policy	303

## **5 Managing Filters 309**

5.1	Filter Tasks in Designer	309
5.1.1	Accessing the Filter Editor	309
5.1.2	Editing the Filter	310
5.1.3	Testing Filters	315
5.1.4	Viewing the Filter XML Source	319
5.2	Filter Tasks in iManager	321
5.2.1	Accessing the Filter	322
5.2.2	Editing the Filter	322

<b>6</b>	<b>Managing Schema Mapping Policies</b>	<b>327</b>
6.1	Schema Mapping Policy Tasks in Designer . . . . .	327
6.1.1	Accessing the Schema Map Editor . . . . .	327
6.1.2	Editing a Schema Mapping Policy . . . . .	329
6.1.3	Testing Schema Mapping Policies . . . . .	331
6.1.4	Viewing the Schema Mapping Policy XML Source . . . . .	336
6.2	Schema Mapping Policy Tasks in iManager . . . . .	338
6.2.1	Accessing Schema Mapping Policies . . . . .	339
6.2.2	Editing the Schema Mapping Policy . . . . .	339

# About This Guide

Novell® Identity Manager 3.0® 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 3.0.

- [Chapter 1, “Policies and Filters,” on page 13](#)
- [Chapter 2, “Defining Policies By Using The Policy Builder With Designer,” on page 37](#)
- [Chapter 3, “Defining Policies By Using The Policy Builder In iManager,” on page 183](#)
- [Chapter 4, “Defining Policies using XSLT Style Sheets,” on page 295](#)
- [Chapter 5, “Managing Filters,” on page 309](#)
- [Chapter 6, “Managing Schema Mapping Policies,” on page 327](#)

## Audience

This guide is intended for Identity Manager administrators.

## Feedback

We want to hear your comments and suggestions about this manual and the other documentation included with this product. Please use the User Comments feature at the bottom of each page of the online documentation, or go to [www.novell.com/documentation/feedback.html](http://www.novell.com/documentation/feedback.html) and enter your comments there.

## Documentation Updates

For the most recent version of this document, see the [Identity Manager Documentation Web site \(http://www.novell.com/documentation/idm\)](http://www.novell.com/documentation/idm)

For documentation on Identity Manager 2.0, see the [Identity Manager Documentation Web site \(http://www.novell.com/documentation/idm\)](http://www.novell.com/documentation/idm)

## Additional Documentation

For documentation on using the Identity Manager drivers, see the [Identity Manager Documentation Web site \(http://www.novell.com/documentation/idmdrivers/index.html\)](http://www.novell.com/documentation/idmdrivers/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 (<sup>®</sup>, <sup>™</sup>, etc.) denotes a Novell trademark. An asterisk (\*) denotes a third-party trademark.

# Policies and Filters

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

- [Section 1.1, “What Are Policies and Filters?,” on page 13](#)
- [Section 1.2, “Introduction to Policies,” on page 15](#)

## 1.1 What Are Policies and Filters?

At a high level, policies enable you to customize the way Identity Manager 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 the Identity Vault.

This change is contained in an XML document, formatted according to the Identity Manager 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>
```

Drivers are designed to report any relevant changes, then enable you to filter the information. Filters are designed to block information. You modify the filter to allow only the information you desire to enter your environment. 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 isn't very concerned with groups, they can implement a filter to block all operations regarding groups in either the Identity Vault or the connected system. If the company cared about

users and groups, they can implement a filter to allow both types of objects to synchronize between the Identity Vault and the connected system.

Defining filters to allow the synchronization of only objects that are interesting to you is the first step in driver customization.

The next step defines what Identity Manager does with the objects that are allowed by your filter. As an example, 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 allow this operation, Identity Manager needs to decide what to do with this user.

To make this decision, Identity Manager applies a set of policies, in a specific order.

First, a Matching policy 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, because these are usually unique. You can define a policy that says “If two objects have the same e-mail address, they are the same object.”

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

In circumstances where a match is not found, a Creation policy is called on. The Creation policy tells Identity Manager under what conditions you want objects to be created. You can make the existence of certain attributes mandatory in the creation rule. If these attributes do not exist, Identity Manager blocks the creation of the object until the required information is provided.

After the object is created, a Placement policy tells Identity Manager where to put it. You can specify that objects should be created in a hierarchical structure identical to the system they came from, or you can place them somewhere completely different based on an attribute value.

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

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

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

Continue to [Section 1.2, “Introduction to Policies,” on page 15](#) to learn more about the different types of policies, then move on to [Chapter 2, “Defining Policies By Using The Policy Builder With Designer,” on page 37](#) or [Chapter 3, “Defining Policies By Using The Policy Builder In iManager,” on page 183](#) to learn how to use the Policy Builder.

## 1.1.1 Terminology Changes from Earlier Versions

If you have not used DirXML® 1.1a or Identity Manager 2.0, you do not need to review this section.

In DirXML 1.1a, 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 Identity Manager 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, 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 the terminology changes from DirXML 1.1a to Identity Manager 2.x.

**Table 1-1** Terminology Changes from DirXML 1.1a to Identity Manager 2.x

Item being described	DirXML 1.1a Terminology	Identity Manager 2.x Terminology
Set of transformations	Rule	Set of policies
An individual transformation within a set	Rule	Policy
The conditions and actions within an individual transformation	Rule	Rule

The following table shows the terminology changes from Identity Manager 2.x to Identity Manager 3.0.

**Table 1-2** Terminology Changes from Identity Manager 2.x to Identity Manager 3.0

Item being described	Identity Manager 2.x Terminology	Identity Manager 3 Terminology
The product	DirXML	Identity Manager
A server that has the product installed	DirXML server	Metadirectory server
A server in the application or database the data is synchronizing with	DirXML connected system server	Connected system server
Where the objects are stored	eDirectory™	Identity Vault
The processing component	DirXML engine	Metadirectory engine

## 1.1.2 DirXML Script

DirXML<sup>®</sup> Script is the primary method of implementing Identity Manager policies. It describes a policy that is implemented by an ordered set of rules. A rule consists of a set of conditions to be tested and an ordered set of actions to be performed when the conditions are met.

DirXML Script is created using the Policy Builder, which provides a GUI interface for easy of use.

## 1.2 Introduction to Policies

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

- Section 1.2.1, “Policies,” on page 16
- Section 1.2.2, “Defining Policies,” on page 35

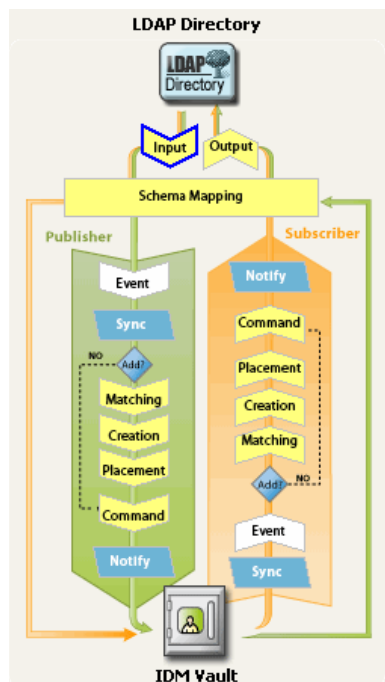
## 1.2.1 Policies

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.

The order of execution of the policies on the channel are:

- Event Transformation Policy
- Creation Policy
- Matching Policies
- Placement Policy
- Command Transformation Policy
- Schema Mapping Policy
- Output Transformation Policy
- Input Transformation Policy

**Figure 1-1** Order of Execution of the Policies



### Event Transformation Policy

Event Transformation policies alter the Metadirectory engine's view of the events that happen in the Identity Vault or the connected application. The most common task performed in an Event Transformation policy is custom filtering, such as scope filtering and event-type filtering.



Scope filtering removes unwanted events based on event location or an attribute value. For example, removing the event if the department attribute is not equal to a specific value or is not a member of a specific group.

Event-type filtering removes unwanted events based on event type. For example, removing all delete events.

Examples:

- Scope Filtering
- Type Filtering

**Scope Filtering:** This example DirXML Script policy allows events through only for users who are contained within the Users subtree, are not disabled, and do not contain the word Consultant or Manager in the Title attribute. It also generates a status document indicating when an operation has been blocked.

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

This DirXML Script policy votes modify operations on User objects except for modifies of objects that are already associated.

```
<policy>
  <rule>
    <description>Veto all operation on User except modifies
of already associated objects</description>
    <conditions>
```

```

        <or>
            <if-class-name op="equal">User</if-class-name>
        </or>
        <or>
            <if-operation op="not-equal">modify</if-
operation>
            <if-association op="not-associated"/>
        </or>
    </conditions>
    <actions>
        <do-veto/>
    </actions>
</rule>
</policy>

```

**Type Filtering** - The first rule of this example DirXML Script policy allows only objects in the Employee and Contractor containers to be synchronized. The second rule blocks all Rename and Move operations.

```

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

```

```

                                <arg-string>
                                    <token-text>Change ignored:
We don't like you to do that.</token-text>
                                </arg-string>
                                </do-status>
                                <do-veto/>
                            </actions>
                    </rule>
</policy>

```

This DirXML Script policy blocks all Add events.

```

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

```

## Creation Policy

Creation policies, such as a Subscriber Creation policy and a Publisher Creation policy, define the conditions that must be met to create a new object. The absence of a Creation policy implies that the object can be created.

For example, you create a new user in the Identity Vault, 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 the Identity Vault because you have a Creation policy specifying that only User objects with a more complete definition are allowed.

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

Template objects can be specified for use in the creation process when the object is to be created in eDirectory.

Creation policies are commonly used to:

- Veto creation of objects that don't qualify, possibly due to a missing attribute.
- Provide default attribute values.
- Provide a default password.

Examples:

- Required Attributes
- Default Attribute Values
- Default Password
- Specify Template

**Required Attributes:** The first rule of this example DirXML Script policy requires that a User object contain a CN, Given Name, Surname, and Internet EMail Address attribute before the user can be created. The second rule requires an OU attribute for all Organizational Unit objects. The final rule vetoes all User objects with a name of Fred.

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

**Default Attribute Values:** This example DirXML Script policy adds a default value for a user's Description attribute.

```
<policy>
  <rule>
    <description>Default Description of New Employee</
description>
    <conditions>
      <or>
```

```

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

```

**Default Password:** This example DirXML Script policy provides creates a password value comprised of the first two characters of the first name and the first six characters of the last name, all in lower case.

```

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

```

**Specify Template:** This example DirXML Script policy specifies a template object if a user's Title attribute indicates that the user is a Manager (contains "Manager").

```

<policy>
    <rule>
        <description>Assign Manager Template if Title
contains Manager</description>
        <conditions>
            <and>

```

```

                                <if-class-name op="equal">User</if-class-
name>
                                <if-op-attr name="Title" op="available"/
>
                                <if-op-attr mode="regex" name="Title"
op="equal">.*Manager.*</if-op-attr>
                                </and>
                                </conditions>
                                <actions>
                                    <do-set-op-template-dn>
                                        <arg-dn>
                                            <token-text>Users\Manager
Template</token-text>
                                        </arg-dn>
                                    </do-set-op-template-dn>
                                </actions>
                            </rule>
</policy>

```

## Matching Policies

Matching policies, such as Subscriber Matching and Publisher Matching, look for an object in the destination data store that corresponds to an unassociated object in the source data store. It is important to note that Matching policies are not always needed or desired.

For example, a Matching policy might not be desired in the following situation:

- Performing an initial migration when there are not preexisting or corresponding objects

A Matching policy must be carefully crafted to ensure that the Matching policy doesn't find false matches.

Examples:

- Match by Internet Email Address
- Match by Common Name

**Match by ID:** This example DirXML Script policy matches users based on the Internet Email Address.

```

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

```

```

        </do-find-matching-object>
    </actions>
</rule>
</policy>

```

**Match by Name:** This example DirXML Script policy matches a Group object based on its Common Name attribute.

```

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

```

## Placement Policy

Placement policies determine where new objects are placed and what they are named in the Identity Vault and the connected application.

A Placement policy is required on the Publisher channel if you want object creations to occur in the Identity Vault. A Placement policy might not be necessary on the Subscriber channel even if you want object creations to occur in the connected application, depending on the nature of the destination datastore. For example, no Placement policy is needed when synchronizing to a relational database because rows in a relational database do not have a location or a name.

Example:

- Placement by Attribute Value
- Placement by Name

**Placement By Attribute Value:** This example DirXML Script policy creates the user in a specific container based on the value of the Department attribute.

```

<policy>
  <rule>
    <description>Department Engineering</description>
    <conditions>
      <and>
        <if-class-name op="equal">User</if-class-name>
        <if-op-attr mode="regex" name="Department"
op="equal">.*Engineering.*</if-op-attr>
      </and>
    </conditions>

```

```

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

```

This DirXML Script policy determines placement of a User or Organization Unit by the src-dn in the input document.

```

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

```



```

                <token-text>\</token-text>
                <token-unmatched-src-dn convert="true"/>
            </arg-dn>
        </do-set-op-dest-dn>
    </actions>
</rule>
</policy>

```

**Placement By Name:** This example DirXML Script policy creates the user in a specific container based on the first letter of the user's last name. Users with a last name beginning with A-I are placed in the container Users1, while J-R are placed in Users2, and S-Z in Users3.

```

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

```

```

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

```

## Command Transformation Policy

Command Transformation policies alter the commands that Identity Manager is sending to the destination datastore by either substituting or adding commands. Intercepting a Delete command and replacing it with Modify, Move, or Disable command is an example of substituting commands in a Command Transformation policy. Creating a Modify command based on the attribute value of an Add command is a common example of adding commands in a Command Transformation policy.

In the most general terms, Command Transformation policies are used to alter the commands that Identity Manager executes as a result of the default processing of events that were submitted to the Metadirectory engine.

It is also common practice to include policies here that do not fit neatly into the descriptions of any other policy.

Examples:

- Convert Delete to Modify and Move
- Create Additional Operation
- Set Password Expiration Time

**Convert Delete to Modify:** This DirXML Script policy converts a Delete operation to a Modify operation of the Login Disabled attribute.

```

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

```

```

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

```

**Create Additional Operation:** This DirXML Script policy determines if the destination container for the user already exists. If the container doesn't exist, the policy creates an Add operation to create the Container object.

```

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

```

```

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

```

**Setting Password Expiration Time:** This DirXML Script policy modifies an eDirectory user's Password Expiration Time attribute.

```

<?xml version="1.0" encoding="UTF-8"?>
<policy xmlns:jssystem="http://www.novell.com/nxsl/java/
java.lang.System">
    <rule>
        <description>Set password expiration time for a given
interval from current day</description>
        <conditions>
            <and>
                <if-operation op="equal">modify-password</if-
operation>
            </and>
        </conditions>
        <actions>
            <do-set-local-variable name="interval">
                <arg-string>
                    <token-text>30</token-text>
                </arg-string>
            </do-set-local-variable>
            <do-set-dest-attr-value class-name="User"
name="Password Expiration Time" when="after">
                <arg-association>
                    <token-association/>
                </arg-association>
                <arg-value type="string">
                    <token-

```

```

xpath expression="round(jsystem:currentTimeMillis() div 1000 +
(86400*$interval))"/>
    </arg-value>
    </do-set-dest-attr-value>
  </actions>
</rule>
</policy>

```

## Schema Mapping Policy

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

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

After a Schema Mapping policy is defined in the Identity Manager driver configuration, the corresponding data can be mapped.

It is important to note the following:

- The same policies are applied in both directions.
- All documents that are passed in either direction on either channel between the Metadirectory engine and the application shim are passed through the Schema Mapping policies.

See [Chapter 6, “Managing Schema Mapping Policies,” on page 327](#) for administrative information.

Examples:

- Basic Schema Mapping policy
- Custom Schema Mapping policy

**Basic Schema Mapping Policy:** This example DirXML Script policy shows the raw XML source of a basic Schema Mapping policy. However when you edit the policy through Designer for Identity Manager, the default Schema Mapping editor allows the policy to be displayed and edited graphically.

```

<?xml version="1.0" encoding="UTF-8"?><attr-name-map>
  <class-name>
    <app-name>WorkOrder</app-name>
    <nds-name>DirXML-nwoWorkOrder</nds-name>
  </class-name>
  <class-name>
    <app-name>PbxSite</app-name>
    <nds-name>DirXML-pbxSite</nds-name>
  </class-name>
  <attr-name class-name="DirXML-pbxSite">
    <app-name>PBXName</app-name>
    <nds-name>DirXML-pbxName</nds-name>
  </attr-name>
  <attr-name class-name="DirXML-pbxSite">
    <app-name>TelephoneNumber</app-name>
    <nds-name>Telephone Number</nds-name>
  </attr-name>

```

```

<attr-name class-name="DirXML-pbxSite">
    <app-name>LoginName</app-name>
    <nds-name>DirXML-pbxLoginName</nds-name>
</attr-name>
<attr-name class-name="DirXML-pbxSite">
    <app-name>Password</app-name>
    <nds-name>DirXML-pbxPassword</nds-name>
</attr-name>
<attr-name class-name="DirXML-pbxSite">
    <app-name>Nodes</app-name>
    <nds-name>DirXML-pbxNodesNew</nds-name>
</attr-name>
</attr-name-map>

```

**Custom Schema Mapping Policy:** This example DirXML Script policy uses DirXML Script to perform custom Schema Mapping.

```

<?xml version="1.0" encoding="UTF-8"?><policy>
  <rule>
    <!--
      The Schema Mapping Policy can only handle one-to-one
mappings.
      That Mapping Policy maps StudentPersonal addresses.
      This rule maps StaffPersonal addresses.
    -->
    <description>Publisher Staff Address Mappings</
description>
    <conditions>
      <and>
        <if-local-variable name="fromNds"
op="equal">false</if-local-variable>
        <if-xpath op="true">@original-class-name =
'StaffPersonal'</if-xpath>
      </and>
    </conditions>
    <actions>
      <do-rename-op-attr dest-name="SA" src-name="Address/
Street/Line1"/>
      <do-rename-op-attr dest-name="Postal Office Box"
src-name="Address/Street/Line2"/>
      <do-rename-op-attr dest-name="Physical Delivery
Office Name" src-name="Address/City"/>
      <do-rename-op-attr dest-name="S" src-name="Address/
StatePr"/>
      <do-rename-op-attr dest-name="Postal Code" src-
name="Address/PostalCode"/>
    </actions>
  </rule>
  <rule>
    <description>Subscriber Staff Address Mappings</
description>
    <!--
      The Schema Mapping Policy has already mapped addresses to
StudentPersonal.

```

This rule maps StudentPersonal to StaffPersonal.

```
-->
  <conditions>
    <and>
      <if-local-variable name="fromNds"
op="equal">true</if-local-variable>
      <if-op-attr name="DirXML-sifIsStaff"
op="equal">true</if-op-attr>
    </and>
  </conditions>
  <actions>
    <do-rename-op-attr dest-name="Address/Street/Line1"
src-name="StudentAddress/Address/Street/Line1"/>
    <do-rename-op-attr dest-name="Address/Street/Line2"
src-name="StudentAddress/Address/Street/Line2"/>
    <do-rename-op-attr dest-name="Address/City" src-
name="StudentAddress/Address/City"/>
    <do-rename-op-attr dest-name="Address/StatePr" src-
name="StudentAddress/Address/StatePr"/>
    <do-rename-op-attr dest-name="Address/PostalCode"
src-name="StudentAddress/Address/PostalCode"/>
  </actions>
</rule>
</policy>
```

## Output Transformation Policy

Output Transformation policies primarily handle the conversion of data formats from data that the Metadirectory engine provides to data that the application shim expects. Examples of these conversions include:

- Attribute value format conversion
- XML vocabulary conversion
- Output Transformation policies can also provide custom handling of status messages returned from the Metadirectory engine to the application shim

All documents that the Metadirectory engine supplies to the application shim on either channel pass through the Output Transformation policies. Since the Output Transformation happens after schema mapping, all schema names are in the application namespace.

Examples:

- Attribute Value Format Conversion
- Custom Handling of Status Messages

**Attribute Value Conversion:** This example DirXML Script policy reformats the telephone number from the (nnn) nnn-nnnn format to the nnn.nnn.nnnn format. The reverse transformation can be found in the Input Transformation policy examples.

```
<policy>
  <rule>
    <description>Reformat all telephone numbers from (nnn)
nnn-nnnn to nnn.nnn.nnnn</description>
    <conditions/>
```

```

        <actions>
            <do-reformat-op-attr name="telephoneNumber">
                <arg-value type="string">
                    <token-replace-first
regex="^\((\d\d\d)\) *(\d\d\d)-(\d\d\d\d)$" replace-with="$1.$2.$3">
                        <token-local-
variable name="current-value"/>
                    </token-replace-first>
                </arg-value>
            </do-reformat-op-attr>
        </actions>
    </rule>
</policy>

```

**Custom Handling of Status Messages:** This example DirXML Script policy detects status documents with a level not equal to success that also contain a child password-publish-status element within the operation data and then generate an e-mail message using the DoSendEmailFromTemplate action.

```

<?xml version="1.0" encoding="UTF-8"?>
    <policy>
        <description>Email notifications for failed password
publications</description>
        <rule>
            <description>Send e-mail for a failed publish
password operation</description>
            <conditions>
                <and>
                    <if-global-variable
mode="nocase" name="notify-user-on-password-dist-failure"
op="equal">true</if-global-variable>
                    <if-operation
op="equal">status</if-operation>
                    <if-xpath
op="true">self::status[@level != 'success']/operation-data/password-
publish-status</if-xpath>
                </and>
            </conditions>
            <actions>
                <!-- generate email notification -->
                <do-send-email-from-template notification-
dn="\cn=security\cn=Default Notification Collection" template-
dn="\cn=security\cn=Default Notification Collection\cn>Password Sync
Fail">
                    <arg-string name="UserFullName">
                        <token-src-attr name="Full Name">
                            <arg-association>
                                <token-xpath
expression="self::status/operation-data/password-publish-status/
association"/>
                            </arg-association>
                        </token-src-attr>
                    </arg-string>
                    <arg-string name="UserGivenName">

```



```

        <token-src-attr name="Given Name">
            <arg-association>
                <token-xpath
expression="self::status/operation-data/password-publish-status/
association"/>
            </arg-association>
        </token-src-attr>
    </arg-string>
    <arg-string name="UserLastName">
        <token-src-attr name="Surname">
            <arg-association>
                <token-xpath
expression="self::status/operation-data/password-publish-status/
association"/>
            </arg-association>
        </token-src-attr>
    </arg-string>
    <arg-string name="ConnectedSystemName">
        <token-global-variable
name="ConnectedSystemName"/>
    </arg-string>
    <arg-string name="to">
        <token-src-attr name="Internet Email
Address">
            <arg-association>
                <token-xpath
expression="self::status/operation-data/password-publish-status/
association"/>
            </arg-association>
        </token-src-attr>
    </arg-string>
    <arg-string name="FailureReason">
        <token-text/>
        <token-xpath
expression="self::status/child::text()"/>
    </arg-string>
</do-send-email-from-template>
</actions>
</rule>
</policy>

```

## Input Transformation Policy

Input Transformation policies primarily handle the conversion of data formats from data that the application shim provides to data that the Metadirectory engine expects. Examples of these conversions include:

- Attribute value format conversion
- XML vocabulary conversion
- Driver Heartbeat
- Input Transformation policies can also provide custom handling of status messages returned from the application shim to the Metadirectory engine.

All documents supplied to the Metadirectory engine by the application shim on either channel pass through the Input Transformation policies.

Examples:

- Attribute Value Format Conversion
- Driver Heartbeat

**Attribute Value Format Conversion:** This example DirXML Script policy reformats the telephone number from the nnn.nnn.nnnn format to the (nnn) nnn-nnnn format. The reverse transformation can be found in the Output Transformation policy examples.

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

**Driver Heartbeat:** This DirXML Script policy creates a status heartbeat event. The driver's heartbeat functionality is used to send a success message (HEARTBEAT: \$driver) at each heartbeat interval. The message can be monitored by Novell Audit. The Identity Manager driver must support heartbeat, and heartbeat must be enabled at the driver configuration page.

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

```

        <do-set-xml-attr expression="." name="text2">
            <arg-string>
                <token-text>HEARTBEAT</token-text>
            </arg-string>
        </do-set-xml-attr>
    </actions>
</rule>
</policy>

```

## 1.2.2 Defining Policies

All 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.

Schema Mapping policies can also be defined (and usually are) using a schema mapping table.

### 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 By Using The Policy Builder With Designer,” on page 37](#) and [Chapter 3, “Defining Policies By Using The Policy Builder In iManager,” on page 183](#), for more information on Policy Builder. See [Section 1.1.2, “DirXML Script,” on page 15](#) for more information on DirXML Script.

---

**TIP:** Although it is not necessary for using Policy Builder, a complete DirXML Script reference is available at the [DirXML Driver Developer Kit Documentation \(http://developer.novell.com/ndk/doc/dirxml/dirxmlbk/ref/index.html\)](http://developer.novell.com/ndk/doc/dirxml/dirxmlbk/ref/index.html) Web site.

---

### 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 thorough understanding of XSLT, the nds.dtd, and the commands and events transferred to and from the Metadirectory engine. For detailed nds.dtd

reference, see the [NDS DTD reference \(http://developer.novell.com/ndk/doc/dirxml/dirxmlbk/ref/ndsdttd/index.html\)](http://developer.novell.com/ndk/doc/dirxml/dirxmlbk/ref/ndsdttd/index.html).

See [Chapter 4, “Defining Policies using XSLT Style Sheets,” on page 295](#) for more information on XSLT style sheets.

## Downloadable Identity Manager Policies

Novell has provided sample policies you can download and use in your environment. The policies are available at [Novell’s support Web site \(http://support.novell.com/filefinder/20607/index.html\)](http://support.novell.com/filefinder/20607/index.html). Download the files and extract them. The How\_To\_Install.rtf files contain the installation instructions.

To import the files using Designer, see [“Importing a Policy From an XML File” on page 50](#). To import the files using iManager, see [Section 3.2.9, “Importing a Policy from an XML File,” on page 195](#).

## 1.3 Filters

Filters specify the object classes and the attributes for which the Metadirectory engine processes events and how changes to those classes and attributes are handled.

Filters only pass events occurring on objects whose base class matches one of those classes specified by the filter. 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. There are separate filter settings for each channel, which allows the control of the synchronization direction and the authoritative data source for each class and attribute.

---

**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 or the auxiliary classes from which additional attributes may come.

---

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

Filters must be configured to include the following:

- Attributes that are to be synchronized
- Attributes that are not synchronized, but are used to trigger policies to do something

See [Chapter 5, “Managing Filters,” on page 309](#) for information on defining filters.

# Defining Policies By Using The Policy Builder With Designer

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

This section gives the following information on policies and how to use the Policy Builder:

- [Section 2.1, “Policies,” on page 37](#)
- [Section 2.2, “Policy Builder Tasks in Designer,” on page 38](#)

This section also contains the following detailed reference sections:

- [Section 2.3, “Regular Expressions,” on page 101](#)
- [Section 2.4, “XPath 1.0 Expressions,” on page 101](#)
- [Section 2.5, “Conditions,” on page 102](#)
- [Section 2.6, “Actions,” on page 119](#)
- [Section 2.7, “Noun Tokens,” on page 159](#)
- [Section 2.8, “Verb Tokens,” on page 173](#)

## 2.1 Policies

As part of understanding how policies work, it is important to understand the components of policies.

- Policies are made up of rules.
- A rule is a set of conditions (see [“Conditions” on page 102](#)) that must be met before a defined action (see [“Actions” on page 119](#)) occurs.
- Actions can have dynamic arguments that derive from tokens that are expanded at run time.
- Tokens are broken up into two classifications: nouns (see [“Noun Tokens” on page 159](#)) and verbs (see [“Verb Tokens” on page 173](#)).
  - Noun tokens expand to values that are derived from the current operation, the source or destination data stores, or some external source.
  - Verb tokens modify the concatenated results of other tokens that are subordinate to them.
- Regular expressions (see [“Regular Expressions” on page 101](#)) and XPath 1.0 expressions (see [“XPath 1.0 Expressions” on page 101](#)) are commonly used in the rules to create the desired results for the policies.

A policy operates on an XDS document and its primary purpose is to examine and modify that document. A policy can also get additional context from outside of the document and cause side effects that are not reflected in the result document.

The following outline describes the different elements of a policy:

- [Section 2.1.1, “Operation,” on page 38](#)

- [Section 2.1.2, “Current Operation,” on page 38](#)
- [Section 2.1.3, “Current Object,” on page 38](#)

## 2.1.1 Operation

An operation is any element that is a child of the input element and the output element. The elements are part of Novell’s `nds.dtd`, for more information, see [NDS DTD \(http://developer.novell.com/ndk/doc/dirxml/dirxmlbk/ref/ndsstd/index.html\)](http://developer.novell.com/ndk/doc/dirxml/dirxmlbk/ref/ndsstd/index.html). An operation usually represents an event, a command, or a status.

## 2.1.2 Current Operation

The policy is applied separately to each operation. As the policy is applied to each operation in turn, that operation becomes the current operation. Each rule is applied sequentially to the current operation. All of the rules are applied to the current operation unless an action is executed by a prior rule that causes subsequent rules to no longer be applied.

## 2.1.3 Current Object

The object that is described by the `src-dn`, `src-entry-id`, `dest-dn`, `dest-entry-id` and `association` becomes the current object. For more information about the different elements, see the [NDS DTD \(http://developer.novell.com/ndk/doc/dirxml/dirxmlbk/ref/ndsstd/index.html\)](http://developer.novell.com/ndk/doc/dirxml/dirxmlbk/ref/ndsstd/index.html).

# 2.2 Policy Builder Tasks in Designer

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

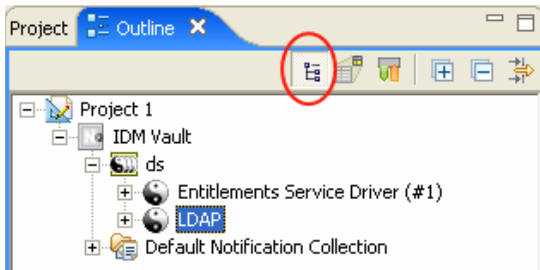
- [Section 2.2.1, “Opening Policy Builder,” on page 38](#)
- [Section 2.2.2, “Creating a Policy,” on page 40](#)
- [Section 2.2.3, “Creating a Rule,” on page 44](#)
- [Section 2.2.4, “Creating an Argument,” on page 50](#)
- [Section 2.2.5, “Modifying a Policy,” on page 58](#)
- [Section 2.2.6, “Using Predefined Rules,” on page 61](#)
- [Section 2.2.7, “Testing Policies with the Policy Simulator,” on page 90](#)
- [Section 2.2.8, “Editing the DirXML Script,” on page 97](#)

## 2.2.1 Opening Policy Builder

There are two different ways the Policy Builder can be opened. It can be opened from the Model Outline view or from the Policy Flow view.

### Model Outline View

- 1 Open a project in Designer.
- 2 Select the Outline Tab > select the Show Model Outline icon.



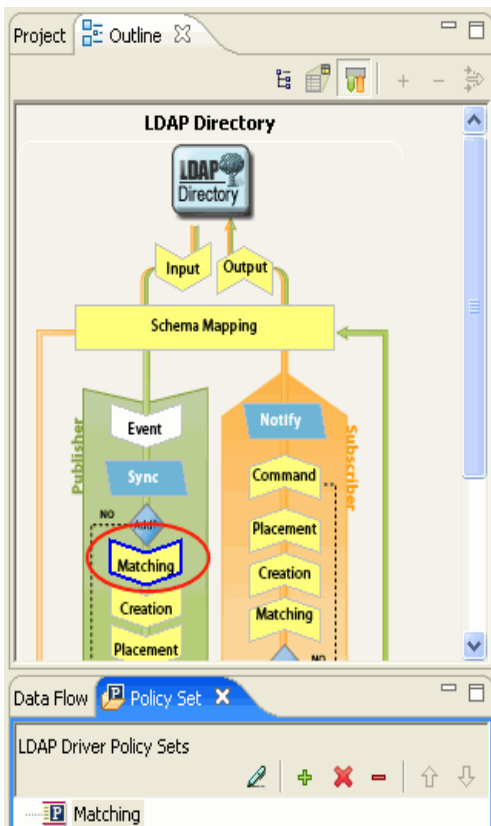
3 Double-click a policy listed in the Model Outline view.

### Policy Flow View

- 1 Open a project in Designer.
- 2 Select the Outline Tab > select the Policy Flow icon.



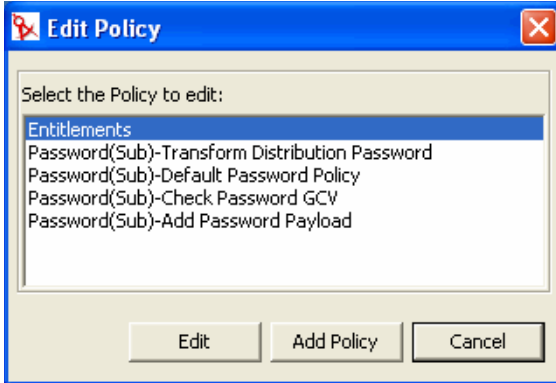
3 Click on a policy (for example, the Command policy) in the Policy Flow view.



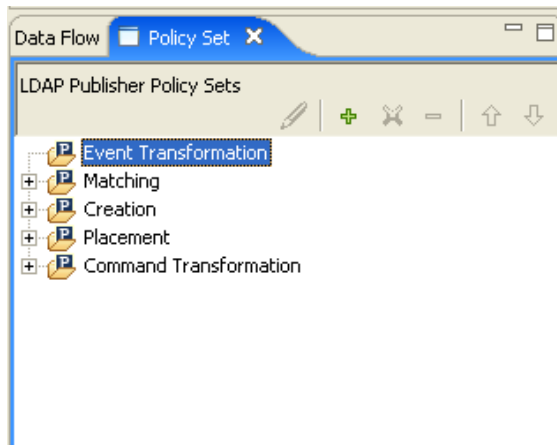
The policy is listed in the Policy Set in the lower-left corner.

Use any of the following methods to launch the Policy Builder:

- Double click the Command Policy object in the Policy Flow, select the policy, then click *Edit*.



- Double-click a policy in the Policy Set.



- Right-click the policy in the Policy Set, then *Edit*.
- Select the policy in the Policy Set, then click the *Edit the Policy* icon.

To see all of the information in the Policy Builder window, without scrolling double-click the policy tab so the Policy Builder fills the entire window. To minimize the window, double-click the policy tab.

## 2.2.2 Creating a Policy

Policies are created in through the following two tools:

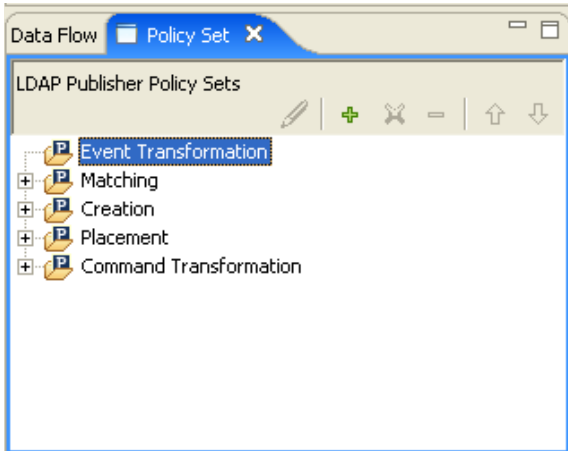
- “Policy Set” on page 40
- “Add Policy Wizard” on page 42

### Policy Set

The Policy Set contains a toolbar and a list of policies.

The policy list displays all the policies contained in the selected policy set. During a transformation, the policies within the list are executed from top to bottom. The toolbar contains buttons and a drop-down menu that you can use to manage policies displayed in the list. Managing of policies includes editing, adding, deleting, renaming, and changing the processing order of the policies.











## Policy Set Toolbar

The Policy Set displays a copy of the policy. The buttons on the toolbar are enabled or disabled depending upon what item you have selected. The different icons are described below.

**Table 2-1** Policy Set Toolbar

Operation	Description
Edit a policy 	Launches Policy Builder.
Add a policy 	Launches the Add Policy Wizard.
Delete a policy 	Deletes the policy from the project.
Remove a policy 	Removes the policy from the selected Policy Set object but doesn't delete the policy.
Move up 	Moves the policy up in the processing order.
Move down 	Moves the policy down in the processing order.

## Keyboard Support

You can move through the Policy Set with keystrokes as well as use the mouse. Below is listed the supported keystrokes.

**Table 2-2** Keyboard Support

Keystroke	Description
Up-Arrow	Moves the selected policy up in the processing order.
Down-Arrow	Moves the selected policy down in the processing order.

Keystroke	Description
Delete	Deletes the policy from the project.
Minus	Removes the policy from the selected policy set, but does not delete it.
Plus	Launches the Add Policy Wizard.
Ctrl+Z	Undoes the last operation.
Ctrl+Y	Redoes the last operation.

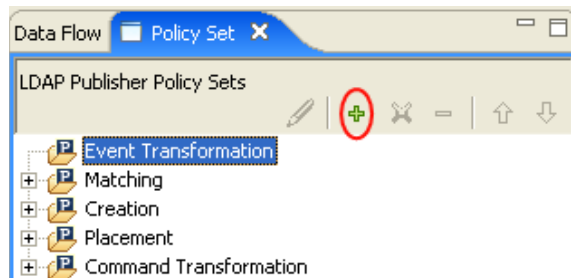
## Add Policy Wizard

The Add Policy Wizard enables you to do the following tasks:

- “Creating a Policy” on page 42
- “Copying a Policy” on page 43
- “Linking to a Policy” on page 43

To launch the Add Policy Wizard:

- 1 Select a driver in the Outline view.
- 2 Select a Set item in the Policy Set, then *click the Add icon* in the toolbar.



### Creating a Policy

- 1 In the Add Policy Wizard, select *Create a new policy*, then click *Next*.
- 2 Provide a policy name.
- 3 Accept the default container, or browse to and select the Driver, Publisher, or Subscriber object where you want the policy to be created.

This decision depends on how you want to organize the policies. By default, policies are placed under the container object that is selected in the Outline view when the Add Policy Wizard is launched.

For example, if you move to a Publisher object in the Outline view and then add a policy to a policy set, the policy defaults to the Publisher container.

You can change this setting if you want to create policies in a different container. For example, you can set up a policy library under a dummy driver, put all of the common policies under this driver, and then simply reference the policies from the other drivers. That way, the policy is common. If you need to change a policy, you need to do it only once.

If a policy is not reused by multiple drivers, you typically create that policy under the driver or channel that is using it.

- 4 Select the type of policy you want to implement. The policy type defaults to *DirXML Script*. You can select *XSLT* or *Schema Mapping*, if you don't want to use DirXML Script.
- 5 Click *Finish*.

If the Schema Mapping policy set is selected, then an additional option is available for Schema Mapping. The new policy appears in the expanded Set item.

You can also add a policy by right-clicking a Set item in the Policy Set.

- 1 Right-click a Set item (for example, Input Transformation Set).
- 2 Select *Add Policy*.
- 3 Select how to implement the policy.
- 4 Name the policy.
- 5 Select *Open Editor after creating policy*.
- 6 Click *OK*.

### Copying a Policy

- 1 In the Add Policy Wizard, select *Copy a policy*, then click *Next*.
- 2 Name the policy.
- 3 Accept the default container, or browse to and select the Driver, Publisher or Subscriber object where you want the policy to be created.
- 4 Browse to and select the policy you want to copy, then click *OK*.
- 5 Click *Finish* to make a copy of the selected policy.

### Linking to a Policy

- 1 In the Add Policy Wizard, select *Link in a policy*, then click *Next*.
- 2 Launch the Modeler browser by selecting the Browse button.
- 3 Browse to and select the Policy object you want to link into the policy set, then click *OK*.

Linking a policy into a policy set doesn't create a new Policy object. Instead, it adds a reference to an existing policy. This reference can be to any existing policy within the current Identity Vault. It doesn't need to be contained within the current Driver object, but the policy type must be valid for the policy set that it is being linked to. For example, you can't link a Schema Mapping policy into an Input Policy Set.

---

**NOTE:** Linking a policy into a policy set is not permitted when viewing all policies.

---

- 4 Click *Finish* to link to the selected policy.

### Renaming a Policy

- 1 In the Outline View, select the policy you want to rename.
- 2 Right-click and select *Properties*.
- 3 Change the name of the policy in the *Policy Name* field.
- 4 Click *OK*.

## 2.2.3 Creating a Rule

Rules are created from condition groups, conditions, and actions. A rule is defined as a set of conditions that must be met before a defined action occurs.

Rules can be created in three different ways:

- “Creating a New Rule” on page 44
- “Using Predefined Rules” on page 48
- “Including a Rule” on page 48

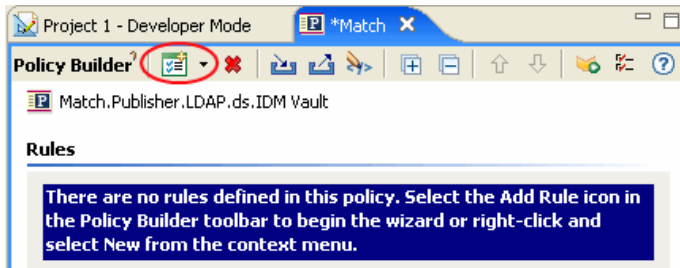
### Creating a New Rule

When you create a rule, you create condition groups, conditions, and actions.

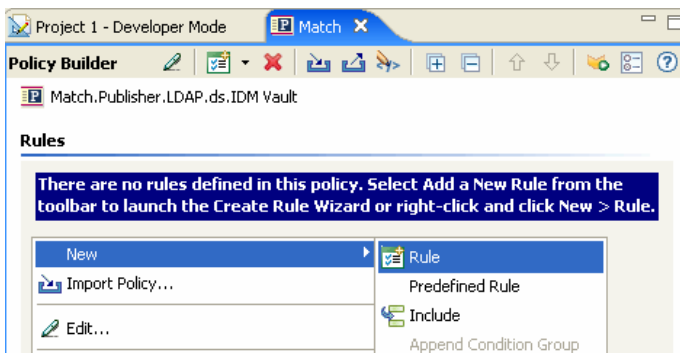
- “Creating a Rule” on page 44
- “Creating Condition Groups” on page 47
- “Creating a Condition” on page 47
- “Creating an Action” on page 48

### Creating a Rule

- 1 From the Policy Builder toolbar, select *Add a new rule*.



You can also right-click and click *New > Rule*.

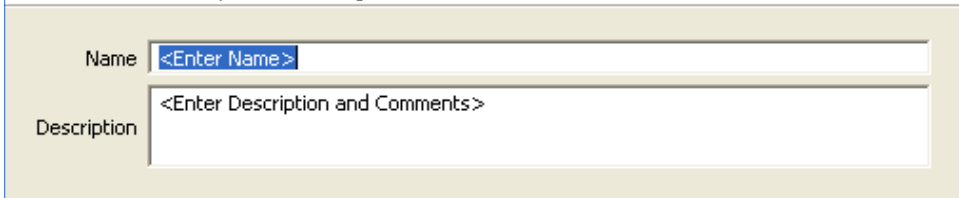


Either option launches the Create Rule Wizard.

- 2 Specify the name of the rule, then click *Next*.

### Name and Describe Rule

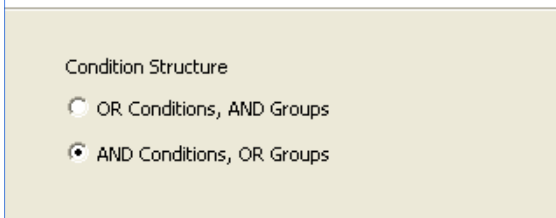
The rule name and description display on the rule in the Rule Builder editor. Both can be edited by double-clicking the rule name in Rule Builder.



**3** Select the *Condition Structure*, then click *Next*.

### Select the Condition Structure

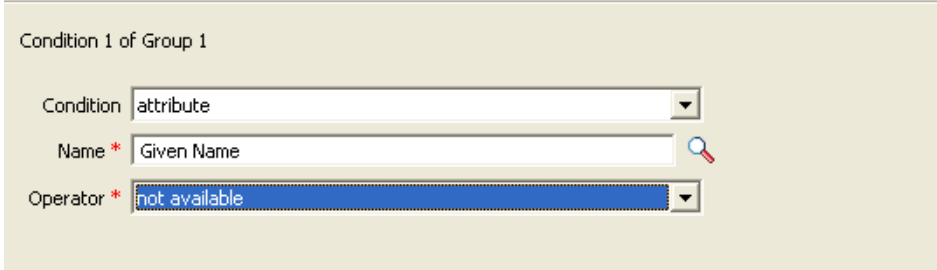
Condition structures define the logic of condition groups.



**4** Select the condition you want, specify the appropriate information, then click *Next*.

### Define the Condition

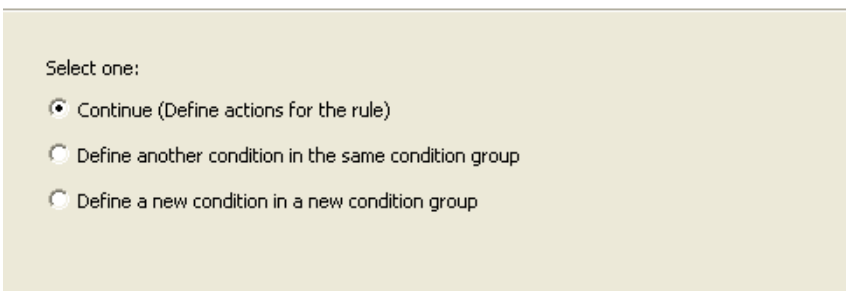
Select the values to complete the syntax of the condition. Values with an \* are required for a valid condition. The first condition is automatically inserted into a new condition group.



**5** (Optional) You can define an additional condition or condition group at this point. For this example, there is only one condition.

### Continue Defining Conditions?

Select whether to continue defining your condition or proceed to defining actions for your rule.



**6** When you are finished defining conditions, select *Continue*, then click *Next*.

7 Select the action that you want, then click *Next*.

**Define the Action**

Select the values for the syntax of your action. Values with an \* are required.

Action 1

Do

8 (Optional) You can define additional actions at this point. For this example, there is only one action.

**Continue Defining Actions?**

Select whether to define a new action or select finish to complete the rule.

Select one;

Continue (Go to Summary Page)

Define another action

9 When you are finished defining actions, select *Continue*, then click *Next*.

10 The summary screen displays the rule that you are creating. Click *Finish* to complete the creation of the rule.

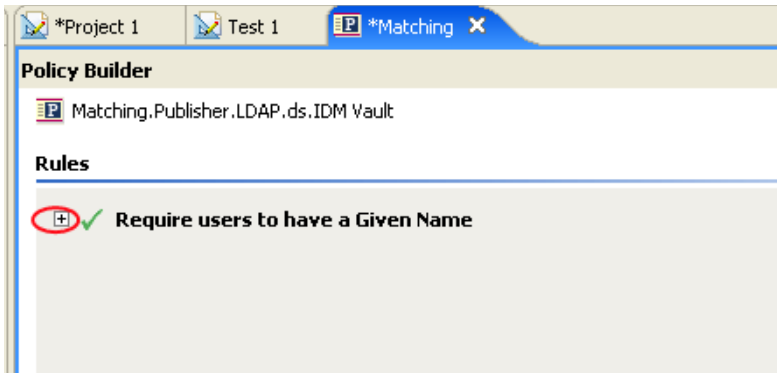
**Summary**

The following is a summary of the new rule to be created.

**Rule Summary**

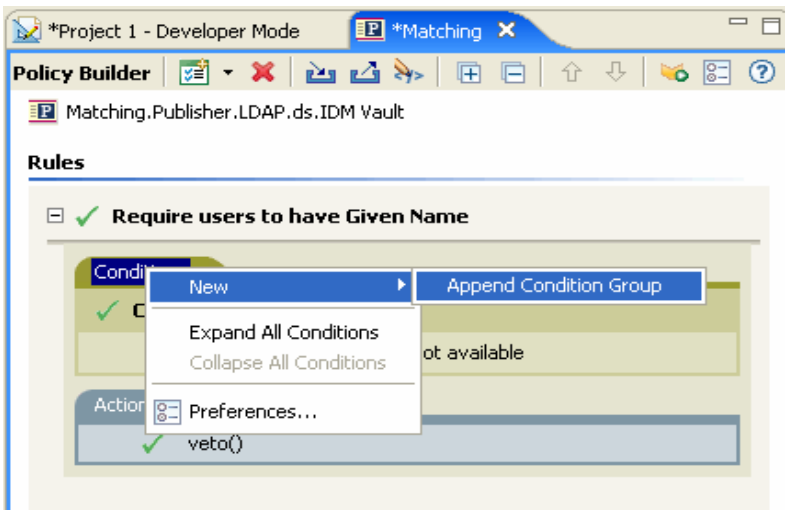
- [-] Given Name
  - [-] Conditions
    - [-] Group 1
      - if attribute 'Given Name' not available
  - [-] Actions
    - veto()

You can expand or collapse the view of the rule by clicking on the plus/minus sign.



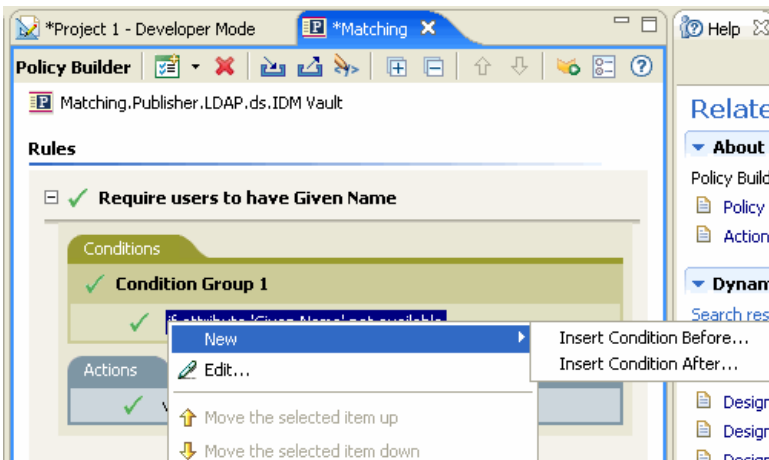
## Creating Condition Groups

Right-click the *Conditions* tab or right-click the name of the Conditional group, then click *New > Append Condition Group*.



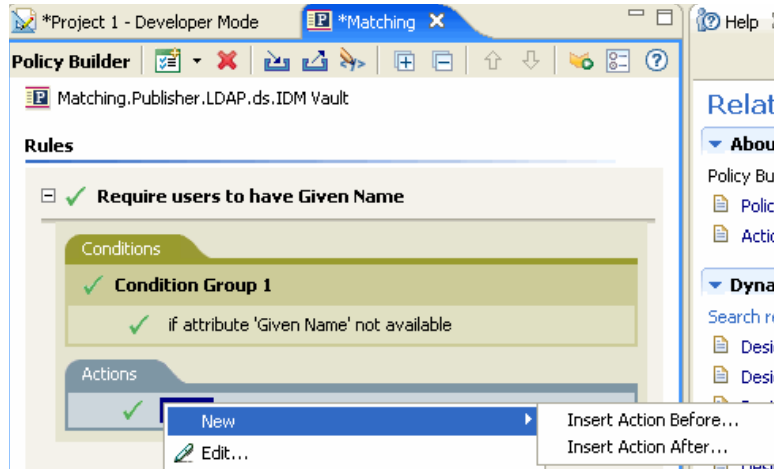
## Creating a Condition

Right-click the condition, then click *Insert Condition Before* or *Insert Condition After*.



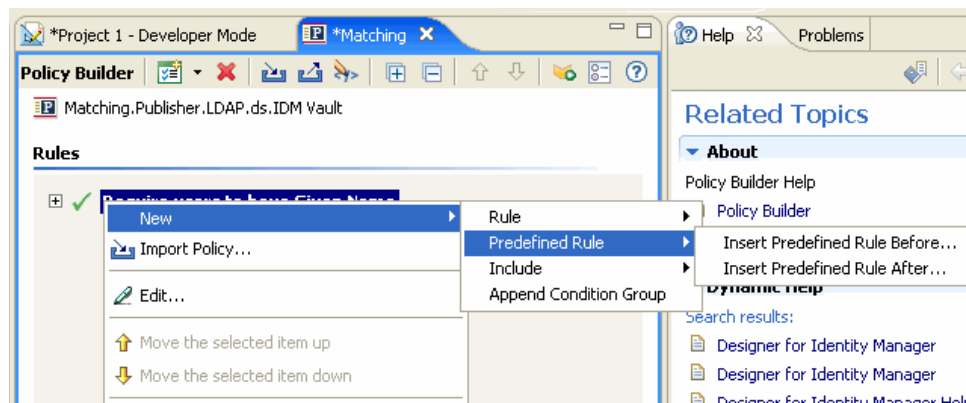
## Creating an Action

Right-click the action, then click *Insert Action Before* or *Insert Action After*.



## Using Predefined Rules

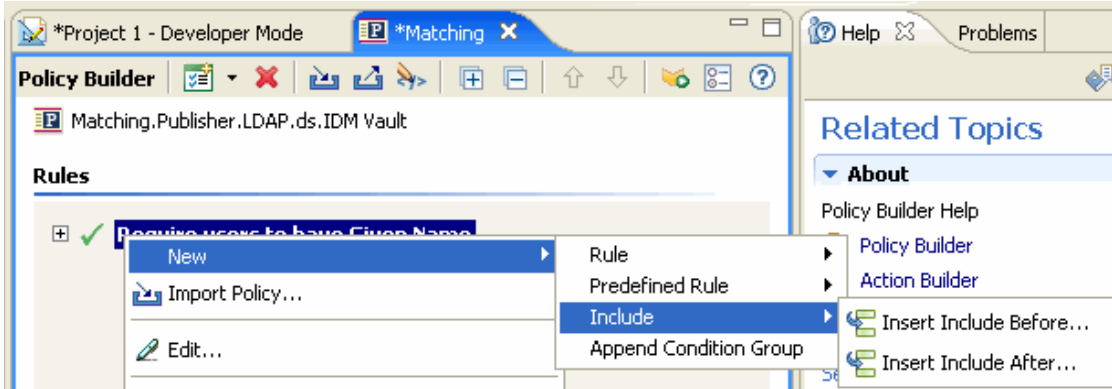
Designer includes some predefined rules. You can import and use these rules as well as create your own rules. Right-click in the Policy Builder, then select *New > Predefined Rules > Insert Predefined Rule Before* or *Insert Predefined Rule After* see [Using Predefined Rules \(page 61\)](#) for more information.



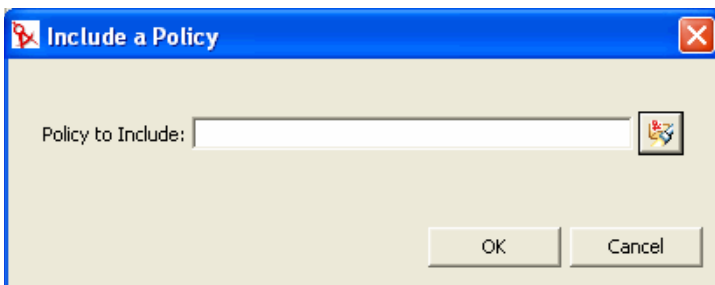
## Including a Rule

Designer allows you to include the rules from another policy.

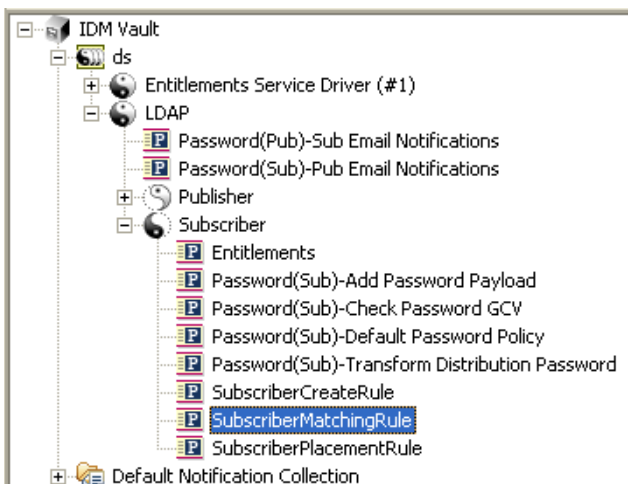




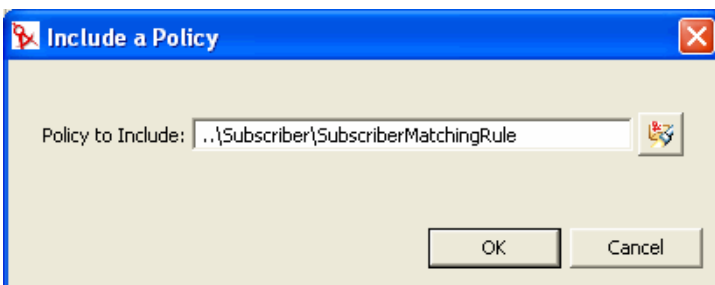
1 Right-click, then click *New > Include*.



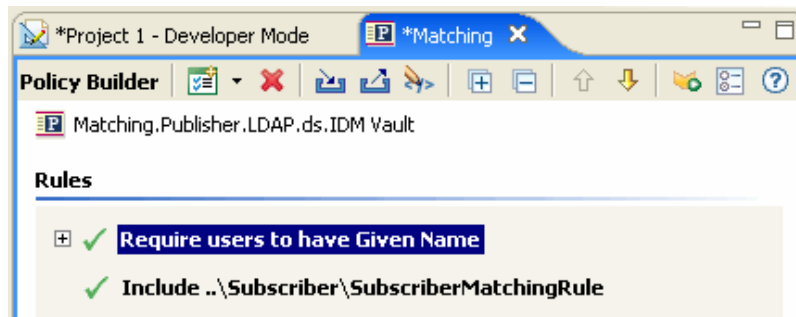
2 Select the browse icon, browse to the policy you want to include, then click *OK*.



3 The field is now populated with the path to the policy. Click *OK*.



There is now a link to the original policy. You cannot edit the policy in this location. Access the original policy to make changes.



## Importing a Policy From an XML File

- 1 In the Policy Builder, right-click and select Import Policy or select the Import Policy icon from the toolbar.
- 2 Select one of the two options:
  - *Append the rules from the imported policy*
  - *Replace the rules from the imported policy*
- 3 Click the browse icon and select the file that contains the DirXML<sup>®</sup> Script, then click *Open*.
- 4 Click *OK*.

## 2.2.4 Creating an Argument

The Argument Builder provides a dynamic graphical interface that enables you to construct complex argument expressions for use within the Policy Builder. To access the Argument Builder, see [“Argument Builder” on page 52](#).

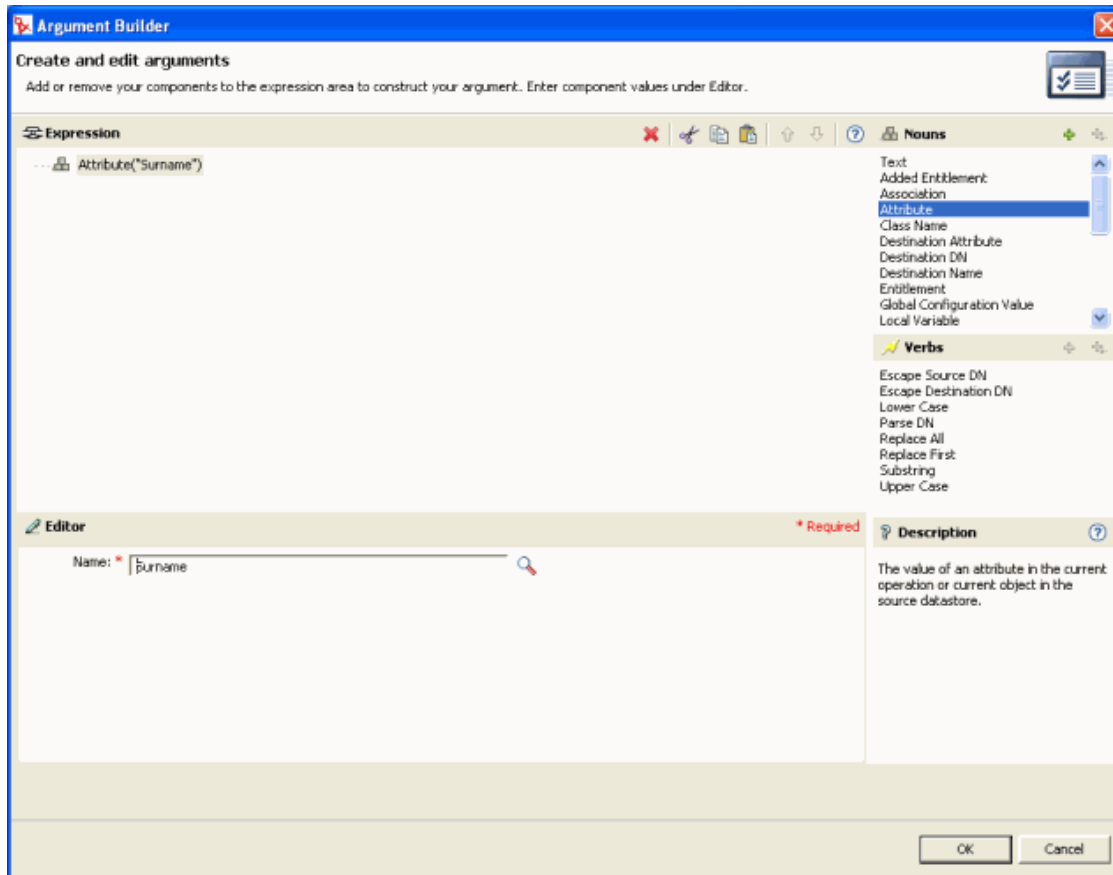
Arguments are dynamically used by actions and are derived from tokens that are expanded at run time.

Tokens are broken up into two classifications: nouns and verbs. Noun tokens expand to values that are derived from the current operation, the source or destination data stores, or some external source. Verb tokens modify the concatenated results of other tokens that are subordinate to them.

To define an expression, select one or more nouns tokens (values, objects, variables, etc.), and combine them with verb tokens (substring, escape, uppercase, and lowercase) to construct arguments. Multiple tokens are combined to construct complex arguments.

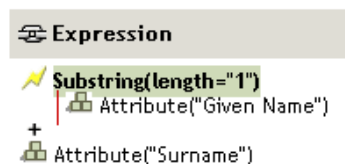
For example, if you want the argument set to an attribute value, you select the attribute token, then select the attribute name:

Figure 2-1 Argument Builder



If you only want a portion of an attribute, you can combine the attribute token with the substring token:

Figure 2-2 Expression



After you add a token, you can edit its fields in the editor.

See [Noun Tokens \(page 159\)](#) and [Verb Tokens \(page 173\)](#) for a detailed reference on tokens available in the Argument Builder.

Although you define most arguments using the Argument Builder, there are several more builders that are used by the Condition Editor and Action Editor in the Policy Builder. Each builder can recursively call anyone of the builders in the following list:

- “Argument Actions Builder” on page 52
- “Argument Builder” on page 52
- “Match Attribute Builder” on page 53

- “Action Argument Component Builder” on page 54
- “Argument Value List Builder” on page 55
- “Named String Builder” on page 56
- “Condition Argument Component Builder” on page 56
- “Pattern String Builder” on page 57

The information below describes how to access each Builder.

## Argument Actions Builder

The Argument Actions Builder enables you to set or edit the actions subordinate to a **For Each** (page 132) action or an **Implement Entitlement** (page 135) action.

In the following example the add destination attribute value action is performed for each Group entitlement that is being added in the current operation.

**Figure 2-3** For Each Action

To define the action of the add destination attribute value, click the icon that launches the Argument Actions Builder. In the Argument Actions Builder, you define the desired action. In the following example, the member attribute is added to the destination object for each added Group entitlement.

**Figure 2-4** Argument Action Builder

## Argument Builder

Launch the Argument Builder from the following actions by clicking the Edit Arguments icon.

- “Add Association” on page 120
- “Add Destination Attribute Value” on page 120

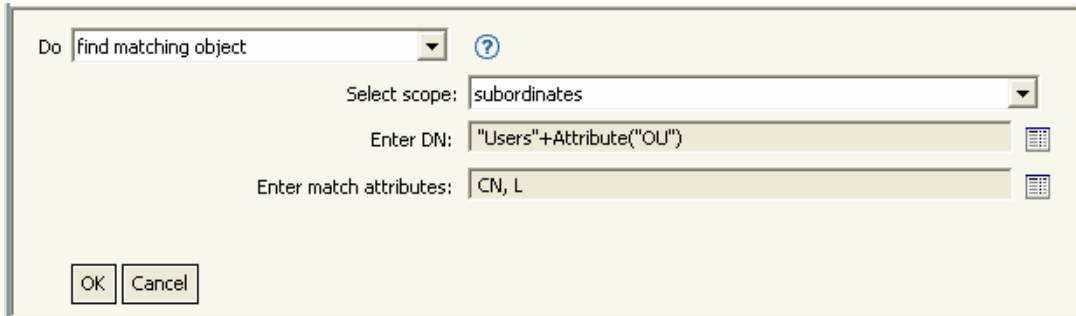
- “Add Destination Object” on page 122
- “Add Source Attribute Value” on page 123
- “Append XML Text” on page 125
- “Clear Destination Attribute Value” on page 126 When the selected object is DN or Association.
- “Clear Source Attribute Value” on page 127 When the selected object is DN or Association.
- “Delete Destination Object” on page 129 When the selected object is DN or Association.
- “Delete Source Object” on page 130 When the select object is DN or Association.
- “Find Matching Object” on page 130
- “For Each” on page 132
- “Move Destination Object” on page 136
- “Move Source Object” on page 137
- “Reformat Operation Attribute” on page 138
- “Remove Association” on page 139
- “Remove Destination Attribute Value” on page 139
- “Remove Source Attribute Value” on page 140
- “Rename Destination Object” on page 141 When the selected object is DN or Association and Enter String.
- “Rename Source Object” on page 142 When the selected object is DN or Association and Enter String.
- “Set Destination Attribute Value” on page 146 When the selected object is DN or Association and Enter Value Type is not structured.
- “Set Destination Password” on page 148
- “Set Local Variable” on page 148
- “Set Operation Association” on page 149
- “Set Operation Class Name” on page 150
- “Set Operation Destination DN” on page 150
- “Set Operation Property” on page 151
- “Set Operation Source DN” on page 151
- “Set Operation Template DN” on page 152
- “Set Source Attribute Value” on page 153
- “Set Source Password” on page 154
- “Set XML Attribute” on page 154
- “Status” on page 155
- “Trace Message” on page 157


### Match Attribute Builder

The Match Attribute Builder enables you to select attributes and values used by the “[Find Matching Object](#)” on page 130 action to determine if a matching object exists in a data store.

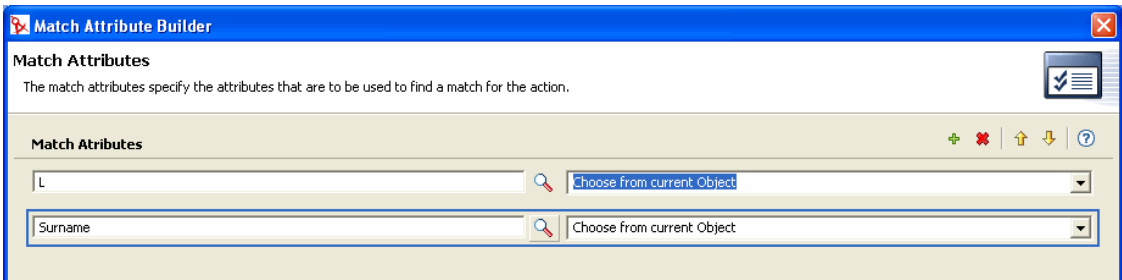
For example, if you wanted to match users based on a common name and a location, you would select the following condition:

**Figure 2-5** Find Matching Object Action



You then click the Edit Arguments icon  next to the Enter Match Attributes field to launch the Match Attribute Builder interface:

**Figure 2-6** Match Attribute Builder



Select the *Browse attributes* icon to browse to and select the attributes you want to match. In this example they are L and Surname.

The second column allows you to match the current value stored in the attribute by selecting *Use value(s) from current Object*. You can match against another value by selecting *Other Value*. You can create any value you want to match. Select the value type, and the appropriate builder is available through the *Enter State* field.

## Action Argument Component Builder

Launch the Action Argument Component Builder by selecting the following actions when the Enter Value Type selection is the Structured selection.

- “Add Destination Attribute Value” on page 120
- “Add Source Attribute Value” on page 123
- “Reformat Operation Attribute” on page 138
- “Remove Destination Attribute Value” on page 139
- “Remove Source Attribute Value” on page 140
- “Set Destination Attribute Value” on page 146
- “Set Source Attribute Value” on page 153

**Figure 2-7** Add Destination Attribute Value Action

Do  ?

Enter attribute name: \*  🔍

Enter class name:  🔍

Select mode:  ▼

Select object:  ▼

Enter DN: \*  📄

**Enter value type:**  ▼

Enter components: \*  📄

**Figure 2-8** Action Argument Component Builder

## Argument Value List Builder

The Argument Value List Builder enables you to construct default argument values for the **Set Default Attribute Value** (page 145) action.

For example, if you want to set a default location of Unknown, you select the following action:

**Figure 2-9** Set Default Attribute Value Action

Do  ?

Enter attribute name: \*  🔍

Write back:  ▼

Enter argument values: \*  📄

You then click the icon next to the Enter Values field to launch the Argument Value List Builder interface, and construct an argument similar to the following:

**Figure 2-10** Argument Value List Builder

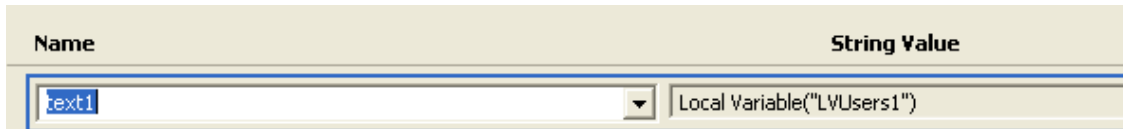
Type	Argument Values
<input type="text" value="string"/> ▼	"Digital Airlines Inc."

## Named String Builder

The Named String Builder enables you to construct name/value pairs for use in certain actions, such as [Generate Event \(page 133\)](#), [Send Email \(page 143\)](#), and [Send Email From Template \(page 144\)](#).

For a Generate Event action, the named strings correspond to the custom value fields you can provide with an event:

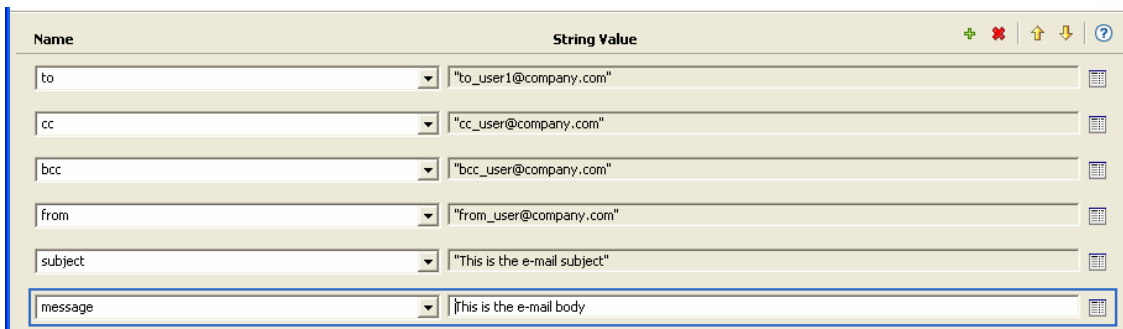
**Figure 2-11** *Named String Builder*



Name	String Value
text1	Local Variable("LVUsers1")

For a Send Mail action, the named strings correspond to the elements of the e-mail:

**Figure 2-12** *E-mail Elements in the Send Mail Action*



Name	String Value
to	"to_user1@company.com"
cc	"cc_user@company.com"
bcc	"bcc_user@company.com"
from	"from_user@company.com"
subject	"This is the e-mail subject"
message	This is the e-mail body

A complete list of possible values is contained in the help file corresponding to the action that launches the Named String Builder.

## Condition Argument Component Builder

Launch the Condition Argument Component Builder by clicking the Edit Arguments Icon. 

In order to see the icon, you must select the Structured selection for Mode with the following conditions:

- [“If Attribute” on page 103](#)
- [“If Destination Attribute” on page 105](#)
- [“If Source Attribute” on page 116](#)
- [“If Operation Attribute” on page 113](#)



**Figure 2-13** Structured Mode for the Condition

The screenshot shows a dialog box titled "Condition" with a dropdown menu set to "attribute". Below this, there are four fields: "Name \*" containing "Given Name", "Operator \*" set to "equal", "Mode" set to "structured" (highlighted with a red circle), and "Value". At the bottom left are "OK" and "Cancel" buttons, and at the bottom right is a red asterisk followed by the text "\* Required".

**Figure 2-14** Condition Argument Component Builder

The screenshot shows a dialog box titled "Condition Argument Component Builder". It has a blue header bar with a close button. Below the header, it says "Argument Components" and "The condition argument components are name/value pairs." There is a table with two columns: "Name" and "Values". The table is currently empty. To the right of the table are four small icons: a plus sign, a red X, an up arrow, and a down arrow. At the bottom right of the table area is a small icon of a document with a checkmark.

## Pattern String Builder

Launch the Pattern String Builder from the Argument Builder Editor when the Unique Name token is selected. The Argument Builder Editor pane shows a Pattern field that launches the Pattern String Builder.

- [“Unique Name” on page 171](#)

Figure 2-15 Unique Name Token in the Argument Builder

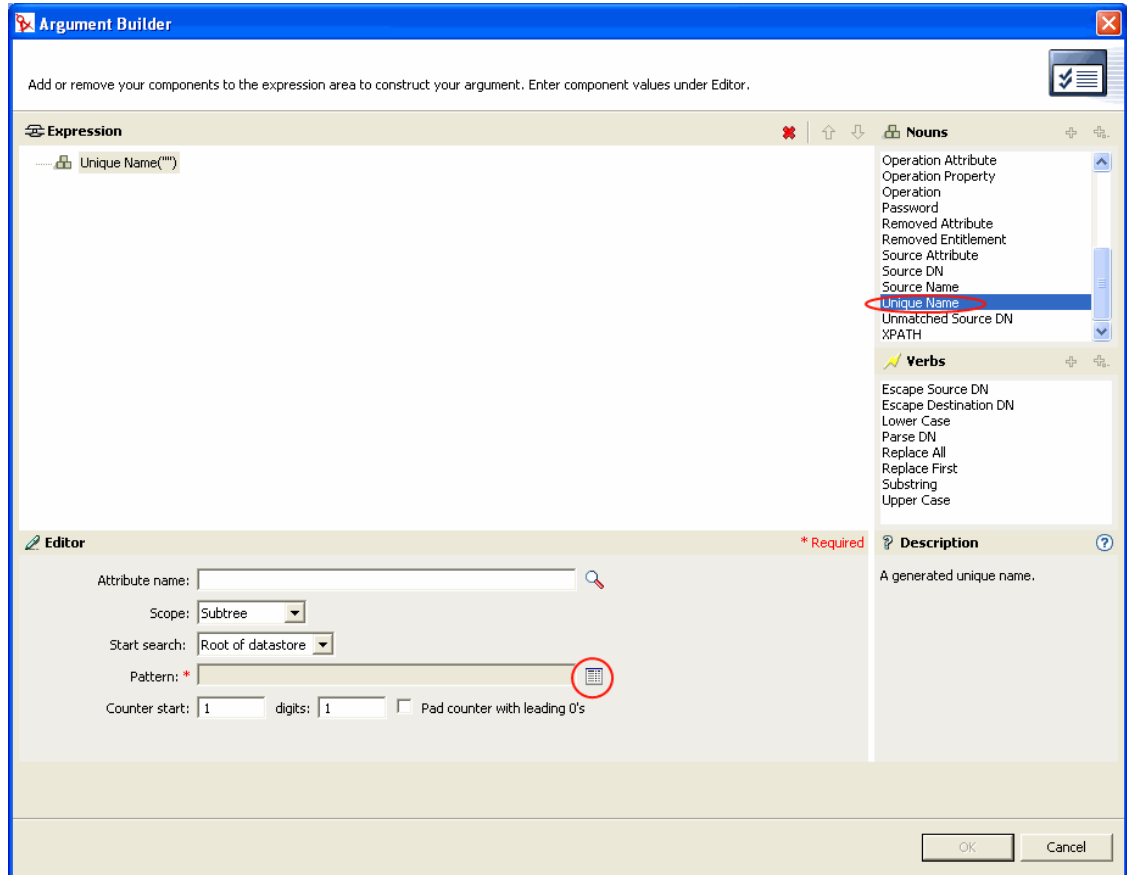
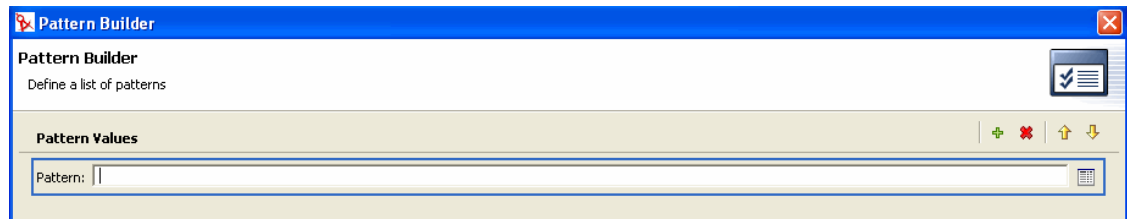


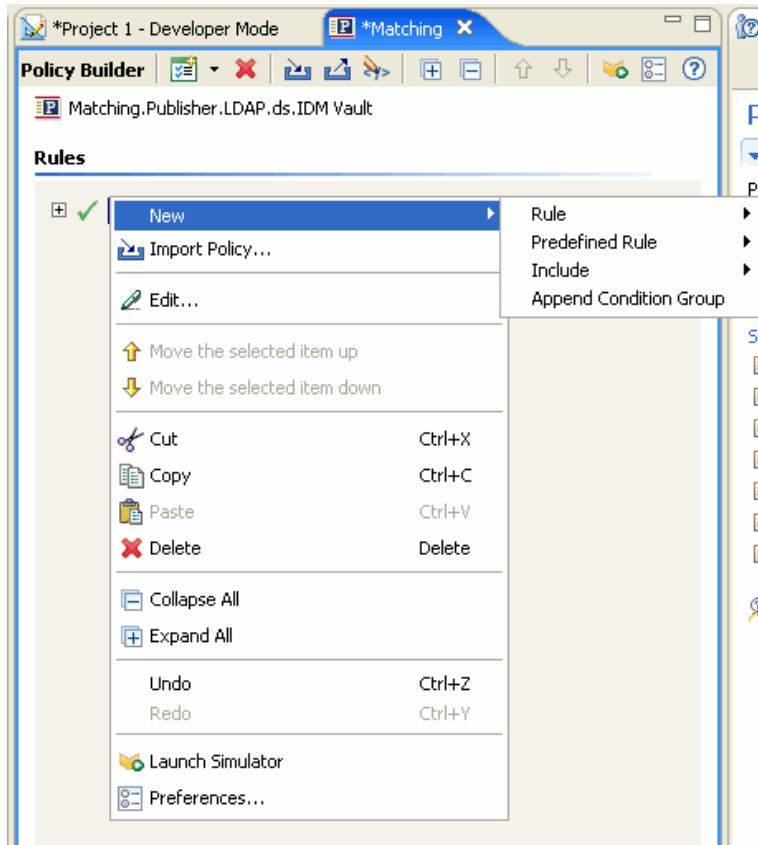
Figure 2-16 Pattern String Builder



## 2.2.5 Modifying a Policy

The Policy Builder allows you to create and edit policies. You can drag and drop rules, conditions and actions. For additional operations, access the Policy Builder toolbar. To display a context menu, right-click an item.

**Figure 2-17** Policy Builder Context Menu and Toolbar





### Actions and Menu Items in the Policy Builder

The table contains a list of the different actions and menu items that are possible in the Policy Builder.

The table contains a list of the different actions and menu items that are possible in the Policy Builder.

**Table 2-3** Policy Builder Actions and Menu Items

Operation	Description
<i>Collapse All</i>	Collapses all expanded rules.
<i>Copy</i>	Copies the selected item to the Clipboard.
Copy and drop	Select the item, press Ctrl, then drag the item.
<i>Cut</i>	Cuts the selected item and copies it to the Clipboard.
<i>Delete</i>	Deletes the selected item.
Disable	Disables a rule, condition, or action. Click the  icon.

<b>Operation</b>	<b>Description</b>
Drag and drop	Enables you to select an item, then relocate it. Select the item, then drag it to the new location.
<i>Edit</i>	Enables you to edit the selected item. To open the Rule Builder, select a rule, then click Edit.
Enable	Enables a rule, condition, or action. Click the  icon.
<i>Expand All</i>	Expands all the rules so that you can view the conditions and actions of each rule.
<i>Import Policy</i>	Imports a policy from the file system and appends it to the policy, or replaces all the rules of the policy.
<i>Launch Simulator</i>	Launches the Policy Simulator.
Move and drop	Enables you to select and move an item. Select the item, then drag the item.
<i>Move the selected item down</i>	Moves the item down in the list of policies.
<i>Move the selected item up</i>	Moves the item up in the list of policies.
<i>New &gt; Condition Group</i>	Creates a new condition group after a selected item.
<i>New &gt; Include</i>	Creates a new Include after a selected item.
<i>New &gt; Predefined Rule</i>	Inserts a predefined rule.
<i>New &gt; Rule</i>	Creates a new rule after a selected item.
<i>Paste</i>	Pastes the contents of the Clipboard after the selected item.
<i>Preferences</i>	Enables you to change how the information is displayed.
Select	Click any item to select it.

## **Policy Description**

The Policy Description field provides a place to add notes about the functionality of the policy.

## **KeyBoard Support**

You can move through the Policy Builder with keystrokes as well as use the mouse. Below is listed the supported keystrokes.

**Table 2-4** *Keyboard Support in the Policy Builder*

<b>Keystroke</b>	<b>Description</b>
Ctrl+C	Copies the selected item into the Clipboard.
Ctrl+X	Cuts the selected item and adds it to the Clipboard.

Keystroke	Description
Ctrl+V	Pastes the contents of the Clipboard after the selected item.
Delete	Deletes the selected Item.
Left-Arrow	Collapses a rule node.
Right-Arrow	Expands a rule node.
Up-Arrow	Navigates up.
Down-Arrow	Navigates down.
Undo	Ctrl+Z
Redo	Ctrl+Y

## Saving Your Work

Do one of the following:

- From the Main menu, click *File > Save* (or *Save All*).
- Close the editor by clicking the *X* in the editor's tab.
- Select *Close* from the Main Menu's file menu.

## 2.2.6 Using Predefined Rules

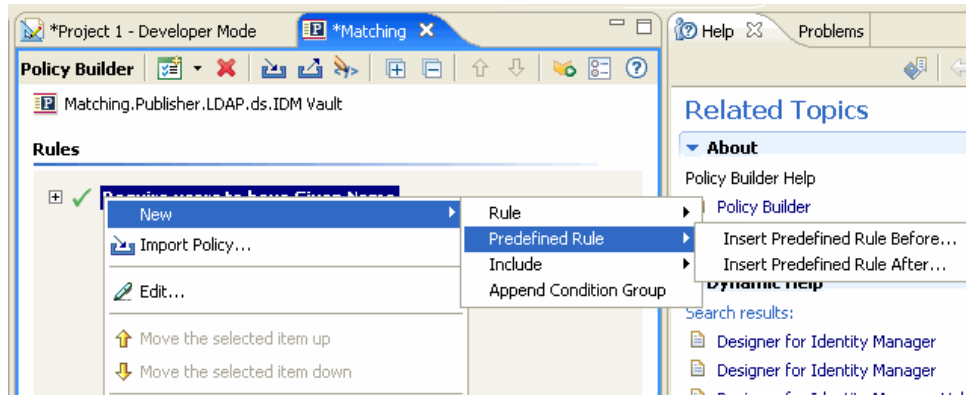
Designer includes twenty predefined rules. You can import and use these rules as well as create your own rules. These rules include common tasks that administrators use. You need to provide information specific to your environment to customize the rules.

- [“Command Transformation - Create Departmental Container - Part 1 and Part 2” on page 62](#)
- [“Command Transformation - Publisher Delete to Disable” on page 64](#)
- [“Creation - Require Attributes” on page 65](#)
- [“Creation - Publisher - Use Template” on page 67](#)
- [“Creation - Set Default Attribute Value” on page 68](#)
- [“Creation - Set Default Password” on page 70](#)
- [“Event Transformation - Scope Filtering - Include Subtrees” on page 71](#)
- [“Event Transformation - Scope Filtering - Exclude Subtrees” on page 72](#)
- [“Input or Output Transformation - Reformat Telephone Number from \(nnn\) nnn-nnnn to nnn-nnn-nnnn” on page 74](#)
- [“Input or Output Transformation - Reformat Telephone Number from nnn-nnn-nnnn to \(nnn\) nnn-nnnn” on page 75](#)
- [“Matching - Publisher Mirrored” on page 76](#)
- [“Matching - Subscriber Mirrored - LDAP Format” on page 78](#)
- [“Matching - By Attribute Value” on page 79](#)
- [“Placement - Publisher Mirrored” on page 81](#)

- “Placement - Subscriber Mirrored - LDAP Format” on page 82
- “Placement - Publisher Flat” on page 84
- “Placement - Subscriber Flat - LDAP Format” on page 85
- “Placement - Publisher By Dept” on page 87
- “Placement - Subscriber By Dept - LDAP Format” on page 88

To access the predefined rules: right-click in the Policy Builder, then click *New > Predefined Rules*.

**Figure 2-18** *Predefined Rules*



## Command Transformation - Create Departmental Container - Part 1 and Part 2

Creates a department container in the destination data store, if one does not exist. Implement the rule on the Subscriber Command Transformation policy or Publisher Command Transformation policy in the driver.

There are two steps involved in using the predefined rules: creating a policy in the Command Transformation policy set, and importing the predefined rule. If you already have a Command Transformation policy that you want to add this rule to, skip to “[Importing the Predefined Rule](#)” on page 63.

### Creating a Policy

- 1 From the Outline view or the Policy Flow view, select the Publisher or Subscriber channel.
- 2 Select the Command Transformation policy set in the Policy Set, then click *Create or add a new policy to the Policy Set* + to create a new policy.
- 3 Click *Create a new policy*, then click *Next*.
- 4 Name the policy.
- 5 Use the location that is populated to place the policy in the driver.

**Create Policy**

Specify the name and parent container for the new policy.

Enter the name that will be used for the new policy.

Create Container

Select the container where the policy will be created.

Publisher.LDAP.ds.IDM Vault

Open Editor after creating policy.

- 6 Select *Open Editor after creating policy*, then click *Next*.
- 7 Select *DirXML Script* for the type of policy, then click *Finish*.
- 8 A file conflict window appears with the message “*Before editing this item you need to save. Do you wish to save the editor’s changes and continue?*” Click *Yes*. The Policy Builder is launched and the new Command Transformation policy is saved.

**Importing the Predefined Rule**

- 1 Right-click in the Policy Builder, then click *New > Predefined Rule*.
- 2 Select *Command Transformation - Create Department Container - Part 1*, then click *OK*.
- 3 Right-click in Policy Builder and click *New > Predefined Rule*.
- 4 Select *Command Transformation - Create Department Container - Part 2*, then click *OK*.
- 5 Save the rule by clicking *File > Save*.

There is no information to change in the rules that are specific to your environment.

Command Transformation - Create Departmental Container - Part 1

Conditions

Condition Group 1

- if operation equal "add"

Actions

- set local variable("target-container", Destination DN(length="-2"))
- set local variable("does-target-exist", Destination Attribute("objectclass", class name="OrganizationalUnit", dn(Local Variable("target-container"))))

Command Transformation - Create Departmental Container - Part 2

Conditions

Condition Group 1

- if local variable 'does-target-exist' available

And

- if local variable 'does-target-exist' equal ""

Actions

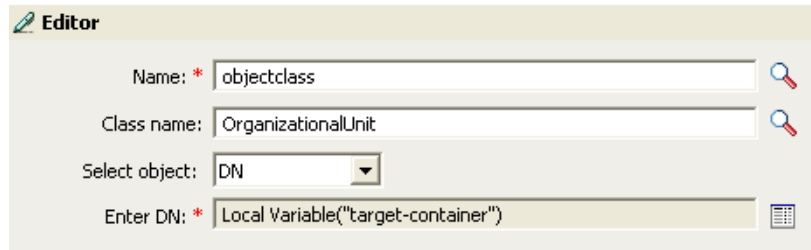
- add destination object(class name="organizationalUnit", direct="true", dn(Local Variable("target-container")))
- add destination attribute value("ou", direct="true", dn(Local Variable("target-container")), Parse DN("dest-dn", "dot", length="1", start="-1", Local Variable("target-container")))

**IMPORTANT:** Make sure that the rules are listed in order. Part 1 must be executed before Part 2.

## How the Logic in the Rule Works

The rule is used when the destination location for an object does not exist. Instead of getting a veto because the object cannot be placed, this rule creates the container and places the object in the container.

Part 1 looks for any Add operation. When the Add operation occurs, two local variables are set. The first local variable is named target-container. The value of target-container is set to the destination DN. The second local variable is named does-target-exist. The value of does-target-exist is set to the destination attribute value of objectclass. The class is set to OrganizationalUnit. The DN of the OrganizationalUnit is set to the local variable of target-container.



The screenshot shows a software interface titled "Editor" with a pencil icon. It contains four input fields for configuring a rule:

- Name:** \* objectclass
- Class name:** OrganizationalUnit
- Select object:** DN (with a dropdown arrow)
- Enter DN:** \* Local Variable("target-container")

Part 2 checks to see if the local variable does-target-exist is available. It also checks to see if the value of the local variable does-target-exist is set to a blank value. If the value is blank, then an Organizational Unit object is created. The DN of the organizational unit is set to the value of the local variable target-container. It also adds the value for the OU attribute. The value of the OU attribute is set to the name of the new organizational unit, which is obtained by parsing the value of the local variable target-container.

## Command Transformation - Publisher Delete to Disable

Transforms a Delete operation for a User object into a Modify operation that disables the target User object in eDirectory. Implement the rule on the Publisher Command Transformation policy in the driver.

There are two steps involved in using the predefined rules: creating a policy in the Command Transformation policy set, and importing the predefined rule. If you already have a Command Transformation policy that you want to add this rule to, skip to [Importing the Predefined Rule \(page 65\)](#).

### Creating a Policy

- 1 From the Outline view or the Policy Flow view, select the Publisher channel.
- 2 Select the Command Transformation policy set in the Policy Set, then click *Create or add a new policy to the Policy Set* + to create a new policy.
- 3 Click *Create a new policy*, then click *Next*.
- 4 Name the policy.
- 5 Use the location that is populated to place the policy in the driver.




## Create Policy

Specify the name and parent container for the new policy.

Enter the name that will be used for the new policy.

Select the container where the policy will be created.

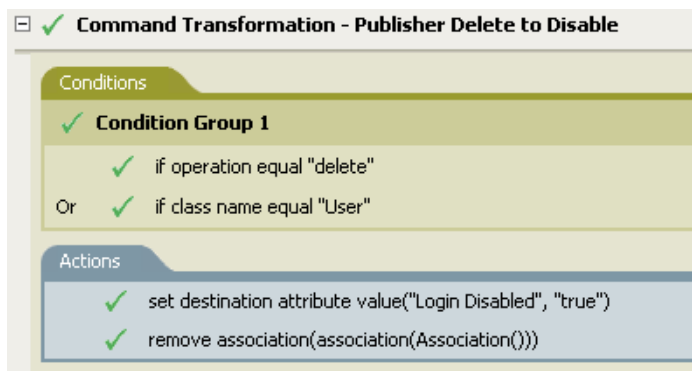
Open Editor after creating policy.

- 6 Select *Open Editor after creating policy*, then click *Next*.
- 7 Select *DirXML Script* for the type of policy, then click *Finish*.
- 8 A file conflict window appears with the message “*Before editing this item you need to save. Do you wish to save the editor’s changes and continue?*” Click *Yes*. The Policy Builder is launched and the new Command Transformation policy is saved.

## Importing the Predefined Rule

- 1 Right-click in the Policy Builder, then click *New > Predefined Rule*.
- 2 Select *Command Transformation - Publisher Delete to Disable*, then click *OK*.
- 3 Save the rule by clicking *File > Save*.

There is no information to change in the rule that is specific to your environment.



Command Transformation - Publisher Delete to Disable

Conditions

Condition Group 1

- if operation equal "delete"
- Or if class name equal "User"

Actions

- set destination attribute value("Login Disabled", "true")
- remove association(association(Association()))

## How the Logic in the Rule Works


The rule is used when a Delete command is going to be sent to the Identity Vault, usually in response to a Delete event that occurred in the connected system. Instead of the User object being deleted in the Identity Vault, the User object is disabled. When a Delete command is processed for a User object, the destination attribute value of Login Disabled is set to true, the association is removed from the User object, and the Delete command is vetoed. The User object can no longer log in to the Novell® eDirectory™ tree, but the User object was not deleted.

## Creation - Require Attributes

Prevents User objects from being created unless the required attributes are populated. Implement the rule on the Subscriber Creation policy or the Publisher Creation policy in the driver.

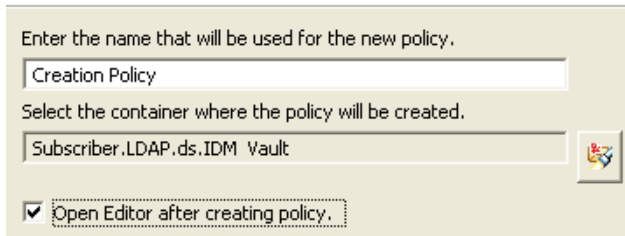
There are two steps involved in using the predefined rules: creating a policy in the Creation policy set, and importing the predefined rule. If you already have a Creation policy that you want to add this rule to, skip to [Importing the Predefined Rule \(page 66\)](#).

### Creating a Policy

- 1 From the Outline view or the Policy Flow view, select the Publisher or Subscriber channel.
- 2 Select the Creation policy set in the Policy Set, then click *Create or add a new policy to the Policy Set*  to create a new policy.
- 3 Click *Create a new policy*, then click *Next*.
- 4 Name the policy.
- 5 Use the location that is populated to place the policy in the driver.

#### Create Policy

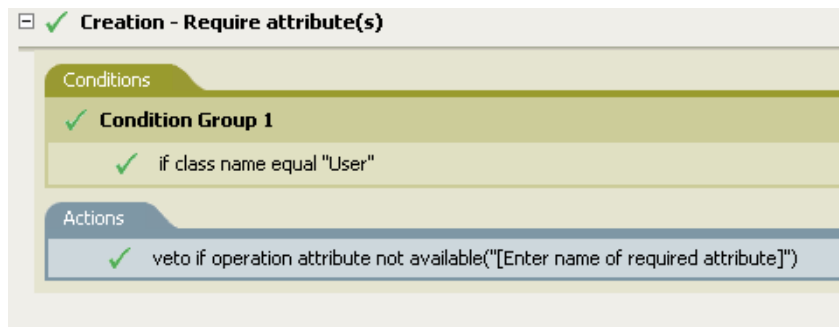
Specify the name and parent container for the new policy.



- 6 Select *DirXML Script* for the type of policy, then click *Finish*.
- 7 A file conflict window appears with the message “*Before editing this item you need to save. Do you wish to save the editor’s changes and continue?*” Click *Yes*. The Policy Builder is launched and the new Creation policy is saved.

### Importing the Predefined Rule

- 1 Right-click in the Policy Builder and click *New > Predefined Rule*.
- 2 Select *Creation - Require attributes*, then click *OK*.
- 3 Edit the action by double-clicking the Actions tab.
- 4 Delete *[Enter name of required attribute]* from the *Enter Name field*.
- 5 Browse to the attributes you require for a User object to be created, then click *OK*.
- 6 Click *OK*.
- 7 Save the rule by selecting *File > Save*.



## How the Logic in the Rule Works


The rule is used when your business processes require that a user has specific attributes populated in the source User object before the destination the User object can be created. When a User object is created in the source data store, the rule vetoes the creation of the object in the destination data store unless the required attributes are provided when the User object is created. You can have one or more required attributes.

### Creation - Publisher - Use Template

Allows for the use of a Novell eDirectory template object during the creation of a User object. Implement the rule on the Publisher Creation policy in the driver.

There are two steps involved in using the predefined rules: creating a policy in the Creation policy set, and importing the predefined rule. If you already have a Creation policy that you want to add this rule to, skip to [Importing the Predefined Rule \(page 67\)](#).

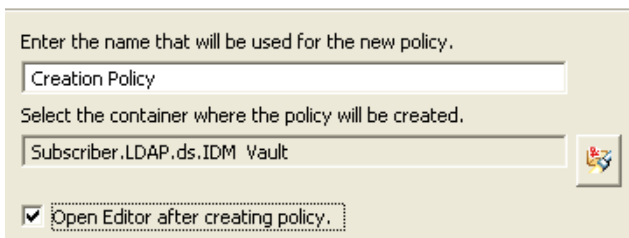
### Creating a Policy

- 1 From the Outline view or the Policy Flow view, select the Publisher channel.
- 2 Select the Creation policy set in the Policy Set, then click *Create or add a new policy to the Policy Set*  to create a new policy.
- 3 Click *Create a new policy*, then click *Next*.
- 4 Name the policy.
- 5 Use the location that is populated to place the policy in the driver.

---


#### Create Policy

Specify the name and parent container for the new policy.

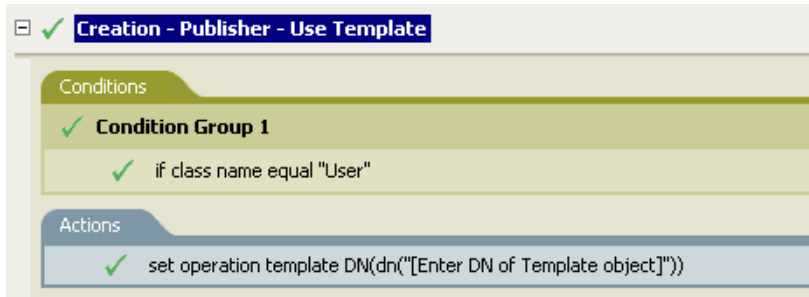


- 6 Select *Open Editor after creating policy*, then click *Next*.
- 7 Select *DirXML Script* for the type of policy, then click *Finish*.
- 8 A file conflict window appears with the message “*Before editing this item you need to save. Do you wish to save the editor’s changes and continue?*” Click *Yes*. The Policy Builder is launched and the new Creation policy is saved.

### Importing the Predefined Rule

- 1 Right-click in the Policy Builder, then click *New > Predefined Rule*.
- 2 Select *Creation - Publisher - Use Template*, then click *OK*.
- 3 Edit the action by double-clicking the Actions tab.
- 4 Delete *[Enter DN of Template object]* from the *Enter DN field*.
- 5 Click the *Edit Arguments icon*  to launch the Argument Builder.

- 6 Select *Text* in the Noun list.
- 7 Double-click *Text* to add it to the argument.
- 8 In the Editor, click the browse icon, browse to and select the template object, then click *OK*.
- 9 Click *OK*.
- 10 Save the rule by clicking *File > Save*.



### How the Logic in the Rule Works

The rule is used when you want to create a user in the Identity Vault based on a template object. If you have attributes that are the same for users, using the template saves time. You fill in the information in the template object and when the User object is created, Identity Manager uses the attribute values from the template to create the User object.


During the creation of User objects, the rule does the action of the set operation template DN, which instructs the Identity Manager to use the referenced template when creating the object.

### Creation - Set Default Attribute Value

Allows you to set default values for attributes that are assigned during the creation of User objects. Implement the rule on the Subscriber Creation policy or Publisher Creation policy in the driver.

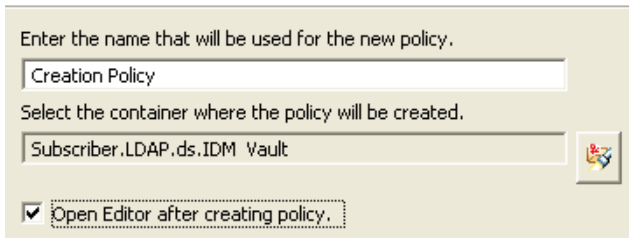
There are two steps involved in using the predefined rules: creating a policy in the Creation policy set, and importing the predefined rule. If you already have a Creation policy that you want to add this rule to, skip to [Importing the Predefined Rule \(page 69\)](#).

### Creating a Policy

- 1 From the Outline view or the Policy Flow view, select the Publisher or Subscriber channel.
- 2 Select the Creation policy set in the Policy Set, then click the *Create or add a new policy to the Policy Set* icon  to create a new policy.
- 3 Click *Create a new policy*, then click *Next*.
- 4 Name the policy.
- 5 Use the location that is populated to place the policy in the driver.



## Create Policy

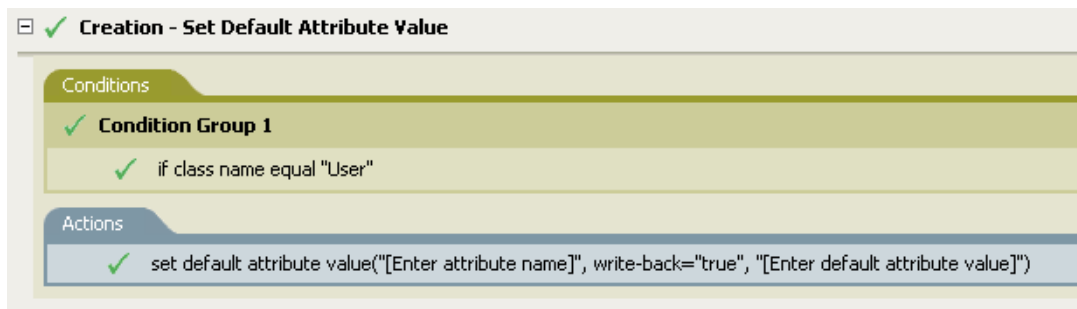
Specify the name and parent container for the new policy.



- 6 Select *Open Editor after creating policy*, then click *Next*.
- 7 Select *DirXML Script* for the type of policy, then click *Finish*.
- 8 A file conflict window appears with the message “*Before editing this item you need to save. Do you wish to save the editor’s changes and continue?*” Click *Yes*. The Policy Builder is launched and the new Creation policy is saved.

## Importing the Predefined Rule

- 1 Right-click in the Policy Builder, then click *New > Predefined Rule*.
- 2 Select *Creation - Set Default Attribute Value*, then click *OK*.
- 3 Edit the action by double-clicking the *Actions* tab.
- 4 Delete *[Enter attribute name]* from the *Enter attribute name* field.
- 5 Click the browse icon, then browse to and select the attribute you want to have created.
- 6 Delete *[Enter default attribute value]* from the *Enter arguments values* field.
- 7 Click the *Edit Arguments* icon  to launch the Argument Values List Builder.
- 8 Select the type of data you want the value to be.
- 9 Click the *Edit Arguments* icon  to launch the Argument Builder.
- 10 Create the value you want the attribute to be through the Argument Builder, then click *OK*.
- 11 Click *OK*.
- 12 Save the rule by clicking *File > Save*.



## How the Logic in the Rule Works

The rule is used when you want to populate default attribute values when creating a User object. When a User object is created, the rule adds the specified attribute values if and only if the attribute has no values supplied by the source object.


If you want more than one attribute value defined, right-click the action and click *New > Action*. Select the action, set the default attribute value, and follow the steps above to assign the value to the attribute.

## Creation - Set Default Password

During the creation of User objects, it sets a default password for User objects. Implement the rule on the Subscriber Creation policy or Publisher Creation policy in the driver.

There are two steps involved in using the predefined rules: creating a policy in the Creation policy set, and importing the predefined rule. If you already have a Creation policy that you want to add this rule to, skip to [Importing the Predefined Rule \(page 70\)](#).

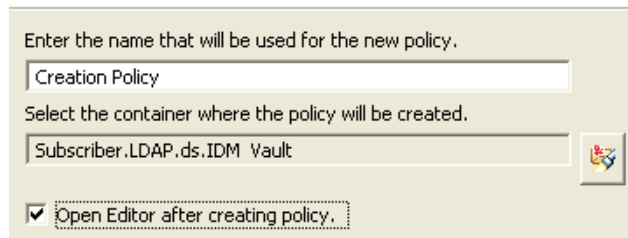
### Creating a Policy

- 1 From the Outline view or the Policy Flow view, select the Publisher or Subscriber channel.
- 2 Select the Creation policy set in Policy Set, then click *Create or add a new policy to the Policy Set*  to create a new policy.
- 3 Click *Create a new policy*, then click *Next*.
- 4 Name the policy.
- 5 Use the location that is populated to place the policy in the driver.

---

#### Create Policy

Specify the name and parent container for the new policy.



Enter the name that will be used for the new policy.

Creation Policy

Select the container where the policy will be created.

Subscriber.LDAP.ds.IDM Vault

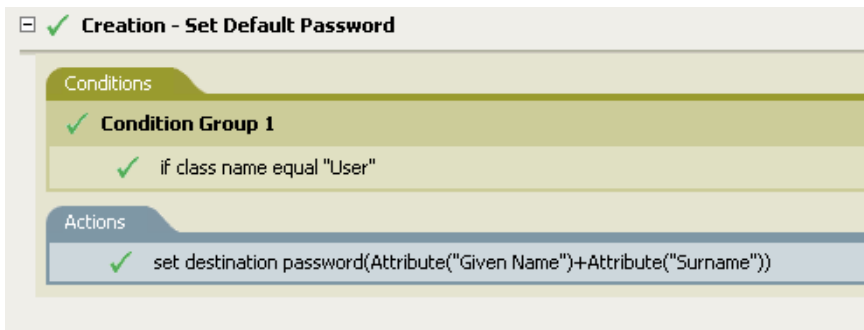
Open Editor after creating policy.

- 6 Select *Open Editor after creating policy*, then click *Next*.
- 7 Select *DirXML Script* for the type of policy, then click *Finish*.
- 8 A file conflict window appears with the message “*Before editing this item you need to save. Do you wish to save the editor’s changes and continue?*” Click *Yes*. The Policy Builder is launched and the new Creation policy is saved.

### Importing the Predefined Rule

- 1 Right-click in the Policy Builder, then click *New > Predefined Rule*.
- 2 Select *Creation - Set Default Password*, then click *OK*.
- 3 Save the rule by clicking *File > Save*.

There is no information to change in the rule that is specific to your environment.



## How the Logic in the Rule Works

The rule is used when you want User objects to be created with a default password. During the creation of a User object, the password that is set for the User object is the Given Name attribute plus the Surname attribute of the User object.


You can change the value of the default password by editing the argument. You can set the password to any other value you want through the Argument Builder.

## Event Transformation - Scope Filtering - Include Subtrees

Excludes all events that occur outside of the specific subtrees. Implement the rule on the Subscriber Event Transformation policy or the Publisher Event Transformation policy in the driver.

There are two steps involved in using the predefined rules: creating a policy in the Event Transformation policy set, and importing the predefined rule. If you already have an Event Transformation policy that you want to add this rule to, skip to [Importing the Predefined Rule \(page 72\)](#).

## Creating a Policy


- 1 From the Outline view or the Policy Flow view, select the Publisher or Subscriber channel.
- 2 Select the Event Transformation policy set in the Policy Set, then click *Create or add a new policy to the Policy Set*  to create a new policy.
- 3 Click *Create a new policy*, then click *Next*.
- 4 Name the policy.
- 5 Use the location that is populated to place the policy in the driver.

**Create Policy**

Specify the name and parent container for the new policy.

Enter the name that will be used for the new policy.

Select the container where the policy will be created.

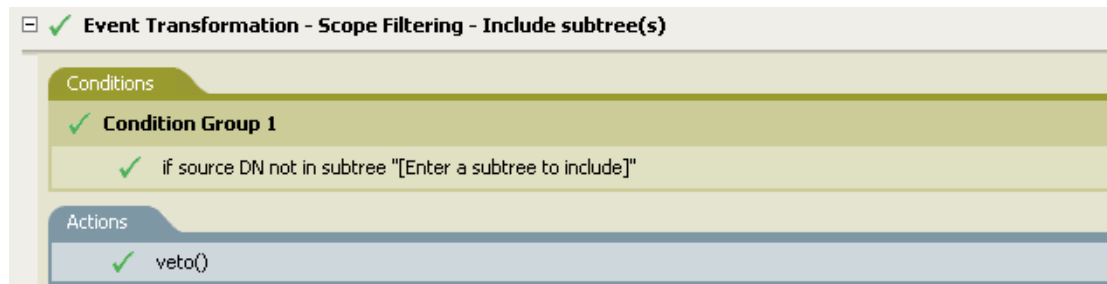
Open Editor after creating policy.

- 6 Select *Open Editor after creating policy*, then click *Next*.

- 7 Select *DirXML Script* for the type of policy, then click *Finish*.
- 8 A file conflict window appears with the message “*Before editing this item you need to save. Do you wish to save the editor’s changes and continue?*” Click *Yes*. The Policy Builder is launched and the new Event Transformation policy is saved.

### Importing the Predefined Rule

- 1 Right-click in the Policy Builder, then select *New > Predefined Rule*.
- 2 Select *Event Transformation - Scope Filtering - Include subtrees*, then click *OK*.
- 3 Edit the condition by double-clicking the Conditions tab.
- 4 Delete *[Enter a subtree to include]* in the *Value* field.
- 5 Click the browse button to browse the Identity Vault for the part of the tree you were you want events to synchronize, then click *OK*.
- 6 Click *OK*.
- 7 Save the rule by click *File > Save*.



### How the Logic in the Rule Works


The rule is used when you only want to synchronize specific subtrees between the Identity vault and the connected system. When an event occurs anywhere but in that specific part of the Identity Vault, it is vetoed. You can add additional subtrees to be synchronized by copying and pasting the “**If Source DN**” on page 117 condition.

### Event Transformation - Scope Filtering - Exclude Subtrees

Excludes all events that occur in a specific subtree. Implement the rule on the Subscriber Event Transformation or the Publisher Event Transformation policy in the driver.

There are two steps involved in using the predefined rules: creating a policy in the Event Transformation policy set, and importing the predefined rule. If you already have an Event Transformation policy that you want to add this rule to, skip to [Importing the Predefined Rule \(page 73\)](#).

### Creating a Policy


- 1 From the Outline view or the Policy Flow view, select the Publisher or Subscriber channel.
- 2 Select the Event Transformation policy set in Policy Set, then click *Create or add a new policy to the Policy Set*  to create a new policy.
- 3 Click *Create a new policy*, then click *Next*.
- 4 Name the policy.



- 5 Use the location that is populated to place the policy in the driver.

**Create Policy**  
Specify the name and parent container for the new policy.

Enter the name that will be used for the new policy.

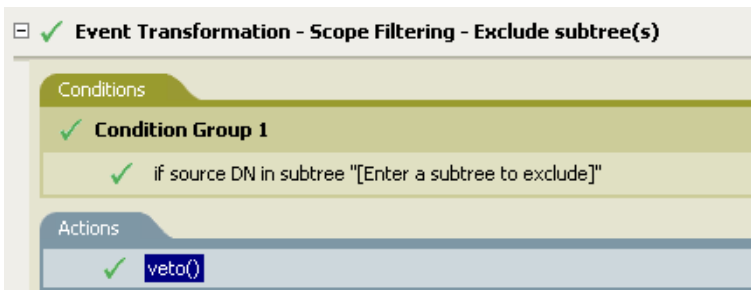
Select the container where the policy will be created.  
 

Open Editor after creating policy.

- 6 Select *Open Editor after creating policy*, then click *Next*.
- 7 Select *DirXML Script* for the type of policy, then click *Finish*.
- 8 A file conflict window appears with the message “*Before editing this item you need to save. Do you wish to save the editor’s changes and continue?*” Click *Yes*. The Policy Builder is launched and the new Event Transformation policy is saved.

### Importing the Predefined Rule

- 1 Right-click in the Policy Builder, then click *New > Predefined Rule*.
- 2 Select *Event Transformation - Scope Filtering - Exclude subtrees*, then click *OK*.
- 3 Edit the condition by double-clicking the Conditions tab.
- 4 Delete *[Enter a subtree to exclude]* in the *Value field*.
- 5 Click the browse button to browse the Identity Vault for the part of the tree you want to exclude events from synchronizing, then click *OK*.
- 6 Click *OK*.
- 7 Save the rule by clicking *File > Save*.



Event Transformation - Scope Filtering - Exclude subtree(s)

Conditions

✓ Condition Group 1

✓ if source DN in subtree "[Enter a subtree to exclude]"

Actions

✓ veto()

### How the Logic in the Rule Works


The rule is used when you want to exclude part of the Identity Vault or connected system from synchronizing. When an event occurs in that specific part of the Identity Vault, it is vetoed. You can add additional subtrees to be excluded by copying and pasting the if source DN condition.

## Input or Output Transformation - Reformat Telephone Number from (nnn) nnn-nnnn to nnn-xxx-xxxx

Converts the format of the telephone number. Implement the rule on the Input or Output Transformation policy in the driver. Typically, if this rule is used on an Input Transformation, you would then use the rule Reformat Telephone Number from nnn-xxx-xxxx to (nnn) nnn-nnnn on the Output Transformation and vice versa to convert the format back and forth.

There are two steps involved in using the predefined rules: creating a policy in the Input or Output Transformation policy set, and importing the predefined rule. If you already have an Input or Output Transformation policy that you want to add this rule to, skip to [Importing the Predefined Rule \(page 74\)](#).

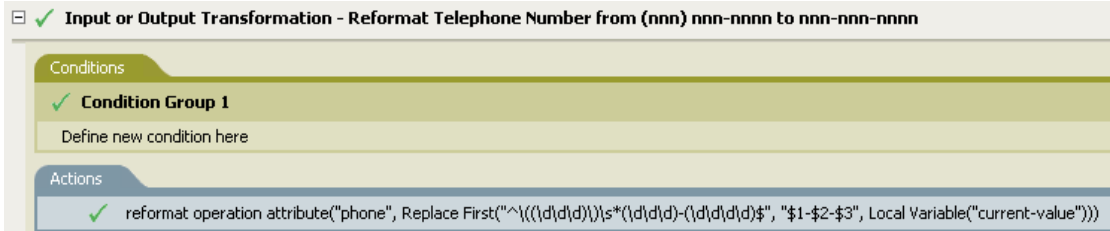
### Creating a Policy

- 1 From the Outline view or the Policy Flow view, select the Publisher or Subscriber channel.
- 2 Select the Input or Output Transformation policy set in the Policy Set, then click *Create or add a new policy to the Policy Set*  to create a new policy.
- 3 Click *Create a new policy*, then click *Next*.
- 4 Name the policy.
- 5 Use the location that is populated to place the policy in the driver.

- 6 Select *Open Editor after creating policy*, then click *Next*.
- 7 Select *DirXML Script* for the type of policy, then click *Finish*.
- 8 A file conflict window appears with the message “*Before editing this item you need to save. Do you wish to save the editor’s changes and continue?*” Click *Yes*. Policy Builder is launched and the new Input or Output Transformation policy is saved.

### Importing the Predefined Rule

- 1 Right-click in the Policy Builder, then click *New > Predefined Rule*.
- 2 Select *Input or Output Transformation - Reformat Telephone Number from (nnn) nnn-nnnn to nnn-xxx-xxxx*, then click *OK*.
- 3 Edit the condition by double-clicking the Conditions tab.
- 4 Define the condition you want to have occur when the telephone number is reformatted.
- 5 Click *OK*.
- 6 Save the rule by clicking *File > Save*.



## How the Logic in the Rule Works


The rule is used when you want to reformat the telephone number. It finds all the values for the attribute phone in the current operation that match the pattern (nnn) nnn-nnnn and replaces each with nnn-xxx-xxxx.

## Input or Output Transformation - Reformat Telephone Number from nnn-xxx-xxxx to (nnn) nnn-nnnn

Transforms the format of the telephone number. Implement the rule on the Input or Output Transformation policy. Typically, if you use this rule on an Output Transformation, you would use the rule Reformat Telephone Number from (nnn) nnn-nnnn to nnn-xxx-xxxx on the Input Transformation and vice versa to convert the format back and forth.

There are two steps involved in using the predefined rules; creating a policy in the Input or Output Transformation policy set, and importing the predefined rule. If you already have an Input or Output Transformation policy that you want to add this rule to, skip to [Importing the Predefined Rule \(page 76\)](#).

### Creating a Policy


- 1 From the Outline view or the Policy Flow view, select the Publisher or Subscriber channel.
- 2 Select the Input or Output Transformation policy set in the Policy Set, then click *Create or add a new policy to the Policy Set*  to create a new policy.
- 3 Click *Create a new policy*, then click *Next*.
- 4 Name the policy.
- 5 Use the location that is populated to place the policy in the driver.

**Create Policy**

Specify the name and parent container for the new policy.

Enter the name that will be used for the new policy.

Select the container where the policy will be created.

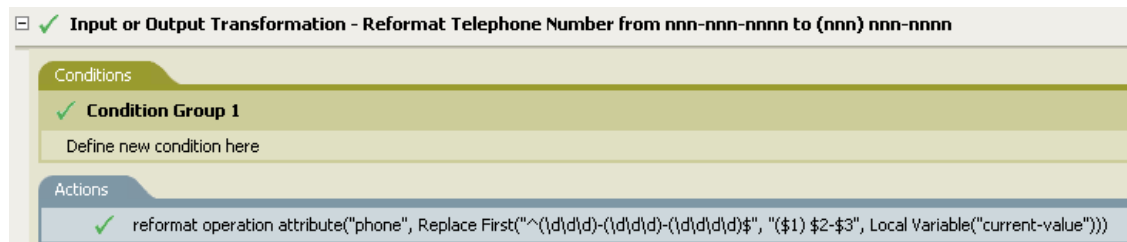
Open Editor after creating policy.

- 6 Select *Open Editor after creating policy*, then click *Next*.
- 7 Select *DirXML Script* for the type of policy, then click *Finish*.

- 8 A file conflict window appears with the message “Before editing this item you need to save. Do you wish to save the editor’s changes and continue?” Click *Yes*. Policy Builder is launched and the new Input or Output Transformation policy is saved.

### Importing the Predefined Rule

- 1 Right-click in the Policy Builder and click *New > Predefined Rule*.
- 2 Click *Input or Output Transformation - Reformat Telephone Number from nnn-xxx-nnnn to (nnn) nnn-nnnn*, then click *OK*.
- 3 Edit the condition by double-clicking the Conditions tab.
- 4 Define the condition you want to have occur when the telephone number is reformatted.
- 5 Click *OK*.
- 6 Save the rule by clicking *File > Save*.



### How the Logic in the Rule Works

The rule is used when you want to reformat the telephone number. It finds all the values for the attribute phone in the current operation that match the pattern (nnn) xxx-nnnn and replaces each with nnn-xxx-nnnn.

### Matching - Publisher Mirrored

Finds matches in the Identity Vault for objects in the connected system based on their name and location. Implement the rule on the Publisher Matching policy in the driver.

There are two steps involved in using the predefined rules: creating a policy in the Matching policy set, and importing the predefined rule. If you already have a Matching policy that you want to add this rule to, skip to [Importing the Predefined Rule \(page 77\)](#).

### Creating a Policy

- 1 From the Outline view or the Policy Flow view, select the Publisher channel.
- 2 Select the Matching policy set in the Policy Set, then click *Create or add a new policy to the Policy Set* + to create a new policy.
- 3 Click *Create a new policy*, then click *Next*.
- 4 Name the policy.
- 5 Use the location that is populated to place the policy in the driver.

## Create Policy

Specify the name and parent container for the new policy.


Enter the name that will be used for the new policy.  
Matching

Select the container where the policy will be created.  
Publisher.LDAP.ds.IDM Vault

Open Editor after creating policy.

- 6 Select *Open Editor after creating policy*, then click *Next*.
- 7 Select *DirXML Script* for the type of policy, then click *Finish*.
- 8 A file conflict window appears with the message “*Before editing this item you need to save. Do you wish to save the editor’s changes and continue?*” Click *Yes*. The Policy Builder is launched and the new Matching policy is saved.

## Importing the Predefined Rule

- 1 Right-click in the Policy Builder, then click *New > Predefined Rule*.
- 2 Select *Matching - Publisher Mirrored*, then click *OK*.
- 3 Edit the condition by double-clicking the Conditions tab.
- 4 Delete *[Enter base of source hierarchy]* from the *Value* field.
- 5 Browse to and select the container in the source hierarchy where you want the matching to start, then click *OK*.
- 6 Click *OK*.
- 7 Edit the action by double-clicking the Actions tab.
- 8 Delete *[Enter base of destination hierarchy]* from the *Enter string* field.
- 9 Click on the *Edit Arguments* icon  to launch the Argument Builder.
- 10 Select *Text* in the Noun list.
- 11 Double-click *Text* to add it to the argument.
- 12 In the Editor, click the browse icon and browse to the container in the destination hierarchy where you want the source structure to be matched, then click *OK*.
- 13 Click *OK*.
- 14 Save the rule by clicking *File > Save*.

Matching - Publisher Mirrored

Conditions

Condition Group 1

if source DN in subtree "[Enter base of source hierarchy]"

Actions

set local variable("dest-base", "[Enter base of destination hierarchy]")

find matching object(scope="entry", dn(Local Variable("dest-base")+";"+Unmatched Source DN(convert="true")))

## How the Logic in the Rule Works


When an Add event occurs on an object in the connected system that is located within the specified source subtree, the rule constructs a DN that represents the same object name and location within the Identity Vault relative to the specified destination subtree. If the destination objects exists and is of the desired object class then it is considered a match. You must supply the DN's of the source (connected system) and destination (Identity Vault) subtrees.

### Matching - Subscriber Mirrored - LDAP Format

Finds matches in a connected system that uses LDAP format DN's for objects in the Identity Vault based on their name and location. Implement the rule on the Subscriber Matching policy in the driver.

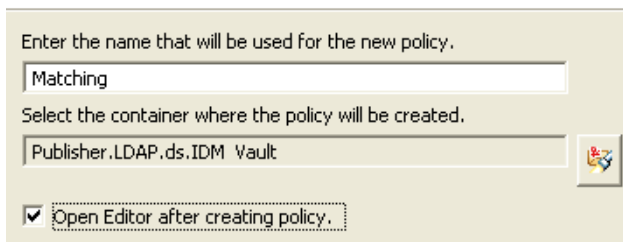
There are two steps involved in using the predefined rules: creating a policy in the Matching policy set, and importing the predefined rule. If you already have a Matching policy that you want to add this rule to, skip to [Importing the Predefined Rule \(page 78\)](#).

#### Creating a Policy

- 1 From the Outline view or the Policy Flow view, select the Publisher channel.
- 2 Select the Matching policy set in the Policy Set, then click *Create or add a new policy to the Policy Set*  to create a new policy.
- 3 Click *Create a new policy*, then click *Next*.
- 4 Name the policy.
- 5 Use the location that is populated to place the policy in the driver.

#### Create Policy


Specify the name and parent container for the new policy.

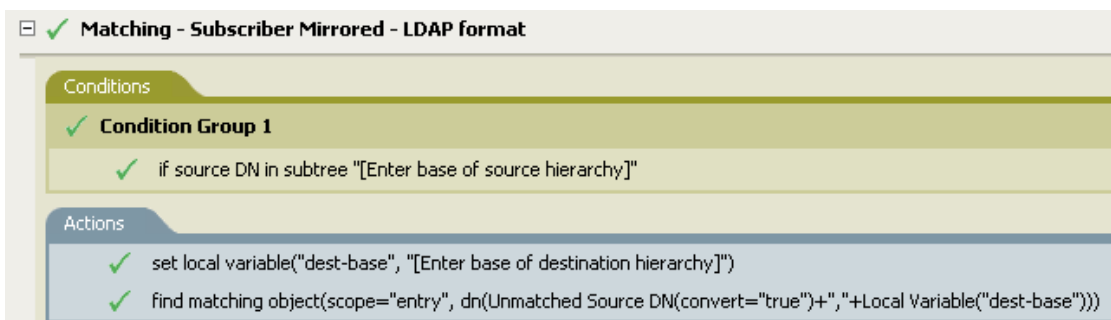


- 6 Select *Open Editor after creating policy*, then click *Next*.
- 7 Select *DirXML Script* for the type of policy, then click *Finish*.
- 8 A file conflict window appears with the message “*Before editing this item you need to save. Do you wish to save the editor’s changes and continue?*” Click *Yes*. The Policy Builder is launched and the new Matching policy is saved.

#### Importing the Predefined Rule

- 1 Right-click in the Policy Builder, then click *New > Predefined Rule*.
- 2 Select *Matching - Subscriber Mirrored - LDAP format*, then click *OK*.
- 3 Edit the condition by double-clicking the Conditions tab.
- 4 Delete *[Enter base of source hierarchy]* from the *Value* field.

- 5 Browse to and select the container in the source hierarchy where you want the matching to start, then click *OK*.
- 6 Click *OK*.
- 7 Edit the action by double-clicking the Actions tab.
- 8 Delete *[Enter base of destination hierarchy]* from the *Enter String field*.
- 9 Click the *Edit Arguments* icon  to launch the Argument Builder.
- 10 Select *Text* in the Noun list.
- 11 Double-click *Text* to add it to the argument.
- 12 In the Editor, click the browse icon, browse to and select the container in the destination hierarchy where you want the source structure to be matched, then click *OK*.
- 13 Click *OK*.
- 14 Save the rule by clicking *File > Save*.



### How the Logic in the Rule Works


When an Add event occurs on an object in the Identity Vault that is located within the specified source subtree, the rule constructs a DN that represents the same object name and location within the connected system relative to the specified destination subtree. If the destination objects exists and is of the desired object class then it is considered a match. You must supply the DN's of the source (Identity Vault) and destination (connected system) subtrees. The connected system must use an LDAP formatted DN.

### Matching - By Attribute Value

Finds matches for objects by specific attribute values. Implement the rule on the Subscriber Matching policy or the Publisher Matching policy in the driver.

There are two steps involved in using the predefined rules; creating a policy in the Matching policy set, and importing the predefined rule. If you already have a Matching policy that you would like to add this rule to, skip to [Importing the Predefined Rule \(page 80\)](#).

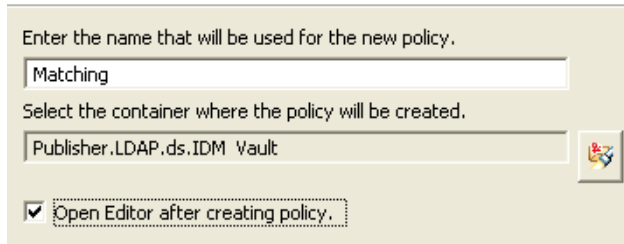
### Creating a Policy

- 1 From the Outline view or the Policy Flow view, select the Publisher channel.
- 2 Select the Matching policy set in the Policy Set, then click *Create or add a new policy to the Policy Set*  to create a new policy.
- 3 Click *Create a new policy*, then click *Next*.
- 4 Name the policy.

- 5 Use the location that is populated to place the policy in the driver.

### Create Policy

Specify the name and parent container for the new policy.





Enter the name that will be used for the new policy.  
Matching

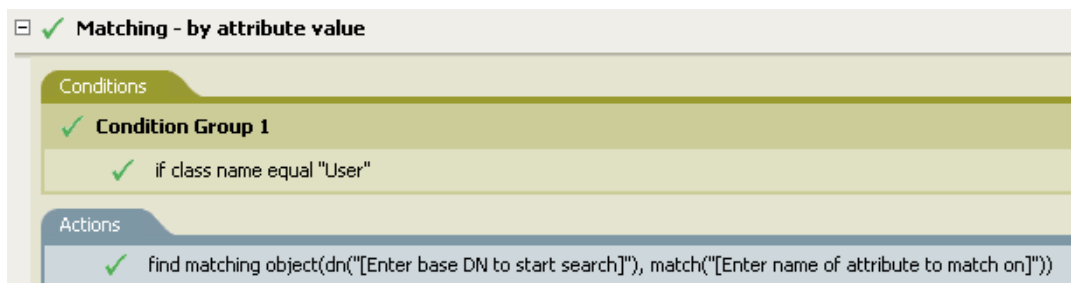
Select the container where the policy will be created.  
Publisher.LDAP.ds.IDM Vault

Open Editor after creating policy.

- 6 Select *Open Editor after creating policy*, then click *Next*.
- 7 Select *DirXML Script* for the type of policy, then click *Finish*.
- 8 A file conflict window appears with the message “*Before editing this item you need to save. Do you wish to save the editor’s changes and continue?*” Click *Yes*. The Policy Builder is launched and the new Matching policy is saved.

### Importing the Predefined Rule

- 1 Right-click in the Policy Builder, then click *New > Predefined Rule*.
- 2 Select *Matching - by attribute value*, then click *OK*.
- 3 Edit the action by double-clicking the Actions tab.
- 4 Delete *[Enter base DN to start search]* from the *Enter DN field*.
- 5 Click the *Edit Arguments icon*  to launch the Argument Builder.
- 6 Select *Text* in the Noun list.
- 7 Double-click *Text* to add it to the argument.
- 8 In the Editor, click the browse icon, then browse to and select the container where you want the search to start, then click *OK*.
- 9 Delete *[Enter name of attribute to match on]* from the *Enter Match Attributes field*.
- 10 Click the *Edit Arguments icon*  to launch the Match Attributes Builder.
- 11 Click the browse icon and select the attributes you want to match. You can select one or more attributes to match against, then click *OK*.
- 12 Click *OK*.
- 13 Save the rule by clicking *File > Save*.



Matching - by attribute value

Conditions

Condition Group 1

if class name equal "User"

Actions

find matching object(dn("[Enter base DN to start search]"), match("[Enter name of attribute to match on]))



## How the Logic in the Rule Works


When an Add event occurs on an object in the source data store, rule searches for an object in the destination data store that has the same values for the specified attribute. You must supply the DN of the base of the subtree to search in the connected system and the name of the attribute to match on.

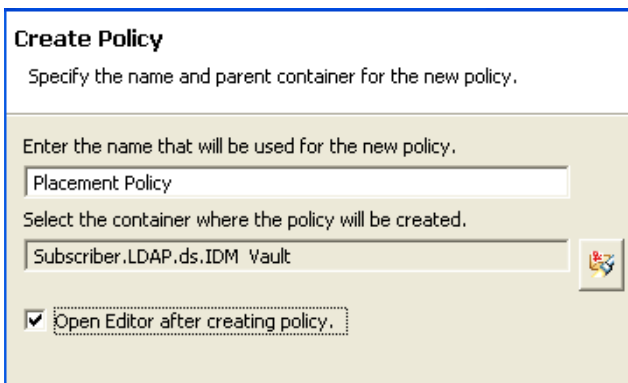
### Placement - Publisher Mirrored

Places objects in the Identity Vault by based on the name and location from the connected system. Implement the rule on the Publisher Placement policy in the driver.

There are two steps involved in using the predefined rules: creating a policy in the Placement policy set, and importing the predefined rule. If you already have a Placement policy that you would like to add this rule to, skip to [Importing the Predefined Rule \(page 81\)](#).

### Creating a Policy


- 1 From the Outline view or the Policy Flow view, select the Publisher channel.
- 2 Select the Placement policy set in the Policy Set, then click *Create or add a new policy to the Policy Set*  to create a new policy.
- 3 Click *Create a new policy*, then click *Next*.
- 4 Name the policy.
- 5 Use the location that is populated to place the policy in the driver.

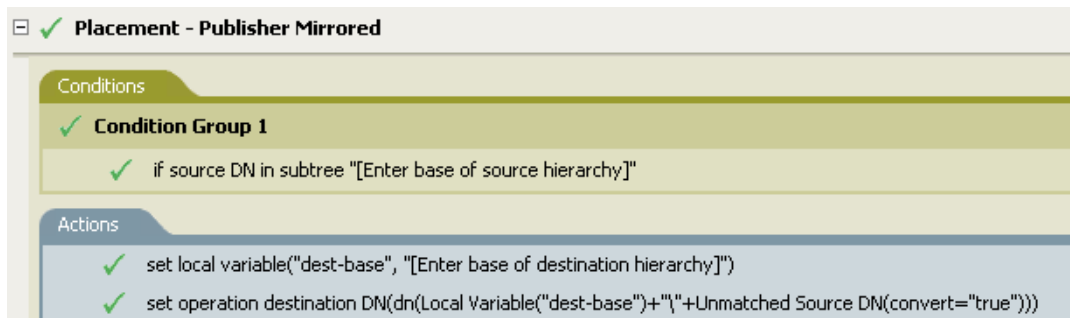


- 6 Select *Open Editor after creating policy*, then click *Next*.
- 7 Select *DirXML Script* for the type of policy, then click *Finish*.
- 8 A file conflict window appears with the message “*Before editing this item you need to save. Do you wish to save the editor’s changes and continue?*” Click *Yes*. The Policy Builder is launched and the new Placement policy is saved.

### Importing the Predefined Rule

- 1 Right-click in the Policy Builder, then click *New > Predefined Rule*.
- 2 Select *Placement - Publisher Mirrored*, then click *OK*.
- 3 Edit the condition by double-clicking the Conditions tab.
- 4 Delete *[Enter base of source hierarchy]* from the *Value field*.
- 5 Browse to and select the container in the source hierarchy where you want the object to be acted upon, then click *OK*.

- 6 Edit the action by double-clicking the Actions tab.
- 7 Delete *[Enter base of destination hierarchy]* from the *Enter String field*.
- 8 Click the *Edit Arguments* icon  to launch the Argument Builder.
- 9 Select *Text* in the Noun list.
- 10 Double-click *Text* to add it to the argument.
- 11 In the Editor, click the browse icon, browse to and select the container in the destination hierarchy where you want the object to be placed, then click *OK*.
- 12 Click *OK*.
- 13 Save the rule by clicking *File > Save*.



### How the Logic in the Rule Works


If the User object resides in the specified source subtree in the connected system, then the object is placed at the same relative name and location within the Identity Vault. You must supply the DN's of the source (connected system) and destination (Identity Vault) subtrees.

### Placement - Subscriber Mirrored - LDAP Format

Places objects in the data store by using the mirrored structure in the Identity Vault from a specified point. Implement the rule on the Placement policy in the driver. You can implement the rule only on the Subscriber channel.


There are two steps involved in using the predefined rules: creating a policy in the Placement policy set, and importing the predefined rule. If you already have a Placement policy that you want to add this rule to, skip to [Importing the Predefined Rule \(page 83\)](#).

### Creating a Policy

- 1 From the Outline view or the Policy Flow view, select the Publisher channel.
- 2 Select the Placement policy set in the Policy Set, then click *Create or add a new policy to the Policy Set*  to create a new policy.
- 3 Click *Create a new policy*, then click *Next*.
- 4 Name the policy.
- 5 Use the location that is populated to place the policy in the driver.

**Create Policy**  
Specify the name and parent container for the new policy.


Enter the name that will be used for the new policy.



Select the container where the policy will be created.  
 

Open Editor after creating policy.


- 6 Select *Open Editor after creating policy*, then click *Next*.
- 7 Select *DirXML Script* for the type of policy, then click *Finish*.
- 8 A file conflict window appears with the message “*Before editing this item you need to save. Do you wish to save the editor’s changes and continue?*” Click *Yes*. The Policy Builder is launched and the new Placement policy is saved.


### Importing the Predefined Rule

- 1 Right-click in the Policy Builder, then click *New > Predefined Rule*.
- 2 Select *Placement - Subscriber Mirrored - LDAP* format, then click *OK*.
- 3 Edit the condition by double-clicking the *Conditions* tab.
- 4 Delete *[Enter base of source hierarchy]* from the *Value* field.
- 5 Browse to the container in the source hierarchy where you want the object to be acted upon, then click *OK*.
- 6 Edit the action by double-clicking the *Actions* tab.
- 7 Delete *[Enter base of destination hierarchy]* from the *Enter String* field.
- 8 Click the *Edit Arguments* icon  to launch the Argument Builder.
- 9 Select *Text* in the Noun list.
- 10 Double-click *Text* to add it to the argument.
- 11 In the Editor, click the browse icon and browse to the container in the destination hierarchy where you want the object to be placed, then click *OK*.
- 12 Click *OK*.
- 13 Save the rule by clicking *File > Save*.


  **Placement - Subscriber Mirrored - LDAP format**


**Conditions**

 **Condition Group 1**

 if source DN in subtree "[Enter base of source hierarchy]"

**Actions**

 set local variable("dest-base", "[Enter base of destination hierarchy]")

 set operation destination DN(dn(Unmatched Source DN(convert="true")+";"+Local Variable("dest-base")))

## How the Logic in the Rule Works


If the User object resides in the specified source subtree, then the object is placed at the same relative name and location within the Identity Vault. You must supply the DN's of the source (Identity Vault) and destination (connected system) subtrees. The connected system must use an LDAP formatted DN.

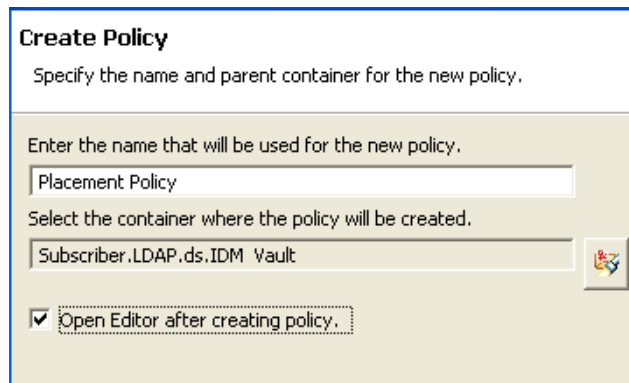
### Placement - Publisher Flat

Places objects from the data store into one container in the Identity Vault. Implement the rule on the Publisher Placement policy in the driver.

There are two steps involved in using the predefined rules: creating a policy in the Placement policy set, and importing the predefined rule. If you already have a Placement policy that you want to add this rule to, skip to [Importing the Predefined Rule \(page 84\)](#).


### Creating a Policy


- 1 From the Outline view or the Policy Flow view, select the Publisher channel.
- 2 Select the Placement policy set in the Policy Set, then click *Create or add a new policy to the Policy Set*  to create a new policy.
- 3 Click *Create a new policy*, then click *Next*.
- 4 Name the policy.
- 5 Use the location that is populated to place the policy in the driver.



**Create Policy**  
Specify the name and parent container for the new policy.


Enter the name that will be used for the new policy.  
Placement Policy

Select the container where the policy will be created.  
Subscriber.LDAP.ds.IDM Vault 

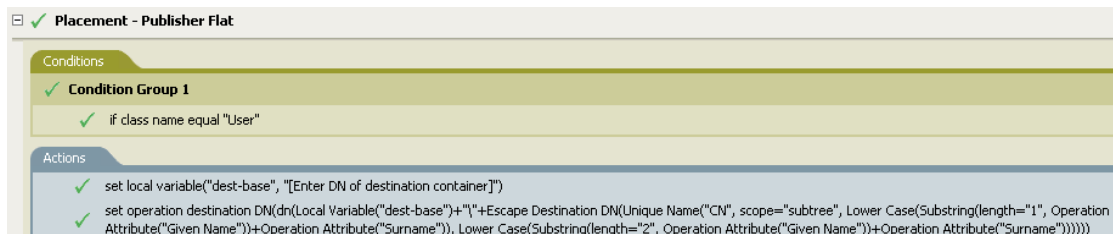
Open Editor after creating policy. 

- 6 Select *Open Editor after creating policy*, then click *Next*.
- 7 Select *DirXML Script* for the type of policy, then click *Finish*.
- 8 A file conflict window appears with the message “*Before editing this item you need to save. Do you wish to save the editor’s changes and continue?*” Click *Yes*. The Policy Builder is launched and the new Placement policy is saved.

### Importing the Predefined Rule

- 1 Right-click in the Policy Builder, then click *New > Predefined Rule*.
- 2 Select *Placement - Publisher Flat*, then click *OK*.
- 3 Edit the action by double-clicking the Actions tab.
- 4 Delete *[Enter DN of destination container]* from the *Enter String* field.
- 5 Click the *Edit Arguments* icon  to launch the Argument Builder.

- 6 Select *Text* in the Noun list.
- 7 Double-click *Text* to add it to the argument.
- 8 In the Editor, click the browse icon, then browse to and select the destination container where you want all of the User objects to be placed, then click *OK*.
- 9 Click *OK*.
- 10 Save the rule by clicking *File > Save*.



### How the Logic in the Rule Works


The rule places all User objects in the destination DN. The rule sets the DN of the destination container as the local variable `dest-base`. The rule then sets the destination DN to be `dest-base\CN` attribute. The CN attribute of the User object is the first two letters of the Given Name attribute plus the Surname attribute as lowercase. The rule uses slash format.

### Placement - Subscriber Flat - LDAP Format

Places objects from the Identity Vault into one container in the data store. Implement the rule on the Subscriber Placement policy in the driver.

There are two steps involved in using the predefined rules: creating a policy in the Placement policy set, and importing the predefined rule. If you already have a Placement policy that you want to add this rule to, skip to [Importing the Predefined Rule \(page 86\)](#).

### Creating a Policy

- 1 From the Outline view or the Policy Flow view, select the Publisher channel.
- 2 Select the Placement policy set in Policy Set, then click *Create or add a new policy to the Policy Set*  to create a new policy.
- 3 Click *Create a new policy*, then click *Next*.
- 4 Name the policy.
- 5 Use the location that is populated to place the policy in the driver.

**Create Policy**

Specify the name and parent container for the new policy.

Enter the name that will be used for the new policy.

Placement Policy


Select the container where the policy will be created.

Subscriber.LDAP.ds.IDM Vault

Open Editor after creating policy.

- 6 Select *Open Editor after creating policy*, then click *Next*.
- 7 Select *DirXML Script* for the type of policy, then click *Finish*.
- 8 A file conflict window appears with the message “*Before editing this item you need to save. Do you wish to save the editor’s changes and continue?*” Click *Yes*. The Policy Builder is launched and the new Placement policy is saved.

### Importing the Predefined Rule

- 1 Right-click in the Policy Builder, then click *New > Predefined Rule*.
- 2 Select *Placement - Subscriber Flat - LDAP format*, then click *OK*.
- 3 Edit the action by double-clicking the *Actions* tab.
- 4 Delete *[Enter DN of destination container]* from the *Enter String* field.
- 5 Click the *Edit Arguments* icon  to launch the Argument Builder.
- 6 Select *Text* in the Noun list.
- 7 Double-click *Text* to add it to the argument.
- 8 In the Editor, add the destination container were you want all of the User objects to be placed. Make sure the container is specified in LDAP format, then click *OK*.
- 9 Click *OK*.
- 10 Save the rule by clicking *File > Save*.

Placement - Subscriber Flat - LDAP format

Conditions

Condition Group 1

if class name equal "User"

Actions

set local variable("dest-base", "[Enter DN of destination container]")

set operation destination DN(dn("uid="+Escape Destination DN(Unique Name("uid", scope="subtree", Lower Case(Substring(length="1", Operation Attribute("Given Name")))+Operation Attribute("Surname")), Lower Case(Substring(length="2", Operation Attribute("Given Name"))+Operation Attribute("Surname")))+","+Local Variable("dest-base")))

### How the Logic in the Rule Works


The rule places all User objects in the destination DN. The rule sets the DN of the destination container as the local variable *dest-base*. The rule then sets the destination DN to be *uid=unique name,dest-base*. The *uid* attribute of the User object is the first two letters of the *Given Name* attribute plus the *Surname* attribute as lowercase. The rule uses LDAP format.

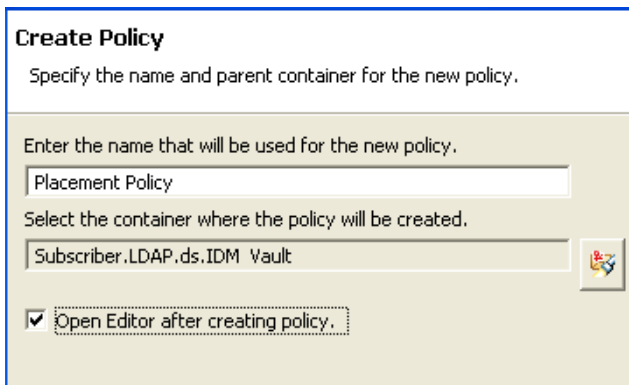
## Placement - Publisher By Dept

Places objects from one container in the data store into multiple containers in the Identity Vault based on the value of the OU attribute. Implement the rule on the Publisher Placement policy in the driver.

There are two steps involved in using the predefined rules: creating a policy in the Placement policy set, and importing the predefined rule. If you already have a Placement policy that you want to add this rule to, skip to [Importing the Predefined Rule \(page 87\)](#).


### Creating a Policy


- 1 From the Outline view or the Policy Flow view, select the Publisher channel.
- 2 Select the Placement policy set in the Policy Set, then click *Create or add a new policy to the Policy Set*  to create a new policy.
- 3 Click *Create a new policy*, then click *Next*.
- 4 Name the policy.
- 5 Use the location that is populated to place the policy in the driver.



**Create Policy**  
Specify the name and parent container for the new policy.


Enter the name that will be used for the new policy.  
Placement Policy

Select the container where the policy will be created.  
Subscriber.LDAP.ds.IDM Vault 

Open Editor after creating policy. 

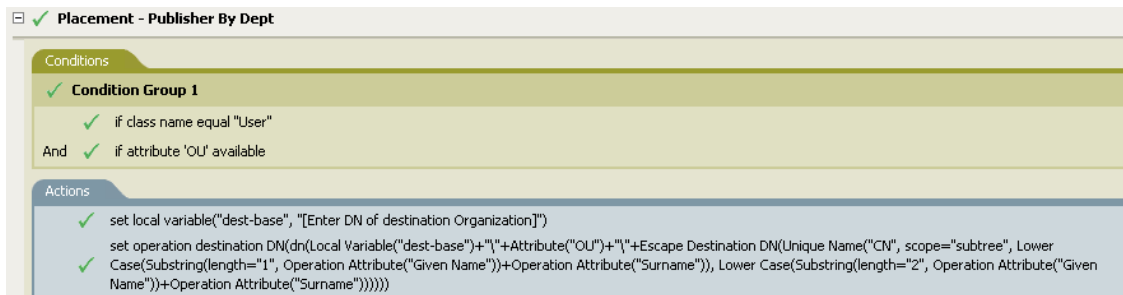
- 6 Select *Open Editor after creating policy*, then click *Next*.
- 7 Select *DirXML Script* for the type of policy, then click *Finish*.
- 8 A file conflict window appears with the message “Before editing this item you need to save. Do you wish to save the editor’s changes and continue?” Click *Yes*. The Policy Builder is launched and the new Placement policy is saved.

### Importing the Predefined Rule

- 1 Right-click in Policy Builder, then click *New > Predefined Rule*.
- 2 Select *Placement - Publisher By Dept*, then click *OK*.
- 3 Edit the action by double-clicking the Actions tab.
- 4 Delete *[Enter DN of destination Organization]* from the *Enter String* field.
- 5 Click the *Edit Arguments* icon  to launch the Argument Builder.
- 6 Select *Text* in the Noun list.
- 7 Double-click *Text* to add it to the argument.
- 8 In the Editor, click the browse icon, then browse to and select the parent container in the Identity Vault. Make sure all of the department containers are child containers of this DN, then click *OK*.

9 Click *OK*.

10 Save the rule by clicking *File > Save*.



## How the Logic in the Rule Works

The rule places User objects in proper department containers depending upon what value is stored in the OU attribute. If a User object needs to be placed and has the OU attribute available, then the User object is placed in the dest-base\value of OU attribute\CN attribute.

The dest-base is a local variable. The DN must be the relative root path of the department containers. It can be an organization or an organizational unit. The value stored in the OU attribute must be the name of a child container of the dest-base local variable.

The value of the OU attribute must be the name of the child container. If the OU attribute is not present, this rule is not executed.


The CN attribute of the User object is the first two letters of the Given Name attribute plus the Surname attribute as lowercase. The rule uses slash format.

## Placement - Subscriber By Dept - LDAP Format

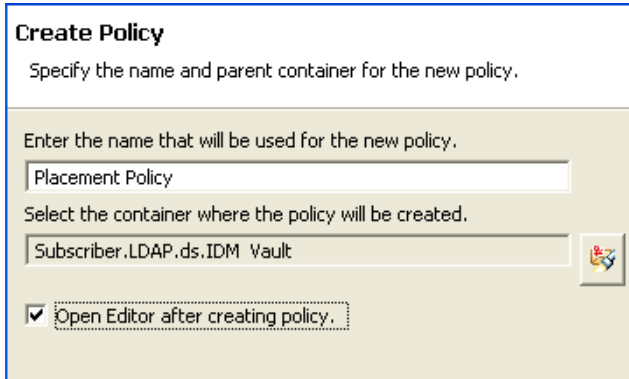
Places objects from one container in the Identity Vault into multiple containers in the data store base on the OU attribute. Implement the rule on the Placement policy in the driver. You can implement the rule only on the Subscriber channel.

There are two steps involved in using the predefined rules: creating a policy in the Placement policy set, and importing the predefined rule. If you already have a Placement policy that you want to add this rule to, skip to [Importing the Predefined Rule \(page 89\)](#).

### Creating a Policy


- 1 From the Outline view or the Policy Flow view, select the Publisher channel.
- 2 Select the Placement policy set in the Policy Set, then click *Create or add a new policy to the Policy Set*  to create a new policy.
- 3 Click *Create a new policy*, then click *Next*.
- 4 Name the policy.
- 5 Use the location that is populated to place the policy in the driver.

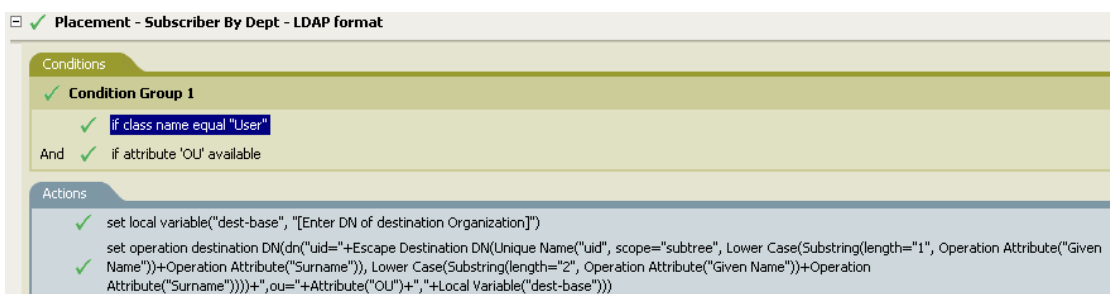




- 6 Select *Open Editor after creating policy*, then click *Next*.
- 7 Select *DirXML Script* for the type of policy, then click *Finish*.
- 8 A file conflict window appears with the message “*Before editing this item you need to save. Do you wish to save the editor’s changes and continue?*” Click *Yes*. The Policy Builder is launched and the new Placement policy is saved.

### Importing the Predefined Rule

- 1 Right-click in the Policy Builder, then click *New > Predefined Rule*.
- 2 Select *Placement - Subscriber By Dept - LDAP format*, then click *OK*.
- 3 Edit the action by double-clicking the *Actions* tab.
- 4 Delete *[Enter DN of destination Organization]* from the *Enter string* field.
- 5 Click the *Edit Arguments* icon  to launch the Argument Builder.
- 6 Select *Text* in the Noun list.
- 7 Double-click *Text* to add it to the argument.
- 8 In the Editor, add the parent container in the data store. The parent container must be specified in LDAP format. Make sure all of the department containers are child containers of this DN, then click *OK*.
- 9 Click *OK*.
- 10 Save the rule by clicking *File > Save*.



### How the Logic in the Rule Works

The rule places User objects in proper department containers depending upon what value is stored in the OU attribute. If a User object needs to be placed and has the OU attribute available, then the User object is placed in the uid=unique name,ou=value of OU attribute,dest-base.

The dest-base is a local variable. The DN must be the relative root path of the department containers. It can be an organization or an organizational unit. The value stored in the OU attribute must be the name of a child container of the dest-base local variable.

The value of the OU attribute must be the name of the child container. If the OU attribute is not present, then this rule is not executed.

The uid attribute of the User object is the first two letters of the Given Name attribute plus the Surname attribute as lowercase. The rule uses LDAP format.

## 2.2.7 Testing Policies with the Policy Simulator

Designer includes a new tool called the Policy Simulator. It allows you to test policies before deploying them. You can thoroughly test the policies and the drivers without using your existing data in the Identity Vault or the connected system.

For more information about common tasks with the Policy Simulator, see the following sections:

- “[Accessing the Policy Simulator](#)” on page 90
- “[Using the Policy Simulator](#)” on page 92

The Policy Simulator requires you to provide input well-formed XML documents for the policy to operate on. Typically the documents that policies operate on use an XML vocabulary called XDS that represents events and commands as they flow through the Metadirectory engine. To learn more about XDS, see [eDirectory DTD Commands and Events \(http://developer.novell.com/ndk/doc/dirxml/index.html?page=/ndk/doc/dirxml/dirxmlbk/data/a36pjzu.html\)](http://developer.novell.com/ndk/doc/dirxml/index.html?page=/ndk/doc/dirxml/dirxmlbk/data/a36pjzu.html).


All policies that are implemented in Policy Builder expect an XDS document as input. Most policies implemented in XSLT also expect an XDS document as input, however some policies (such as the initial Input Transformations for the Delimited Text and SOAP drivers) may expect a different XML vocabulary as input.

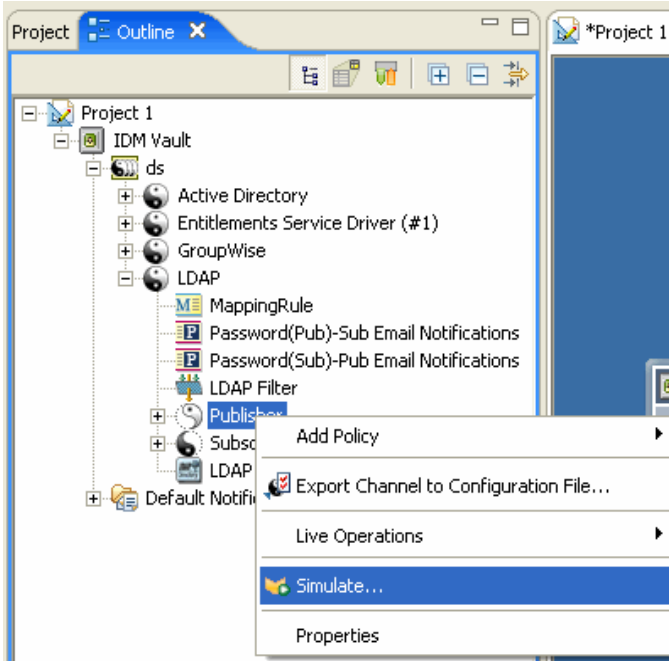
### Accessing the Policy Simulator

The Policy Simulator can be accessed in three different ways:


- “[Outline View](#)” on page 90
- “[Policy Flow](#)” on page 91
- “[Editors](#)” on page 92

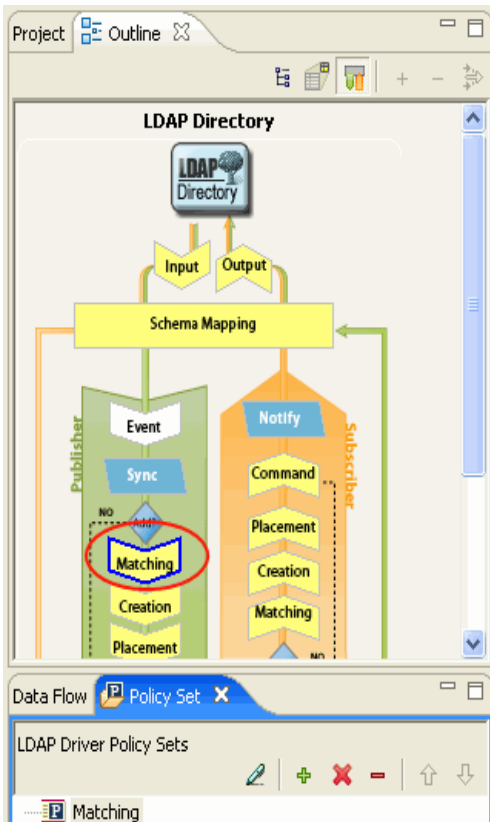
#### Outline View

- 1 Click the *Model Outline* icon. 
- 2 Right-click the driver, publisher, subscriber, mapping rule, filter, or any policy you want to simulate, then click *Simulate*.



## Policy Flow

- 1 Click the *Policy Flow* icon. 
- 2 Right-click the input, output, schemaMapping, filter, and any policy set icons you want to simulate, then click *Simulate*.



## Editors

You can access the Policy Simulator through the Policy Builder, the Schema Mapping Editor, or the Filter Editor by selecting the *Policy Simulator icon*  in the toolbar of each editor.

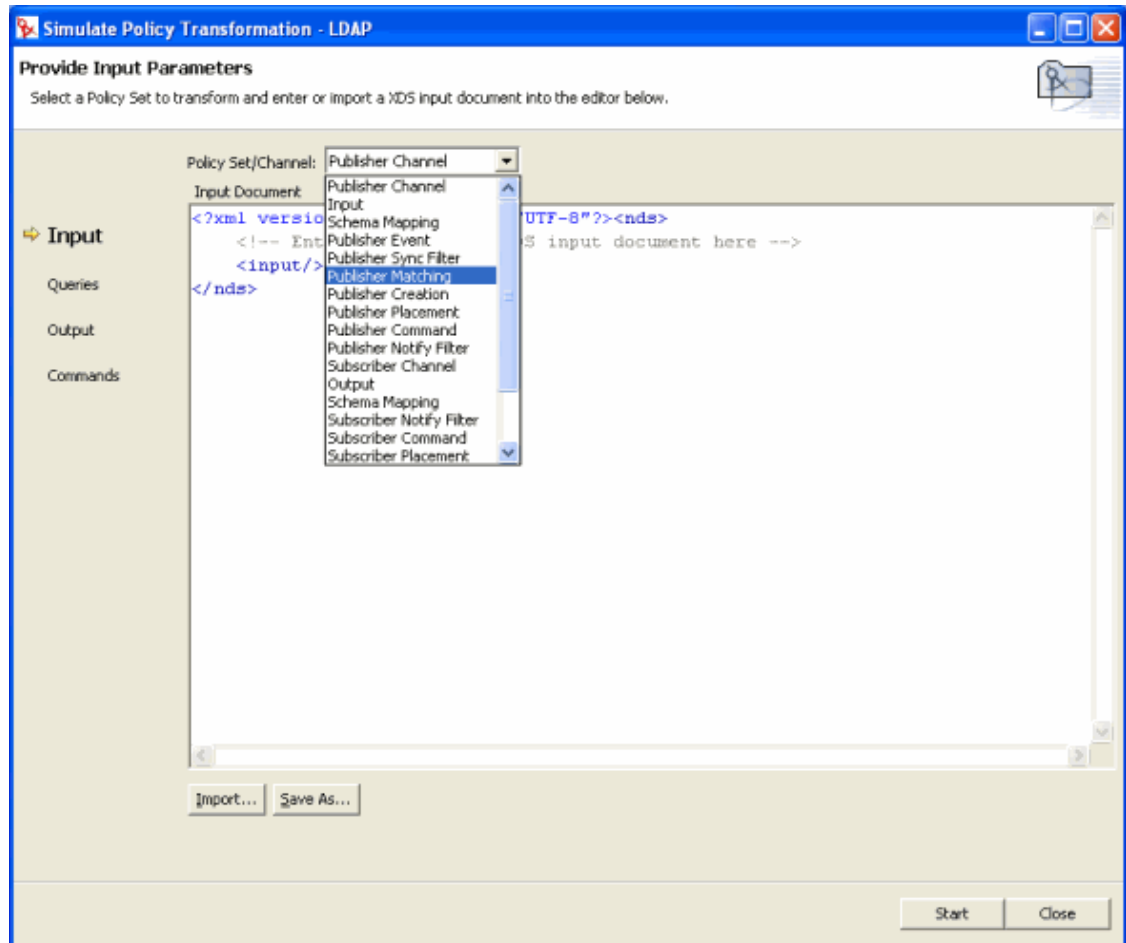
### Using the Policy Simulator

The Policy Simulator allows you to select a point in the driver flow to test the policy with a specific operation. It allows you to edit the input and output documents, while you are testing. If you want to keep the changes, select the Save As icon to save the document as an XML file.

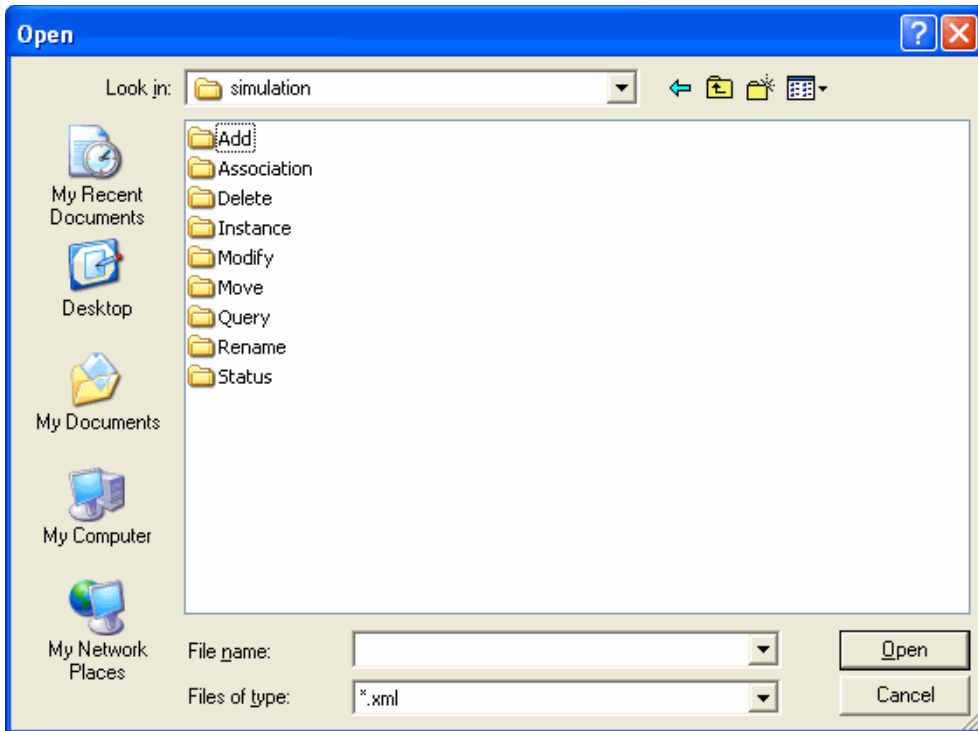
The Policy Simulator gives you a powerful tool to test and edit policies before implementing them in lab or production environments.

To use the Policy Simulator, please follow the steps listed below:

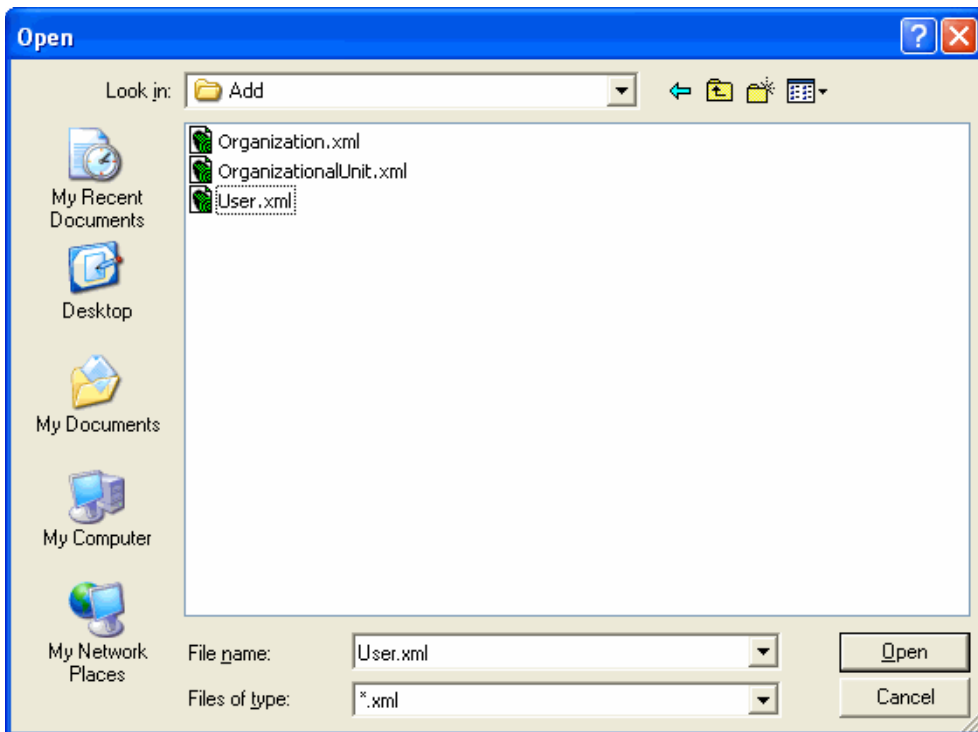
- 1 From the Policy Set drop-down list, select the place in the driver flow that you want to test the policy. You can select any of the following items: Publisher Channel, Subscriber Channel, Input, Schema Mapping, Event, Sync Filter, Matching, Creation, Placement, Command and Notify Filter. If you select a specific policy or rule to test, this option does not appear.



- 2 Select *Import* to display the list of operations you can test the policy with. The operations are Add, Association, Delete Instance, Modify, Move, Query, Rename, and Status.

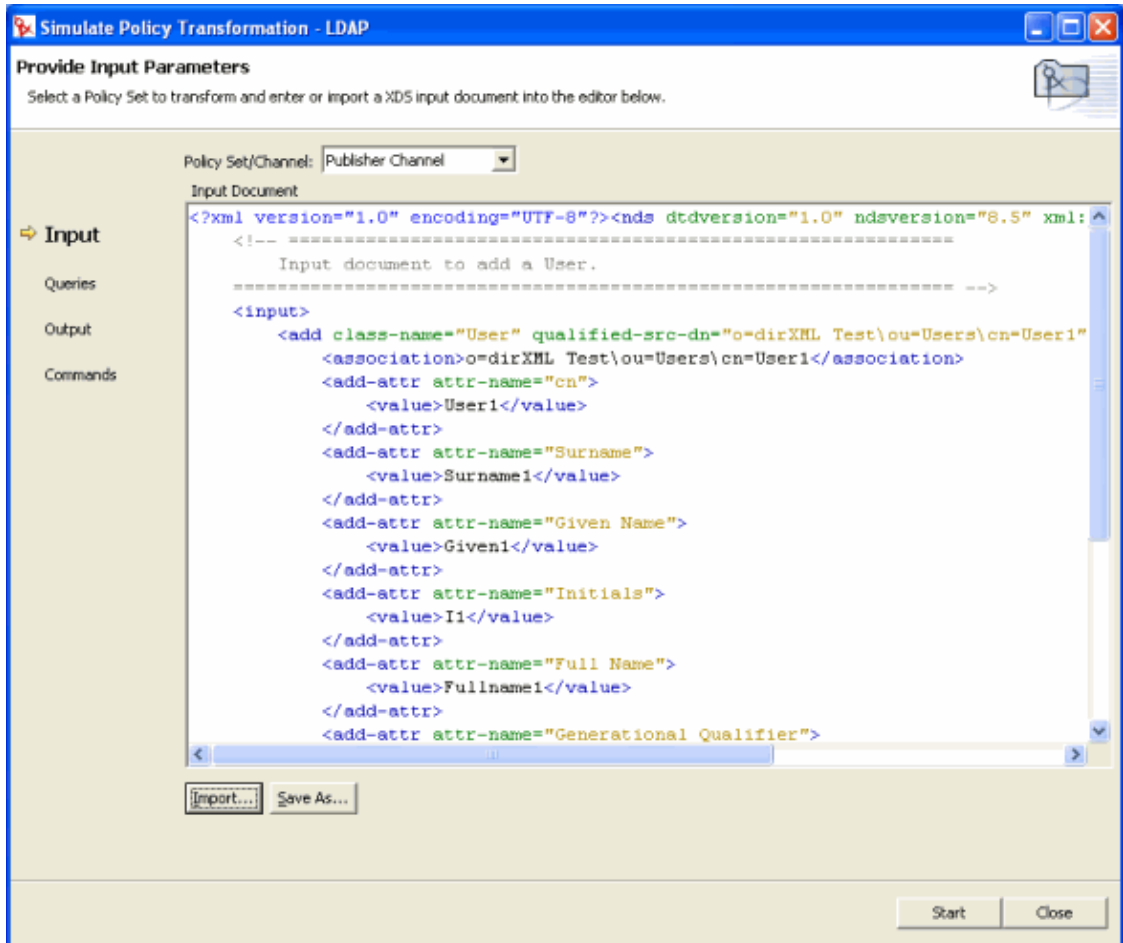


- 3 Double-click the folder to display the available events. Each event has different files you can select. For example, if you select Add you have three options: Organization.xml, OrganizationalUnit.xml and User.xml. The file indicates the object class for the operation. If you select User.xml, it will be an Add operation for a User object.

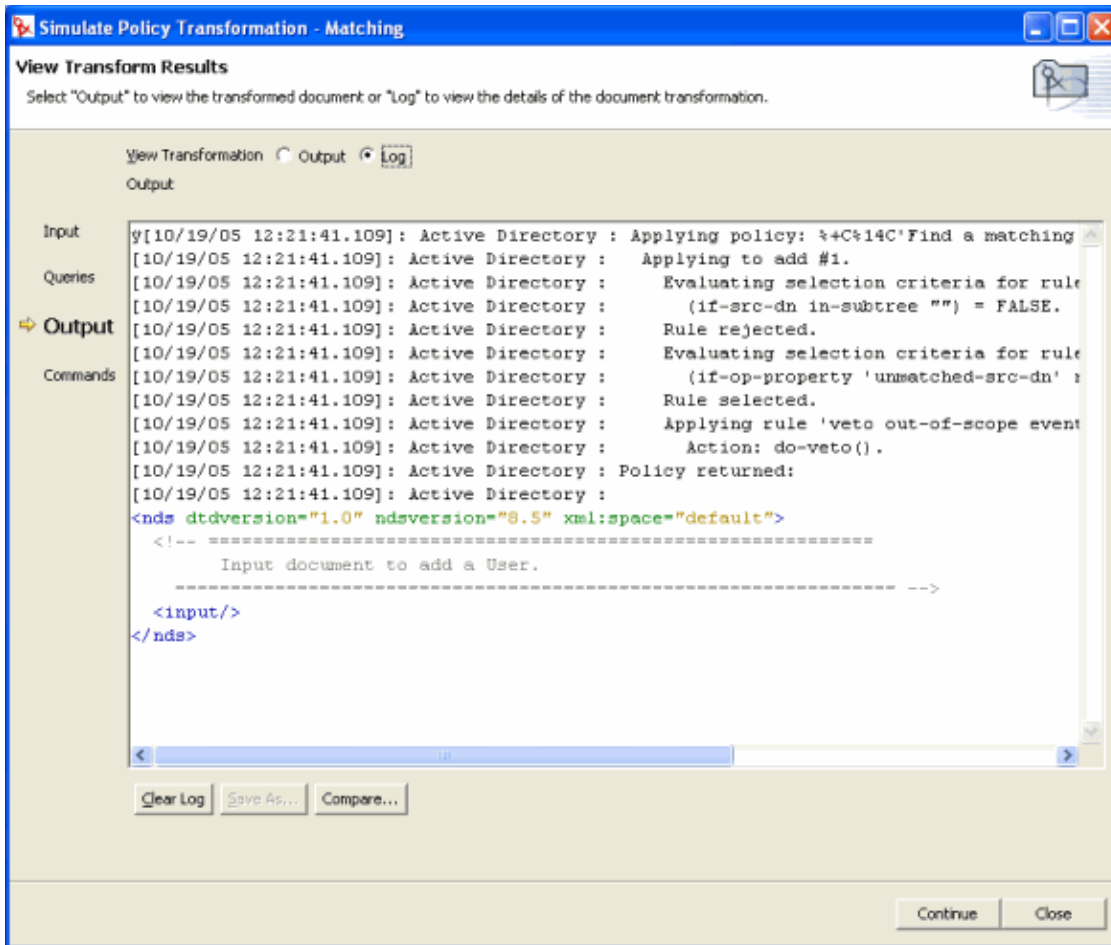


- 4 Select a file, then click *Open*.

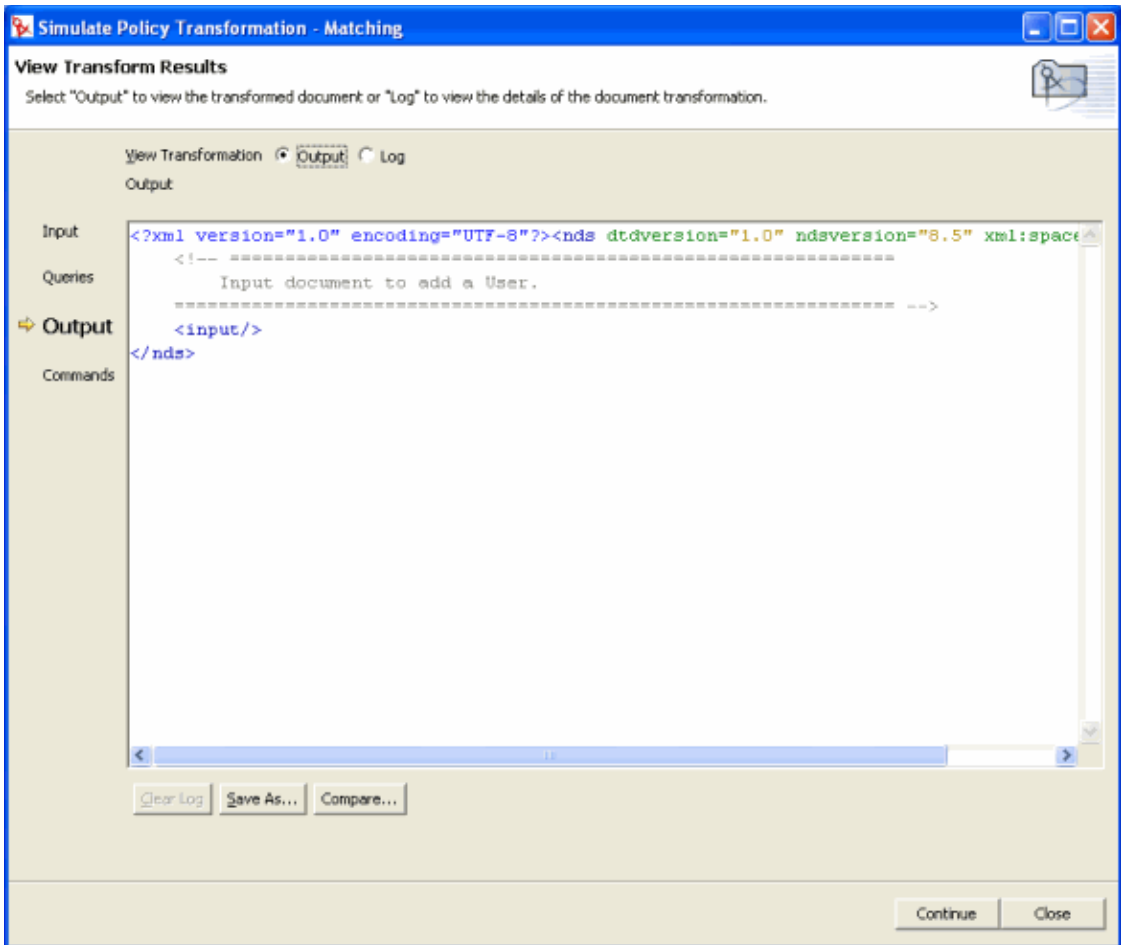
- 5 The input document is now displayed in the window. Click *Start*.



- 6 If you select *View Transformation Log*, you see a trace log of the policy execution. The information in this window is the same information that you see in DSTRACE.

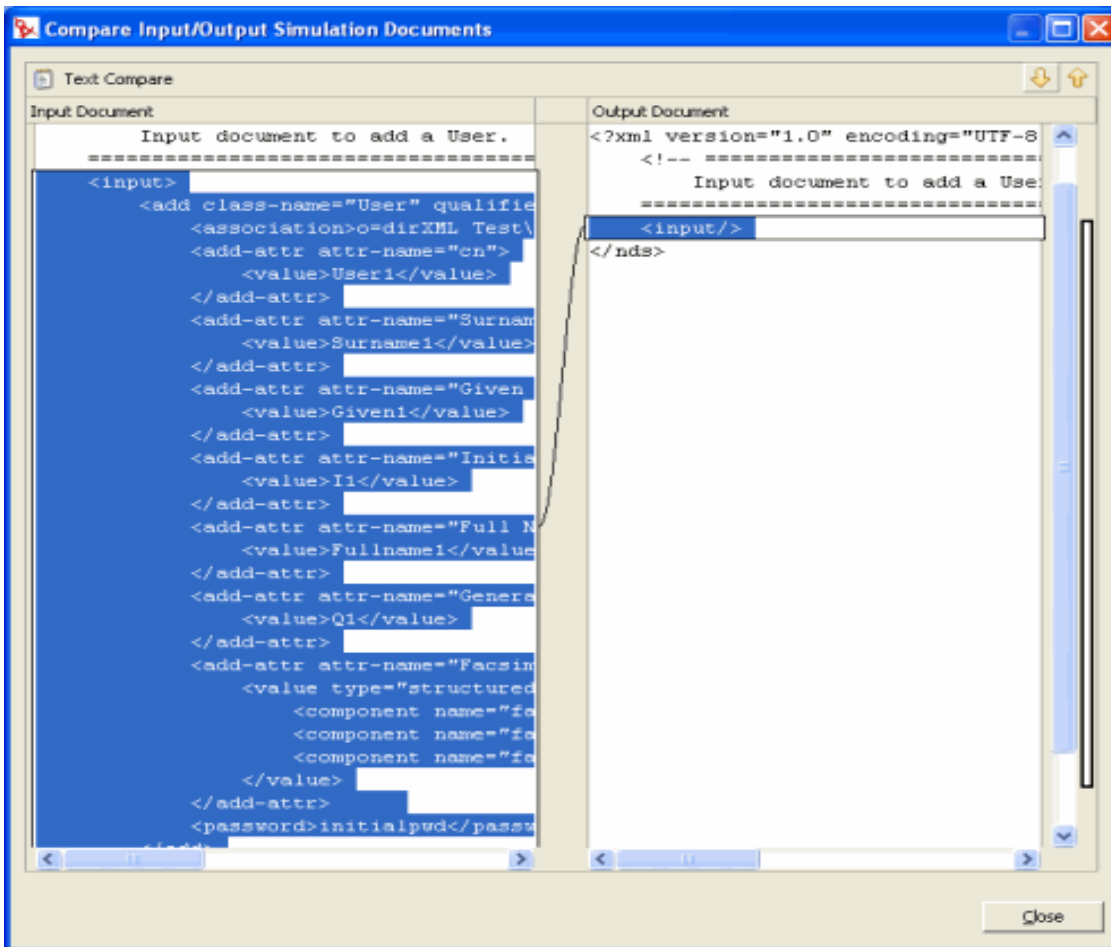


7 If you select *Output*, you see the output document that was generated.



- 8 When you are finished looking at the results, click *Continue* to test another event against the policy.
- 9 Select *Compare* to compare the output document to the input document. The comparison is displayed in a dialog using the Eclipse text compare editor.





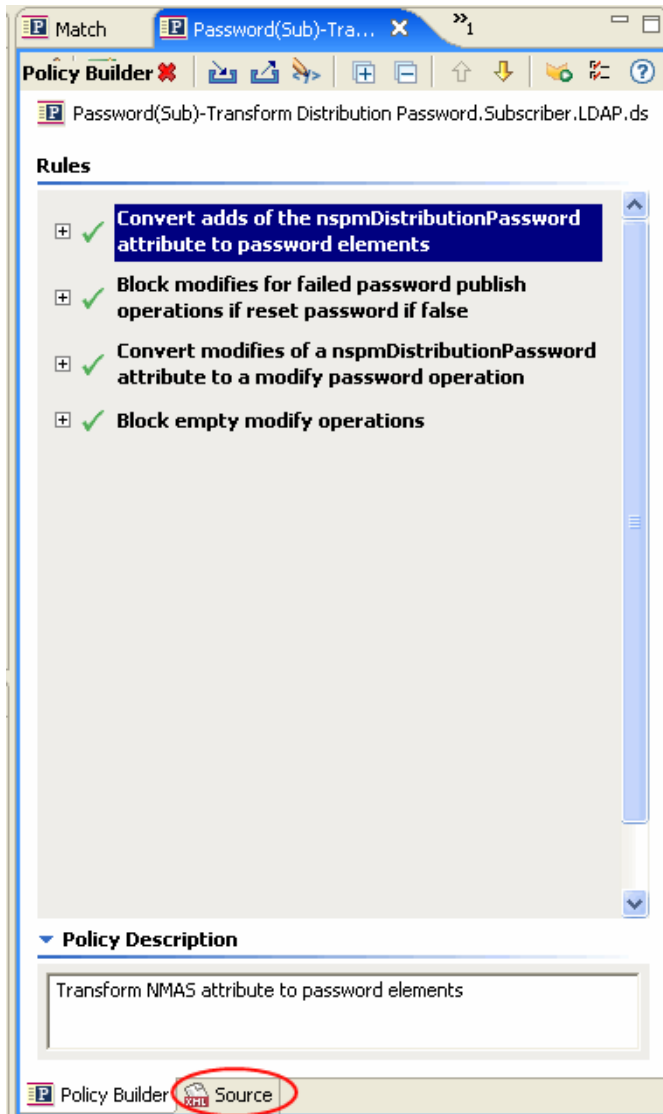
10 When you are finished testing, click *Close* and to close the Policy Simulator.

## 2.2.8 Editing the DirXML Script

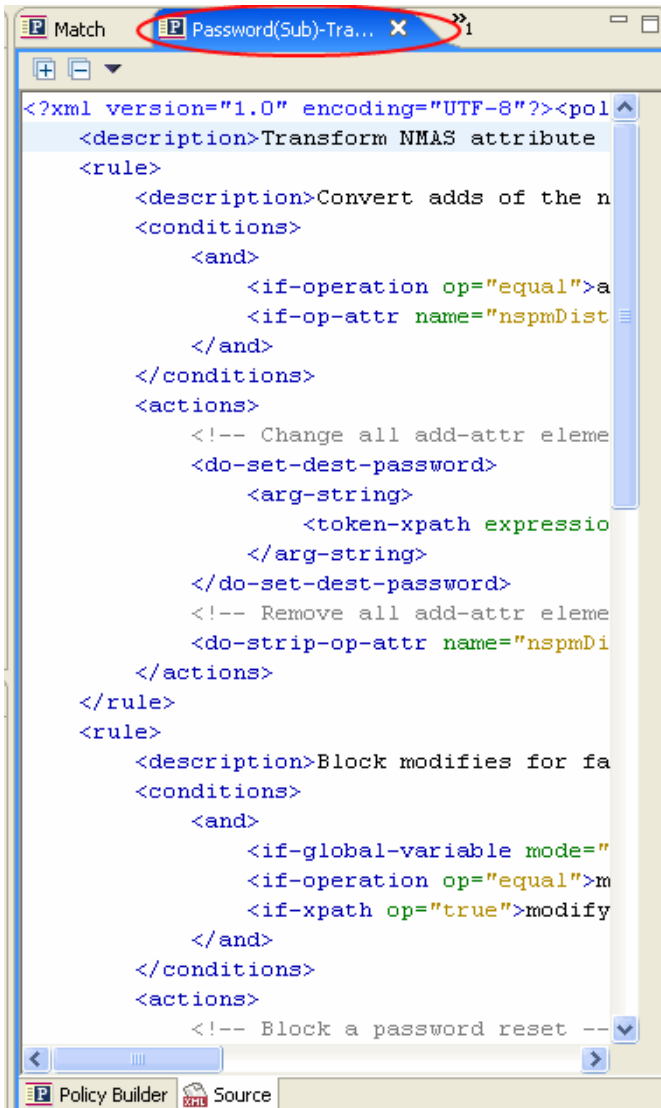
Designer enables you to view and edit the source DirXML Script by using an XML editor or text editor. Designer has a built in XML editor. The Source view also allows you to validate against the document type definition file (DTD). For more information about the DTD see [DirXML Driver Developer Kit Documentation \(http://developer.novell.com/ndk/doc/dirxml/dirxmlbk/ref/index.html\)](http://developer.novell.com/ndk/doc/dirxml/dirxmlbk/ref/index.html).

### Edit XML

- 1 To open the Source view, select Source at the bottom of the Policy Builder.



The DirXML Script is displayed in an XML editor. You can edit the DirXML Script. To see the information without scrolling, double-click the active tab and the current panel expands to fill the entire window. To minimize the view, double-click the active tab.



To select a different XML editor for your Source view:

- 1 From the Main menu, select *Window > Preferences*.
- 2 Select *General > Editors > File Associations*.
- 3 Select *\*.xml* from the list under *File Types*.
- 4 From the list in *Associated Editors*, select the editor that you want. If the editor you want isn't in the list, you might need to click *Add*, then add it to the list.
- 5 Click *OK*.
- 6 Close and reopen the Policy Builder.

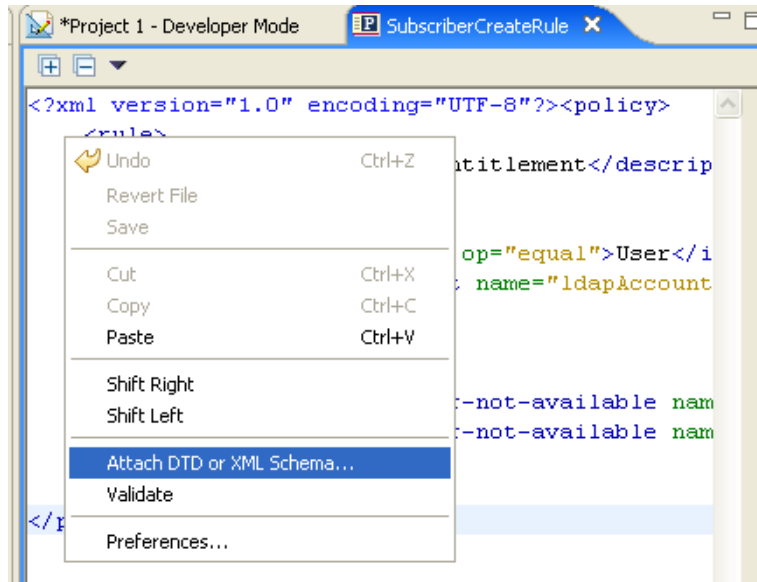
## Validate DTD

You can validate any DTD file with Designer. The following examples uses Novell's DTD. If you validate a policy created with the Policy Builder, use Novell's DTD file. The DTD file is shipped with Designer. The file is located `software\designer\eclipse\plugins\com.novell.soa.xml\resolver_x.x.`

x\xmlcatalog\dtddcatalog\dirxmlscript.dtd where *software* is where you have Designer installed.

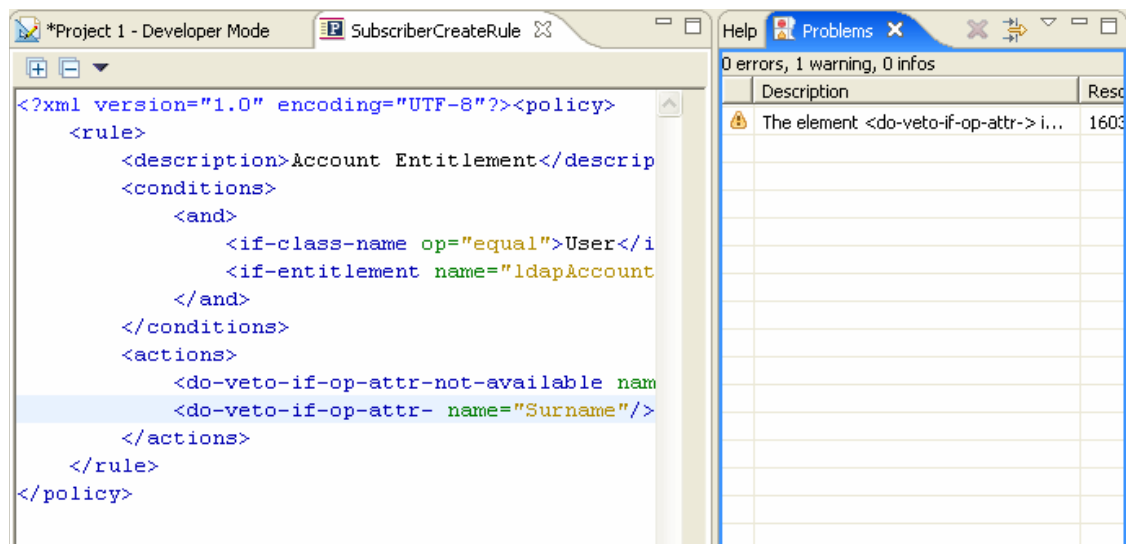
For the DTD validation to work, the policy must be saved. If you have been working on a policy, make sure it is saved before validating.

- 1 Select the policy you want to validate the DTD against in the Policy Builder.
- 2 Click the *Source* tab, then right-click in the Source View > *Attach DTD* or *XML Schema*.



- 3 Browse to the DTD file, then click *Open*. In this example the file is located at C:\Program Files\Novell\designer\eclipse\plugins\com.novell.soa.xml.resolver\_1.0.0\XmlCatalog\DTDCatalog\dirxmlscript.dtd.

- 4 Right-click in the Source View again, then click *Validate*.



The Problems window opens. If there is no error, nothing is listed in the Problems window. If there is an error, information about the error is listed in the window. It explains why, in what project, and on what line the error occurred. Click the error and Designer highlights the line of code where the problem is. You can fix the problem in the XML editor and save the policy.

## 2.3 Regular Expressions

A regular expression is a formula for matching text strings that follow some pattern. Regular expressions are made up of normal characters and metacharacters. Normal characters include uppercase and lowercase letters and digits. Metacharacters have special meanings. The following table contains some of the most common metacharacters and their meanings.

Metacharacter	Description
.	Matches any single character.
\$	Matches the end of the line.
^	Matches the beginning of a line.
*	Matches zero or more occurrences of the character immediately preceding.
\	Literal escape character. It allows you to search for any of the metacharacters. For example \\$ finds \$1000 instead of matching at the end of the line.
[ ]	Matches any one of the characters between the brackets.
[0-9]	Matches a range of characters with the hyphen. The example matches any digit.
[A-Za-z]	Matches multiple ranges as well. The example matches all uppercase and lowercase letters.

The Argument Builder is designed to use regular expressions as defined in Java. The [Java Web site](http://java.sun.com/j2se/1.4.2/docs/api/java/util/regex/Pattern.html) (<http://java.sun.com/j2se/1.4.2/docs/api/java/util/regex/Pattern.html>) contains further information.

## 2.4 XPath 1.0 Expressions

Arguments to some conditions, actions, and tokens use XPath 1.0 expressions. XPath is a language created to provide a common syntax and semantics for functionality shared between XSLT and XPointer. It is used primarily for addressing parts of an XML document, but also provides basic facilities for manipulation of strings, numbers and Booleans.

The XPath specification requires that the embedding application provide a context with several application-defined pieces of information. In DirXML Script (see [Section 1.1.2, “DirXML Script,” on page 15](#)), XPath is evaluated with the following context:

- The context node is the current operation.
- The context position and size are 1.
- There are several available variables:
  - Those available as parameters to style sheets within Identity Manager (currently fromNDS, srcQueryProcessor, destQueryProcessor, srcCommandProcessor, destCommandProcessor, and dnConverter).
  - Global configuration variables.
  - Local policy variables.

- If there is a name conflict between the different variable sources, then the order of precedence is local variable, style sheet parameters, global variables.
- Namespaces are declared on the policy element.
- There are several available functions:
  - All built-in XPath 1.0 functions.
  - Java extension functions as provided by NXSL.

Namespaces declarations to associate a prefix with a Java class must be declared on the policy element.

The [W3 Web site \(http://www.w3.org/TR/1999/REC-xpath-19991116\)](http://www.w3.org/TR/1999/REC-xpath-19991116) contains further information.

## 2.5 Conditions

This section contains detailed information on all conditions available using the Policy Builder interface.

- [Section 2.5.1, “If Association,” on page 102](#)
- [Section 2.5.2, “If Attribute,” on page 103](#)
- [Section 2.5.3, “If Class Name,” on page 104](#)
- [Section 2.5.4, “If Destination Attribute,” on page 105](#)
- [Section 2.5.5, “If Destination DN,” on page 107](#)
- [Section 2.5.6, “If Entitlement,” on page 108](#)
- [Section 2.5.7, “If Global Configuration Value,” on page 109](#)
- [Section 2.5.8, “If Local Variable,” on page 110](#)
- [Section 2.5.9, “If Named Password,” on page 112](#)
- [Section 2.5.10, “If Operation,” on page 112](#)
- [Section 2.5.11, “If Operation Attribute,” on page 113](#)
- [Section 2.5.12, “If Operation Property,” on page 115](#)
- [Section 2.5.13, “If Password,” on page 116](#)
- [Section 2.5.14, “If Source Attribute,” on page 116](#)
- [Section 2.5.15, “If Source DN,” on page 117](#)
- [Section 2.5.16, “If XPath Expression,” on page 118](#)

### 2.5.1 If Association

Performs a test on the association value of the current operation or the current object.

#### Fields

#### Operator Condition is Met When...

Operator	Condition is met when...
associated	There is an established association for the current object.

Operator	Condition is met when...
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 the if association.
not-associated	There is not an established association for the current object.
not available	The association is not available for the current object.
not-equal	The association value specified by the current operation is not equal to the content of the if association.

## Example

This example tests to see if the association is available. When this condition is met, the actions that are defined are executed.

The screenshot shows a dialog box titled "Condition Group 1" with a green checkmark icon. Inside, there is a sub-section with a green checkmark and the text "if association available". Below this, there are two dropdown menus: "Condition" is set to "association" and "Operator \*" is set to "available". A question mark icon is next to the "Condition" dropdown. At the bottom of the dialog are "OK" and "Cancel" buttons.

## 2.5.2 If Attribute

Performs a test on attribute values of the current object in either the current operation or the source data store. It can be logically thought of as If Operation Attribute or If Source Attribute, because the test is satisfied if the condition is met in the source data store or in the operation.

### Fields

#### Name

Specify the name of the attribute to test.

#### Operator

Select the condition test type.

#### Compare Mode

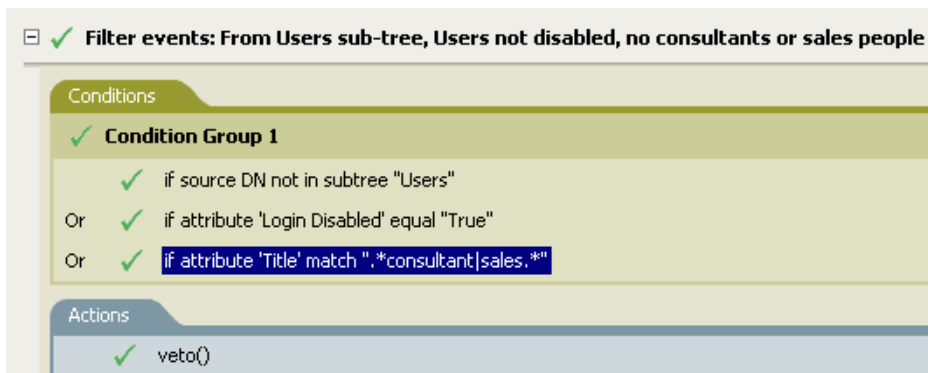
Select the comparison mode. See [“Comparison Modes” on page 180](#).

#### Operator Condition is Met When...

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, which equals the specified value when compared using the specified comparison mode.
not available	Available would return False.
not-equal	Equal would return False.

## Example

The example uses the condition If Attribute when filtering for User objects that are disabled or have a certain title. The policy is Policy to Filter Events, and it is available for download from Novell's support Web site. For more information, see [“Downloadable Identity Manager Policies” on page 36](#).



The close-up shows the configuration for the "if attribute" condition:

- Condition: attribute
- Name: Title
- Operator: equal
- Mode: regular expression
- Value: .\*consultant|sales.\*

The condition is looking for any User object that has an attribute of Title with a value of consultant or sales.

## 2.5.3 If Class Name

Performs a test on the object class name in the current operation.

### Fields

#### Operator

Select the condition test type.



## Compare Mode

Select the comparison mode. See [“Comparison Modes” on page 180](#).

## Operator Condition is Met When...

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.

## Example

The example uses the condition If Class Name to govern group membership for a User object based on their title. The policy is Govern Groups for User Based on Title Attribute, and it is available for download from Novell’s support Web site. For more information, see [“Downloadable Identity Manager Policies” on page 36](#).

The screenshot displays a policy configuration window with the following sections:

- Conditions:**
  - Condition Group 1
    - if class name equal "User"
    - And if destination attribute 'Title' match ".\*manager.\*"
    - And if operation attribute 'Title' not-match ".\*manager.\*"
- Actions:**
  - set destination attribute value("Group Membership", "Users\EmployeesGroup")
  - clone operation attribute("Group Membership", "Security Equals")

The dialog box shows the configuration for a condition:

- Condition: class name
- Operator: equal
- Mode: case insensitive
- Value: User

Checks to see if the class name of the current object is User.

## 2.5.4 If Destination Attribute

Performs a test on attribute values of the current object in the destination data store.

## Fields

### Name

Specify the name of the attribute to test.

### Operator

Select the condition test type.

### Compare Mode

Select the comparison mode. See [“Comparison Modes” on page 180](#).

### Operator Condition is Met When...

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.

## Example

The example uses the condition If Attribute to govern group membership for a User object based on the title. The policy is Govern Groups for User Based on Title Attribute, and it is available for download from Novell’s support Web site. For more information, see [“Downloadable Identity Manager Policies” on page 36](#).

The screenshot displays a policy configuration window titled "User changing from Manager to Employee". It is divided into two main sections: "Conditions" and "Actions".


**Conditions:**


- Condition Group 1
  - if class name equal "User"
  - And if destination attribute 'Title' match ".\*manager.\*"
  - And if operation attribute 'Title' not-match ".\*manager.\*"

**Actions:**

- set destination attribute value("Group Membership", "Users\EmployeesGroup")
- clone operation attribute("Group Membership", "Security Equals")


The interface also shows a summary bar at the bottom: "User changing from Employee to Manager".

Condition destination attribute 

Name \* Title 

Operator \* equal

Mode regular expression

Value .\*manager.\* 

The policy checks to see if the value of the title attribute contains manager.

## 2.5.5 If Destination DN

Performs a test on the destination DN in the current operation.

### Fields

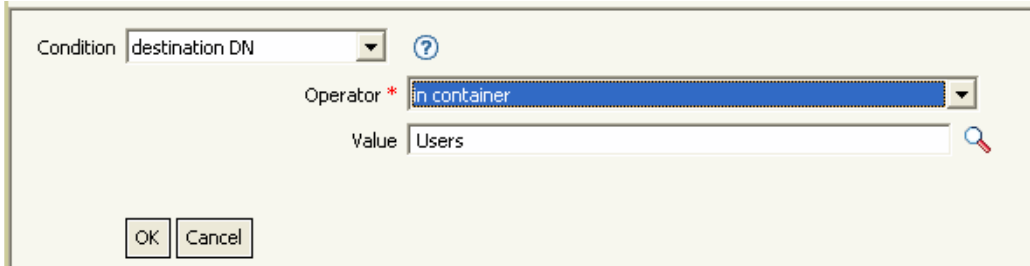
#### Operator

Select the condition test type.

#### Operator Condition is Met When...

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.

## Example



Condition: destination DN

Operator \*: in container

Value: Users

OK Cancel

## 2.5.6 If Entitlement

Performs a test on entitlements of the current object, in either the current operation or the Identity Vault.

### 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 180](#).

#### Operator Condition is Met When...

Operator	Condition is met when...
available	The named entitlement is available in either the current operation or the Identity Vault.
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, which has a value 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 named entitlement. It has a value that 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.

Operator	Condition is met when...
not-equal	Equal would return False.

## Example

## 2.5.7 If Global Configuration Value

Performs a test on a global configuration variable.

### Fields

#### Name

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

#### Operator

Select the condition test type.

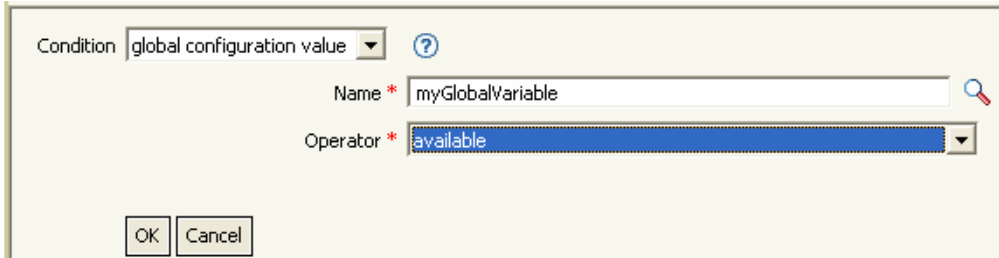
#### Compare Mode

Select the comparison mode. See [“Comparison Modes” on page 180](#).

#### Operator Condition is Met When...

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.

## Example



Condition: global configuration value

Name \*: myGlobalVariable

Operator \*: available

OK Cancel

## 2.5.8 If Local Variable

Performs a test on a local variable.

### 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 180](#).

#### Operator Condition is Met When...

Operator	Condition is met when...
available	There is a local variable with the specified name that has been defined by an action of an 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.

## Example

The example adds a User object to the appropriate group, Employee or Manager, based on Title. It also creates the group, if needed, and sets up security equal to that group. The policy is Govern Groups for User Based on Title Attribute, and it is available for download from Novell’s support Web site. For more information, see [“Downloadable Identity Manager Policies” on page 36](#).

- ⊕ ✓ **Set local variables to test existence of groups and for placement**
- ⊖ ✓ **Create ManagersGroup, if needed**

---

Conditions

- ✓ **Condition Group 1**
  - ✓ if local variable 'manager-group-info' available
- And ✓ if local variable 'manager-group-info' not equal "group"

Actions

- ✓ add destination object(class name="Group", when="before", dn(Local Variable("manager-group-dn")))

- ⊕ ✓ **Create EmployeesGroup, if needed**
- ⊕ ✓ **If Title indicates Manager, add to ManagerGroup and set rights**
- ⊕ ✓ **If Title does not indicate Manager, add to EmployeeGroup and set rights**

The policy contains five rules that are dependent on each other.

- ⊖ ✓ **Set local variables to test existence of groups and for placement**

---

Conditions

- ✓ **Condition Group 1**
  - ✓ if class name equal "User"
- And
- ✓ **Condition Group 2**
  - ✓ if operation equal "add"
- Or ✓ if operation equal "modify"

Actions

- ✓ set local variable("manager-group-dn", "Users\ManagersGroup")
- ✓ set local variable("manager-group-info", Destination Attribute("Object Class", dn(Local Variable("manager-group-dn"))))
- ✓ set local variable("employee-group-dn", "Users\EmployeesGroup")
- ✓ set local variable("employee-group-info", Destination Attribute("Object Class", dn(Local Variable("employee-group-dn"))))

For the If Locate Variable condition to work, the first rule sets four different local variables to test for groups and where to place the groups.

Condition local variable ?

Name \*  🔍

Operator \*  ▼

Mode  ▼

Value  🔍

The condition the rule is looking for is to see if the local variable of manager-group-info is available and if manager-group-info is not equal to group. If these conditions are met, then the destination object of group is added.

## 2.5.9 If Named Password

Performs a test on a password in the current operation with the specified name.

### Fields

#### Name

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

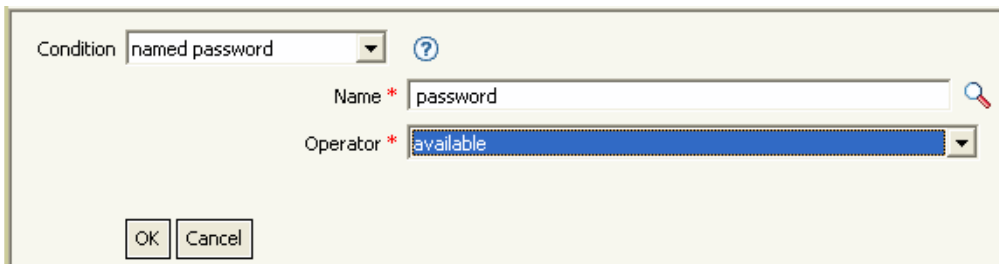
#### Operator

Select the condition test type.

#### Operator Condition is Met When...

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

### Example



Condition: named password

Name \*: password

Operator \*: available

OK Cancel

## 2.5.10 If Operation

Performs a test on the name of the current operation.

### Fields

#### Operator

Select the condition test type.

#### Operator Condition is Met When...

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.



## Example

The example adds a User object to the appropriate group, Employee or Manager, based on Title. It also creates the group, if needed, and sets up security equal to that group. The policy name is Govern Groups for User Based on Title Attribute, and it is available for download from Novell's support Web site. For more information, see [“Downloadable Identity Manager Policies” on page 36](#).

The screenshot shows a policy configuration window with the title "Set local variables to test existence of groups and for placement". It is divided into two main sections: "Conditions" and "Actions".

**Conditions:**

- Condition Group 1:** if class name equal "User"
- And**
- Condition Group 2:**
  - if operation equal "add"
  - Or if operation equal "modify"

**Actions:**

- set local variable("manager-group-dn", "Users\ManagersGroup")
- set local variable("manager-group-info", Destination Attribute("Object Class", dn(Local Variable("manager-group-dn"))))
- set local variable("employee-group-dn", "Users\EmployeesGroup")
- set local variable("employee-group-info", Destination Attribute("Object Class", dn(Local Variable("employee-group-dn"))))

This close-up shows the configuration for a condition of type "operation". It includes a dropdown menu for the "Operator" set to "equal" and a text input field for the "Value" set to "add". There are also help icons and a search icon.

The condition is checking to see if an add or modify operation has occurred. When one of these occurs, it sets the local variables.

## 2.5.11 If Operation Attribute

Performs a test on attribute values in the current operation.

### Fields

#### Name

Specify the name of the attribute to test.

#### Operator

Select the condition test type.

#### Compare Mode

Select the comparison mode. See [“Comparison Modes” on page 180](#).

#### Operator Condition is Met When...

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. It 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. It 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. It 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.

## Example

The example adds a User object to the appropriate group, Employee or Manager, based on Title. It also creates the group, if needed, and sets up security equal to that group. The policy name is Govern Groups for User Based on Title Attribute, and it is available for download from Novell's support Web site. For more information, see [“Downloadable Identity Manager Policies” on page 36](#).

The screenshot displays a policy configuration window with the following structure:

- Policy Name:** Set local variables to test existence of groups and for placement
- Conditions:**
  - Condition Group 1
    - if class name equal "User"
    - And if operation attribute 'Title' match ".\*manager.\*"
- Actions:**
  - set destination attribute value("Group Membership", Local Variable("manager-group-dn"))
  - clone operation attribute("Group Membership", "Security Equals")
- Additional Policy Rule:** If Title does not indicate Manager, add to EmployeeGroup and set rights

Condition: destination attribute ?

Name \*: Title 🔍

Operator \*: equal

Mode: regular expression

Value: .\*manager.\* 🔍

The condition is checking to see if the attribute of Title is equal to .\*manager\*, which is a regular expression. It is looking for a title that has zero or more characters before manager and a single character after manager. It finds a match if the User object's title was sales managers.

## 2.5.12 If Operation Property

Performs a test on an operation property on the current operation.

### Fields

#### Name

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

#### Operator

Select the condition test type.

#### Compare Mode

Select the comparison mode. See [“Comparison Modes” on page 180](#).

#### Operator Condition is Met When...

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.

### Example

Condition: operation property ?

Name \*: myStoredVariable

Operator \*: available

OK Cancel

## 2.5.13 If Password

Performs a test on a password in the current operation.

### Fields

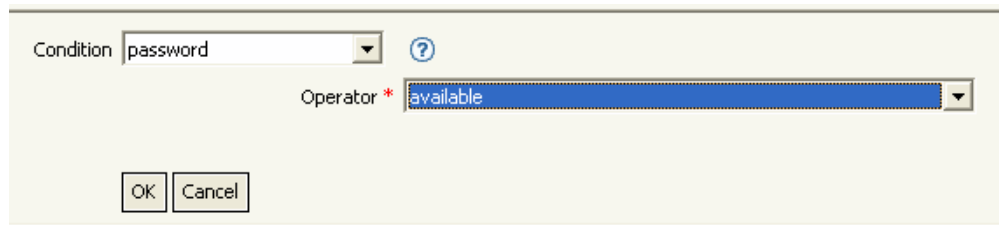
#### Operator

Select the condition test type.

#### Operator Condition is Met When...

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

### Example



The screenshot shows a configuration dialog box for the 'If Password' condition. It features two dropdown menus: 'Condition' is set to 'password' and 'Operator \*' is set to 'available'. A help icon (?) is visible next to the 'Condition' dropdown. At the bottom of the dialog are 'OK' and 'Cancel' buttons.

## 2.5.14 If Source Attribute

Performs a test on attribute values of the current object in the source data store.

### 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 180](#).

#### Operator Condition is Met When...

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. It equals the specified value when compared using the specified comparison mode.

Operator	Condition is met when...
not available	Available would return False.
not-equal	Equal would return False.

### Example

Condition: password [?] Operator\*: available [v]

OK Cancel

## 2.5.15 If Source DN

Performs a test on the source DN in the current operation.

### Fields

#### Operator

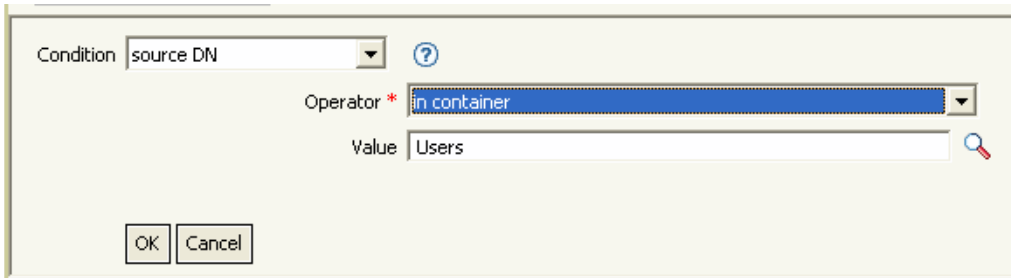
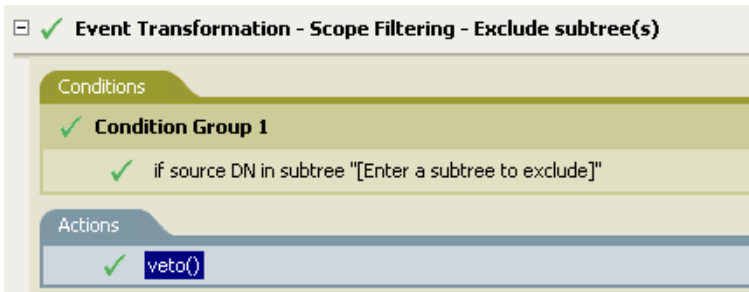
Select the condition test type.

#### Operator Condition is Met When...

Operator	Condition is met when...
available	DN available.
equal	There is a source DN available, and it equals the content of the specified value in-container There is a source DN available, and it represents an object in the container identified by the specified value.
in-subtree	There is a source DN available, and it represents an object in the subtree identified by the specified value.
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.

### Example

The example uses the condition If Source DN to check if the User object is in the source DN. The rule is from the predefined rules that come with Identity Manager 3.0. For more information, see [“Event Transformation - Scope Filtering - Exclude Subtrees” on page 72](#).



The condition is checking to see if the source DN is in the Users container. If the object is coming from that container, it is vetoed.

## 2.5.16 If XPath Expression

Performs a test on the results of evaluating an XPath 1.0 expression.

### Fields

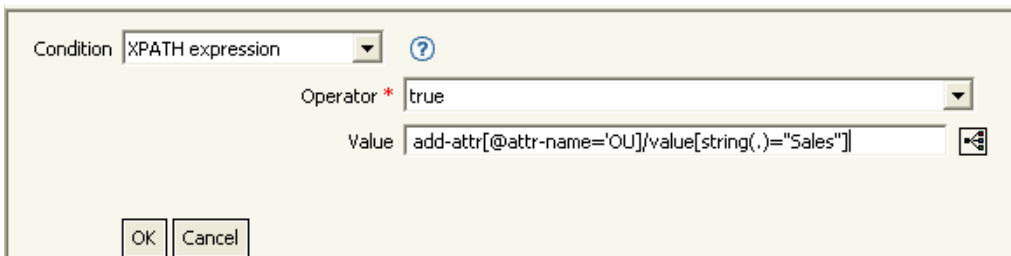
#### Operator

Select the condition test type.

#### Operator Condition is Met When...

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

### Example



## 2.6 Actions

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

- [Section 2.6.1, “Add Association,” on page 120](#)
- [Section 2.6.2, “Add Destination Attribute Value,” on page 120](#)
- [Section 2.6.3, “Add Destination Object,” on page 122](#)
- [Section 2.6.4, “Add Source Attribute Value,” on page 123](#)
- [Section 2.6.5, “Add Source Object,” on page 124](#)
- [Section 2.6.6, “Append XML Element,” on page 125](#)
- [Section 2.6.7, “Append XML Text,” on page 125](#)
- [Section 2.6.8, “Break,” on page 126](#)
- [Section 2.6.9, “Clear Destination Attribute Value,” on page 126](#)
- [Section 2.6.10, “Clear Operation Property,” on page 127](#)
- [Section 2.6.11, “Clear Source Attribute Value,” on page 127](#)
- [Section 2.6.12, “Clone By XPath Expressions,” on page 128](#)
- [Section 2.6.13, “Clone Operation Attribute,” on page 128](#)
- [Section 2.6.14, “Delete Destination Object,” on page 129](#)
- [Section 2.6.15, “Delete Source Object,” on page 130](#)
- [Section 2.6.16, “Find Matching Object,” on page 130](#)
- [Section 2.6.17, “For Each,” on page 132](#)
- [Section 2.6.18, “Generate Event,” on page 133](#)
- [Section 2.6.19, “Implement Entitlement,” on page 135](#)
- [Section 2.6.20, “Move Destination Object,” on page 136](#)
- [Section 2.6.21, “Move Source Object,” on page 137](#)
- [Section 2.6.22, “Reformat Operation Attribute,” on page 138](#)
- [Section 2.6.23, “Remove Association,” on page 139](#)
- [Section 2.6.24, “Remove Destination Attribute Value,” on page 139](#)
- [Section 2.6.25, “Remove Source Attribute Value,” on page 140](#)
- [Section 2.6.26, “Rename Destination Object,” on page 141](#)
- [Section 2.6.27, “Rename Operation Attribute,” on page 142](#)
- [Section 2.6.28, “Rename Source Object,” on page 142](#)
- [Section 2.6.29, “Send Email,” on page 143](#)
- [Section 2.6.30, “Send Email From Template,” on page 144](#)
- [Section 2.6.31, “Set Default Attribute Value,” on page 145](#)
- [Section 2.6.32, “Set Destination Attribute Value,” on page 146](#)
- [Section 2.6.33, “Set Destination Password,” on page 148](#)
- [Section 2.6.34, “Set Local Variable,” on page 148](#)
- [Section 2.6.35, “Set Operation Association,” on page 149](#)

- Section 2.6.36, “Set Operation Class Name,” on page 150
- Section 2.6.37, “Set Operation Destination DN,” on page 150
- Section 2.6.38, “Set Operation Property,” on page 151
- Section 2.6.39, “Set Operation Source DN,” on page 151
- Section 2.6.40, “Set Operation Template DN,” on page 152
- Section 2.6.41, “Set Source Attribute Value,” on page 153
- Section 2.6.42, “Set Source Password,” on page 154
- Section 2.6.43, “Set XML Attribute,” on page 154
- Section 2.6.44, “Status,” on page 155
- Section 2.6.45, “Strip Operation Attribute,” on page 156
- Section 2.6.46, “Strip XPath,” on page 156
- Section 2.6.47, “Trace Message,” on page 157
- Section 2.6.48, “Veto,” on page 158
- Section 2.6.49, “Veto If Operational Attribute Not Available,” on page 159

## 2.6.1 Add Association

Sends an add association command to the Identity Vault, with the specified association.

### Fields

#### Mode

Select whether this actions should be added to the current operation, or written directly to the Identity Vault.

#### DN

Specify the DN of the target object or leave blank to use the current object.

#### Association

Specify the value of the association to be added.

### Example

## 2.6.2 Add Destination Attribute Value

Adds a value to an attribute on an object in the destination data store.



## Fields

### Attribute Name

Specify the name of the attribute.

### Class Name

(Optional) Specify the class name of the target object. Leave blank to use the class name from the current object.

### 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. This object can be the current object, or be specified by a DN or an association.

### Value Type

Select the syntax of the attribute value to be added.

### Value

Specify the attribute value to be added.

## Example

The example adds the destination attribute value to the OU attribute. It creates the value from the local variables that are created. The rule is from the predefined rules that come with Identity Manager 3.0. For more information, see [“Command Transformation - Create Departmental Container - Part 1 and Part 2”](#) on page 62.

☐ ✓ **Command Transformation - Create Departmental Container - Part 1**

Conditions

- ✓ **Condition Group 1**
  - ✓ if operation equal "add"

Actions

- ✓ set local variable("target-container", Destination DN(length="-2"))
- ✓ set local variable("does-target-exist", Destination Attribute("objectclass", class name="OrganizationalUnit", dn(Local Variable("target-container"))))

☐ ✓ **Command Transformation - Create Departmental Container - Part 2**

Conditions

- ✓ **Condition Group 1**
  - ✓ if local variable 'does-target-exist' available
  - And ✓ if local variable 'does-target-exist' equal ""

Actions

- ✓ add destination object(class name="organizationalUnit", direct="true", dn(Local Variable("target-container")))
- ✓ add destination attribute value("ou", direct="true", dn(Local Variable("target-container")), Parse DN("dest-dn", "dot", length="1", start="-1", Local Variable("target-container")))

Do **add destination attribute value** ?

Enter attribute name: \*  🔍

Enter class name:

Select mode:

Select object:

Enter DN: \*  📄

Enter value type:

Enter string: \*  📄

## 2.6.3 Add Destination Object

Creates a new object in the destination data store.

### Fields

#### Class Name

Specify the class name of the object to be created.

#### 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 object to be created.

### Remarks

Any attribute values to be added as part of the object creation must be done in subsequent [Section 2.6.2, “Add Destination Attribute Value,” on page 120](#) actions using the same DN.

### Example

The example creates the department container that is needed. The rule is from the predefined rules that come with Identity Manager 3.0. For more information, see [“Command Transformation - Create Departmental Container - Part 1 and Part 2” on page 62](#).

📄 ✓ **Command Transformation - Create Departmental Container - Part 1**

Conditions

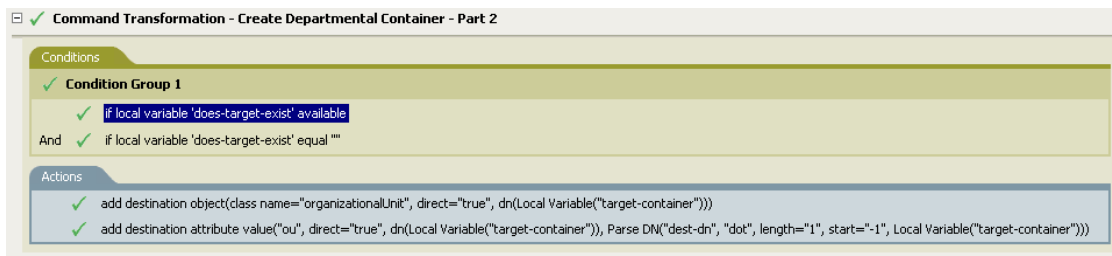
✓ **Condition Group 1**

✓ if operation equal "add"

Actions

✓ set local variable("target-container", Destination DN(length="-2"))

✓ set local variable("does-target-exist", Destination Attribute("objectclass", class name="OrganizationalUnit", dn(Local Variable("target-container"))))



The Organizational Unit object is created. The value for the OU attribute is created from the destination attribute value action that occurs after this action.

## 2.6.4 Add Source Attribute Value

Adds a value to an attribute on an object in the source data store.

### Fields

#### Attribute Name

Specify the name of the attribute.

#### Class Name

(Optional) Specify the class name of the target object. Leave blank to use the class name from the current object.

#### Object

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

#### Value Type

Select the syntax of the attribute value to be added.

#### Value

Specify the attribute value to be added.

## Example

The screenshot shows a dialog box titled "add source attribute value". It contains the following fields and controls:

- Do: add source attribute value (dropdown menu)
- Enter attribute name: \* Title (text input)
- Enter class name: User (text input)
- Select object: Current object (dropdown menu)
- Enter value type: string (dropdown menu)
- Enter string: \* Manager (text input)
- OK and Cancel buttons at the bottom left.

## 2.6.5 Add Source Object

Creates 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 \(page 123\)](#) actions using the same DN.

### Fields

#### Class Name

Specify the class name of the object to be added.

#### DN

Specify the DN of the object to be added.

## Example

The screenshot shows a sequence of three dialog boxes:

- add source object**:
  - Do: add source object (dropdown menu)
  - Enter class name: \* User (text input)
  - Enter DN: \* Users\John Smith (text input)
  - OK and Cancel buttons at the bottom left.
- A confirmation message: `add source attribute value("Title", class name="User", "Manager")` with a green checkmark icon.
- add source attribute value**:
  - Do: add source attribute value (dropdown menu)
  - Enter attribute name: \* Title (text input)
  - Enter class name: User (text input)
  - Select object: Current object (dropdown menu)
  - Enter value type: string (dropdown menu)
  - Enter string: \* "Manager" (text input)
  - OK and Cancel buttons at the bottom left.

## 2.6.6 Append XML Element

Appends an element to a set of elements selected by an XPath expression.

### Fields

#### Name

Specify the tag name of the XML element. This name can contain a namespace prefix if the prefix has been previously defined in this policy.

#### XPATH Expression

Specify an XPath 1.0 expression that returns a node set containing the elements to which the new elements should be appended.

### Example

The screenshot displays two configuration windows from the Policy Builder. The top window is titled 'append XML element'. It features a dropdown menu set to 'append XML element' with a help icon. Below this, there are two input fields: 'Enter name: \*' with the value 'jdbc:sql' and 'Enter XPATH expression: \*' with the value '..//jdbc:statement[last()]'. At the bottom of this window are 'OK' and 'Cancel' buttons. The bottom window is titled 'append XML text'. It also has a dropdown menu set to 'append XML text' with a help icon. It contains two input fields: 'Enter XPATH expression: \*' with the value '..//jdbc:statement[last()]/jdbc:sql' and 'Enter string: \*' with the value 'UPDATE dixml.emp SET fname'+Operation Attribute('Member'. At the bottom of this window are 'OK' and 'Cancel' buttons. A horizontal line separates the two windows, with the text 'Define new action below' centered above the second window.

## 2.6.7 Append XML Text

Appends text to a set of elements selected by an XPath expression.

### Fields

#### XPATH Expression

XPath 1.0 expression that returns a node set containing the elements to which the text should be appended.

#### String

Specify the text to be appended.

## Example

The image shows two dialog boxes for defining new actions. The first dialog box is titled "Define new action below" and has a dropdown menu set to "append XML element". It contains a "Do" dropdown with a question mark icon, an "Enter name:" field with the value "jdbc:sql", and an "Enter XPATH expression:" field with the value "../jdbc:statement[last()]". There are "OK" and "Cancel" buttons at the bottom. The second dialog box is also titled "Define new action below" and has a dropdown menu set to "append XML text". It contains a "Do" dropdown with a question mark icon, an "Enter XPATH expression:" field with the value "../jdbc:statement[last()]/jdbc:sql", and an "Enter string:" field with the value "'UPDATE dixml.emp SET fname"+Operation Attribute("Member". There are "OK" and "Cancel" buttons at the bottom.

## 2.6.8 Break

Ends processing of the current operation by the current policy.

### Example

The image shows a dialog box for defining a new action. The "Do" dropdown menu is set to "break". There are "OK" and "Cancel" buttons at the bottom.

## 2.6.9 Clear Destination Attribute Value

Removes the all values of an attribute from an object in the destination data store.

### Fields

#### Attribute Name

Specify the name of the attribute.

#### Class Name

(Optional) Specify the class name of the target object. Leave blank to use the class name from the current object.

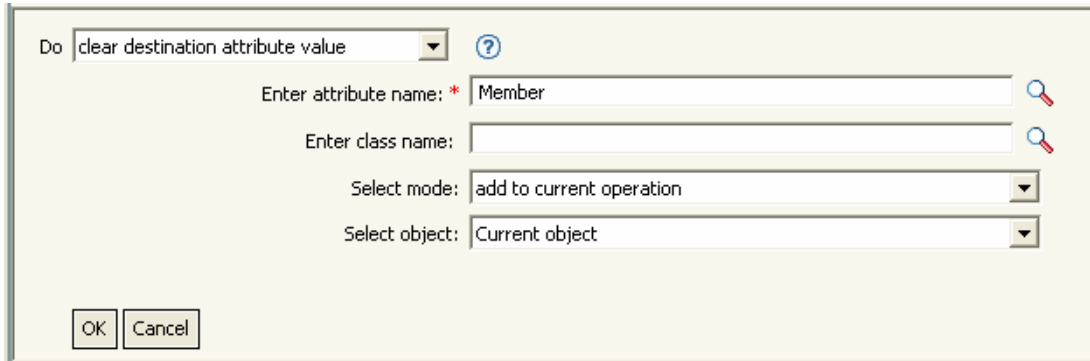
#### Mode

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. This object can be the current object, or be specified by a DN or an association.

### Example



The screenshot shows a dialog box titled "Do" with a dropdown menu set to "clear destination attribute value" and a help icon. Below the title bar, there are four input fields: "Enter attribute name: \*" with the value "Member", "Enter class name:" (empty), "Select mode:" with a dropdown menu set to "add to current operation", and "Select object:" with a dropdown menu set to "Current object". At the bottom left, there are "OK" and "Cancel" buttons.

## 2.6.10 Clear Operation Property

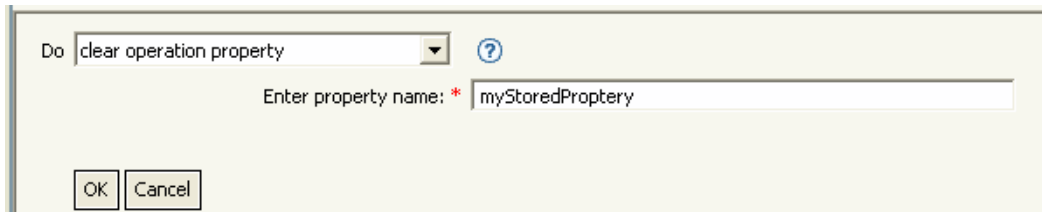
Clears any operation property the current operation.

### Fields

#### Property Name

Specify the name of the operation property to clear.

### Example



The screenshot shows a dialog box titled "Do" with a dropdown menu set to "clear operation property" and a help icon. Below the title bar, there is one input field: "Enter property name: \*" with the value "myStoredProptery". At the bottom left, there are "OK" and "Cancel" buttons.

## 2.6.11 Clear Source Attribute Value

Removes the all values of an attribute from an object in the source data store.

### Fields

#### Attribute Name

Specify the name of the attribute.

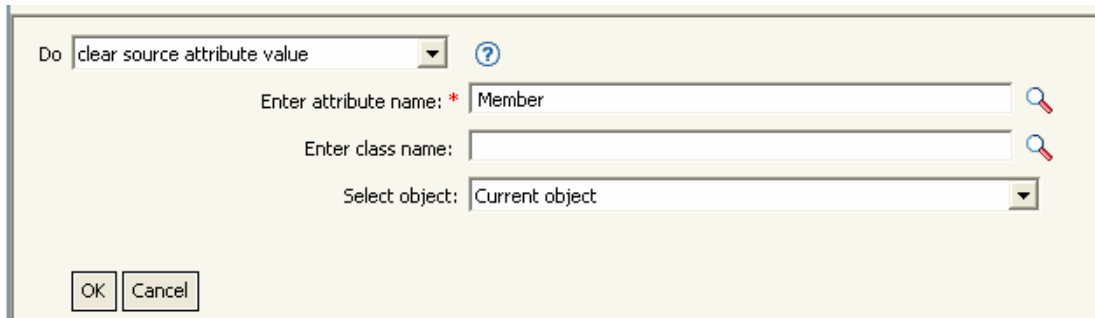
#### Class Name

(Optional) Specify the class name of the target object. Leave blank to use the class name from the current object.

#### Object

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

### Example



The dialog box has a title bar and a 'Do' dropdown menu set to 'clear source attribute value'. To the right of the dropdown is a help icon. Below the dropdown are three input fields: 'Enter attribute name: \*' with the value 'Member', 'Enter class name:' which is empty, and 'Select object:' with a dropdown menu set to 'Current object'. Each input field has a search icon to its right. At the bottom left are 'OK' and 'Cancel' buttons.

## 2.6.12 Clone By XPath Expressions

Appends deep copies of a set of XML nodes selected by an XPath expression to a set of elements selected by another XPath expression.

### Fields

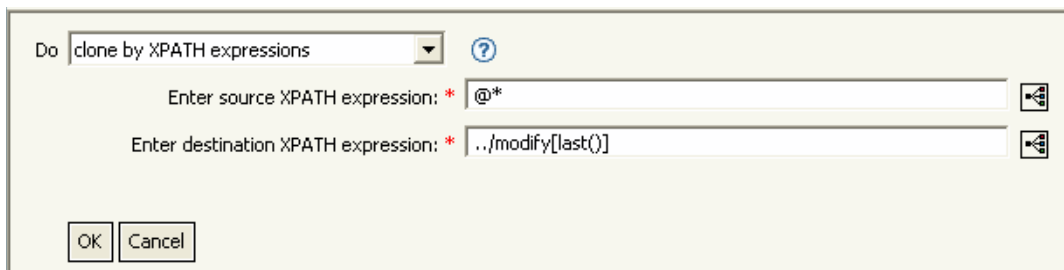
#### Source XPath Expression

Specify the XPath 1.0 expression that returns the node set containing the nodes to be copied.

#### Destination XPath Expression

Specify the XPath 1.0 expression that returns a node set containing the elements to which the copied nodes are to be appended.

### Example



The dialog box has a title bar and a 'Do' dropdown menu set to 'clone by XPATH expressions'. To the right of the dropdown is a help icon. Below the dropdown are two input fields: 'Enter source XPath expression: \*' with the value '@\*' and 'Enter destination XPath expression: \*' with the value '../modify[last()]'. Each input field has a search icon to its right. At the bottom left are 'OK' and 'Cancel' buttons.

## 2.6.13 Clone Operation Attribute

Copies all occurrences of an attribute within the current operation to a different attribute within the current operation.

### Fields

#### Source Name

Specify the name of the attribute to be copied from.



## Destination Name

Specify the name of the attribute to be copied to.

## Example

The example adds a User object to the appropriate group, Employee or Manager, based on Title. It also creates the group, if needed, and sets up security equal to that group. The policy is Govern Groups for User Based on Title Attribute, and it is available for download from Novell's support Web site. For more information, see [“Downloadable Identity Manager Policies” on page 36](#).

The screenshot displays a policy configuration window with the following elements:

- Four main steps, each with a checkmark and a plus icon:
  - Set local variables to test existence of groups and for placement** (highlighted in blue)
  - Create ManagersGroup, if needed**
  - Create EmployeesGroup, if needed**
  - If Title indicates Manager, add to ManagerGroup and set rights** (expanded)
- Conditions** section:
  - Condition Group 1** (expanded)
    - if class name equal "User"
    - And if operation attribute 'Title' match ".\*manager.\*"
- Actions** section:
  - set destination attribute value("Group Membership", Local Variable("manager-group-dn"))
  - clone operation attribute("Group Membership", "Security Equals")
- Bottom step: **If Title does not indicate Manager, add to EmployeeGroup and set rights** (collapsed)

Do

Enter source name: \*

Enter destination name:

The Clone Operation Attribute is taking the information from the Group Membership attribute and adding that to the Security Equals attribute so the values are the same.

## 2.6.14 Delete Destination Object

Deletes an object in the destination data store.

### Fields

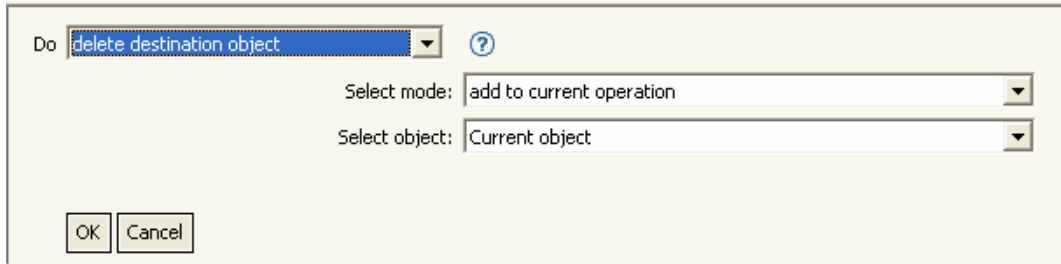
### Mode

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. This object can be the current object, or be specified by a DN or an association.

## Example



The screenshot shows a dialog box titled "Delete Destination Object". It features a "Do" dropdown menu with "delete destination object" selected. To the right of the dropdown is a question mark icon. Below the dropdown are two more dropdown menus: "Select mode:" with "add to current operation" selected, and "Select object:" with "Current object" selected. At the bottom left are "OK" and "Cancel" buttons.

## 2.6.15 Delete Source Object

Deletes an object in the source data store.

### Fields

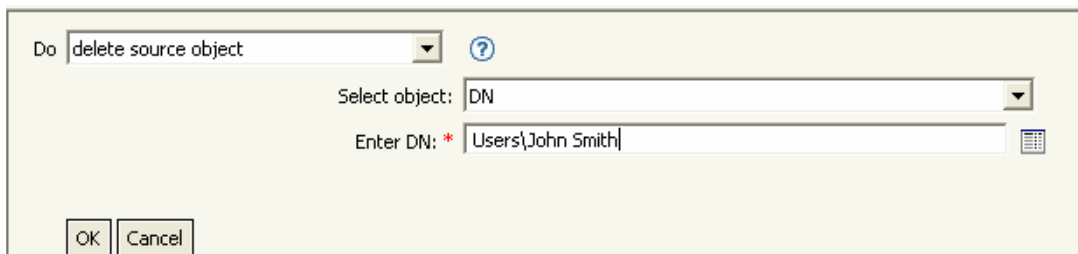
#### Mode

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

## Example



The screenshot shows a dialog box titled "Delete Source Object". It features a "Do" dropdown menu with "delete source object" selected. To the right of the dropdown is a question mark icon. Below the dropdown are two input fields: "Select object:" with "DN" selected in a dropdown, and "Enter DN: \*" with "Users\John Smith" entered in a text box. At the bottom left are "OK" and "Cancel" buttons.

## 2.6.16 Find Matching Object

Finds a match for the current object in the destination data store.

### Fields

#### Scope

Select the scope of the search. The scope might be an entry, a subordinates, or a subtree.

#### DN

Specify the DN that is the base of the search.

#### Match Attributes

Specify the attribute values to search for.

## Remarks

Find Matching Object is only valid when the current operation is an add.

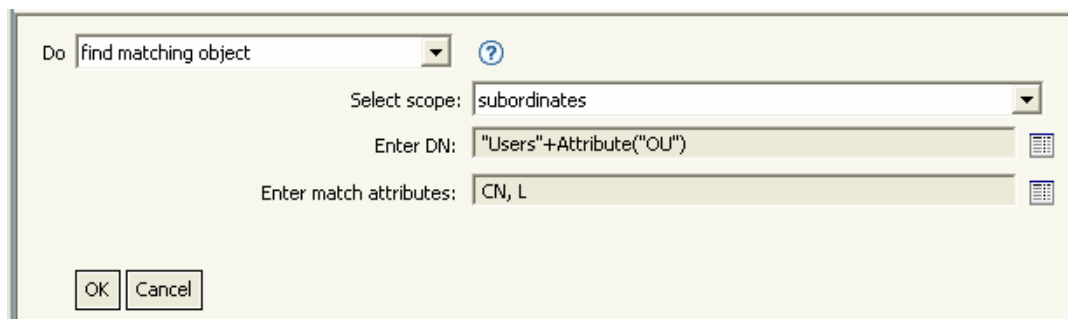
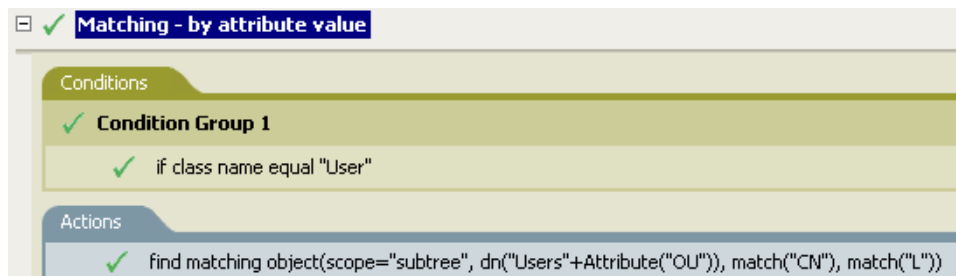
The DN argument is required when scope is “entry”, and is optional otherwise. At least one match attribute is required when scope is “subtree” or “subordinates”.

The results are undefined if scope is entry and there are match attributes specified. If the destination data store is the connected application, then an association is added to the current operation for each successful match that is returned. No query is 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 the Identity Vault, 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 `&#x0000`; If multiple results are returned, then the destination DN of the current operation is set to the single character `&#x0000`;

## Example

The example matches on Users objects using the attributes CN and L. The location where the rule is searching starts at the Users container and adds the information stored in the OU attribute to the DN. The rule is from the predefined rules that come with Identity Manager 3.0. For more information, see [“Matching - By Attribute Value” on page 79](#).



When you click the Argument Builder icon, the Match Attribute Builder comes up. You specify the attribute you want to match in the builder. This examples uses the CN and L attributes.

## 2.6.17 For Each

Repeats a set of actions for each node in a node set.

### Fields

#### Node Set

Specify the node set.

#### Action

Specify the actions to perform on each node in the node set.

### Remarks

The current node is a different value for each iteration of the actions, if a local variable is used.

If a node in the node set is an entitlement, then the for each implicitly performs an **“Implement Entitlement”** on page 135 action.

### Example

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

**Action List** + ✖ ⬆ ⬇ ?

---

✓ Do add destination attribute value ?

Enter attribute name: \* Member 🔍

Enter class name: Group 🔍

Select mode: add to current operation

Select object: DN

Enter DN: \* Local Variable("current-node") 📄

Enter value type: string

Enter string: \* Destination DN() 📄

\* Required

## 2.6.18 Generate Event

Sends a user-defined event to Novell Audit.

### Fields

#### ID

Specify the ID of the event. The ID must be an integer in the range of 1000-1999.

#### Level

Select the level of the event.

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

### Strings

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

String Name	Description
target	The object being acted upon.

String Name	Description
target-type	Integer specifying a predefined format for the target. Predefined values for target-type are currently: <ul style="list-style-type: none"> <li>• 0 = None</li> <li>• 1 = Slash Notation</li> <li>• 2 = Dot Notation</li> <li>• 3 = LDAP Notation</li> </ul>
subTarget	The subcomponent of the target being acted upon.
text1	Text entered here is stored in the text1 event field.
text2	Text entered here is stored in the text2 event field.
text3	Text entered here is stored in the text3 event field.
value	Any number entered here is stored in the value event field.
value3	Any number entered here is stored in the value3 event field.
data	Data entered here is stored in the blob event field.

### Remarks

The Novell Audit event structure contains a target, a subTarget, three strings (text1, text2, text3), two integers (value, value3), and a generic field (data). The text fields are limited to 256 bytes, and the data field can contain up to 3 KB of information, unless a larger data field is enabled in your environment.

### Example

The example has four rules that implement a placement policy for User objects based on the first character of the Surname attribute. It generates both a trace message and a custom Novell Audit event. The Generate Event action is used to send an event Novell Audit. The policy name is Policy to Place by Surname, and it is available for download from Novell's support Web site. For more information [“Downloadable Identity Manager Policies” on page 36](#).

**Setup Local Variables**  
 **Surname A-I: place in Users1**

**Conditions**  
 **Condition Group 1**  
 if class name equal "User"  
 And  if operation attribute 'Surname' match "[a-i].\*"

**Actions**  
 set operation destination DN(dn("Training\Users\Active\Users1"+"\"+Operation Attribute("CN")))
   
 trace message(color="yellow", Local Variable("LVUsers1"))
   
 generate event(id="1000", text1=Local Variable("LVUsers1"))

**Surname J-R: place in Users2**  
 **Surname S-Z: place in Users3**

Do  ?

Enter ID: \*

Select level:

Enter strings:

The following is an example of the Named String Builder being used to provide the strings argument.

Name	String Value
<input type="text" value="text1"/>	<input lvusers1")"="" type="text" value="Local Variable("/>

Generate Event is creating an event with the ID 1000 and displaying the text that is generated by the local variable of LVUser1. The local variable LVUser1 is the string of User:Operation Attribute “cn” +” added to the “+”Training\Users\Active\Users1”+” container”. The event reads User:jsmith added to the Training\Users\Active\Users1 container.

## 2.6.19 Implement Entitlement

Designates actions that implement an entitlement so that the status of those entitlements might be reported to the agent that granted or revoked the entitlement.

### Fields

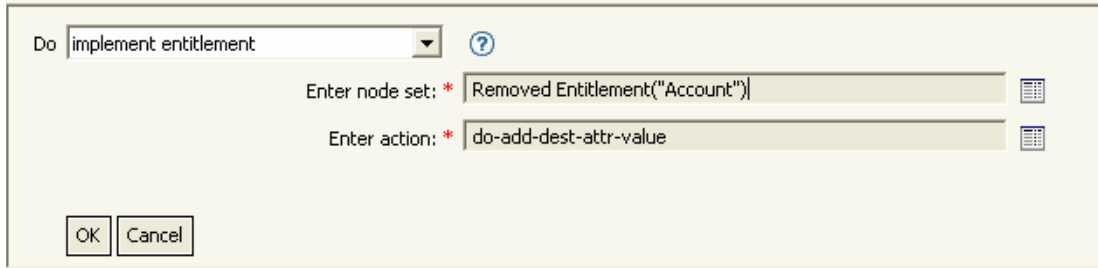
#### Node Set

Node set containing the entitlements being implemented by the specified actions.

#### Action

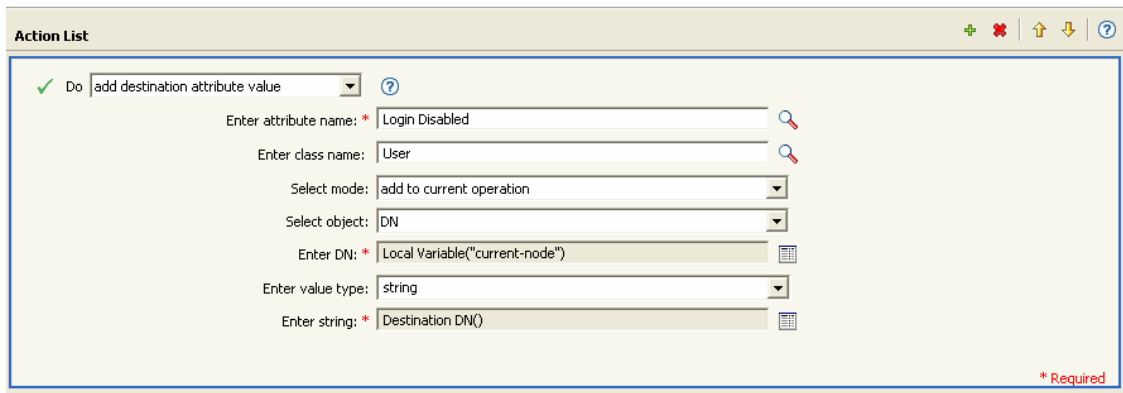
Actions that implement the specified entitlements.

## Example



The screenshot shows a configuration window for an action. At the top, there is a dropdown menu labeled "Do" with the value "implement entitlement" and a help icon. Below this are two input fields: "Enter node set: \*" with the value "Removed Entitlement('Account')" and "Enter action: \*" with the value "do-add-dest-attr-value". At the bottom left, there are "OK" and "Cancel" buttons.

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



The screenshot shows the "Action List" window. It contains a configuration for an action. The "Do" dropdown is set to "add destination attribute value". The configuration includes: "Enter attribute name: \*" with "Login Disabled", "Enter class name: \*" with "User", "Select mode:" with "add to current operation", "Select object:" with "DN", "Enter DN: \*" with "Local Variable('current-node')", "Enter value type:" with "string", and "Enter string: \*" with "Destination DN()". A red asterisk and the text "\* Required" are visible in the bottom right corner.

## 2.6.20 Move Destination Object

Moves an object in the destination data store.

### Fields

#### Mode

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

#### Class Name

(Optional) Specify the class name of the object to be moved. Leave blank to use the class name from the current object.

#### Object to Move

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

#### Container to Move to

Select the target container. This container is specified by a DN or an association.

## Example

The example contains a single rule that disables a user's account and moves it to a disabled container when the Description attribute indicates the user is terminated. The policy is named



Disable User Account and Move When Terminated, and it is available for download from Novell's support Web site. For more information, see ["Downloadable Identity Manager Policies"](#) on page 36.

The screenshot shows a policy configuration window with a title bar that reads "On Termination, disable user and move to Disabled container". The window is divided into two main sections: "Conditions" and "Actions".

**Conditions:**

- Condition Group 1
  - if operation equal "modify"
  - And if class name equal "User"
  - And if operation attribute 'Description' match "^terminated.\*"

**Actions:**

- set destination attribute value("Login Disabled", direct="true", "True")
- move destination object(when="after", dn("Users\Disabled"))

The screenshot shows the "Do" section of the policy configuration. It includes a dropdown menu for "Do" set to "move destination object". Below this are several configuration fields:

- Select mode: add after current operation
- Select object to move: Current object
- Select container to move to: DN
- Enter DN: \* "Users\Disabled"

The policy checks to see if it is a modify event on a User object and if the attribute Description contains the value of terminated. If that is the case, then it sets the attribute of Login Disabled to true and moves the object to the User\Disabled container.

## 2.6.21 Move Source Object

Moves an object in the source data store.

### Fields

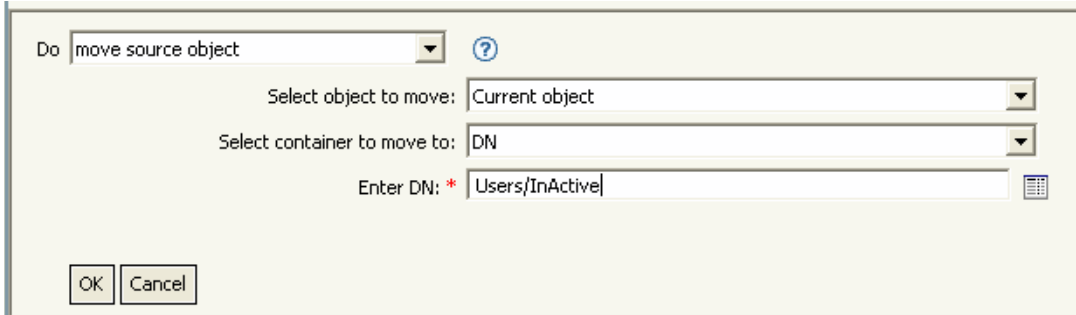
#### Object to Move

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

#### Container to Move to

Select the target container. This container is specified by a DN or an association.

## Example



## 2.6.22 Reformat Operation Attribute

Reformats all values of an attribute within the current operation using a pattern.

### Fields

#### Name

Specify the name of the attribute.

#### Value Type

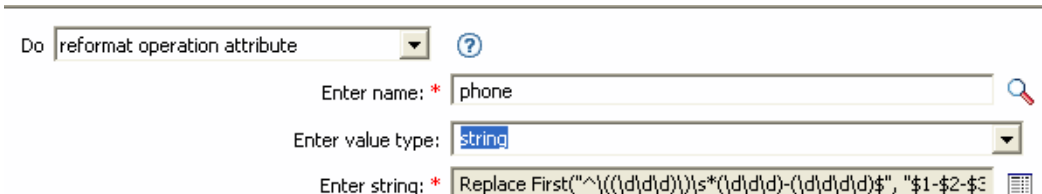
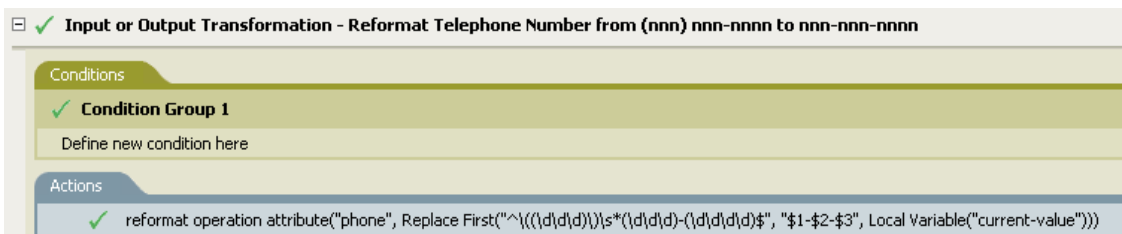
Specify the syntax of the new attribute values.

#### Value

Specify a value to use as a pattern for the new format of the attribute values. If the original value is needed to constructed the new value, it must be obtained by referencing the local variable current-value.

## Example

The example reformats the telephone number. It changes it from (nnn)-nnn-nnnn to nnn-xxx-xxxx. The rule is from the predefined rules that come with Identity Manager 3.0. For more information, see [“Input or Output Transformation - Reformat Telephone Number from \(nnn\) nnn-nnnn to nnn-xxx-xxxx”](#) on page 74.



The action reformat operation attribute changes the format of the telephone number. The rule uses the Argument Builder and regular expressions to change how the information is displayed.

## 2.6.23 Remove Association

Sends a remove association command to the Identity Vault.

### 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

Specify the value of the association to be removed.

### Example

The example takes a delete operation and disables the User object instead. It transforms the event. The rule is from the predefined rules that come with Identity Manager 3.0. For more information, see [“Command Transformation - Publisher Delete to Disable” on page 64](#).

Do

Select mode:

Enter association: \*

When a delete operation occurs for a User object, the value of the attribute Login Disabled is set to true and the association is removed from the object. The association is removed because the associated object in the connected application no longer exists.

## 2.6.24 Remove Destination Attribute Value

Removes an attribute value from an object in the destination data store.

### Fields

#### Attribute Name

Specify the name of the attribute.

### Class Name

(Optional) Specify the class name of the target object. Leave blank to use the class name from the current object.

### 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. This object can be the current object, or can be specified by a DN or an association.

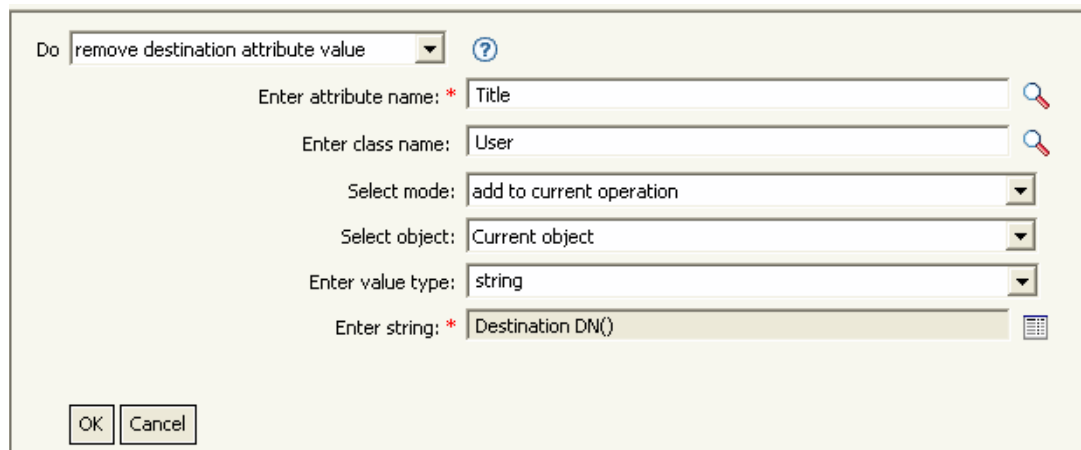
### Value Type

Specify the syntax of the attribute value to be removed.

### Value

Specify the value of the new attribute.

### Example



The screenshot shows a configuration dialog box with the following fields and values:

- Do: remove destination attribute value
- Enter attribute name: \* Title
- Enter class name: User
- Select mode: add to current operation
- Select object: Current object
- Enter value type: string
- Enter string: \* Destination DN()

Buttons: OK, Cancel

## 2.6.25 Remove Source Attribute Value

Removes the specified value from the named attribute on an object in the source data store.

### Fields

#### Attribute Name

Specify the name of the attribute.

#### Class Name

(Optional) Specify the class name of the target object. Leave blank to use the class name from the current object.

#### Object

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

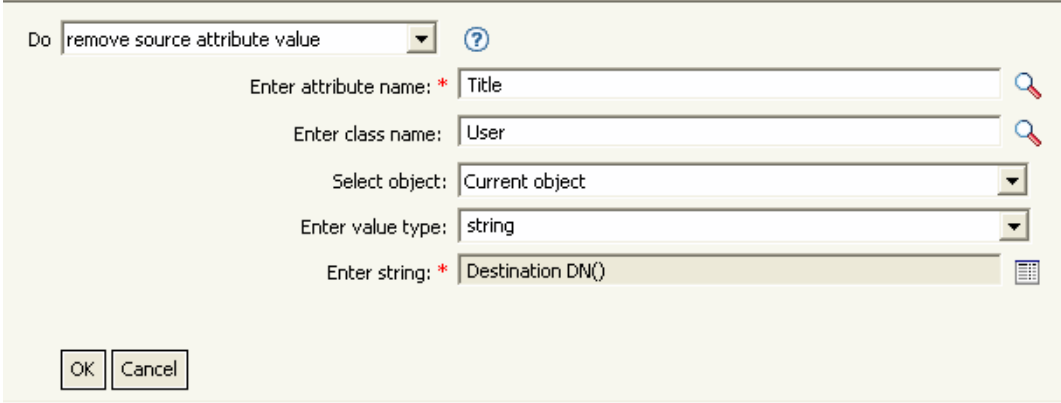
### Value Type

Specify the syntax of the attribute value to be removed

### Value

Specify the attribute value to be removed.

### Example



The screenshot shows a configuration dialog box with a light green background. At the top left, there is a dropdown menu labeled "Do" with the text "remove source attribute value" and a question mark icon to its right. Below this, there are five input fields, each with a label and a red asterisk indicating a required field:

- "Enter attribute name: \* Title" with a search icon to the right.
- "Enter class name: User" with a search icon to the right.
- "Select object: Current object" with a dropdown arrow to the right.
- "Enter value type: string" with a dropdown arrow to the right.
- "Enter string: \* Destination DN()" with a list icon to the right.

At the bottom left of the dialog, there are two buttons: "OK" and "Cancel".

## 2.6.26 Rename Destination Object

Renames an object in the destination data store

### 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. This object can be the current object, or can be specified by a DN or an association.

### String

Specify the new name of the object.

## Example

The screenshot shows a dialog box titled "rename destination object". It features a dropdown menu with "rename destination object" selected, a help icon, and a "Select mode:" dropdown set to "add to current operation". Below that is a "Select object:" dropdown set to "DN". There are two text input fields: "Enter DN: \* Users\John Smith" and "Enter string: \* Johnny". Both input fields have a search icon to their right. At the bottom are "OK" and "Cancel" buttons.

## 2.6.27 Rename Operation Attribute

Renames all occurrences of an attribute within the current operation.

### Fields

#### Source Name

Specify the original attribute name.

#### Destination Name

Specify the new attribute name.

## Example

The screenshot shows a dialog box titled "rename operation attribute". It features a dropdown menu with "rename operation attribute" selected, a help icon, and two text input fields: "Enter source name: \* Surname" and "Enter destination name: \* sn". Both input fields have a search icon to their right. At the bottom are "OK" and "Cancel" buttons.

## 2.6.28 Rename Source Object

Renames an object in the source data store.

### Fields

#### Object

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

#### String

Specify the new name of the object.

## Example

Do: rename source object

Select object: DN

Enter DN: \* "Users\John Smith"

Enter string: \* "Johnny"

OK Cancel

## 2.6.29 Send Email

Sends an e-mail notification.

### Fields

#### ID

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

#### Server

Specify the SMTP server name.

#### Password

(Optional) Specify the SMTP server account password.

---

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

---

#### Type

Select the e-mail message type.

#### Strings

Specify the values containing 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.

String Name	Description
message	Specifies the content of the e-mail message.
encoding	Specifies the character encoding to use for the e-mail message.

### Example

The following is an example of the Named String Builder being used to provide the strings arguments:

Name	String Value
to	"to_user1@company.com"
cc	"cc_user@company.com"
bcc	"bcc_user@company.com"
from	"from_user@company.com"
subject	"This is the e-mail subject"
message	This is the e-mail body

## 2.6.30 Send Email From Template

Generates an e-mail notification using a template.

### Fields

#### Notification DN

Specify the slash form DN of the SMTP notification configuration object.

#### Template DN

Specify the slash form DN of the e-mail template object.

#### Password

(Optional) Specify the SMTP server account password.

---

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

---



## Strings

Specify additional fields for the e-mail message. The following table contains reserved field names, 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.
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.

Each template might also define fields that can be replaced in the subject and body of the email message.

## Example

The screenshot shows a configuration dialog box with the following fields:

- Do: send email from template (dropdown menu)
- Enter notification DN: \* Security\Default Notification Collection (text input)
- Enter template DN: \* Security\Default Notification Collection\Password Set Fail (text input)
- Enter password: (text input)
- Enter strings: to, cc, manager, surname, given-name (text input)

Buttons: OK, Cancel

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

The screenshot shows the Named String Builder interface with the following entries:

Name	String Value
to	"to_user@company.com"
cc	"cc_user@company.com"
manager	"Bill Jones"
surname	"Smith"
given-name	"John"

## 2.6.31 Set Default Attribute Value

Adds default values to the current operation (and optionally to the current object in the source data store) if no values for that attribute already exist. It is only valid when the current operation is add.

## Fields

### Attribute Name

Specify the name of the default attribute.

### Write Back

Select whether or not to also write back the default values to source data store.

### Values

Specify the default values of the attribute.

## Example

The example sets the default value for the attribute company. You can set the value for an attribute of your choice. The rule is from the predefined rules that come with Identity Manager 3.0. For more information, see [“Creation - Set Default Attribute Value” on page 68](#).

Creation - Set Default Attribute Value

Conditions

Condition Group 1

if class name equal "User"

Actions

set default attribute value("company", write-back="true", "Digital Airlines Inc.")

Do: set default attribute value

Enter attribute name: \* company

Write back: true

Enter argument values: \* Digital Airlines Inc.

Type	Argument Values
string	"Digital Airlines Inc."

To build the value, the Argument Value List Builder is launched. See [“Argument Value List Builder” on page 55](#) for more information on the builder. You can set the value to what is needed. In this case, the Argument Builder is used and the text is set to be the name of the company.

## 2.6.32 Set Destination Attribute Value

Adds a value to an attribute on an object in the destination data store, and removes all other values for that attribute.

## Fields

### Attribute Name

Specify the name of the attribute.

## Class Name

(Optional) Specify the class name of the target object in the destination data store. Leave blank to use the class name from the current object.

## 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. This object can be the current object, or be specified by a DN or an association.

## Value Type

Select the syntax of the attribute value to set.

## Value

Specify the attribute values to set.

## Example

The example takes a delete operation and disables the User object instead. The rule is from the predefined rules that come with Identity Manager 3.0. For more information, see [“Command Transformation - Publisher Delete to Disable”](#) on page 64.

The screenshot shows a rule configuration window titled "Command Transformation - Publisher Delete to Disable". It is divided into two main sections: "Conditions" and "Actions".

- Conditions:** A "Condition Group 1" is defined with two conditions:
  - if operation equal "delete"
  - Or if class name equal "User"
- Actions:** Two actions are listed:
  - set destination attribute value("Login Disabled", "true")
  - remove association(association(Association()))

Do  

Enter attribute name: \*

Enter class name:

Select mode:

Select object:

Enter value type:

Enter string: \*

The rule sets the value for the attribute of Login Disabled to true. The rule uses the Argument Builder to add the text of true for the value of the attribute. See [“Argument Builder”](#) on page 52 for more information about the builder.

## 2.6.33 Set Destination Password

Sets the password for the current object in the destination data store.

### 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

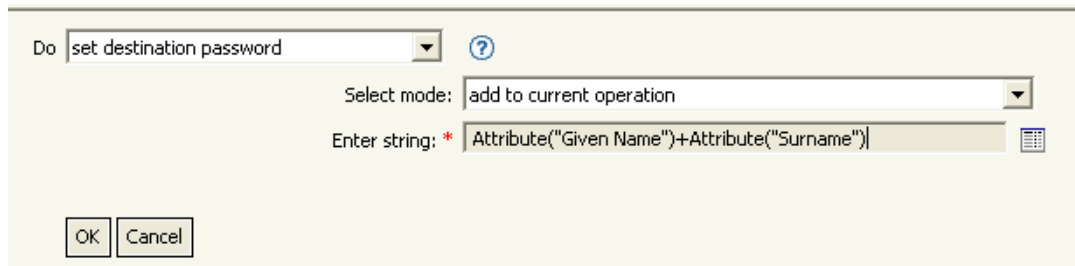
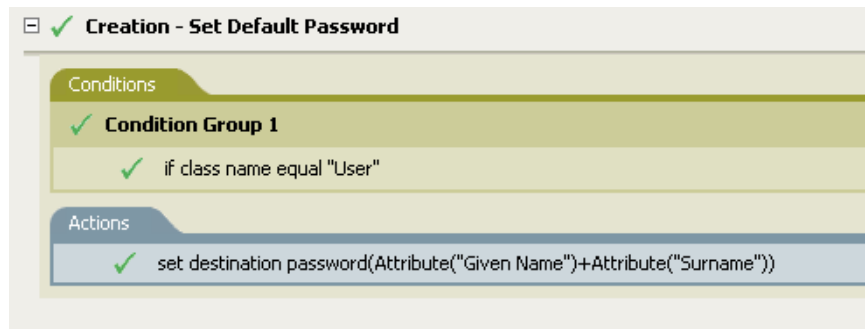
Select the target object. This object can be the current object, or be specified by a DN or an association.

#### String

Specify the password to be set.

### Example

The example sets a default password for a User object that is created. The rule is from the predefined rules that come with Identity Manager 3.0. For more information, see [“Creation - Set Default Password” on page 70](#).



When a User object is created, the password is set to the Given Name attribute plus the Surname attribute.

## 2.6.34 Set Local Variable

Sets a local variable.

## Fields

### Variable Name

Specify the name of the local variable.

### Variable Type

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

### Value

Specify the value of the local variable.

## Example

The example adds a User object to the appropriate group, Employee or Manager, based on Title. It also creates the group, if needed, and sets up security equal to that group. The policy name is Govern Groups for User Based on Title, and it is available for download from Novell's support Web site. For more information, see [“Downloadable Identity Manager Policies” on page 36](#).

The screenshot shows a policy configuration window with the following structure:

- Conditions:**
  - Condition Group 1:** if class name equal "User"
  - And**
  - Condition Group 2:**
    - if operation equal "add"
    - Or if operation equal "modify"
- Actions:**
  - set local variable("manager-group-dn", "Users\ManagersGroup")
  - set local variable("manager-group-info", Destination Attribute("Object Class", dn(Local Variable("manager-group-dn"))))
  - set local variable("employee-group-dn", "Users\EmployeesGroup")
  - set local variable("employee-group-info", Destination Attribute("Object Class", dn(Local Variable("employee-group-dn"))))

Do **set local variable**

Enter variable name: \*

Select variable type:

Enter string: \*

The local variable is set to the value that is in the User object's destination attribute of Object Class plus the Local Variable of manager-group-info. The Argument Builder is used to construct the local variable. See [“Argument Builder” on page 52](#) for more information.

## 2.6.35 Set Operation Association

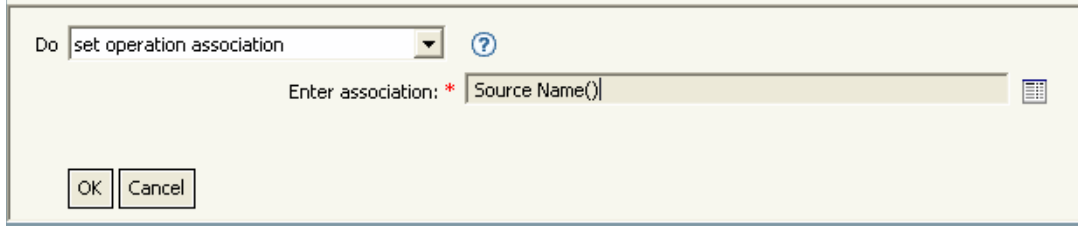
Sets the association value for the current operation.

## Fields

### Association

Provide the new association value.

### Example



The screenshot shows a dialog box titled "set operation association". It features a dropdown menu with the text "set operation association" and a question mark icon. Below the dropdown is a text input field labeled "Enter association: \*" containing the text "Source Name()". At the bottom of the dialog are "OK" and "Cancel" buttons.

## 2.6.36 Set Operation Class Name

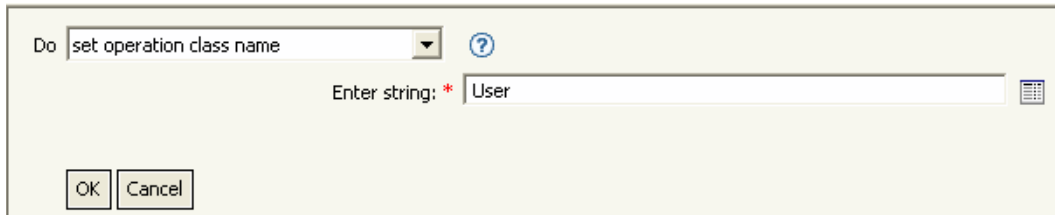
Sets the object class name for the current operation.

### Fields

#### String

Provide the new class name.

### Example



The screenshot shows a dialog box titled "set operation class name". It features a dropdown menu with the text "set operation class name" and a question mark icon. Below the dropdown is a text input field labeled "Enter string: \*" containing the text "User". At the bottom of the dialog are "OK" and "Cancel" buttons.

## 2.6.37 Set Operation Destination DN

Sets the destination DN for the current operation.

### Fields

#### DN

Specify the new destination DN.

### Example

The example places the objects in the Identity Vault using the structure that is mirrored from the connected system. You need to define at what point the mirroring begins in the source and destination data stores. The rule is from the predefined rules that come with Identity Manager 3.0. For more information, see [“Creation - Set Default Attribute Value” on page 68](#).

Placement - Publisher Mirrored

Conditions

Condition Group 1

- if source DN in subtree "[Enter base of source hierarchy]"

Actions

- set local variable("dest-base", "[Enter base of destination hierarchy]")
- set operation destination DN(dn(Local Variable("dest-base")+\"\\\"+Unmatched Source DN(convert=\"true\")))

Do  ?

Enter DN: \*  [icon]

The rule sets the operation destination DN to be the local variable of the destination base location plus the source DN.

## 2.6.38 Set Operation Property

Sets an operation property. An operation property is a named value that is stored within an operation. It is typically used to supply additional context that might be needed by the policy that handles the results of an operation.

### Fields

#### Property Name

Specify the name of the operation property.

#### String

Specify the name of the operation property.

### Example

Do  ?

Enter property name: \*

Enter string: \*  [icon]

OK Cancel

## 2.6.39 Set Operation Source DN

Sets the source DN for the current operation.

### Fields

#### DN

Specify the new source DN.

### Example

Do  

Enter DN: \*

## 2.6.40 Set Operation Template DN

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

### Fields

#### DN

Specify the template DN.

### Example

The example applies the Manager template if the Title attribute contains the word Manager. The name of the policy is Policy: Assign Template to User Based on Title, and it is available for download from Novell's support Web site. For more information, see [“Downloadable Identity Manager Policies” on page 36](#).

**Assign Manager template if Title contains "Manager"**

Conditions


**Condition Group 1**


- if class name equal "User"
- And  if operation attribute 'Title' available
- And  if operation attribute 'Title' match ".\*manager.\*"

Actions

- set operation template DN(dn("Users\ManagerTemplate"))

**Assign Employee template if Title does not contain "Manager"**

Do  

Enter DN: \*  

The template Manager Template is applied to any User object that has the attribute of Title available and it contains the word manager somewhere in the title. The policy uses regular expressions to find all possible matches.



## 2.6.41 Set Source Attribute Value

Adds a value to an attribute on an object in the source data store, and removes all other values for that attribute.

### Fields

#### Attribute Name

Specify the name of the attribute.

#### Class Name

(Optional) Specify the class name of the target object in the source data store. Leave blank to use the class name from the current object.

#### Object

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

#### Value Type

Select the syntax of the attribute value.

#### Value

Specify the attribute value to be set.

### Example

The example detects when an e-mail address is changed and sets it back to what it was. The policy name is Policy: Reset Value of the E-mail Attribute, and it is available for download from Novell's support Web site. For more information, see [“Downloadable Identity Manager Policies” on page 36](#).

Policy: **Push back on email changing**

Conditions

- Condition Group 1
  - if class name equal "User"
  - And if operation attribute "Email" changing

Actions

- set source attribute value("Email", Destination Attribute("Internet EMail Address"))
- strip operation attribute("Email")

Do  

Enter attribute value: \*

Enter class name:

Select object:

Enter value type:

Enter string: \*

The action takes the value of the destination attribute Internet EMail Address and sets the source attribute of Email to this same value.

## 2.6.42 Set Source Password

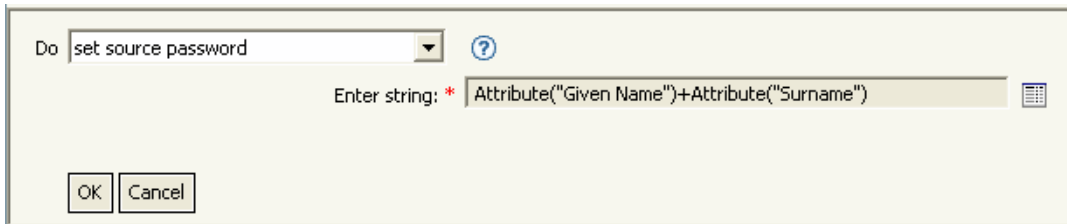
Sets the password for the current object in the source data store.

### Fields

#### String

Specify the password to be set.

### Example



The screenshot shows a dialog box with a title bar. Inside, there is a dropdown menu labeled 'Do' with the text 'set source password' and a question mark icon to its right. Below this, there is a text input field labeled 'Enter string: \*' containing the XPath expression 'Attribute("Given Name")+Attribute("Surname")'. At the bottom of the dialog, there are two buttons: 'OK' and 'Cancel'.

## 2.6.43 Set XML Attribute

Sets an XML attribute on a set of elements selected by an XPath expression.

### Fields

#### Name

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

#### XPATH Expression

XPath 1.0 expression that returns a node set containing the elements on which the XML attribute should be set.

#### String

Specify the value of the XML attribute.

## Example

The image displays two screenshots of a dialog box titled "set XML attribute". Each dialog box has a "Do" dropdown menu set to "set XML attribute" and a help icon. The first dialog box has the following fields: "Enter name: \*" with the value "cert-id", "Enter XPATH expression: \*" with the value ".", and "Enter string: \*" with the value "c:\lotus\domino\data\eng.id". The second dialog box has: "Enter name: \*" with the value "cet-pwd", "Enter XPATH expression: \*" with the value ".", and "Enter string: \*" with the value "certify2eng". Both dialog boxes include "OK" and "Cancel" buttons.

## 2.6.44 Status

Generates a status notification.

### Fields

#### Level

Specify the status level of the notification.

#### Message

Provide the status message by using the Argument Builder.

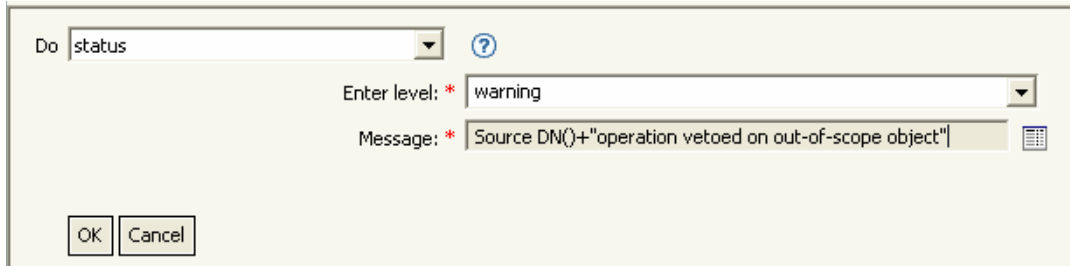
### Remarks

If level is retry, then the policy immediately halts processing of the input document and schedules a retry of the event currently being processed.

If level is fatal, then the policy immediately halts processing of the input document and initiates a shutdown of the driver.

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

## Example



### 2.6.45 Strip Operation Attribute

Strips all occurrences of an attribute from the current operation.

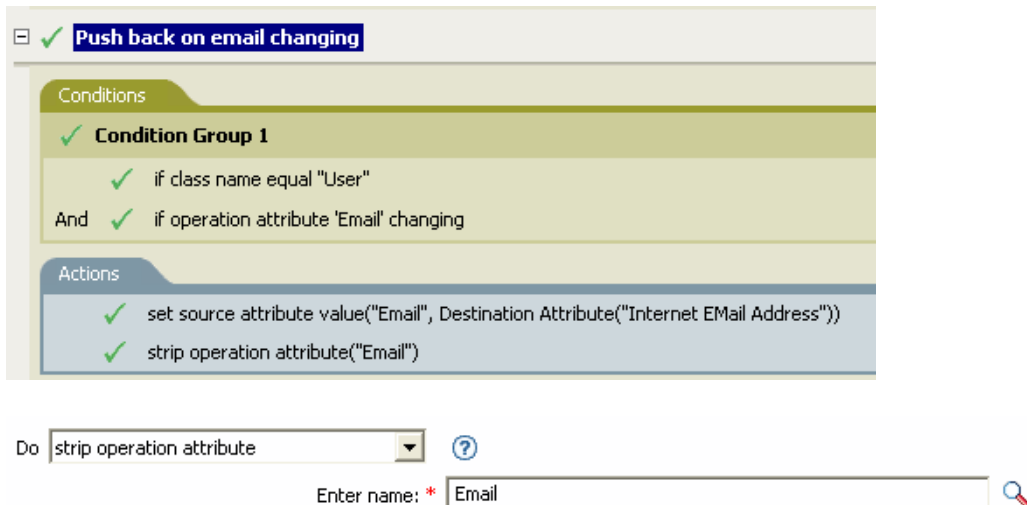
#### Fields

##### Name

Specify the name of the attribute to be stripped.

#### Example

The example detects when an e-mail address is changed and sets it back to what it was. The policy name is Policy: Reset Value of the E-mail Attribute, and it is available for download from Novell's support Web site. For more information, see [“Downloadable Identity Manager Policies” on page 36](#).



The action strips the attribute of Email. The value that is kept is what was in the destination Email attribute.

### 2.6.46 Strip XPath

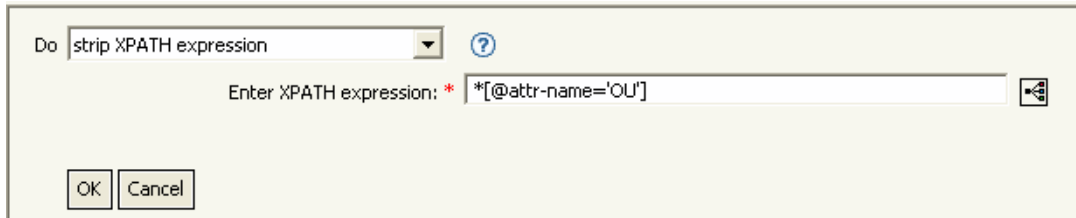
Strips nodes selected by an XPath expression.

## Fields

### XPATH Expression

Specify the XPath 1.0 expression that returns the node set containing the nodes to be stripped.

### Example



The screenshot shows a dialog box with a light green background. At the top left, there is a dropdown menu with the text "strip XPath expression" and a question mark icon to its right. Below this, there is a text input field with the label "Enter XPath expression: \*" and the text "\*[@attr-name='OU']" entered. To the right of the input field is a small icon of a document with a magnifying glass. At the bottom left, there are two buttons: "OK" and "Cancel".

## 2.6.47 Trace Message

Sends a message to DSTRACE.

### Fields

#### Level

Specify the trace level of the message. The default level is 0. The message only appears if the specified trace level is less than or equal to the trace level configured in the driver.

For information on how to set the trace level on the driver, see “[Viewing Identity Manager Processes](#)” in the *Novell Identity Manager 3.0 Administration Guide*.

#### Color

Select the color of the trace message.

#### String

Specify the value of the trace message.

### Example

The example has four rules that implement a Placement policy for User objects based on the first character of the Surname attribute. It generates both a trace message and a custom Novell Audit event. The Trace Message action is used to send a trace message into DSTRACE. The policy name is Policy to Place by Surname, and it is available for download from Novell’s support Web site. For more information “[Downloadable Identity Manager Policies](#)” on page 36.

**Setup Local Variables**  
 **Surname A-I: place in Users1**

**Conditions**  
 **Condition Group 1**  
 if class name equal "User"  
 And  if operation attribute 'Surname' match "[a-i].\*"

**Actions**  
 set operation destination DN(dn("Training\Users\Active\Users1"+"\"+Operation Attribute("CN")))

trace message(color="yellow", Local Variable("LVUsers1"))

generate event(id="1000", text1=Local Variable("LVUsers1"))

**Surname J-R: place in Users2**  
 **Surname S-Z: place in Users3**

Do  

Enter level:

Select color:

Enter string: \*

The action sends a trace message to DSTRACE. The contents of the local variable is LVUsers1 and it shows up in yellow in DSTRACE.

## 2.6.48 Veto

Vetoes the current operation.

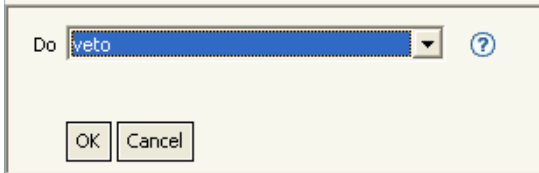
### Example

The example excludes all events that come from the specified subtree. The rule is from the predefined rules that come with Identity Manager 3.0. For more information, see [“Event Transformation - Scope Filtering - Exclude Subtrees”](#) on page 72.

**Event Transformation - Scope Filtering - Exclude subtree(s)**

**Conditions**  
 **Condition Group 1**  
 if source DN in subtree "[Enter a subtree to exclude]"

**Actions**  
 **veto()**



The action vetoes all events that come from the specified subtree.

## 2.6.49 Veto If Operational Attribute Not Available

Conditionally cancels the current operation and ends processing of the current policy, based on the availability of an attribute in the current operation.

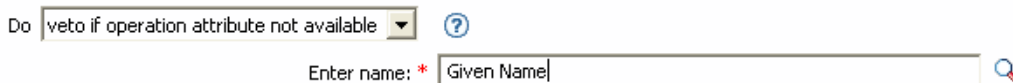
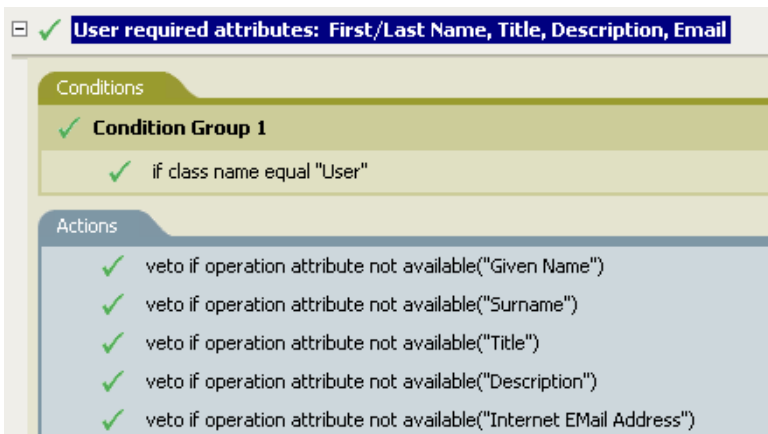
### Fields

#### Name

Specify the name of the attribute.

### Example

The example does not allow all User objects to be created unless the attributes Given Name, Surname, Title, Description, and Internet EMail Address are available. The policy name is Policy to Enforce the Presences of Attributes and it is available for download from Novell's support Web site. For more information, see [“Downloadable Identity Manager Policies” on page 36](#).



The actions vetoes the operation if the attributes of Given Name, Surname, Title, Description, and Internet Email Address are not available.

## 2.7 Noun Tokens

This section contains detailed reference to all noun tokens available using the Argument Builder interface.

- Section 2.7.1, “Added Entitlement,” on page 160
- Section 2.7.2, “Association,” on page 160
- Section 2.7.3, “Attribute,” on page 161
- Section 2.7.4, “Class Name,” on page 162
- Section 2.7.5, “Destination Attribute,” on page 162
- Section 2.7.6, “Destination DN,” on page 163
- Section 2.7.7, “Destination Name,” on page 164
- Section 2.7.8, “Entitlement,” on page 164
- Section 2.7.9, “Global Configuration Value,” on page 165
- Section 2.7.10, “Local Variable,” on page 165
- Section 2.7.11, “Named Password,” on page 166
- Section 2.7.12, “Operation,” on page 166
- Section 2.7.13, “Operation Attribute,” on page 167
- Section 2.7.14, “Operation Property,” on page 167
- Section 2.7.15, “Password,” on page 168
- Section 2.7.16, “Removed Attribute,” on page 168
- Section 2.7.17, “Removed Entitlement,” on page 168
- Section 2.7.18, “Source Attribute,” on page 168
- Section 2.7.19, “Source DN,” on page 169
- Section 2.7.20, “Source Name,” on page 169
- Section 2.7.21, “Text,” on page 170
- Section 2.7.22, “Unique Name,” on page 171
- Section 2.7.23, “Unmatched Source DN,” on page 172
- Section 2.7.24, “XPath,” on page 173

## 2.7.1 Added Entitlement

Expands to the values of an entitlement granted in the current operation.

### Fields

#### Name

Name of the entitlement.

### Example

 Added Entitlement("manager")

## 2.7.2 Association

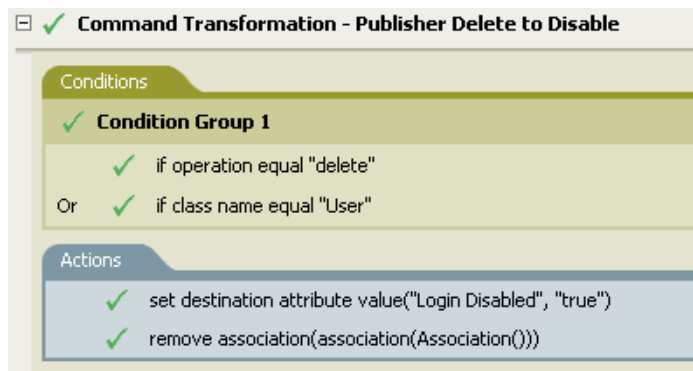
Expands to the association value from the current operation.




## Example

The example is from the predefined rules that come with Identity Manager 3.0. For more information on the predefined rule, see [“Command Transformation - Publisher Delete to Disable” on page 64](#).

The action of Remove Association uses the Association token to retrieve the value from the current operation. The rule removes the association from the User object so that any new events coming through do not affect the User object.



 Association()

## 2.7.3 Attribute

Expands to the value of an attribute from the current object in current operation and in the source data store. It can be logically thought of as the union of the operation attribute token and the source attribute token. It does not include the removed values from a modify operation.

### Fields

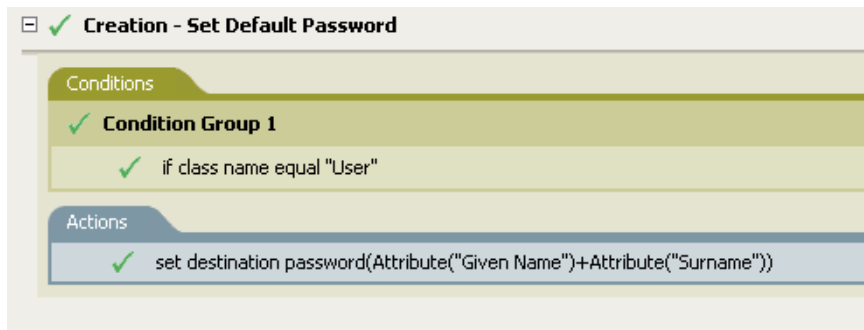
#### Name



Specify the name of the attribute.

### Example

The example is from the predefined rules that come with Identity Manager 3.0. For more information, see [“Creation - Set Default Password” on page 70](#).

The action of Set Destination Password uses the attribute token to create the password. The password is made up of the Given Name attribute and the Surname attribute. When you are in the Argument Builder Editor, you browse and select the attribute you want to use.




 Attribute("Given Name")  
 Attribute("Surname")



## 2.7.4 Class Name

Expands to the object class name from the current operation.

### Example

 Class Name()

## 2.7.5 Destination Attribute

Expands to the specified attribute value of the current object, a DN, or association, in the destination data store.

### Fields

#### Class Name

(Optional) Specify the class name of the target object. Leave blank to use the class name from the current object.

#### Name

Name of the attribute.

### Example

The example is from the Govern Groups for User Based on Title policy which is available for download from Novell's support Web site. For more information, see [“Downloadable Identity Manager Policies” on page 36](#).

The policy creates the Destination Attribute with the Argument Builder. The action of Set Local Variable contains the Destination Attribute token.

☐ ✓ **Set local variables to test existence of groups and for placement**

Conditions

✓ **Condition Group 1**

✓ if class name equal "User"

**And**

✓ **Condition Group 2**

✓ if operation equal "add"

Or ✓ if operation equal "modify"

Actions

✓ set local variable("manager-group-dn", "Users\ManagersGroup")

✓ set local variable("manager-group-info", Destination Attribute("Object Class", dn(Local Variable("manager-group-dn"))))

✓ set local variable("employee-group-dn", "Users\EmployeesGroup")

✓ set local variable("employee-group-info", Destination Attribute("Object Class", dn(Local Variable("employee-group-dn"))))

..... Destination Attribute("Object Class", dn())

**Editor**

Name: \* Object Class 🔍

Class name: 🔍

Select object: DN ▾

Enter DN: \* Local Variable("manager-group-dn") 📄

You build the Destination Attribute through the Editor. In this example, the attribute of Object Class is set. DN is used to select the target object. The value of DN is the Local Variable of manager-group-dn.

## 2.7.6 Destination DN

Expands to the destination DN from the current operation.

### Fields

#### Convert

Select whether or not to convert the DN to the format used by the source data store.

#### Start

Specify the RDN index to start with:

- Index 0 is the root-most RDN
- Positive indexes are an offset from the root-most RDN
- Index -1 is the leaf-most segment
- Negative indexes are an offset from the leaf-most RDN towards the root-most RDN

#### Length

Specify the number of RDN to include. Negative numbers are interpreted as (total # of segments + length) + 1. For example, 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}, the entire DN is used; otherwise only the portion of the DN specified by start and length is used.

## Example

The example uses the Destination DN token to set the value for the local variable of target-container. The policy creates a department container for the User object if it does not exist. The policy is from the predefined rules that come with Identity Manager 3.0. For more information, see [“Command Transformation - Create Departmental Container - Part 1 and Part 2”](#) on page 62.

☐ ✓ **Command Transformation - Create Departmental Container - Part 1**

Conditions

✓ **Condition Group 1**

✓ if operation equal "add"

Actions

✓ set local variable("target-container", Destination DN(length="-2"))

✓ set local variable("does-target-exist", Destination Attribute("objectclass", class name="OrganizationalUnit", dn(Local Variable("target-container"))))

..... Destination DN(length="-2")

## 2.7.7 Destination Name

Expands to the unqualified Relative Distinguished Name (RDN) of the destination DN specified from the current operation.

## Example

Destination Name()

## 2.7.8 Entitlement

Expands to the values of a granted entitlement from the current object.

## Fields

### Name

Specify the name of the entitlement.

## Example

Entitlement("manager")

## 2.7.9 Global Configuration Value


Expands to the value of a global configuration value.

### Fields

#### Name

Name of the global configuration value.

### Example

 Global Configuration Value("Fred")

## 2.7.10 Local Variable

Expands to the value of a local variable.

### Fields

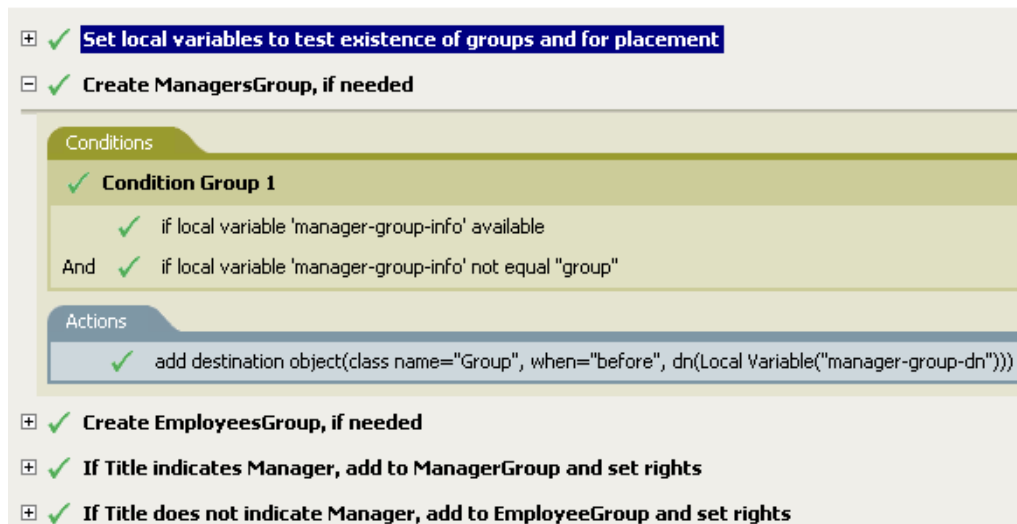
#### Name

Specify the name of the local variable.

### Example


The example is from the Govern Groups for User Based on Title policy which is available for download from Novell's support Web site. For more information, see [“Downloadable Identity Manager Policies” on page 36](#).

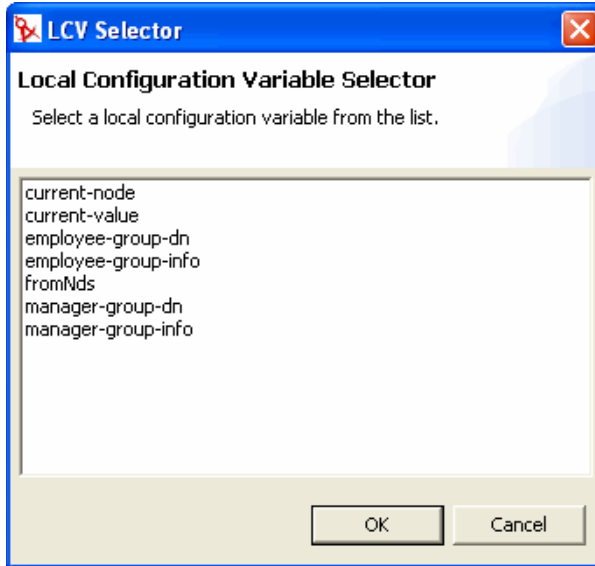
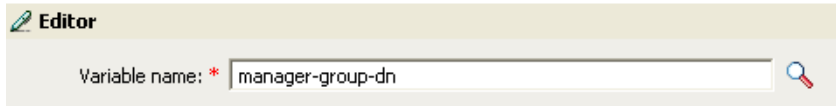
The action Add Destination Object uses the Local Variable token.



The screenshot displays a policy configuration window with the following elements:

- Two checked items at the top: **Set local variables to test existence of groups and for placement** and **Create ManagersGroup, if needed**.
- A **Conditions** section containing:
  - Condition Group 1** (checked):
    - if local variable 'manager-group-info' available (checked)
    - And if local variable 'manager-group-info' not equal "group" (checked)
- An **Actions** section containing:
  - add destination object(class name="Group", when="before", dn(Local Variable("manager-group-dn")) (checked)
- Three checked items at the bottom:
  - Create EmployeesGroup, if needed**
  - If Title indicates Manager, add to ManagerGroup and set rights**
  - If Title does not indicate Manager, add to EmployeeGroup and set rights**

 Local Variable("manager-group-dn")



The Local Variable can only be used if the action Set Local Variable has been used previously in the policy. It sets the value that is stored in the Local Variable. In the Editor, you click the browse icon and all of the local variables that have been defined are listed. Select the correct local variable.

The value of the local variable is group-manager-dn. In the rule before this one, the Set Local Variable action defined group-manager-dn as DN of the manager's group Users\ManagersGroup.

## 2.7.11 Named Password


Expands to the named password from the driver.

### Fields

#### Name

Specify the name of the password.

### Example

 Named Password("password")

## 2.7.12 Operation

Expands to the name of the current operation.

### Example

 Operation()

## 2.7.13 Operation Attribute

Expands to the value of an attribute from the current operation. It does not include the removed values from a modify operation.

### Fields

#### Name

Specify the name of the attribute.

### Example

The example has four rules that implement a Placement policy for User objects based on the first character of the Surname attribute. It generates both a trace message and a custom Novell Audit event. The policy name is Policy to Place by Surname, and it is available for download from Novell's support Web site. For more information "[Downloadable Identity Manager Policies](#)" on [page 36](#).

The screenshot displays the Policy Builder interface. At the top, there are three expandable sections: 'Setup Local Variables', 'Surname A-I: place in Users1', and 'Surname J-R: place in Users2'. The 'Surname A-I: place in Users1' section is expanded, showing a 'Conditions' tab with a 'Condition Group 1' containing two conditions: 'if class name equal "User"' and 'if operation attribute "Surname" match "[a-i].\*"'. Below the conditions is an 'Actions' tab with three actions: 'set operation destination DN(dn("Training\Users\Active\Users1"+"\"+Operation Attribute("CN")))', 'trace message(color="yellow", Local Variable("LVUsers1"))', and 'generate event(id="1000", text1=Local Variable("LVUsers1"))'. Below the actions are two more expandable sections: 'Surname S-Z: place in Users3' and 'Surname J-R: place in Users2'. At the bottom left, there is a tree view showing the context path: 'Training\Users\Active\Users1', '\', and 'Operation Attribute("CN")'. At the bottom, there is an 'Editor' section with a text input field labeled 'Name: \*' containing the text 'CN' and a search icon.

The action Set Operation Destination DN contains the Operation Attribute token. The Operation Attribute token sets the Destination DN to the CN attribute. The rule takes the context of Training\Users\Active\Users and adds a \ plus the value of the CN attribute.

## 2.7.14 Operation Property

Expands to the value of an operation property from the current operation.

## Fields

### Name

Specify the name of the operation property


### Example

```
 Operation Property("myStoredProperty")
```

## 2.7.15 Password

Expands to the password from the current operation.

### Example

```
 Password()
```

## 2.7.16 Removed Attribute

Expands to the values of an attribute being removed in the current operation. It only applies to modify operations.

## Fields

### Name

Specify the name of the attribute

### Example

```
 Removed Attribute("OU")
```

## 2.7.17 Removed Entitlement

Expands to the values of an entitlement revoked in the current operation.

## Fields

### Name

Specify the name of the entitlement.

### Example

```
 Removed Entitlement("manager")
```

## 2.7.18 Source Attribute

Expands to the values of an attribute from an object in the source data store.



## Fields

### Class Name

(Optional) Specify the class name of the target object. Leave blank to use the class name from the current object.

### Name

Name of the attribute.

### Example

```
Source Attribute("OU")
```

## 2.7.19 Source DN

Expands to the source DN from the current operation.

## Fields

### Convert

Select whether or not to convert the DN to the format used by the destination data store.

### Start

Specify the RDN index to start with:

- Index 0 is the root-most RDN
- Positive indexes are an offset from the root-most RDN
- Index -1 is the leaf-most segment
- Negative indexes are an offset from the leaf-most RDN towards the root-most RDN

### Length

Number of RDN's segments to include. Negative numbers are interpreted as (total # of segments + length) + 1. For example, 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.


### Example

```
Source DN()
```

## 2.7.20 Source Name

Expands to the unqualified Relative Distinguished Name (RDN) of the source DN from the current operation.

## Example

 Source Name()

## 2.7.21 Text

Expands to the text.

### Fields

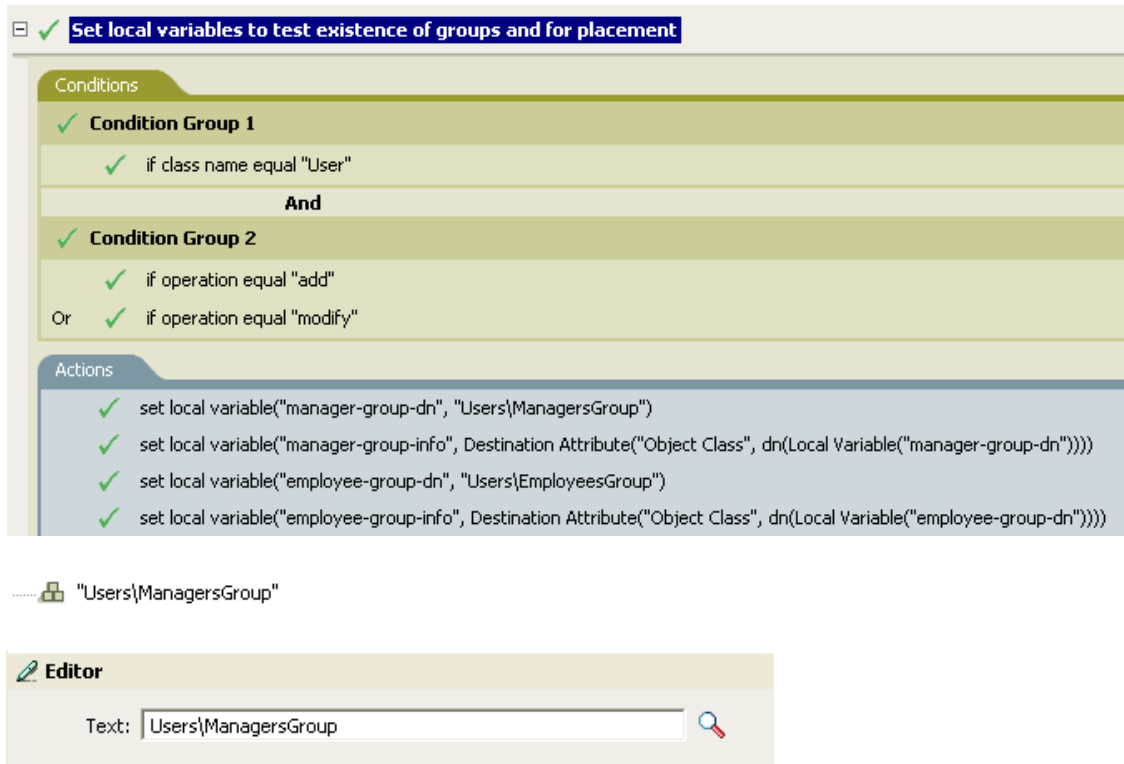
#### Text

Specify the text.

## Example

The example is from the Govern Groups for User Based on Title policy which is available for download from Novell's support Web site. For more information, see ["Downloadable Identity Manager Policies" on page 36](#).

The Text token is used in the action Set Location Variable to define the DN of the manager's group. The Text token can contain objects or plain text.



The screenshot displays the Policy Builder interface for a policy titled "Set local variables to test existence of groups and for placement". The interface is divided into two main sections: "Conditions" and "Actions".

**Conditions:**

- Condition Group 1:** if class name equal "User"
- And**
- Condition Group 2:**
  - if operation equal "add"
  - Or if operation equal "modify"

**Actions:**

- set local variable("manager-group-dn", "Users\ManagersGroup")
- set local variable("manager-group-info", Destination Attribute("Object Class", dn(Local Variable("manager-group-dn"))))
- set local variable("employee-group-dn", "Users\EmployeesGroup")
- set local variable("employee-group-info", Destination Attribute("Object Class", dn(Local Variable("employee-group-dn"))))

Below the actions, there is a field for the "Text" token, which contains the value "Users\ManagersGroup".

The Text noun contains the DN for the manager's group. You can browse to the object you want to use, or type the information into the editor.

## 2.7.22 Unique Name

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

### Fields

#### Name

Specify the name of attribute to check for uniqueness.

#### Scope

Specify the scope in which to check uniqueness.

#### 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

Specify patterns to use to generate unique values by using the Argument Builder.

#### Counter Start

Specify the a number to start counter used when needed to find a unique name.

#### Digits

Specify the width in digits of counter, the 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 is be appended with 001, then 002, and so on.

### Remarks

For each specified pattern, a query is performed against the destination data store, using the supplied attribute name, scope, and search start. Each specified pattern is tried in order until a value is found that does not return any found objects.

If all of the specified patterns are exhausted, the final pattern has a counter appended to it and the pattern is tried repeatedly (increasing the counter each time) until the query does not return any instances.

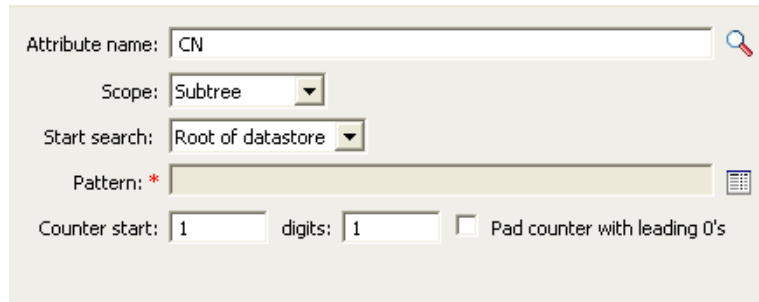
The counter can be set to start at a different number using the counter start field. The counter uses the number of digits specified by the digits field. If the number of digits is less than those specified, then the counter is right padded with zeros. When the number of digits exceeds those specified, then no unique name is generated and the enclosing rule returns an error status.

If the destination data store is the Identity Vault and name field is left blank, 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 connected application, then the name field is required.

### Example

 Unique Name("CN",scope="subtree",Lower Case())

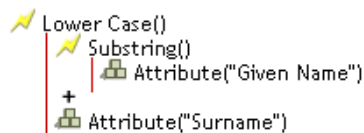
The following is an example of the Editor pane when constructing the unique name argument:



The screenshot shows a configuration window with the following fields and options:

- Attribute name: CN
- Scope: Subtree
- Start search: Root of datastore
- Pattern: \*
- Counter start: 1
- digits: 1
- Pad counter with leading 0's

The following pattern was constructed to provide unique names:



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

## 2.7.23 Unmatched Source DN

Expands to the part 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.

### Fields

#### Convert

Select whether or not to convert the DN to the format used by the destination data store.

### Remarks

If there were no matches, the entire DN is used.

### Example

The example is from the predefined rules that come with Identity Manager 3.0. For more information, see [“Matching - Subscriber Mirrored - LDAP Format” on page 78](#).

The action of Finding Matching Object uses the Unmatched Source DN token to build the matching information in LDAP format. It takes the unmatched portion of the source DN to make a match.

**Matching - Subscriber Mirrored - LDAP format**

**Conditions**

**Condition Group 1**

if source DN in subtree "[Enter base of source hierarchy]"

**Actions**

set local variable("dest-base", "[Enter base of destination hierarchy]")

find matching object(scope="entry", dn(Unmatched Source DN(convert="true")+","+Local Variable("dest-base")))

- Unmatched Source DN(convert="true")
- ","
- Local Variable("dest-base")

**Editor**

Convert to destination DN format:

## 2.7.24 XPath

Expands to results of evaluating an XPath 1.0 expression.

### Example

`XPATH("//*[ @attr-name='OU']/value[starts-with(string(.),'xxx']")`

### Fields

#### Expression

Specify the XPath 1.0 expression to evaluate.

## 2.8 Verb Tokens

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

- [Section 2.8.1, “Escape Destination DN,” on page 174](#)
- [Section 2.8.2, “Escape Source DN,” on page 174](#)
- [Section 2.8.3, “Lower Case,” on page 174](#)
- [Section 2.8.4, “Parse DN,” on page 175](#)
- [Section 2.8.5, “Replace All,” on page 177](#)
- [Section 2.8.6, “Replace First,” on page 177](#)
- [Section 2.8.7, “Substring,” on page 178](#)
- [Section 2.8.8, “Upper Case,” on page 179](#)

## 2.8.1 Escape Destination DN

Escapes a string according to the rules of the DN format of the destination data store.

### Example

The example is from the predefined rules that come with Identity Manager 3.0. For more information, see [“Placement - Publisher Flat” on page 84](#).

The action of Set Operation Destination DN uses the Escape Destination DN token to build the destination DN of the User object.

Placement - Publisher Flat

Conditions

- Condition Group 1
  - if class name equal "User"

Actions

- set local variable("dest-base", [Enter DN of destination container])
- set operation destination DN(dn(Local Variable("dest-base")+ "|" + Escape Destination DN(Unique Name("CN", scope="subtree", Lower Case(Substring(length="1", Operation Attribute("Given Name"))+Operation Attribute("Surname"))))))

Local Variable("dest-base")

"|"

Escape Destination DN()

Unique Name("CN", scope="subtree", Lower Case(), Lower Case())

The Escape Destination DN token takes the value in Unique Name and sets it to the format for the destination DN.

## 2.8.2 Escape Source DN

Escapes a string according to the rules of the DN format of the source data store.

### Example

Escape Source DN()

Attribute("Surname")

## 2.8.3 Lower Case

Converts the characters in a string to lowercase.

### Example

This example sets the e-mail address to be name@slartybartfast.com where the name equals the first character of the Given Name plus the Surname. The policy name is Policy: Create E-mail from Given Name and Surname, and it is available for download at Novell’s support Web site. For more information, see [“Downloadable Identity Manager Policies” on page 36](#).

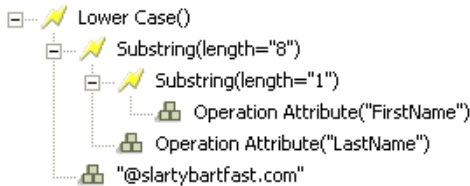
Set email address: name@slartybartfast.com; name = (1 char of Given Name + Surname) <= 8 chars

**Conditions**

- Condition Group 1
  - if class name equal "User"
  - And if operation attribute 'Given Name' available
  - And if operation attribute 'Surname' available

**Actions**

- strip operation attribute("Internet Email Address")
- set destination attribute value("Internet Email Address", Lower Case(Substring(length="8", Substring(length="1", Operation Attribute("FirstName"))+Operation Attribute("LastName"))+"@slartybartfast.com"))



The Lower Case token sets all of the information in the action Set Destination attribute value to lowercase.

## 2.8.4 Parse DN

Converts a DN to an alternate format.

### Example

The example uses the Parse DN token to build the value for the Add Destination Attribute Value action. The example is from the predefined rules that come with Identity Manager 3.0. For more information, see [“Command Transformation - Create Departmental Container - Part 1 and Part 2” on page 62](#).

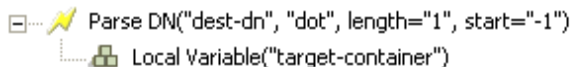
Command Transformation - Create Departmental Container - Part 2

**Conditions**

- Condition Group 1
  - if local variable 'does-target-exist' available
  - And if local variable 'does-target-exist' equal ""

**Actions**

- add destination object(class name="organizationalUnit", direct="true", dn(Local Variable("target-container")))
- add destination attribute value("ou", direct="true", dn(Local Variable("target-container")), Parse DN("dest-dn", "dot", length="1", start="-1", Local Variable("target-container")))



**Editor**

Start:

Length:

Source DN format:

Destination DN format:

The Parse DN token takes the information from the source DN and converts it to the dot notation. The information from the Parse DN is stored in the attribute value of OU.

## Fields

### Start

Specify the RDN index to start with:

- Index 0 is the root-most RDN
- Positive indexes are an offset from the root-most RDN
- Index -1 is the leaf-most segment
- Negative indexes are an offset from the leaf-most RDN towards the root-most RDN

### Length

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

### Source DN Format

Specifies the format used to parse the source DN.

### Destination DN Format

Specify the format used to output the parsed DN.

### Source DN Delimiter

Specify the custom source DN delimiter set if Source DN Format is set to custom.

### Destination DN Delimiter

Specify the custom destination DN delimiter set if Destination DN Format is set to custom.

## 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.

When specifying custom DN formats, the eight characters that make up the delimiter set are defined as follows:

1. Typed Name Boolean Flag: 0 means names are not typed, and 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, such as \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, the orientation of the name is root right, otherwise the orientation is root left.

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

### 2.8.5 Replace All

Replaces all occurrences of a regular expression in a string.

#### Fields

##### Regular Expression

Specify the regular expression that matches the substrings to be replaced.

##### Replace With

Specify the replacement string.



#### Remarks

For details on creating regular expressions, see:

- Sun's Java Web site (<http://java.sun.com/j2se/1.4/docs/api/java/util/regex/Pattern.html>)
- Sun's Java Web site ([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 options `CASE_INSENSITIVE`, `DOTALL`, and `UNICODE_CASE` are used but can be reversed by using the appropriate embedded escapes.

#### Example

 Replace All("(.)", "\$1")  
 Destination DN()

### 2.8.6 Replace First

Replaces the first occurrence of a regular expression in a string.

#### Fields

##### Regular Expression

Specify the regular expression that matches the substring to replace.

##### Replace With

Specify 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)) (<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)))

The pattern option CASE\_INSENSITIVE, DOTALL, and UNICODE\_CASE are used but can be reversed using the appropriate embedded escapes.

## Example

The example reformats the telephone number (nnn)-nnn-nnnn to nnn-nnn-nnnn. The rule is from the predefined rules that come with Identity Manager 3.0. For more information, see [“Input or Output Transformation - Reformat Telephone Number from \(nnn\) nnn-nnnn to nnn-nnn-nnnn” on page 74](#).

The Replace First token is used in the Reformat Operation Attribute action.

The screenshot displays the configuration for a rule titled "Input or Output Transformation - Reformat Telephone Number from (nnn) nnn-nnnn to nnn-nnn-nnnn". It is divided into two main sections: "Conditions" and "Actions".

- Conditions:** A "Condition Group 1" is defined with the instruction "Define new condition here".
- Actions:** A single action is configured: "reformat operation attribute('phone', Replace First('^((\d\d\d))s\*(\d\d\d)-(\d\d\d\d)\$', '\$1-\$2-\$3', Local Variable('current-value')))".

Below the configuration, a detailed view of the "Replace First" action is shown. It includes a lightning bolt icon and the text "Replace First('^((\d\d\d))s\*(\d\d\d)-(\d\d\d\d)\$', '\$1-\$2-\$3')". A local variable named "current-value" is linked to the replacement string.

The "Editor" section at the bottom provides a visual representation of the regular expressions. The "Regular expression:" field contains `^((\d\d\d))s*(\d\d\d)-(\d\d\d\d)$` and the "Replace with:" field contains `$1-$2-$3`.

The regular expression of `^((\d\d\d))s*(\d\d\d)-(\d\d\d\d)$` represents (nnn) nnn-nnnn and the regular expression of `$1-$2-$3` represents nnn. This rule transforms the format of the telephone number from (nnn) nnn-nnnn to nnn-nnn-nnnn.

## 2.8.7 Substring

Extracts a portion of a string.

### Fields

#### Start

Specify the starting character index:

- Index 0 is the first character.
- Positive indexes are an offset from the start of the string
- Index -1 is the last character
- Negative indexes are an offset from the last character toward the start of the string

## Length

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

## Example

This example sets the e-mail address to be name@slartybartfast.com where the name equals the first character of the Given Name plus the Surname. The policy name is Policy: Create E-mail from Given Name and Surname and it is available at Novell's support Web site for download. For more information, see [“Downloadable Identity Manager Policies” on page 36](#).

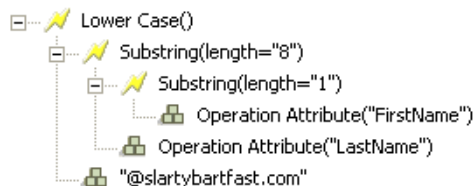
The screenshot shows a policy configuration window titled "Set email address: name@slartybartfast.com; name = (1 char of Given Name + Surname) <= 8 chars". It is divided into two sections: "Conditions" and "Actions".

**Conditions:**

- Condition Group 1
  - if class name equal "User"
  - And if operation attribute 'Given Name' available
  - And if operation attribute 'Surname' available

**Actions:**

- strip operation attribute("Internet Email Address")
- set destination attribute value("Internet Email Address", Lower Case(Substring(length="8", Substring(length="1", Operation Attribute("FirstName"))+Operation Attribute("LastName"))+"@slartybartfast.com"))



The Substring token is used twice in the action Set Destination Attribute Value. It takes the first character of the First Name attribute and adds eight characters of the Last Name attribute together to form one substring.

## 2.8.8 Upper Case

Converts the characters in a string to uppercase.

### Example

The example converts the first and last name attributes of the User object to uppercase. The policy name is Policy: Convert First/Last Name to Upper Case, and it is available for download at Novell's support Web site. For more information, see [“Downloadable Identity Manager Policies” on page 36](#).

☐ ✓ **Convert First/Last name to uppercase**

Conditions

✓ **Condition Group 1**

✓ if class name equal "User"

**And**

✓ **Condition Group 2**

✓ if operation attribute 'Given Name' changing

Or ✓ if operation attribute 'Surname' changing

Actions

✓ reformat operation attribute("Given Name", Upper Case(Operation Attribute("Given Name")))

✓ reformat operation attribute("Surname", Upper Case(Operation Attribute("Surname")))

☐ ⚡ Upper Case()

☐ 🗄️ Operation Attribute("Given Name")

## 2.9 Values

This section contains a list of common policy builder values.

### 2.9.1 Comparison Modes

**Table 2-5** 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 can be changed by an escape in the expression.  See <a href="http://java.sun.com/j2se/1.4/docs/api/java/util/regex/Pattern.html">Sun's Java Web site (http://java.sun.com/j2se/1.4/docs/api/java/util/regex/Pattern.html)</a> and <a href="http://java.sun.com/j2se/1.4/docs/api/java/util/regex/Matcher.html#matches()">Sun's Java Web site (http://java.sun.com/j2se/1.4/docs/api/java/util/regex/Matcher.html#matches())</a> .  The pattern options 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 data store.
dest-dn	Compare using semantics appropriate to the DN format for the destination data store.
numeric	Compare numerically.
octet	Compare octet (Base64 encoded) values.

---

Mode	Description
structured	Compare the structured attribute according to the comparison rules for the structured syntax of the attribute.

---



# Defining Policies By Using The Policy Builder In iManager

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

This section gives the following information on policies and how to use the Policy Builder:

- [Section 2.1, “Policies,” on page 37](#)
- [Section 3.2, “Policy Builder Tasks in iManager,” on page 184](#)

This section also contains the following detailed reference sections:

- [Section 3.5, “Conditions,” on page 218](#)
- [Section 3.6, “Actions,” on page 236](#)
- [Section 3.7, “Noun Tokens,” on page 274](#)
- [Section 3.8, “Verb Tokens,” on page 287](#)

## 3.1 Policies

As part of understanding how policies work, it is important to understand the components of policies.

- Policies are made up of rules.
- A rule is a set of conditions (see [“Conditions” on page 218](#)) that must be met before a defined action (see [“Actions” on page 236](#)) occurs.
- Actions can have dynamic arguments that derive from tokens that are expanded at run time.
- Tokens are broken up into two classifications: nouns (see [“Noun Tokens” on page 274](#)) and verbs (see [“Verb Tokens” on page 287](#)).
  - Noun tokens expand to values that are derived from the current operation, the source or destination data stores, or some external source.
  - Verb tokens modify the concatenated results of other tokens that are subordinate to them.
- Regular expressions (see [“Regular Expressions” on page 216](#)) and XPath 1.0 expressions (see [“XPath 1.0 Expressions” on page 217](#)) are commonly used in the rules to create the desired results for the policies.

A policy operates on an XDS document and its primary purpose is to examine and modify that document. A policy can also get additional context from outside of the document and cause side effects that are not reflected in the result document.

The following outline describes the different elements of a policy:

- [Section 3.1.1, “Operation,” on page 184](#)
- [Section 3.1.2, “Current Operation,” on page 184](#)
- [Section 3.1.3, “Current Object,” on page 184](#)

### 3.1.1 Operation

An operation is any element that is a child of the input element and the output element. The elements are part of Novell®'s `nds.dtd`, for more information, see [NDS DTD \(http://developer.novell.com/ndk/doc/dirxml/dirxmlbk/ref/ndsdttd/index.html\)](http://developer.novell.com/ndk/doc/dirxml/dirxmlbk/ref/ndsdttd/index.html). An operation usually represents an event, a command, or a status.

### 3.1.2 Current Operation

The policy is applied separately to each operation. As the policy is applied to each operation in turn, that operation becomes the current operation. Each rule is applied sequentially to the current operation. All of the rules are applied to the current operation unless an action is executed by a prior rule that causes subsequent rules to no longer be applied.

### 3.1.3 Current Object

The object that is described by the `src-dn`, `src-entry-id`, `dest-dn`, `dest-entry-id` and `association` becomes the current object. For more information about the different elements, see the [NDS DTD \(http://developer.novell.com/ndk/doc/dirxml/dirxmlbk/ref/ndsdttd/index.html\)](http://developer.novell.com/ndk/doc/dirxml/dirxmlbk/ref/ndsdttd/index.html).

## 3.2 Policy Builder Tasks in iManager

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

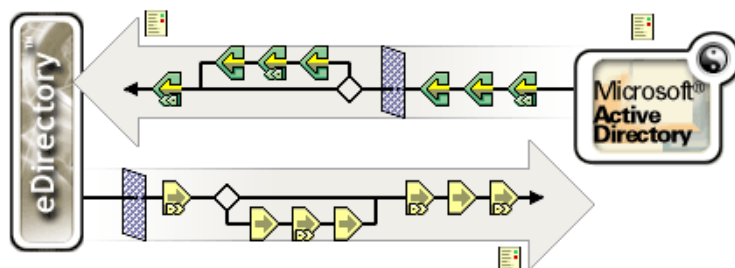
- [Section 3.2.1, “Opening The Policy Builder,” on page 184](#)
- [Section 3.2.2, “Creating a Policy,” on page 185](#)
- [Section 3.2.5, “Modifying a Policy,” on page 194](#)
- [Section 3.2.3, “Defining Individual Rules within a Policy,” on page 185](#)
- [Section 3.2.4, “Defining Individual Arguments within a Rule,” on page 187](#)
- [Section 3.2.12, “Using Predefined Rules,” on page 196](#)

### 3.2.1 Opening The Policy Builder

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




Figure 3-1 Identity Manager Driver Overview




Policies are managed from the Identity Manager Driver Overview.

### 3.2.2 Creating a Policy

- 1 Open the *Identity Manager Driver Overview* for the driver you want to manage.
- 2 Click the icon representing the policy you want to define.

 represents an undefined policy.

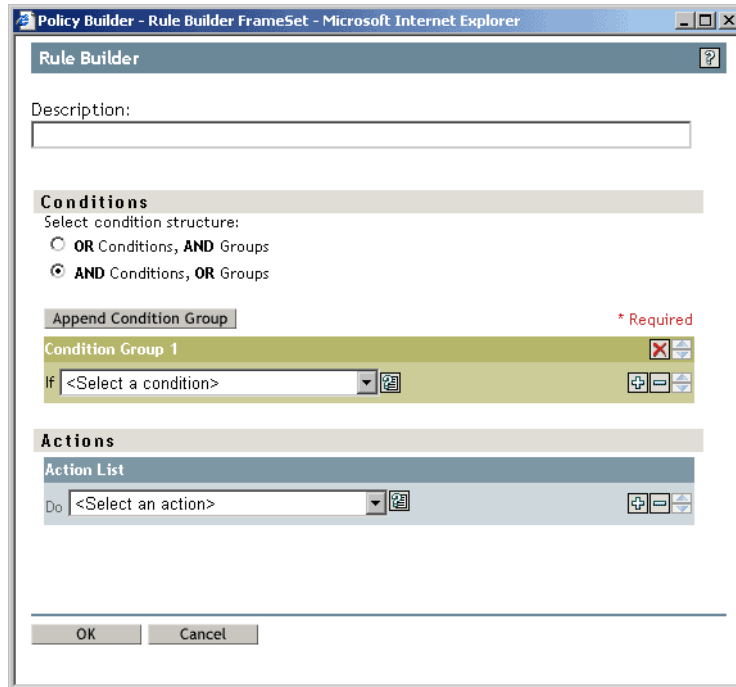
 represents a defined policy.

- 3 Click *Insert*.
- 4 Enter a name for the new policy, then select the 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 [Section 3.2.3, “Defining Individual Rules within a Policy,”](#) on [page 185](#).

### 3.2.3 Defining Individual Rules within a Policy

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

**Figure 3-2** Rule Builder Window of Policy Builder



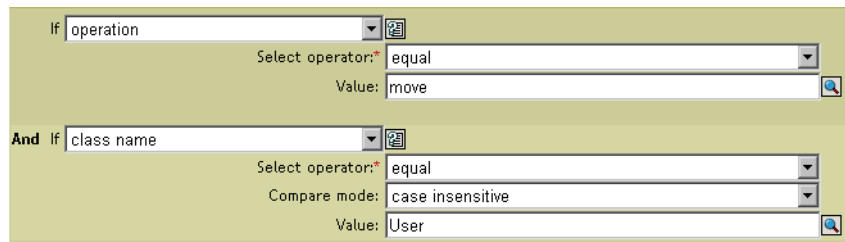
The Rule Builder interface enables you to quickly create and modify rules using intelligent drop-down menus.

In the 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.

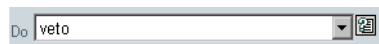
To implement this logic in the Rule Builder, you could select the following condition:

**Figure 3-3** Move User Condition in the Rule Builder Interface



And the following action:

**Figure 3-4** Veto Action in the Rule Builder Interface



See [Section 3.5, “Conditions,”](#) on page 218 and [Section 3.6, “Actions,”](#) on page 236 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:

**Figure 3-5** Condition Structure Radio Buttons

Select condition structure:

- OR Conditions, AND Groups
- AND Conditions, OR Groups

- 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 to add condition groups.
- Use the   icons to remove and position condition groups.

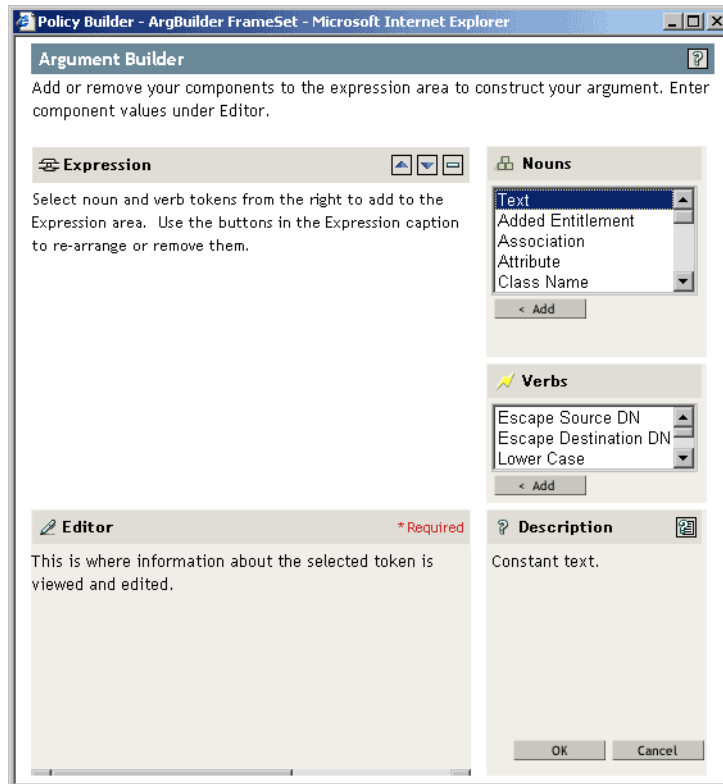
### 3.2.4 Defining Individual Arguments within a Rule

The Argument Builder provides a dynamic graphical interface that enables you to construct complex argument expressions for use within the Rule Builder. To access the Argument Builder, see [“Argument Builder” on page 190](#).

Arguments are dynamically used by actions and are derived from tokens that are expanded at run time.

Tokens are broken up into two classifications: nouns and verbs. Noun tokens expand to values that are derived from the current operation, the source or destination data stores, or some external source. Verb tokens modify the concatenated results of other tokens that are subordinate to them.

**Figure 3-6** *Default Argument Builder Interface*



To define an expression, select one or more nouns tokens (values, objects, variables, etc.), and combine them with verb tokens (substring, escape, uppercase, and lowercase) to construct arguments. Multiple tokens are combined to construct complex arguments.

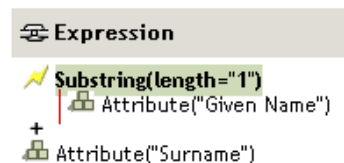
For example, if you want the argument set to an attribute value, you select the attribute token, then select the attribute name:

**Figure 3-7** *Editor Displaying ds.novell as a Text Argument*



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



**Figure 3-8** *Expression Displaying a Substring of Length 1 on the Given Name Attribute, Combined with the Surname Attribute.*



After you add a token, you can edit its fields in the editor.

See [Section 3.7, “Noun Tokens,” on page 274](#) and [Section 3.8, “Verb Tokens,” on page 287](#) 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 noun tokens and verb tokens.
- Click the  icon to see a list of values for a field.
- After you add a noun token or a verb token, you can provide values in the editor, then immediately add another noun token or verb token. 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 the Argument Builder, there are several more builders that are used by the Condition Editor and Action Editor in the Policy Builder. Each builder can recursively call anyone of the builders in the following list:

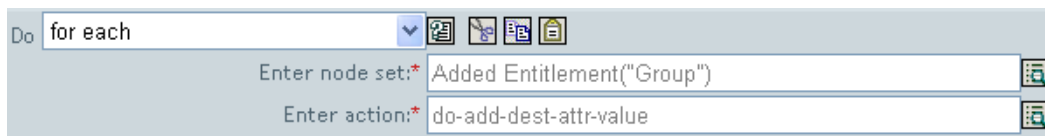
- [“Argument Actions Builder” on page 189](#)
- [“Argument Builder” on page 190](#)
- [“Match Attribute Builder” on page 191](#)
- [“Action Argument Component Builder” on page 192](#)
- [“Argument Value List Builder” on page 192](#)
- [“Named String Builder” on page 193](#)
- [“Condition Argument Component Builder” on page 193](#)

## Argument Actions Builder

The Argument Actions Builder enables you to set the action that is required by the [For Each \(page 249\)](#) action and the [Implement Entitlement \(page 252\)](#) action.

In the following example, the add destination attribute value action is performed for each Group entitlement that is being added in the current operation.

**Figure 3-9** *Argument Actions Builder*



To define the action of add destination attribute value, click the icon that launches the Argument Actions Builder. In the Argument Actions Builder, you define the desired action. In the following example, the member attribute is added to the destination object for each added Group entitlement.

**Figure 3-10** *Argument Actions Builder*

The screenshot shows the 'Argument Actions Builder' window. At the top, there is a tab labeled 'Actions'. Below it is the 'Action List' section. The first action in the list is 'add destination attribute value'. The configuration fields for this action are as follows:

- Enter attribute name: Member
- Enter class name: Group
- Select mode: add to current operation
- Select object: DN
- Enter DN: Local Variable("current-node")
- Enter value type: string
- Enter tokens: Destination DN()

**Figure 3-11** *Argument Actions Builder*

The screenshot shows the 'Argument Actions Builder' window. The 'Action List' section is visible, and the 'Add Destination Attribute Value' action is selected. The configuration fields for this action are as follows:

- Do: <Select an action>

## Argument Builder

Launch the Argument Builder from the following actions by clicking the Edit Arguments icon.

- [Add Association \(page 237\)](#)
- [Add Destination Attribute Value \(page 238\)](#)
- [Add Destination Object \(page 239\)](#)
- [Add Source Attribute Value \(page 240\)](#)
- [Append XML Text \(page 243\)](#)
- [Clear Destination Attribute Value \(page 244\)](#) When the selected object is DN or Association.
- [Clear Source Attribute Value \(page 245\)](#) When the selected object is DN or Association.
- [Delete Destination Object \(page 247\)](#) When the selected object is DN or Association.
- [Delete Source Object \(page 247\)](#) When the selected object is DN or Association.
- [Find Matching Object \(page 248\)](#)
- [For Each \(page 249\)](#)
- [Move Destination Object \(page 253\)](#)
- [Move Source Object \(page 254\)](#)
- [Reformat Operation Attribute \(page 254\)](#)
- [Remove Association \(page 255\)](#)
- [Remove Destination Attribute Value \(page 256\)](#)
- [Remove Source Attribute Value \(page 257\)](#)

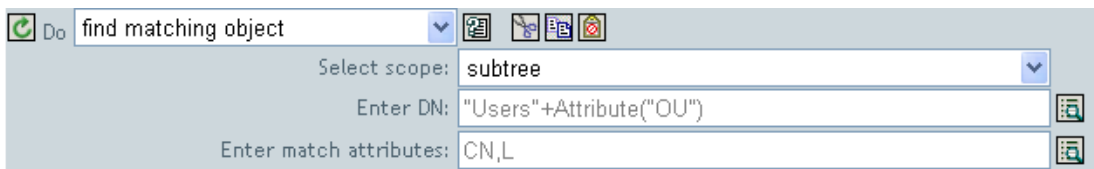
- [Rename Destination Object \(page 258\)](#) When the selected object is DN or Association and Enter String.
- [Rename Source Object \(page 258\)](#) When the selected object is DN or Association and Enter String.
- [Set Destination Attribute Value \(page 262\)](#) When the selected object is DN or Association and Enter Value type is not structured.
- [Set Destination Password \(page 263\)](#)
- [Set Local Variable \(page 264\)](#)
- [Set Operation Association \(page 265\)](#)
- [Set Operation Class Name \(page 265\)](#)
- [Set Operation Destination DN \(page 266\)](#)
- [Set Operation Property \(page 266\)](#)
- [Set Operation Source DN \(page 267\)](#)
- [Set Operation Template DN \(page 267\)](#)
- [Set Source Attribute Value \(page 268\)](#)
- [Set Source Password \(page 269\)](#)
- [Set XML Attribute \(page 269\)](#)
- [Status \(page 270\)](#)
- [Trace Message \(page 271\)](#)

### Match Attribute Builder

The Match Attribute Builder enables you to select attributes and values used by the [Section 3.6.16, “Find Matching Object,” on page 248](#) action to determine if a matching object exists in a data store.

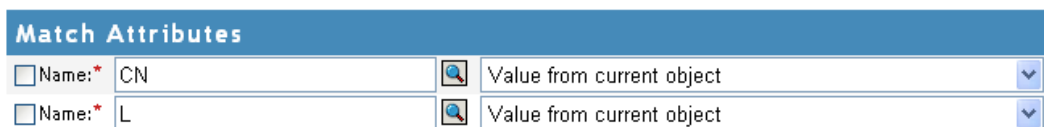
For example, if you want to match users based on a common name and a location, you would select the following condition:

**Figure 3-12** Find Matching Object



You then click the Edit Arguments icon next to the Enter Match Attributes field to launch the Match Attribute Builder interface:

**Figure 3-13** Match Attribute Builder



Select the *Browse attributes* icon to browse to and select the attributes you want to match. In this example they are L and CN.

The second column allows you to match the current value stored in the attribute by selecting *Use value(s) from current Object*. You can match against another value by selecting *Other Value*. You can create any value you want to match. Select the value type, and the appropriate builder is available through the *Enter State* field.

## Action Argument Component Builder

Launch the Action Argument Component Builder by selecting the following actions when the Enter Value Type selection is the Structured selection.

- [Add Destination Attribute Value \(page 238\)](#)
- [Add Source Attribute Value \(page 240\)](#)
- [Reformat Operation Attribute \(page 254\)](#)
- [Remove Destination Attribute Value \(page 256\)](#)
- [Remove Source Attribute Value \(page 257\)](#)
- [Set Default Attribute Value \(page 261\)](#)
- [Set Source Attribute Value \(page 268\)](#)

**Figure 3-14** Action Argument Component Builder

The screenshot shows the 'Action List' dialog box. The selected action is 'add destination attribute value'. The fields are filled with the following values: 'Enter attribute name:\*' is 'Given Name', 'Enter class name:' is 'User', 'Select mode:' is 'write directly to destination datastore', 'Select object:' is 'Current object', 'Enter value type:' is 'structured' (circled in red), and 'Enter components:\*' is 'user'. There are several icons for actions like 'Add', 'Remove', and 'Help'.

**Figure 3-15** Action Argument Component Builder

The screenshot shows the 'Argument Component Builder' dialog box. It contains a text area with the following text: 'The argument components provide values for components of the enclosing <arg-value> when the type attribute of <arg-value> is 'structured'. Each of the enclosed tokens is evaluated and the resulting string values are concatenated to form the value of the component. The name of the component is specified by the name attribute.' Below the text area are two buttons: 'Append New Component' and 'Remove'. At the bottom, there is a section titled 'Argument Components' with two input fields: 'Name:\*' and 'Tokens:\*'. A '\* Required' label is visible in the top right corner.

## Argument Value List Builder

The Argument Value List Builder enables you to construct default argument values for the [Set Default Attribute Value \(page 261\)](#) action.

For example, if you want to set a default location of Unknown, you select the following action:



**Figure 3-16** *Argument Value List Builder*



You then click the icon next to the Enter Values field to launch the Argument Value List Builder interface, and construct an argument similar to the following:

**Figure 3-17** *Argument Value List Builder*



### Named String Builder

The Named String Builder enables you to construct name/value pairs for use in certain actions such as [Generate Event \(page 250\)](#), [Send Email \(page 259\)](#) and [Send Email from Template \(page 260\)](#).

For a Generate Event action, the named strings correspond to the custom value fields you can provide with an event:

**Figure 3-18** *Named String Builder*

Strings			
<input type="checkbox"/> Name:	to	String tokens:	"to_user1@company.com"
<input type="checkbox"/> Name:	to	String tokens:	"to_user2@company.com"
<input type="checkbox"/> Name:	cc	String tokens:	"cc_user@company.com"
<input type="checkbox"/> Name:	bcc	String tokens:	"bcc_user@company.com"
<input type="checkbox"/> Name:	from	String tokens:	"from_user@company.com"
<input type="checkbox"/> Name:	subject	String tokens:	"This is the e-mail subject"
<input type="checkbox"/> Name:	message	String tokens:	"This is the e-mail body"

For a Send Mail action, the named strings correspond to the elements of the e-mail:

**Figure 3-19** *Send Mail Action*

Strings			
<input type="checkbox"/> Name:	manager	String tokens:	"Bill Jones"
<input type="checkbox"/> Name:	surname	String tokens:	"Smith"
<input type="checkbox"/> Name:	given-name	String tokens:	"Joe"
<input type="checkbox"/> Name:	to	String tokens:	"to_user@company.com"
<input type="checkbox"/> Name:	cc	String tokens:	"cc_user@company.com"

A complete list of possible values is contained in the help file corresponding to the action that launches the Named String Builder.

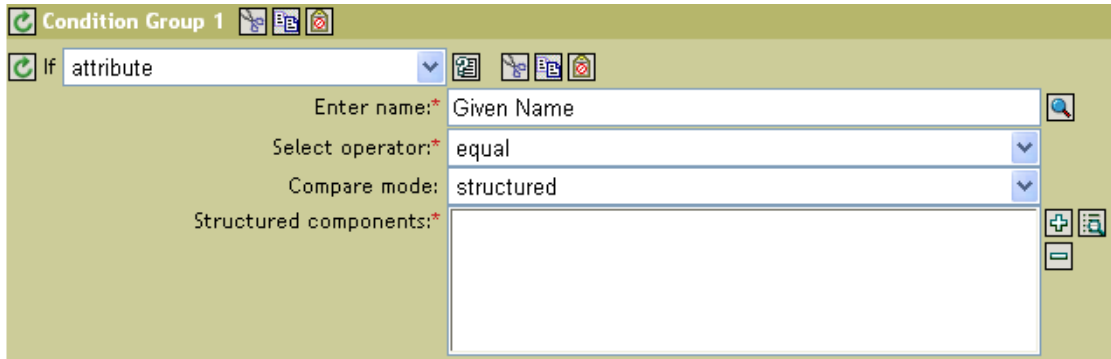
### Condition Argument Component Builder

Launch the Condition Argument Component Builder by clicking the Edit Arguments Icon.

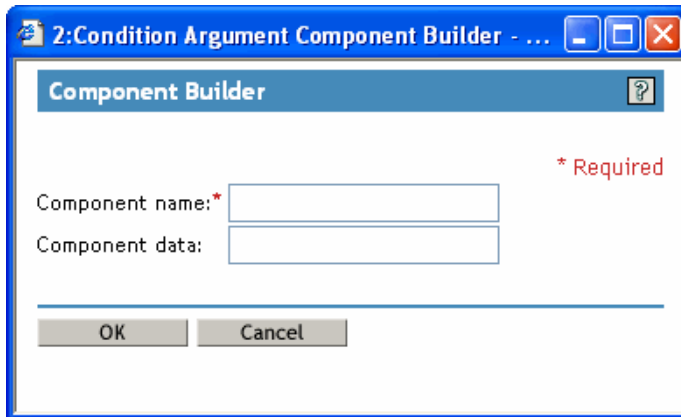
In order to see the icon, you must select the Structured selection for Mode with the following conditions:

- [If Attribute \(page 219\)](#)
- [If Destination Attribute \(page 221\)](#)
- [If Source Attribute \(page 232\)](#)

**Figure 3-20** Structured Option



**Figure 3-21** Condition Argument Component Builder



### 3.2.5 Modifying a Policy


- 1 Open the *Identity Manager 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*.

### 3.2.6 Removing a Policy

Removes the policy from the selected Policy Set but doesn't delete the policy.

- 1 Open the *Identity Manager Driver Overview* for the driver you want to manage.
- 2 Click the icon representing the policy you want to remove.

To view a policy that is not associated with a policy set:

- 1 Open the *Identity Manager Driver Overview* for the driver you want to manage.
- 2 Click the *View All Policies icon* .

To add the removed policy back to the policy set:

- 1 Open the *Identity Manager Driver Overview* for the driver you want to manage.
- 2 Click the policy set where you want to add the policy.
- 3 Click *Insert*.
- 4 Select *Use an existing policy*, then click the browse button.
- 5 Browse to the policy you want to add.

---

**TIP:** Make sure you are in the proper container to see the policy.

---

- 6 Click *OK*.
- 7 Click *Close*.

### 3.2.7 Renaming a Policy

- 1 Open the *Identity Manager Driver Overview* for the driver you want to manage.
- 2 Click the icon representing the policy you want to rename.
- 3 Click *Rename* and rename the policy.
- 4 Click *OK*.
- 5 Click *Close*.

### 3.2.8 Deleting a Policy

- 1 Open the *Identity Manager 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 *Delete*.

### 3.2.9 Importing a Policy from an XML File

- 1 Open the *Identity Manager Driver Overview* for the driver you want to manage.
- 2 Click the icon representing the policy you want to import.
- 3 Select the policy, then click *Edit*.
- 4 Click the *Insert* button, then select *Import an XML file containing DirXML<sup>®</sup> Script*.
- 5 Browse to and select the policy file to import, then click *OK*.

### 3.2.10 Exporting a Policy to an XML File

- 1 Open the *Identity Manager Driver Overview* for the driver you want to manage.
- 2 Click the icon representing the policy you want to export.
- 3 Select the policy, then click *Edit*.
- 4 Click the *Save As* button, then select a location to save the DirXML Script XML file.
- 5 Click *Save*.

### 3.2.11 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 *Identity Manager Driver Overview* for the driver you want to manage.
- 2 Click the icon representing the policy you want to add as a reference.
- 3 Select the policy, then click *Edit*.
- 4 Click the *Insert* button, and select *Append a reference to a policy containing DirXML Script*.
- 5 Browse to and select the policy object to reference, then click *OK*.

### 3.2.12 Using Predefined Rules

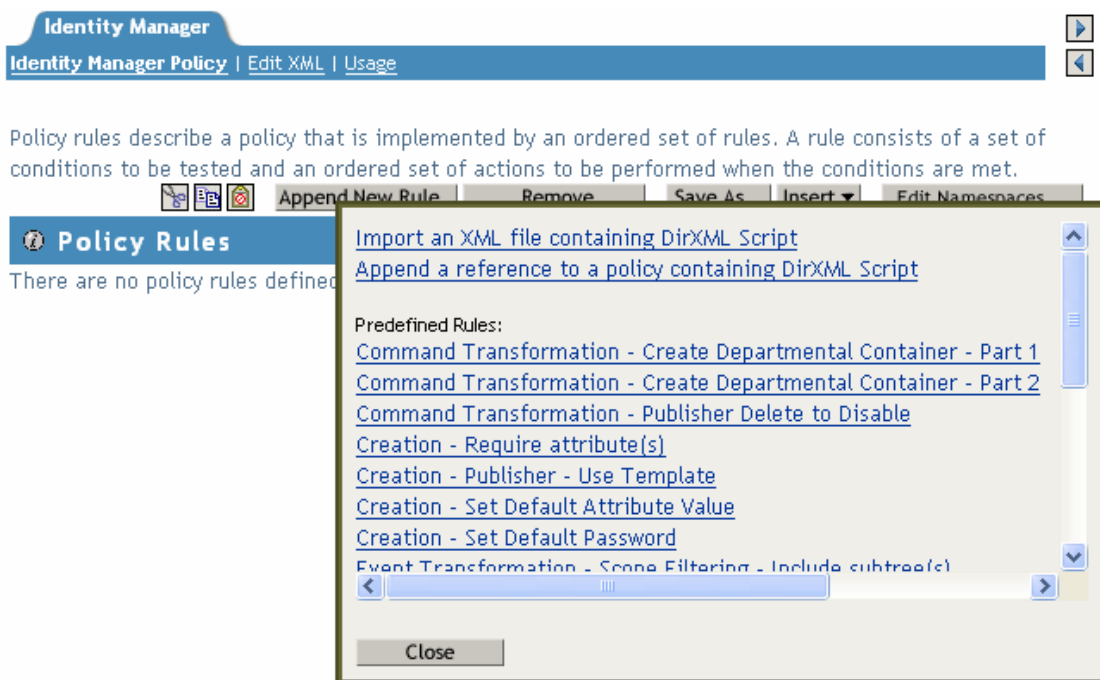
iManager includes twenty predefined rules. You can import and use these rules as well as create your own rules. These rules include common tasks that administrators use. You need to provide information specific to your environment to customize the rules.

- “Command Transformation - Create Departmental Container - Part 1 and Part 2” on page 197
- “Command Transformation - Publisher Delete to Disable” on page 199
- “Creation - Require Attributes” on page 199
- “Creation - Publisher - Use Template” on page 200
- “Creation - Set Default Attribute Value” on page 201
- “Creation - Set Default Password” on page 202
- “Event Transformation - Scope Filtering - Include Subtrees” on page 203
- “Event Transformation - Scope Filtering - Exclude Subtrees” on page 204
- “Input or Output Transformation - Reformat Telephone Number from (nnn) nnn-nnnn to nnn-nnn-nnnn” on page 205
- “Input or Output Transformation - Reformat Telephone Number from nnn-nnn-nnnn to (nnn) nnn-nnnn” on page 206
- “Matching - Publisher Mirrored” on page 207
- “Matching - Subscriber Mirrored - LDAP Format” on page 208
- “Matching - By Attribute Value” on page 209
- “Placement - Publisher Mirrored” on page 210
- “Placement - Subscriber Mirrored - LDAP Format” on page 211
- “Placement - Publisher Flat” on page 212
- “Placement - Subscriber Flat - LDAP Format” on page 213
- “Placement - Publisher By Dept” on page 214
- “Placement - Subscriber By Dept - LDAP Format” on page 215

To access the predefined rules:

- 1 Open the *Identity Manager Driver Overview* for the driver you want to manage.
- 2 Click the icon representing the policy where you want to add the predefined rule.

- 3 Select a policy, then click *Edit*.
- 4 Click *Insert* and select the predefined rule you want to use.



## Command Transformation - Create Departmental Container - Part 1 and Part 2

Creates a department container in the destination data store, if one does not exist. Implement the rule on the Subscriber Command Transformation policy or Publisher Command Transformation policy in the driver.

There are two steps involved in using the predefined rules: creating a policy in the Command Transformation policy set, and importing the predefined rule. If you already have a Command Transformation policy that you want to add this rule to, skip to [“Importing the Predefined Rule” on page 197](#).

### Creating a Policy

- 1 Open the *Identity Manager Driver Overview* for the driver you want to manage.
- 2 Click the Command Transformation Policy object on the Publisher or Subscriber channel.
- 3 Click *Insert*.
- 4 Name the policy, make sure to implement the policy with the Policy Builder, then click *OK*.

The Rule Builder is launched.

### Importing the Predefined Rule

- 1 In the Rule Builder, click *Insert*.
- 2 Select *Command Transformation - Create Department Container - Part 1*.
- 3 Click *Insert*.
- 4 Select *Command Transformation - Create Department Container - Part 2*.

5 Click *OK*.

There is no information to change in the rules that is specific to your environment.

The screenshot displays two parts of a rule configuration in Active Directory Policy Builder. Part 1, titled "Command Transformation - Create Departmental Container - Part 1", has a condition "if operation equal 'add'" and two actions: "set local variable('target-container', Destination DN(length='-2'))" and "set local variable('does-target-exist', Destination Attribute('objectclass', class name='OrganizationalUnit', dn(Local Variable('target-container'))))". Part 2, titled "Command Transformation - Create Departmental Container - Part 2", has a condition "if local variable 'does-target-exist' available" and an "And" condition "if local variable 'does-target-exist' equal ''". Its actions are "add destination object(class name='organizationalUnit', direct='true', dn(Local Variable('target-container')))" and "add destination attribute value('ou', direct='true', dn(Local Variable('target-container')), Parse DN('dest-dn', 'dot', length='1', start='-1', Local Variable('target-container')))"

---

**IMPORTANT:** Make sure that the rules are listed in order. Part 1 must be executed before Part 2.

---

### How the Logic in the Rule Works

The rule is used when the destination location for an object does not exist. Instead of getting a veto because the object cannot be placed, this rule creates the container and places the object in the container.

Part 1 looks for any Add operation. When the Add operation occurs, two local variables are set. The first local variable is named target-container. The value of target-container is set to the destination DN. The second local variable is named does-target-exist. The value of does-target-exist is set to the destination attribute value of objectclass. The class is set to OrganizationalUnit. The DN of the OrganizationalUnit is set to the local variable of target-container.

**Figure 3-22** Create Container

The screenshot shows the "Editor" window in Active Directory Policy Builder. It contains three input fields: "Name:" with the value "objectclass", "Class name:" with the value "OrganizationalUnit", and "Select object:" with a dropdown menu set to "DN" and a text box containing "Local Variable('target-container')".

Part 2 checks to see if the local variable does-target-exist is available. It also checks to see if the value of the local variable does-target-exist is set to a blank value. If the value is blank, then an Organizational Unit object is created. The DN of the organizational unit is set to the value of the local variable target-container. It also adds the value for the OU attribute. The value of the OU attribute is set to the name of the new organizational unit, which is obtained by parsing the value of the local variable target-container.

## Command Transformation - Publisher Delete to Disable

Transforms a Delete operation for a User object into a Modify operation that disables the target User object in eDirectory™. Implement the rule on the Publisher Command Transformation policy in the driver.

There are two steps involved in using the predefined rules: creating a policy in the Command Transformation policy set, and importing the predefined rule. If you already have a Command Transformation policy that you want to add this rule to, skip to [Importing the Predefined Rule \(page 199\)](#).

### Creating a Policy

- 1 Open the *Identity Manager Driver Overview* for the driver you want to manage.
- 2 Click the Command Transformation Policy object on the Publisher channel.
- 3 Click *Insert*.
- 4 Name the policy, make sure to implement the policy with the Policy Builder, then click *OK*.

The Rule Builder is launched.

### Importing the Predefined Rule

- 1 In the Rule Builder, click *Insert*.
- 2 Select *Command Transformation - Publisher Delete to Disable*.
- 3 Click *OK*.

There is no information to change in the rule that is specific to your environment.



### How the Logic in the Rule Works

The rule is used when a Delete command is going to be sent to the Identity Vault, usually in response to a Delete event that occurred in the connected system. Instead of the User object being deleted in the Identity Vault, the User object is disabled. When a Delete command is processed for a User object, the destination attribute value of Login Disabled is set to true, the association is removed from the User object, and the Delete command is vetoed. The User object can no longer log in into the Novell eDirectory tree, but the User object was not deleted.

## Creation - Require Attributes

Prevents User objects from being created unless the required attributes are populated. Implement the rule on the Subscriber Creation policy or the Publisher Creation policy in the driver.

There are two steps involved in using the predefined rules: creating a policy in the Creation policy set, and importing the predefined rule. If you already have a Creation policy that you want to add this rule to, skip to [“Importing the Predefined Rule” on page 200](#).

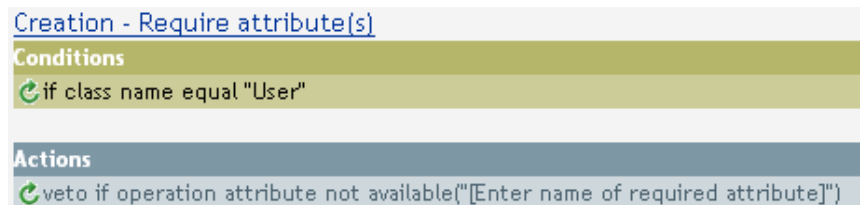
### Creating a Policy

- 1 Open the *Identity Manager Driver Overview* for the driver you want to manage.
- 2 Click the Creation Policy object on the Publisher or Subscriber channel.
- 3 Click *Insert*.
- 4 Name the policy, make sure to implement the policy with the Policy Builder, then click *OK*.

The Rule Builder is launched.

### Importing the Predefined Rule

- 1 In the Rule Builder, click *Insert*.
- 2 Select *Creation - Required Attributes*.
- 3 Click *Creation - Required Attributes* in the Rule Builder, to edit the rule.
- 4 Delete *[Enter name of required attribute]* from the *Enter Name* field.
- 5 Click the browse icon, then browse to and select the attribute you require for a User object to be created.
- 6 (Optional) If you want more than one required attribute, click the *plus icon* to add a new action.
- 7 Select *Veto if operation attribute not available* and browse to the additional required attribute.
- 8 Click *OK*.



### How the Logic in the Rule Works

The rule is used when your business processes require that a user has specific attributes populated in the source User object before the destination the User object can be created. When a User object is created in the source data store, the rule vetoes the creation of the object in the destination data store unless the required attributes are provided when the User object is created. You can have one or more required attributes.

### Creation - Publisher - Use Template

Allows for the use of a Novell eDirectory template object during the creation of a User object. Implement the rule on the Publisher Creation policy in the driver.

There are two steps involved in using the predefined rules: creating a policy in the Creation policy set, and importing the predefined rule. If you already have a Creation policy that you want to add this rule to, skip to [“Importing the Predefined Rule” on page 201](#).



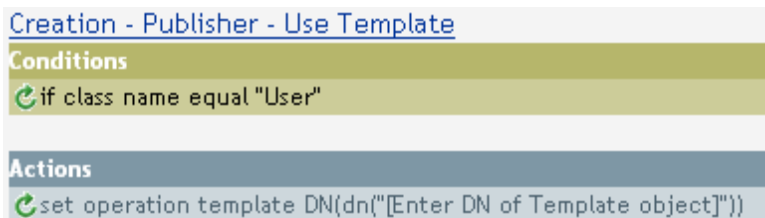
## Creating a Policy

- 1 Open the *Identity Manager Driver Overview* for the driver you want to manage.
- 2 Click the Creation Policy object on the Publisher or Subscriber channel.
- 3 Click *Insert*.
- 4 Name the policy, make sure to implement the policy with the Policy Builder, then click *OK*.

The Rule Builder is launched.

## Importing the Predefined Rule

- 1 In the Rule Builder, click *Insert*.
- 2 Select *Creation - Publisher - Use Template*.
- 3 Click *Creation - Publisher - Use Template* in the Rule Builder, to edit the rule.
- 4 Delete *[Enter DN of Template object]* from the *Enter DN field*.
- 5 Click the *Edit Arguments icon* to launch the Argument Builder.
- 6 Select *Text* in the Noun list, then click *Add*.
- 7 In the Editor, click the browse icon and browse to and select the template object, then click *OK*.
- 8 Click *OK*.



## How the Logic in the Rule Works

The rule is used when you want to create a user in the Identity Vault based on a template object. If you have attributes that are the same for users, using the template saves time. You fill in the information in the template object and when the User object is created, Identity Manager uses the attribute values from the template to create the User object.

During the creation of User objects, the rule does the action of the set operation template DN, which instructs the Identity Manager to use the referenced template when creating the object.

## Creation - Set Default Attribute Value

Allows you to set default values for attributes that are assigned during the creation of User objects. Implement the rule on the Subscriber Creation policy or Publisher Creation policy in the driver.

There are two steps involved in using the predefined rules: creating a policy in the Creation policy set, and importing the predefined rule. If you already have a Creation policy that you want to add this rule to, skip to [“Importing the Predefined Rule” on page 202](#).

## Creating a Policy

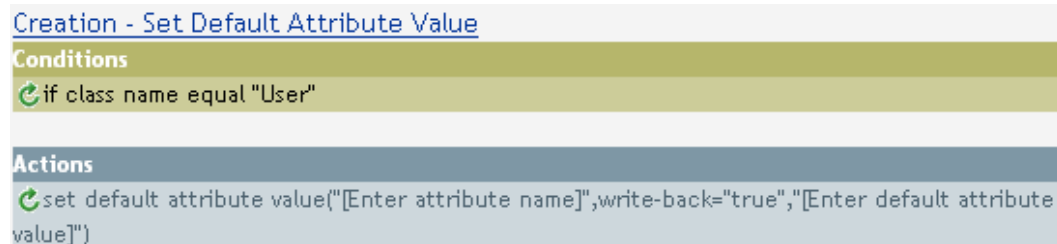
- 1 Open the *Identity Manager Driver Overview* for the driver you want to manage.
- 2 Click the Creation Policy object on the Publisher or Subscriber channel.

- 3 Click *Insert*.
- 4 Name the policy, make sure to implement the policy with the Policy Builder, then click *OK*.

The Rule Builder is launched.

### Importing the Predefined Rule

- 1 In the Rule Builder, click *Insert*.
- 2 Select *Creation - Set Default Attribute Value*.
- 3 Click *Set Default Attribute Value* in the Rule Builder, to edit the rule.
- 4 Delete *[Enter attribute name]* from the *Enter attribute name field*.
- 5 Click the browse icon, then browse to and select the attribute you want to have created.
- 6 Delete *[Enter default attribute value]* from the *Enter arguments values field*.
- 7 Click the *Edit Arguments icon* to launch the Argument Values List Builder.
- 8 Select the type of data you want the value to be.
- 9 Click the *Edit Arguments icon* to launch the Argument Builder.
- 10 Create the value you want the attribute to be through the Argument Builder, then click *OK*.
- 11 Click *OK*.



### How the Logic in the Rule Works

The rule is used when you want to populate default attribute values when creating a User object. When a User object is created, the rule adds the specified attribute values if and only if the attribute has no values supplied by the source object.

If you want more than one attribute value defined, right-click the action and click *New > Action*. Select the action, set the default attribute value, and follow the steps above to assign the value to the attribute.

### Creation - Set Default Password

During the creation of User objects, it sets a default password for User objects. Implement the rule on the Subscriber Creation policy or Publisher Creation policy in the driver.

There are two steps involved in using the predefined rules: creating a policy in the Creation policy set, and importing the predefined rule. If you already have a Creation policy that you want to add this rule to, skip to [“Importing the Predefined Rule” on page 203](#).

### Creating a Policy

- 1 Open the *Identity Manager Driver Overview* for the driver you want to manage.
- 2 Click the Creation Policy object on the Publisher or Subscriber channel.

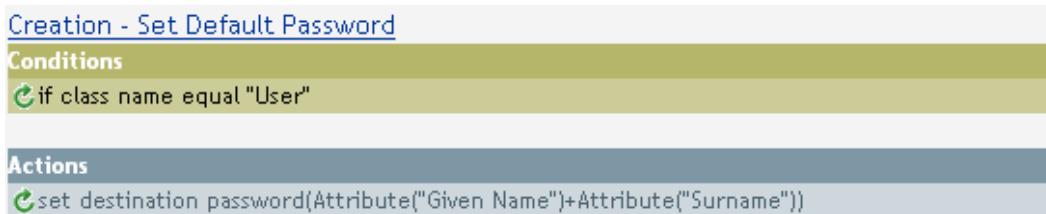
- 3 Click *Insert*.
- 4 Name the policy, make sure to implement the policy with the Policy Builder, then click *OK*.

The Rule Builder is launched.

### Importing the Predefined Rule

- 1 In the Rule Builder, click *Insert*.
- 2 Select *Creation - Set Default Password*.
- 3 Click *OK*.

There is no information to change in the rule that is specific to your environment.



### How the Logic in the Rule Works

The rule is used when you want User objects to be created with a default password. During the creation of a User object, the password that is set for the User object is the Given Name attribute plus the Surname attribute of the User object.

You can change the value of the default password by editing the argument. You can set the password to any other value you want through the Argument Builder.

### Event Transformation - Scope Filtering - Include Subtrees

Excludes all events that occur outside of the specific subtrees. Implement the rule on the Subscriber Event Transformation policy or the Publisher Event Transformation policy in the driver.

There are two steps involved in using the predefined rules: creating a policy in the Event Transformation policy set, and importing the predefined rule. If you already have an Event Transformation policy that you want to add this rule to, skip to [“Importing the Predefined Rule” on page 203](#).

### Creating a Policy

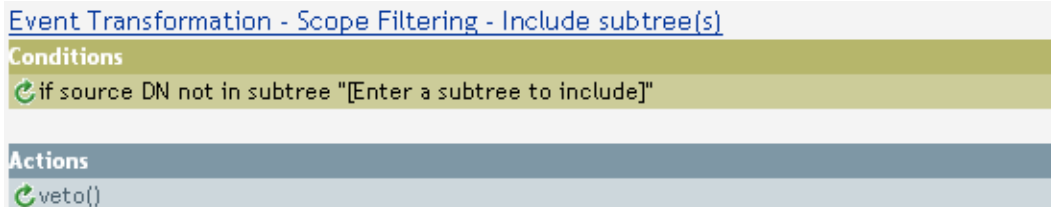
- 1 Open the *Identity Manager Driver Overview* for the driver you want to manage.
- 2 Click the Event Transformation Policy object on the Publisher or Subscriber channel.
- 3 Click *Insert*.
- 4 Name the policy, make sure to implement the policy with the Policy Builder, then click *OK*.

The Rule Builder is launched.

### Importing the Predefined Rule

- 1 In the Rule Builder, click *Insert*.
- 2 Select *Event Transformation - Scope Filtering - Include subtrees*.

- 3 Click *Event Transformation - Scope Filtering - Include subtrees* in the Rule Builder, to edit the rule.
- 4 Delete *[Enter a subtree to include]* in the *Value* field.
- 5 Click the browse button to browse the Identity Vault for the part of the tree you were you want events to synchronize, then click *OK*.
- 6 Click *OK*.



### How the Logic in the Rule Works

The rule is used when you only want to synchronize specific subtrees between the Identity vault and the connected system. When an event occurs anywhere but in that specific part of the Identity Vault, it is vetoed. You can add additional subtrees to be synchronized by copying and pasting the [Section 3.5.15, "If Source DN," on page 234](#) condition.

### Event Transformation - Scope Filtering - Exclude Subtrees

Excludes all events that occur in a specific subtree. Implement the rule on the Subscriber Event Transformation or the Publisher Event Transformation policy in the driver.

There are two steps involved in using the predefined rules: creating a policy in the Event Transformation policy set, and importing the predefined rule. If you already have an Event Transformation policy that you want to add this rule to, skip to ["Importing the Predefined Rule" on page 204](#).

#### Creating a Policy

- 1 Open the Identity Manager Driver Overview for the driver you want to manage.
- 2 Click the Event Transformation Policy object on the Publisher or Subscriber channel.
- 3 Click *Insert*.
- 4 Name the policy, make sure to implement the policy with the Policy Builder, then click *OK*.

The Rule Builder is launched.

#### Importing the Predefined Rule

- 1 In the Rule Builder, click *Insert*.
- 2 Select *Event Transformation - Scope Filtering - Excluding subtrees*.
- 3 Click *Event Transformation - Scope Filtering - Excluding subtrees* in the Rule Builder, to edit the rule.
- 4 Delete *[Enter a subtree to exclude]* in the *Value* field.
- 5 Click the browse button to browse the Identity Vault for the part of the tree you want to exclude events from synchronizing, then click *OK*.
- 6 Click *OK*.

## Event Transformation - Scope Filtering - Exclude subtree(s)

### Conditions

if source DN in subtree "[Enter a subtree to exclude]"

### Actions

veto()

### How the Logic in the Rule Works

The rule is used when you want to exclude part of the Identity Vault or connected system from synchronizing. When an event occurs in that specific part of the Identity Vault, it is vetoed. You can add additional subtrees to be excluded by copying and pasting the if source DN condition.

### Input or Output Transformation - Reformat Telephone Number from (nnn) nnn-nnnn to nnn-xxx-xxxx

Converts the format of the telephone number. Implement the rule on the Input or Output Transformation policy in the driver. Typically, if this rule is used on an Input Transformation, you would then use the rule Reformat Telephone Number from nnn-xxx-xxxx to (nnn) nnn-nnnn on the Output Transformation and vice versa to convert the format back and forth.

There are two steps involved in using the predefined rules: creating a policy in the Input or Output Transformation policy set, and importing the predefined rule. If you already have an Input or Output Transformation policy that you want to add this rule to, skip to [“Importing the Predefined Rule” on page 205](#).

#### Creating a Policy

- 1 Open the *Identity Manager Driver Overview* for the driver you want to manage.
- 2 Click the Input or Output Transformation Policy object on the Publisher or Subscriber channel.
- 3 Click *Insert*.
- 4 Name the policy, make sure to implement the policy with the Policy Builder, then click *OK*.


The Rule Builder is launched.

#### Importing the Predefined Rule


- 1 In the Rule Builder, click *Insert*.
- 2 Select *Input or Output Transformation - Reformat Telephone Number from (nnn) nnn-nnnn to nnn-xxx-xxxx*.
- 3 Click *Input or Output Transformation - Reformat Telephone Number from (nnn) nnn-nnnn to nnn-xxx-xxxx* in the Rule Builder, to edit the rule.
- 4 Define the condition you want to have occur when the telephone number is reformatted.
- 5 Click *OK*.

## [Input or Output Transformation - Reformat Telephone Number from \(nnn\) nnn-nnnn to nnn-xxx-nnnn](#)

### Conditions

 This condition will evaluate to true.

### Actions

 reformat operation attribute("phone",Replace First("^\\(\\d\\d\\d)\\s\*(\\d\\d\\d)-(\\d\\d\\d\\d)\$","\$1-\$2-\$3",Local Variable("current-value")))

## How the Logic in the Rule Works

The rule is used when you want to reformat the telephone number. It finds all the values for the attribute phone in the current operation that match the pattern (nnn) nnn-nnnn and replaces each with nnn-xxx-nnnn.

## Input or Output Transformation - Reformat Telephone Number from nnn-xxx-nnnn to (nnn) nnn-nnnn

Transforms the format of the telephone number. Implement the rule on the Input or Output Transformation policy. Typically, if you use this rule on an Output Transformation, you would use the rule Reformat Telephone Number from (nnn) nnn-nnnn to nnn-xxx-nnnn on the Input Transformation and vice versa to convert the format back and forth.

There are two steps involved in using the predefined rules; creating a policy in the Input or Output Transformation policy set, and importing the predefined rule. If you already have an Input or Output Transformation policy that you want to add this rule to, skip to [“Importing the Predefined Rule” on page 206](#).

### Creating a Policy

- 1 Open the *Identity Manager Driver Overview* for the driver you want to manage.
- 2 Click the Input or Output Transformation Policy object on the Publisher or Subscriber channel.
- 3 Click *Insert*.
- 4 Name the policy, make sure to implement the policy with the Policy Builder, then click *OK*.


The Rule Builder is launched.

### Importing the Predefined Rule


- 1 In the Rule Builder, click *Insert*.
- 2 Select *Input or Output Transformation - Reformat Telephone Number from nnn-xxx-nnnn to (nnn) nnn-nnnn*.
- 3 Click *Input or Output Transformation - Reformat Telephone Number from nnn-xxx-nnnn to (nnn) nnn-nnnn* in the Rule Builder, to edit the rule.
- 4 Define the condition you want to have occur when the telephone number is reformatted.
- 5 Click *OK*.

## [Input or Output Transformation - Reformat Telephone Number from nnn-xxx-xxxx to \(xxx\) xxx-xxxx](#)

### Conditions

 This condition will evaluate to true.

### Actions

 reformat operation attribute("phone",Replace First("^(\d\d\d)-(\d\d\d)-(\d\d\d\d)\$","(\$1) \$2-\$3",Local Variable("current-value")))

## How the Logic in the Rule Works

The rule is used when you want to reformat the telephone number. It finds all the values for the attribute phone in the current operation that match the pattern (nnn) nnn-xxxx and replaces each with nnn-xxx-xxxx.

## Matching - Publisher Mirrored

Finds matches in the Identity Vault for objects in the connected system based on their name and location. Implement the rule on the Publisher Matching policy in the driver.

There are two steps involved in using the predefined rules: creating a policy in the Matching policy set, and importing the predefined rule. If you already have a Matching policy that you want to add this rule to, skip to [“Importing the Predefined Rule” on page 207](#).

## Creating a Policy

- 1 Open the *Identity Manager Driver Overview* for the driver you want to manage.
- 2 Click the Matching Policy object on the Publisher channel.
- 3 Click *Insert*.
- 4 Name the policy, make sure to implement the policy with the Policy Builder, then click *OK*.

The Rule Builder is launched.

## Importing the Predefined Rule

- 1 In the Rule Builder, click *Insert*.
- 2 Select *Matching - Publisher Mirrored*.
- 3 Click *Matching - Publisher Mirrored* in the Rule Builder, to edit the rule.
- 4 Delete *[Enter base of source hierarchy]* from the *Value field*.
- 5 Browse to the container in the source hierarchy where you want the matching to start, then click *OK*.
- 6 Click *OK*.
- 7 Delete *[Enter base of destination hierarchy]* from the *Enter string field*.
- 8 Click on the *Edit Arguments icon* to launch the Argument Builder.
- 9 Select *Text* in the Noun list, then click *Add*.
- 10 In the Editor, click the browse icon and browse to and select the container in the destination hierarchy where you want the source structure to be matched, then click *OK*.
- 11 Click *OK*.

## Matching - Publisher Mirrored

### Conditions

if source DN in subtree "[Enter base of source hierarchy]"

### Actions

set local variable("dest-base","[Enter base of destination hierarchy]")

find matching object(scope="entry",dn{(Local Variable("dest-base")+"\Unmatched Source DN (convert="true")})

## How the Logic in the Rule Works

When an Add event occurs on an object in the connected system that is located within the specified source subtree, the rule constructs a DN that represents the same object name and location within the Identity Vault relative to the specified destination subtree. If the destination objects exists and is of the desired object class then it is considered a match. You must supply the DN's of the source (connected system) and destination (Identity Vault) subtrees.

## Matching - Subscriber Mirrored - LDAP Format

Finds matches in a connected system that uses LDAP format DN's for objects in the Identity Vault based on their name and location. Implement the rule on the Subscriber Matching policy in the driver.

There are two steps involved in using the predefined rules: creating a policy in the Matching policy set, and importing the predefined rule. If you already have a Matching policy that you want to add this rule to, skip to [“Importing the Predefined Rule” on page 208](#).

### Creating a Policy

- 1 Open the *Identity Manager Driver Overview* for the driver you want to manage.
- 2 Click the Matching Policy object on the Subscriber channel.
- 3 Click *Insert*.
- 4 Name the policy, make sure to implement the policy with the Policy Builder, then click *OK*.

The Rule Builder is launched.

### Importing the Predefined Rule

- 1 In the Rule Builder, click *Insert*.
- 2 Select *Matching - Subscriber Mirrored - LDAP format*.
- 3 Click *Matching - Subscriber Mirrored - LDAP format* in the Rule Builder, to edit the rule.
- 4 Delete *[Enter base of source hierarchy]* from the *Value field*.
- 5 Browse to the container in the source hierarchy where you want the matching to start, then click *OK*.
- 6 Click *OK*.
- 7 Delete *[Enter base of destination hierarchy]* from the *Enter String field*.
- 8 Click on the *Edit Arguments icon* to launch the Argument Builder.
- 9 Select *Text* in the Noun list, then click *Add*.



- 10 In the Editor, click the browse icon and browse to and select the container in the destination hierarchy where you want the source structure to be matched, then click OK.
- 11 Click OK.

**Matching - Subscriber Mirrored - LDAP format**

**Conditions**

- if source DN in subtree "[Enter base of source hierarchy]"

**Actions**

- set local variable("dest-base","[Enter base of destination hierarchy]")
- find matching object(scope="entry",dn(Unmatched Source DN(convert="true")+","+Local Variable("dest-base")))

### How the Logic in the Rule Works

When an Add event occurs on an object in the Identity Vault that is located within the specified source subtree, the rule constructs a DN that represents the same object name and location within the connected system relative to the specified destination subtree. If the destination objects exists and is of the desired object class then it is considered a match. You must supply the DN's of the source (Identity Vault) and destination (connected system) subtrees. The connected system must use an LDAP formatted DN.

### Matching - By Attribute Value

Finds matches for objects by specific attribute values. Implement the rule on the Subscriber Matching policy or the Publisher Matching policy in the driver.

There are two steps involved in using the predefined rules; creating a policy in the Matching policy set, and importing the predefined rule. If you already have a Matching policy that you would like to add this rule to, skip to [“Importing the Predefined Rule” on page 209](#).

### Creating a Policy

- 1 Open the *Identity Manager Driver Overview* for the driver you want to manage.
- 2 Click the Matching Policy object on the Publisher channel.
- 3 Click *Insert*.
- 4 Name the policy, make sure to implement the policy with the Policy Builder, then click *OK*.

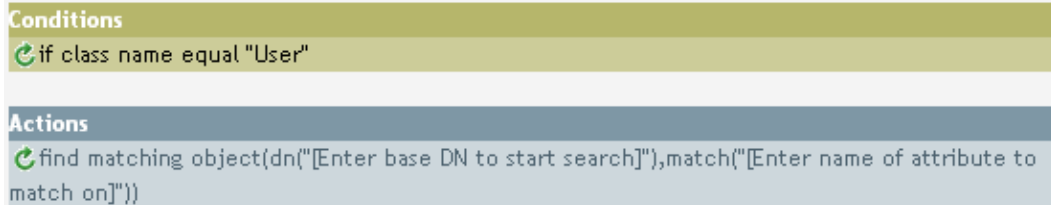
The Rule Builder is launched.

### Importing the Predefined Rule

- 1 In the Rule Builder, click *Insert*.
- 2 Select *Matching - By Attribute Value*.
- 3 Click *Matching - By Attribute Value* in the Rule Builder, to edit the rule.
- 4 Delete *[Enter base DN to start search]* from the *Enter DN field*.
- 5 Click the *Edit Arguments icon* to launch the Argument Builder.
- 6 Select *Text* in the Noun list, then click *Add*.
- 7 In the Editor, click the browse icon and browse to and select the container where you want the search to start, then click *OK*.

- 8 Delete *[Enter name of attribute to match on]* from the *Enter Match Attributes* field.
- 9 Click the *Edit Arguments* icon to launch the Match Attributes Builder.
- 10 Click the browse icon and select the attributes you want to match. You can select one or more attributes to match against, then click *OK*.
- 11 Click *OK*.

#### Matching - by attribute value



#### How the Logic in the Rule Works

When an Add event occurs on an object in the source data store, rule searches for an object in the destination data store that has the same values for the specified attribute. You must supply the DN of the base of the subtree to search in the connected system and the name of the attribute to match on.

#### Placement - Publisher Mirrored

Places objects in the Identity Vault by based on the name and location from the connected system. Implement the rule on the Publisher Placement policy in the driver.

There are two steps involved in using the predefined rules: creating a policy in the Placement policy set, and importing the predefined rule. If you already have a Placement policy that you would like to add this rule to, skip to [“Importing the Predefined Rule” on page 210](#).

#### Creating a Policy

- 1 Open the *Identity Manager Driver Overview* for the driver you want to manage.
- 2 Click the Placement Policy object on the Publisher channel.
- 3 Click *Insert*.
- 4 Name the policy, make sure to implement the policy with the Policy Builder, then click *OK*.

The Rule Builder is launched.

#### Importing the Predefined Rule

- 1 In the Rule Builder, click *Insert*.
- 2 Select *Placement - Publisher Mirrored*.
- 3 Click *Placement - Publisher Mirrored* in the Rule Builder, to edit the rule.
- 4 Delete *[Enter base of source hierarchy]* from the *Value* field.
- 5 Browse to and select the container in the source hierarchy where you want the object to be acted upon, then click *OK*.
- 6 Delete *[Enter base of destination hierarchy]* from the *Enter String* field.
- 7 Click the *Edit Arguments* icon to launch the Argument Builder.
- 8 Select *Text* in the Noun list, then click *Add*.

9 In the Editor, click the browse icon and browse to and select the container in the destination hierarchy where you want the object to be placed, then click *OK*.

10 Click *OK*.

**Placement - Publisher Mirrored**

**Conditions**

- if source DN in subtree "[Enter base of source hierarchy]"

**Actions**

- set local variable("dest-base", "[Enter base of destination hierarchy]")
- set operation destination DN(dn(Local Variable("dest-base")+ "\\" + Unmatched Source DN (convert="true")))

### How the Logic in the Rule Works

If the User object resides in the specified source subtree in the connected system, then the object is placed at the same relative name and location within the Identity Vault. You must supply the DN's of the source (connected system) and destination (Identity Vault) subtrees.

### Placement - Subscriber Mirrored - LDAP Format

Places objects in the data store by using the mirrored structure in the Identity Vault from a specified point. Implement the rule on the Placement policy in the driver. You can implement the rule only on the Subscriber channel.

There are two steps involved in using the predefined rules: creating a policy in the Placement policy set, and importing the predefined rule. If you already have a Placement policy that you want to add this rule to, skip to [“Importing the Predefined Rule” on page 211](#).

### Creating a Policy

- 1 Open the *Identity Manager Driver Overview* for the driver you want to manage.
- 2 Click the Placement Policy object on the Subscriber channel.
- 3 Click *Insert*.
- 4 Name the policy, make sure to implement the policy with the Policy Builder, then click *OK*.

The Rule Builder is launched.

### Importing the Predefined Rule

- 1 In the Rule Builder, click *Insert*.
- 2 Select *Placement - Subscriber Mirrored - LDAP Format*.
- 3 Click *Placement - Subscriber Mirrored - LDAP Format* in the Rule Builder, to edit the rule.
- 4 Delete *[Enter base of source hierarchy]* from the *Value field*.
- 5 Browse to and select the container in the source hierarchy where you want the object to be acted upon, then click *OK*.
- 6 Delete *[Enter base of destination hierarchy]* from the *Enter String field*.
- 7 Click the *Edit Arguments icon* to launch the Argument Builder.
- 8 Select *Text* in the Noun list, then click *Add*.

- 9 In the Editor, click the browse icon and browse to and select the container in the destination hierarchy where you want the object to be placed, then click *OK*.
- 10 Click *OK*.

**Placement - Subscriber Mirrored - LDAP format**

**Conditions**

if source DN in subtree "[Enter base of source hierarchy]"

**Actions**

set local variable("dest-base", "[Enter base of destination hierarchy]")

set operation destination DN(dn(Unmatched Source DN(convert="true")+","+Local Variable ("dest-base")))

### How the Logic in the Rule Works

If the User object resides in the specified source subtree, then the object is placed at the same relative name and location within the Identity Vault. You must supply the DN's of the source (Identity Vault) and destination (connected system) subtrees. The connected system must use an LDAP formatted DN.

### Placement - Publisher Flat

Places objects from the data store into one container in the Identity Vault. Implement the rule on the Publisher Placement policy in the driver.

There are two steps involved in using the predefined rules: creating a policy in the Placement policy set, and importing the predefined rule. If you already have a Placement policy that you want to add this rule to, skip to [“Importing the Predefined Rule” on page 212](#).

### Creating a Policy

- 1 Open the *Identity Manager Driver Overview* for the driver you want to manage.
- 2 Click the Placement Policy object on the Publisher channel.
- 3 Click *Insert*.
- 4 Name the policy, make sure to implement the policy with the Policy Builder, then click *OK*.

The Rule Builder is launched.

### Importing the Predefined Rule

- 1 In the Rule Builder, click *Insert*.
- 2 Select *Placement - Publisher Flat*.
- 3 Click *Placement - Publisher Flat* in the Rule Builder, to edit the rule.
- 4 Delete *[Enter DN of destination container]* from the *Enter String* field.
- 5 Click the *Edit Arguments* icon to launch the Argument Builder.
- 6 Select *Text* in the Noun list, then click *Add*.
- 7 In the Editor, click the browse icon and browse to and select the destination container were you want all of the user objects to be placed, then click *OK*.
- 8 Click *OK*.

## Placement - Publisher Flat

### Conditions

if class name equal "User"

### Actions

```
set local variable("dest-base", "[Enter DN of destination container]")
set operation destination DN(dn(Local Variable("dest-base")+"\"+Escape Destination DN(Unique Name("CN",scope="subtree",Lower Case(Substring(length="1",Operation Attribute("Given Name"))+Operation Attribute("Surname")),Lower Case(Substring(length="2",Operation Attribute("Given Name"))+Operation Attribute("Surname")))))
```

## How the Logic in the Rule Works

The rule places all User objects in the destination DN. The rule sets the DN of the destination container as the local variable dest-base. The rule then sets the destination DN to be dest-base/CN attribute. The CN attribute of the User object is the first two letters of the Given Name attribute plus the Surname attribute as lowercase. The rule uses slash format.

## Placement - Subscriber Flat - LDAP Format

Places objects from the Identity Vault into one container in the data store. Implement the rule on the Subscriber Placement policy in the driver.

There are two steps involved in using the predefined rules: creating a policy in the Placement policy set, and importing the predefined rule. If you already have a Placement policy that you want to add this rule to, skip to [“Importing the Predefined Rule” on page 213](#).

### Creating a Policy

- 1 Open the *Identity Manager Driver Overview* for the driver you want to manage.
- 2 Click the Placement Policy object on the Subscriber channel.
- 3 Click *Insert*.
- 4 Name the policy, make sure to implement the policy with the Policy Builder, then click *OK*.

The Rule Builder is launched.

### Importing the Predefined Rule

- 1 In the Rule Builder, click *Insert*.
- 2 Select *Placement - Subscriber Flat - LDAP Format*.
- 3 Click *Placement - Subscriber Flat - LDAP Format* in the Rule Builder, to edit the rule.
- 4 Delete *[Enter DN of destination container]* from the *Enter String* field.
- 5 Click the *Edit Arguments* icon to launch the Argument Builder.
- 6 Select *Text* in the Noun list, then click *Add*.
- 7 In the Editor, add the destination container were you want all of the User objects to be placed. Make sure the container is specified in LDAP format, then click *OK*.
- 8 Click *OK*.

## Placement - Subscriber Flat - LDAP format

### Conditions

if class name equal "User"

### Actions

```
set local variable("dest-base","[Enter DN of destination container]")
set operation destination DN(dn("uid="+Escape Destination DN(Unique Name
("uid",scope="subtree",Lower Case(Substring(length="1",Operation Attribute("Given Name"))
+Operation Attribute("Surname")),Lower Case(Substring(length="2",Operation Attribute("Given
Name"))+Operation Attribute("Surname")))+","+Local Variable("dest-base")))
```

## How the Logic in the Rule Works

The rule places all User objects in the destination DN. The rule sets the DN of the destination container as the local variable dest-base. The rule then sets the destination DN to be uid=unique name, dest-base. The uid attribute of the User object is the first two letters of the Given Name attribute plus the Surname attribute as lowercase. The rule uses LDAP format.

## Placement - Publisher By Dept

Places objects from one container in the data store into multiple containers in the Identity Vault based on the value of the OU attribute. Implement the rule on the Publisher Placement policy in the driver.

There are two steps involved in using the predefined rules: creating a policy in the Placement policy set, and importing the predefined rule. If you already have a Placement policy that you want to add this rule to, skip to [“Importing the Predefined Rule” on page 214](#).

### Creating a Policy

- 1 Open the *Identity Manager Driver Overview* for the driver you want to manage.
- 2 Click the Placement Policy object on the Publisher channel.
- 3 Click *Insert*.
- 4 Name the policy, make sure to implement the policy with the Policy Builder, then click *OK*.

The Rule Builder is launched.

### Importing the Predefined Rule

- 1 In the Rule Builder, click *Insert*.
- 2 Select *Placement - Publisher By Dept*.
- 3 Click *Placement - Publisher By Dept* to edit the rule.
- 4 Delete *[Enter DN of destination Organization]* from the *Enter String field*.
- 5 Click the *Edit Arguments icon* to launch the Argument Builder.
- 6 Select *Text* in the Noun list, then click *Add*.
- 7 In the Editor, click the browse icon and browse to and select the parent container in the Identity Vault. Make sure all of the department containers are child containers of this DN, then click *OK*.
- 8 Click *OK*.

## Placement - Publisher By Dept

### Conditions

if class name equal "User"

And if attribute 'OU' available

### Actions

set local variable("dest-base","[Enter DN of destination Organization]")

set operation destination DN(dn(Local Variable("dest-base")+Attribute("OU")+Escape Destination DN(Unique Name("CN",scope="subtree",Lower Case(Substring(length="1",Operation Attribute("Given Name"))+Operation Attribute("Surname")),Lower Case(Substring(length="2",Operation Attribute("Given Name"))+Operation Attribute("Surname")))))

## How the Logic in the Rule Works

The rule places User objects in proper department containers depending upon what value is stored in the OU attribute. If a User object needs to be placed and has the OU attribute available, then the User object is placed in the dest-base\value of OU attribute\CN attribute.

The dest-base is a local variable. The DN must be the relative root path of the department containers. It can be an organization or an organizational unit. The value stored in the OU attribute must be the name of a child container of the dest-base local variable.

The value of the OU attribute must be the name of the child container. If the OU attribute is not present, this rule is not executed.

The CN attribute of the User object is the first two letters of the Given Name attribute plus the Surname attribute as lowercase. The rule uses slash format.

## Placement - Subscriber By Dept - LDAP Format

Places objects from one container in the Identity Vault into multiple containers in the data store base on the OU attribute. Implement the rule on the Placement policy in the driver. You can implement the rule only on the Subscriber channel.

There are two steps involved in using the predefined rules: creating a policy in the Placement policy set, and importing the predefined rule. If you already have a Placement policy that you want to add this rule to, skip to [“Importing the Predefined Rule” on page 215](#).

### Creating a Policy

- 1 Open the *Identity Manager Driver Overview* for the driver you want to manage.
- 2 Click the Placement Policy object on the Subscriber channel.
- 3 Click *Insert*.
- 4 Name the policy, make sure to implement the policy with the Policy Builder, then click *OK*.

The Rule Builder is launched.

### Importing the Predefined Rule

- 1 In the Rule Builder, click *Insert*.
- 2 Select *Placement - Subscriber By Dept - LDAP format*.
- 3 Click *Placement - Subscriber By Dept - LDAP format* in the Rule Builder, to edit the rule.
- 4 Delete *[Enter DN of destination Organization]* from the *Enter string field*.

- 5 Click the *Edit Arguments icon* to launch the Argument Builder.
- 6 Select *Text* in the Noun list, then click *Add*.
- 7 In the Editor, add the parent container in the data store. The parent container must be specified in LDAP format. Make sure all of the department containers are child containers of this DN, then click *OK*.
- 8 Click *OK*.

[Placement - Subscriber By Dept - LDAP format](#)

**Conditions**

- if class name equal "User"
- And** if attribute 'OU' available

**Actions**

```

set local variable("dest-base","[Enter DN of destination Organization]")
set operation destination DN(dn("uid="+Escape Destination DN(Unique Name
("uid",scope="subtree",Lower Case(Substring(length="1",Operation Attribute("Given Name"))
+Operation Attribute("Surname")),Lower Case(Substring(length="2",Operation Attribute("Given
Name"))+Operation Attribute("Surname")))+",ou="+Attribute("OU")+","+Local Variable("dest-base"))

```

### How the Logic in the Rule Works

The rule places User objects in proper department containers depending upon what value is stored in the OU attribute. If a User object needs to be placed and has the OU attribute available, then the User object is placed in the uid=unique name,ou=value of OU attribute,dest-base.

The dest-base is a local variable. The DN must be the relative root path of the department containers. It can be an organization or an organizational unit. The value stored in the OU attribute must be the name of a child container of the dest-base local variable.

The value of the OU attribute must be the name of the child container. If the OU attribute is not present, then this rule is not executed.

The uid attribute of the User object is the first two letters of the Given Name attribute plus the Surname attribute as lowercase. The rule uses LDAP format.

## 3.3 Regular Expressions

A regular expression is a formula for matching text strings that follow some pattern. Regular expressions are made up of normal characters and metacharacters. Normal characters include uppercase and lowercase letters and digits. Metacharacters have special meanings. The following table contains some of the most common metacharacters and their meanings.

Metacharacter	Description
.	Matches any single character.
\$	Matches the end of the line.
^	Matches the beginning of a line.
*	Matches zero or more occurrences of the character immediately preceding.



Metacharacter	Description
\	Literal escape character. It allows you to search for any of the metacharacters. For example \\$ finds \$1000 instead of matching at the end of the line.
[ ]	Matches any one of the characters between the brackets.
[0-9]	Matches a range of characters with the hyphen. The example matches any digit.
[A-Za-z]	Matches multiple ranges as well. The example matches all uppercase and lowercase letters.

The Argument Builder is designed to use regular expressions as defined in Java\*. The [Java Web site \(http://java.sun.com/j2se/1.4.2/docs/api/java/util/regex/Pattern.html\)](http://java.sun.com/j2se/1.4.2/docs/api/java/util/regex/Pattern.html) contains further information.

## 3.4 XPath 1.0 Expressions

Arguments to some conditions, actions, and tokens use XPath 1.0 expressions. XPath is a language created to provide a common syntax and semantics for functionality shared between XSLT and XPointer. It is used primarily for addressing parts of an XML document, but also provides basic facilities for manipulation of strings, numbers and booleans.

The XPath specification requires that the embedding application provide a context with several application defined pieces of information. In DirXML Script (see [Section 1.1.2, “DirXML Script,” on page 15](#)), XPath is evaluated with the following context:

- The context node is the current operation.
- The context position and size are 1.
- Available variables
  - Those available as parameters to style sheets within Identity Manager (currently fromNDS, srcQueryProcessor, destQueryProcessor, srcCommandProcessor, destCommandProcessor, and dnConverter).
  - Global configuration variables.
  - Local policy variables.
  - If there is a name conflict between the different variable sources then the order of precedence is local variable, style sheet parameters, global variables.
- Namespaces that are declared on the policy element.
- Available functions
  - All built-in XPath 1.0 functions
  - Java extension functions as provided by NXSL
    - Namespaces declarations to associate a prefix with a Java class must be declared on the policy element.

The [W3 Web site \(http://www.w3.org/TR/1999/REC-xpath-19991116\)](http://www.w3.org/TR/1999/REC-xpath-19991116) contains further information.

## 3.5 Conditions

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

- [Section 3.5.1, “If Association,” on page 218](#)
- [Section 3.5.2, “If Attribute,” on page 219](#)
- [Section 3.5.3, “If Class Name,” on page 220](#)
- [Section 3.5.4, “If Destination Attribute,” on page 221](#)
- [Section 3.5.5, “If Destination DN,” on page 222](#)
- [Section 3.5.6, “If Entitlement,” on page 223](#)
- [Section 3.5.7, “If Global Configuration Value,” on page 225](#)
- [Section 3.5.8, “If Local Variable,” on page 226](#)
- [Section 3.5.9, “If Named Password,” on page 228](#)
- [Section 3.5.10, “If Operation,” on page 228](#)
- [Section 3.5.11, “If Operation Attribute,” on page 229](#)
- [Section 3.5.12, “If Operation Property,” on page 231](#)
- [Section 3.5.13, “If Password,” on page 232](#)
- [Section 3.5.14, “If Source Attribute,” on page 232](#)
- [Section 3.5.15, “If Source DN,” on page 234](#)
- [Section 3.5.16, “If XPath Expression,” on page 235](#)

### 3.5.1 If Association

Performs a test on the association value of current operation or the current object.

#### Fields

#### Operator Condition is Met When...

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 the if association.
not-associated	There is not an established association for the current object.
not available	The association is not available for the current object.
not-equal	The association value specified by the current operation is not equal to the content of the if association.

## Example

This example tests to see if the association is available. When this condition is met, the actions that are defined are executed.

The screenshot shows two rows of configuration for 'If' conditions. Each row starts with a dropdown menu set to 'association'. The first row has a 'Select operator:' dropdown set to 'available'. The second row has a 'Select operator:' dropdown set to 'equal' and a 'Value:' text box containing the GUID '{07414faa-1b38-40ec-8b7c-c20aa21ddafb}'. Both rows include standard UI icons for help, copy, paste, and save.

## 3.5.2 If Attribute

Performs a test on attribute values of the current object in either the current operation or the source data store. It can be logically thought of as If Operation Attribute or If Source Attribute, because the test is satisfied if the condition is met in the source data store or in the operation.

### Fields

#### Name

Specify the name of the attribute to test.

#### Operator

Select the condition test type.

#### Compare Mode

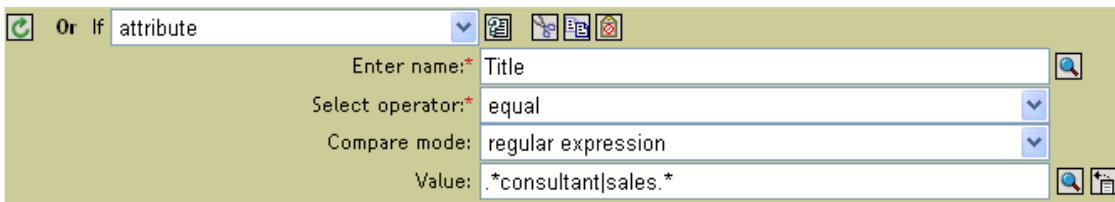
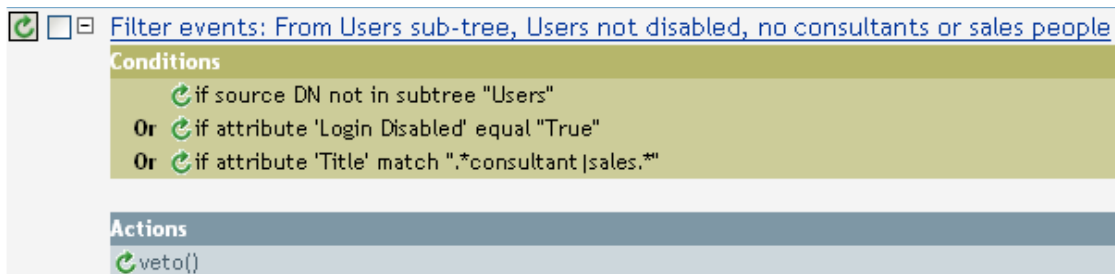
Select the comparison mode. See [“Comparison Modes” on page 294](#).

#### Operator Condition is Met When...

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, which equals the specified value when compared using the specified comparison mode.
not available	Available would return False.
not-equal	Equal would return False.

## Example

The example uses the condition If Attribute when filtering for User objects that are disabled or have a certain title. The policy is Policy to Filter Events, and it is available for download from Novell’s support Web site. For more information, see [“Downloadable Identity Manager Policies” on page 36](#).



The condition is looking for any User object that has an attribute of Title with a value of consultant or sales.

### 3.5.3 If Class Name

Performs a test on the object class name in the current operation.

#### Fields

##### Operator

Select the condition test type.

##### Compare Mode

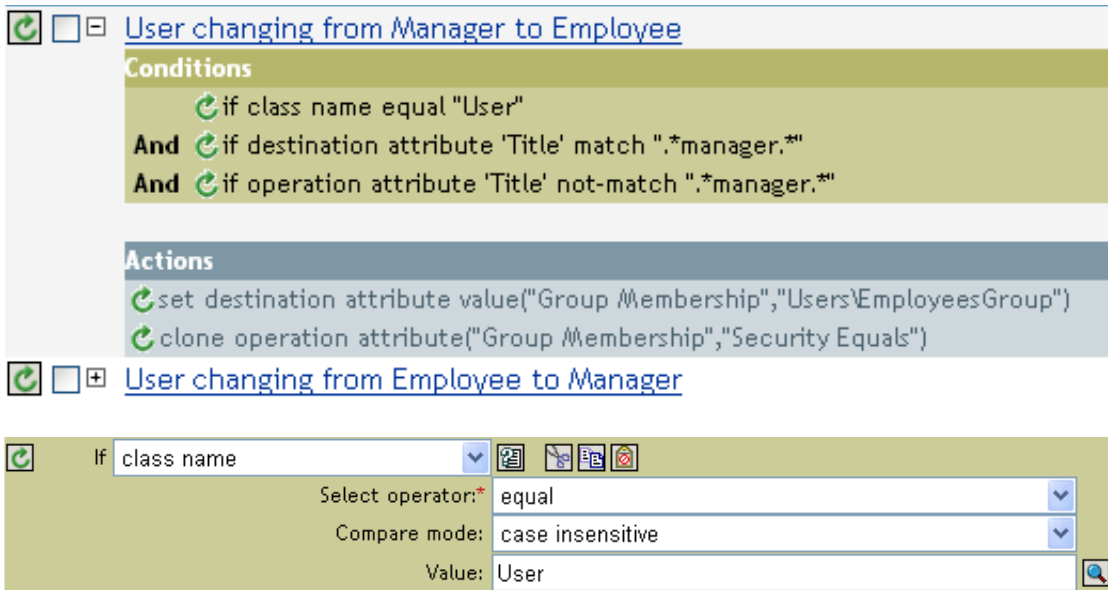
Select the comparison mode. See [Section 3.9.1, “Comparison Modes,” on page 294](#).

##### Operator Condition is Met When...

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.

#### Example

The example uses the condition If Class Name to govern group membership for a User object based on their title. The policy is Govern Groups for User Based on Title Attribute and it is available for download from Novell’s support Web site. For more information, see [“Downloadable Identity Manager Policies” on page 36](#).



Checks to see if the class name of the current object is User.

### 3.5.4 If Destination Attribute

Performs a test on attribute values of the current object in the destination data store.

#### Fields

##### Name

Specify the name of the attribute to test.

##### Operator

Select the condition test type.

##### Compare Mode

Select the comparison mode. See [“Comparison Modes” on page 294](#).

##### Operator Condition is Met When...

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.

## Example

The example uses the condition If Attribute to govern group membership for a User object based on the title. The policy is Govern Groups for User Based on Title Attribute and it is available for download from Novell's support Web site. For more information, see ["Downloadable Identity Manager Policies"](#) on page 36.

The screenshot shows the Identity Manager Policy Builder interface. It displays two policies:

- User changing from Manager to Employee**:
  - Conditions**:
    - if class name equal "User"
    - And if destination attribute 'Title' match ".\*manager.\*"
    - And if operation attribute 'Title' not-match ".\*manager.\*"
  - Actions**:
    - set destination attribute value("Group Membership","Users\EmployeesGroup")
    - clone operation attribute("Group Membership","Security Equals")
- User changing from Employee to Manager**

Below the policies, a configuration dialog for the 'And If' condition is shown. It is configured as follows:

- Condition type: destination attribute
- Enter attribute name: Title
- Select operator: equal
- Compare mode: regular expression
- Value: .\*manager.\*

The policy checks to see if the value of the title attribute contains manager.

## 3.5.5 If Destination DN

Performs a test on the destination DN in the current operation. The test performed depends on the specified operator.

### Fields

#### Operator

Select the condition test type.

#### Operator Condition is Met When...

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.

Operator	Condition is met when...
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.

### Example

The screenshot displays four rows of configuration for an 'If' condition where the field is 'destination DN'. Each row shows a different operator and its corresponding value:

- Row 1: Operator: available
- Row 2: Operator: equal, Value: Novell\Users\Fred
- Row 3: Operator: in container, Value: Novell\Users
- Row 4: Operator: in subtree, Value: Novell

## 3.5.6 If Entitlement

Performs a test on entitlements of the current object, in either the current operation or the Identity Vault.

### 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 294](#).

#### Operator Condition is Met When...

<b>Operator</b>	<b>Condition is met when...</b>
available	The named entitlement is available in either the current operation or the Identity Vault.
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, which has a value 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 named entitlement. It has a value that 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.



## Example

The image shows five examples of the 'If entitlement' configuration interface. Each example consists of a dropdown menu set to 'entitlement', followed by an 'Enter name' text field containing 'notes-group'. Below this is a 'Select operator' dropdown menu. The operators shown are 'available', 'changing', 'changing from', 'changing to', and 'equal'. The third and fourth examples also include a 'Compare mode' dropdown menu set to 'case insensitive' and a 'Value' text field containing 'Sales'. Each configuration panel includes small icons for help, copy, paste, and save.

### 3.5.7 If Global Configuration Value

Performs a test on a global configuration variable.

#### 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 294](#).

##### Operator Condition is Met When...

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.

### Example

The screenshot shows two instances of the 'If' condition configuration in a software interface. Each instance starts with a dropdown menu set to 'global configuration value'. The first instance has 'Enter name:' set to 'myGlobalVariable' and 'Select operator:' set to 'available'. The second instance has 'Enter name:' set to 'myGlobalVariable', 'Select operator:' set to 'equal', 'Compare mode:' set to 'case insensitive', and 'Value:' set to 'enabled'.

## 3.5.8 If Local Variable

Performs a test on a local variable.

### 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 294](#).

#### Operator Condition is Met When...

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.

Operator	Condition is met when...
not-equal	Equal would return False.

## Example

The example adds a User object to the appropriate group, Employee or Manager, based on Title. It also creates the group, if needed, and sets up security equal to that group. The policy is Govern Groups for User Based on Title Attribute and it is available for download from Novell's support Web site. For more information, see ["Downloadable Identity Manager Policies"](#) on page 36.

[Set local variables to test existence of groups and for placement](#)

[Create ManagersGroup, if needed](#)

**Conditions**

- if local variable 'manager-group-info' available
- And**  if local variable 'manager-group-info' not equal "group"

**Actions**

- add destination object(class name="Group",when="before",dn(Local Variable("manager-group-dn")))

[Create EmployeesGroup, if needed](#)

[If Title indicates Manager, add to ManagerGroup and set rights](#)

[If Title does not indicate Manager, add to EmployeeGroup and set rights](#)

The policy contains five rules that are dependent on each other.

[Set local variables to test existence of groups and for placement](#)

**Conditions**

- if class name equal "User"
- And**
- if operation equal "add"
- Or**  if operation equal "modify"

**Actions**

- set local variable("manager-group-dn","Users\ManagersGroup")
- set local variable("manager-group-info",Destination Attribute("Object Class",dn(Local Variable ("manager-group-dn"))))
- set local variable("employee-group-dn","Users\EmployeesGroup")
- set local variable("employee-group-info",Destination Attribute("Object Class",dn(Local Variable ("employee-group-dn"))))

For the If Locate Variable condition to work, the first rule sets four different local variables to test for groups and where to place the groups.

**And** If local variable

Enter name:\* manager-group-info

Select operator:\* not equal

Compare mode: case insensitive

Value: group

The condition the rule is looking for is to see if the local variable of manager-group-info is available and if manger-group-info is not equal to group. If these conditions are met, then the destination object of group is added.

### 3.5.9 If Named Password

Performs a test on a password in the current operation with the specified name.

#### Fields

##### Name

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

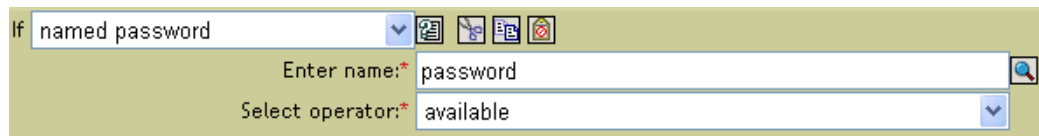
##### Operator

Select the condition test type.

##### Operator Condition is Met When...

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

#### Example



The screenshot shows a configuration window for the 'If named password' condition. The 'If' dropdown menu is set to 'named password'. Below it, there are two input fields: 'Enter name:\*' with the value 'password' and 'Select operator:\*' with the value 'available'. The interface includes standard UI elements like a search icon and a dropdown arrow.

### 3.5.10 If Operation

Performs a test on the name of the current operation.

#### Fields

##### Operator

Select the condition test type.

##### Operator Condition is Met When...

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.

## Example

The example adds a User object to the appropriate group, Employee or Manager, based on Title. It also creates the group, if needed, and sets up security equal to that group. The policy name is Govern Groups for User Based on Title Attribute and it is available for download from Novell's support Web site. For more information, see [“Downloadable Identity Manager Policies” on page 36](#).

Set local variables to test existence of groups and for placement

**Conditions**

- if class name equal "User"
- And**
- if operation equal "add"
- Or** if operation equal "modify"

**Actions**

- set local variable("manager-group-dn","Users\ManagersGroup")
- set local variable("manager-group-info",Destination Attribute("Object Class",dn(Local Variable ("manager-group-dn"))))
- set local variable("employee-group-dn","Users\EmployeesGroup")
- set local variable("employee-group-info",Destination Attribute("Object Class",dn(Local Variable ("employee-group-dn"))))

If operation

Select operator:\* equal

Value: add

The condition is checking to see if an add or modify operation has occurred. Once one of these occurs, then it sets the local variables.

### 3.5.11 If Operation Attribute

Performs a test on attribute values in the current operation. The test performed depends on the specified operator.

#### Fields

##### Name

Specify the name of the attribute to test.

##### Operator

Select the condition test type.

##### Compare Mode















Select the comparison mode. See [“Comparison Modes” on page 294](#).

##### Operator Condition is Met When...

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. It 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. It 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. It 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.

## Example

The example adds a User object to the appropriate group, Employee or Manager, based on Title. It also creates the group, if needed, and sets up security equal to that group. The policy name is Govern Groups for User Based on Title Attribute and it is available for download from Novell's support Web site. For more information, see [“Downloadable Identity Manager Policies” on page 36](#).

-    [Set local variables to test existence of groups and for placement](#)
-    [Create ManagersGroup, if needed](#)
-    [Create EmployeesGroup, if needed](#)
-    [If Title indicates Manager, add to ManagerGroup and set rights](#)
  - Conditions**
  -  if class name equal "User"
  - And**  if operation attribute 'Title' match ".\*manager.\*"
  - Actions**
  -  set destination attribute value("Group Membership",Local Variable("manager-group-dn"))
  -  clone operation attribute("Group Membership","Security Equals")
-    [If Title does not indicate Manager, add to EmployeeGroup and set rights](#)

And If operation attribute

Enter name:\* Title

Select operator:\* equal

Compare mode: regular expression

Value: \*.manager.\*

The condition is checking to see if the attribute of Title is equal to `.*manager*`, which is a regular expression. It means it is looking for a title that has zero or more characters before manager and a single character after manager. It would find a match if the User object's title was sales managers.

### 3.5.12 If Operation Property

Performs a test on an operation property on the current operation.

#### Fields

##### Name

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

##### Operator

Select the condition test type.

##### Compare Mode

Select the comparison mode. See [“Comparison Modes” on page 294](#).

#### Operator Condition is Met When...

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.

## Example

If operation property    
Select operator: available

If operation property    
Select operator: equal   
Compare mode: case insensitive   
Value: true

### 3.5.13 If Password

Performs a test on a password in the current operation.

#### Fields

#### Operator

Select the condition test type.

#### Operator Condition is Met When...

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

## Example

If password    
Select operator: available

### 3.5.14 If Source Attribute

Performs a test on attribute values of the current object in the source data store.

#### 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 [Section 3.9.1, “Comparison Modes,” on page 294](#).



## Operator Condition is Met When...

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. 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

### 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 294](#).

## Operator Condition is Met When...

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. It equals the specified value when compared using the specified comparison mode.
not available	Available would return False.
not-equal	Equal would return False.

## Example

The screenshot displays three 'If' condition configurations in a Policy Builder interface. Each configuration starts with a dropdown menu set to 'source attribute'. The first configuration has 'OU' as the attribute name and 'available' as the operator. The second configuration has 'OU' as the attribute name, 'equal' as the operator, 'case insensitive' as the compare mode, and 'Sales' as the value. The third configuration has 'Language' as the attribute name, 'equal' as the operator, 'structured' as the compare mode, and a list of structured components: 'string(EN)' and 'string(JP)'. The 'string(EN)' component is currently selected in the list.

### 3.5.15 If Source DN

Performs a test on the source DN in the current operation.

#### Fields

##### Operator

Select the condition test type.

##### Operator Condition is Met When...

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 in-container There is a source DN available, and it represents an object in the container identified by the specified value.
in-subtree	There is a source DN available, and it represents an object in the subtree identified by the specified value.
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.

### Operator Condition is Met When...

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 in-container There is a source DN available, and it represents an object in the container identified by the specified value.
in-subtree	There is a source DN available, and it represents an object in the subtree identified by the specified value.
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.

### Example

The example uses the condition If Source DN to check if the User object is in the source DN. The rule is from the predefined rules that come with Identity Manager 3.0. For more information, see [“Event Transformation - Scope Filtering - Exclude Subtrees” on page 204](#).

The screenshot displays the configuration for a rule in the Identity Manager Policy Builder. The rule is titled "Event Transformation - Scope Filtering - Exclude subtree(s)". It consists of a single condition: "if source DN in subtree "[Enter a subtree to exclude]"". The condition is configured with the operator "in container" and the value "Users". The action associated with this condition is "veto()".

The condition is checking to see if the source DN is in the Users container. If the object is coming from that container, it is vetoed.

## 3.5.16 If XPath Expression

Performs a test on the results of evaluating an XPath 1.0 expression.

### Fields

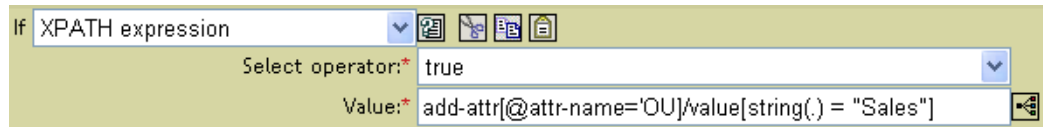
### Operator

Select the condition test type.

### Operator Condition is Met When...

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

### Example



## 3.6 Actions

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

- [Section 3.6.1, “Add Association,” on page 237](#)
- [Section 3.6.2, “Add Destination Attribute Value,” on page 238](#)
- [Section 3.6.3, “Add Destination Object,” on page 239](#)
- [Section 3.6.4, “Add Source Attribute Value,” on page 240](#)
- [Section 3.6.5, “Add Source Object,” on page 241](#)
- [Section 3.6.6, “Append XML Element,” on page 242](#)
- [Section 3.6.7, “Append XML Text,” on page 243](#)
- [Section 3.6.8, “Break,” on page 244](#)
- [Section 3.6.9, “Clear Destination Attribute Value,” on page 244](#)
- [Section 3.6.10, “Clear Operation Property,” on page 245](#)
- [Section 3.6.11, “Clear Source Attribute Value,” on page 245](#)
- [Section 3.6.12, “Clone By XPath Expression,” on page 246](#)
- [Section 3.6.13, “Clone Operation Attribute,” on page 246](#)
- [Section 3.6.14, “Delete Destination Object,” on page 247](#)
- [Section 3.6.15, “Delete Source Object,” on page 247](#)
- [Section 3.6.16, “Find Matching Object,” on page 248](#)
- [Section 3.6.17, “For Each,” on page 249](#)
- [Section 3.6.18, “Generate Event,” on page 250](#)
- [Section 3.6.19, “Implement Entitlement,” on page 252](#)
- [Section 3.6.20, “Move Destination Object,” on page 253](#)
- [Section 3.6.21, “Move Source Object,” on page 254](#)
- [Section 3.6.22, “Reformat Operation Attribute,” on page 254](#)
- [Section 3.6.23, “Remove Association,” on page 255](#)

- Section 3.6.24, “Remove Destination Attribute Value,” on page 256
- Section 3.6.25, “Remove Source Attribute Value,” on page 257
- Section 3.6.26, “Rename Destination Object,” on page 258
- Section 3.6.27, “Rename Operation Attribute,” on page 258
- Section 3.6.28, “Rename Source Object,” on page 258
- Section 3.6.29, “Send Email,” on page 259
- Section 3.6.30, “Send Email from Template,” on page 260
- Section 3.6.31, “Set Default Attribute Value,” on page 261
- Section 3.6.32, “Set Destination Attribute Value,” on page 262
- Section 3.6.33, “Set Destination Password,” on page 263
- Section 3.6.34, “Set Local Variable,” on page 264
- Section 3.6.35, “Set Operation Association,” on page 265
- Section 3.6.36, “Set Operation Class Name,” on page 265
- Section 3.6.37, “Set Operation Destination DN,” on page 266
- Section 3.6.38, “Set Operation Property,” on page 266
- Section 3.6.39, “Set Operation Source DN,” on page 267
- Section 3.6.40, “Set Operation Template DN,” on page 267
- Section 3.6.41, “Set Source Attribute Value,” on page 268
- Section 3.6.42, “Set Source Password,” on page 269
- Section 3.6.43, “Set XML Attribute,” on page 269
- Section 3.6.44, “Status,” on page 270
- Section 3.6.45, “Strip Operation Attribute,” on page 270
- Section 3.6.46, “Strip XPath,” on page 271
- Section 3.6.47, “Trace Message,” on page 271
- Section 3.6.48, “Veto,” on page 272
- Section 3.6.49, “Veto if Operation Attribute Not Available,” on page 273

### 3.6.1 Add Association

Sends an add association command to the Identity Vault, with the specified association.

#### Fields

##### Mode

Select whenter this action should be added to the current operation, or written directly to the Identity Vault.

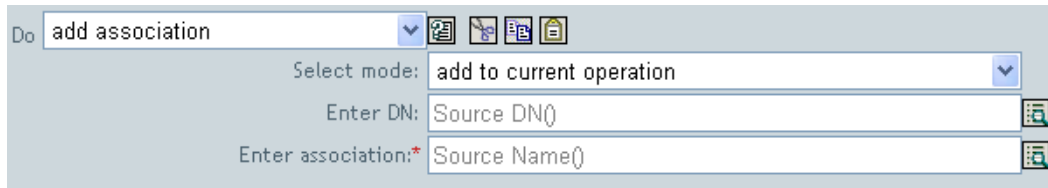
##### DN

Specify the DN of the target object or leave blank to use the current object.

##### Association

Specify the value of the association to be added.

## Example



## 3.6.2 Add Destination Attribute Value

Adds a value to an attribute on an object in the destination data store.

### Fields

#### Attribute Name

Specify the name of the attribute.

#### Class Name

(Optional) Specify the class name of the target object. Leave blank to use the class name from the current object.

#### 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. This object can be the current object, or be specified by a DN or an association.

#### Value Type

Select the syntax of the attribute value to be added.

#### Value

Specify the attribute value to be added.

## Example

The example adds the destination attribute value to the OU attribute. It creates the value from the local variables that are created. The rule is from the predefined rules that come with Identity Manager 3.0. For more information, see [“Command Transformation - Create Departmental Container - Part 1 and Part 2”](#) on page 197.

## Command Transformation - Create Departmental Container - Part 1

### Conditions

if operation equal "add"

### Actions

```
set local variable("target-container",Destination DN(length="-2"))
set local variable("does-target-exist",Destination Attribute("objectclass",class
name="OrganizationalUnit",dn(Local Variable("target-container"))))
```

## Command Transformation - Create Departmental Container - Part 2

### Conditions

if local variable 'does-target-exist' available

And if local variable 'does-target-exist' equal ""

### Actions

```
add destination object(class name="organizationalUnit",direct="true",dn(Local Variable("target-
container")))
add destination attribute value("ou",direct="true",dn(Local Variable("target-container")),Parse
DN("dest-dn","dot",length="1",start="-1",Local Variable("target-container")))
```

Do add destination attribute value

Enter attribute name:\* ou

Enter class name:

Select mode: write directly to destination datastore

Select object: DN

Enter DN:\* Local Variable("target-container")

Enter value type: string

Enter string:\* Parse DN("dest-dn","dot",length="1",start="-1",Local Vari...

### 3.6.3 Add Destination Object

Creates a new object of the specified type in the destination data store.

#### Fields

##### Class Name

Specify the class name of the object to be created.

##### 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 object to be created.

## Remarks

Any attribute values to be added as part of the object creation must be done in subsequent “[Add Destination Attribute Value](#)” on page 238 actions using the same DN.

## Example

The example creates the department container that is needed. The rule is from the predefined rules that come with Identity Manager 3.0. For more information, see “[Command Transformation - Create Departmental Container - Part 1 and Part 2](#)” on page 197 from the predefined rules.

### Command Transformation - Create Departmental Container - Part 1

#### Conditions

if operation equal "add"

#### Actions

```
set local variable("target-container",Destination DN(length="-2"))
set local variable("does-target-exist",Destination Attribute("objectclass",class
name="OrganizationalUnit",dn(Local Variable("target-container"))))
```

### Command Transformation - Create Departmental Container - Part 2

#### Conditions

if local variable 'does-target-exist' available  
And if local variable 'does-target-exist' equal ""

#### Actions

```
add destination object(class name="organizationalUnit",direct="true",dn(Local Variable("target-
container")))
add destination attribute value("ou",direct="true",dn(Local Variable("target-container")),Parse
DN("dest-dn","dot",length="1",start="-1",Local Variable("target-container")))
```

Do add destination object

Enter class name:\* organizationalUnit

Select mode: write directly to destination datastore

Enter DN:\* Local Variable("target-container")

The OU object is created. The value for the OU attribute is created from the destination attribute value action that occurs after this action.

## 3.6.4 Add Source Attribute Value

Adds the specified value the specified attribute on an object in the source data store. The target object is the current object, a DN, or an association.

### Fields

#### Attribute Name

Specify the name of the attribute.



### Class Name

(Optional) Specify the class name of the target object. Leave blank to use the class name from the current object.

### Object

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

### Value Type

Select the syntax of the attribute value to be added.

### Value

Specify the attribute value to be added.

### Example

Do add source attribute value

Enter attribute name:\* Member

Enter class name:

Select object: DN

Enter DN:\* "Users/ManagerGroup"

Enter value type: string

Enter string:\* Destination DN()

## 3.6.5 Add Source Object

Creates 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 \(page 240\)](#) actions using the same DN.

### Fields

#### Class Name

Specify the class name of the object to be added.

#### DN

Specify the DN of the object to be added.

## Example

The screenshot shows a configuration interface with two main sections, each starting with a 'Do' label and a dropdown menu.

**Section 1: add source object**

- Enter class name: \* User
- Enter DN: \* "Users/Fred Flintstone"

**Section 2: add source attribute value**

- Enter attribute name: \* Surname
- Enter class name: (empty)
- Select object: DN
- Enter DN: \* "Users/Fred Flintstone"
- Enter value type: string
- Enter string: \* "Flintstone"

## 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.

## 3.6.6 Append XML Element

Appends an element to a set of elements selected by the XPath expression.

## Fields

### Name

Specify the tag name of the XML element. This name can contain a namespace prefix if the prefix has been previously defined in this policy.

### XPATH Expression

Specify an XPath 1.0 expression that returns a node set containing the elements to which the new elements should be appended.

## Example

The screenshot displays a sequence of four actions in a policy builder interface:

- Action 1:** Do append XML element. Enter name: \* jdbc:statement. Enter XPATH expression: \* ..
- Action 2:** Do append XML element. Enter name: \* jdbc:sql. Enter XPATH expression: \* ../jdbc:statement[last()]
- Action 3:** Do append XML text. Enter XPATH expression: \* ../jdbc:statement[last()]/jdbc:sql. Enter string: \* " UPDATE dirxml.emp SET fname = "+Operation Attribute
- Action 4:** Do append XML text. Enter XPATH expression: \* ../jdbc:statement[last()]/jdbc:sql. Enter string: \* " UPDATE dirxml.emp SET fname = "+Operation Attribute

### 3.6.7 Append XML Text

Appends text to a set of elements selected by the XPath expression.

#### Fields

##### XPATH Expression

XPath 1.0 expression that returns a node set containing the elements to which the new elements should be appended.

##### String

Specify the text to be appended.

## Example

The screenshot displays four sequential 'Do' operations in a policy builder interface:

- Operation 1:** 'append XML element'. Enter name: \* jdbc:statement. Enter XPATH expression: \* ..
- Operation 2:** 'append XML element'. Enter name: \* jdbc:sql. Enter XPATH expression: \* ../jdbc:statement[last()]
- Operation 3:** 'append XML text'. Enter XPATH expression: \* ../jdbc:statement[last()]/jdbc:sql. Enter string: \* " UPDATE dirxml.emp SET fname = "+Operation Attribute
- Operation 4:** 'append XML text'. Enter XPATH expression: \* ../jdbc:statement[last()]/jdbc:sql. Enter string: \* " UPDATE dirxml.emp SET fname = "+Operation Attribute

## 3.6.8 Break

Ends processing of the current operation by the current policy.

### Example

The screenshot shows a single 'Do' operation in a policy builder interface:

- Operation:** 'break'.

## 3.6.9 Clear Destination Attribute Value

Removes the all values for the named attribute from an object in the destination data store.

### Fields

#### Attribute Name

Specify the name of the attribute.

#### Class Name

(Optional) Specify the class name of the target object. Leave blank to use the class name from the current object.

#### 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. This object can be the current object, or be specified by a DN or an association.

## Example

Do clear destination attribute value

Enter attribute name:\* Member

Enter class name:

Select mode: add to current operation

Select object: DN

Enter DN:\* "Users/ManagerGroup"

### 3.6.10 Clear Operation Property

Clears any operation property current operation.

#### Fields

##### Property Name

Specify the name of the operation property to clear.

#### Example

Do clear operation property

Enter property name:\* myStoredProperty

### 3.6.11 Clear Source Attribute Value

Removes the all values of an attribute from an object in the source data store.

#### Fields

##### Attribute Name

Specify the name of the attribute.

##### Class Name

(Optional) Specify the class name of the target object. Leave blank to use the class name from the current object.

##### Object

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

## Example

The screenshot shows a configuration window for the operation 'clear source attribute value'. It includes the following fields:

- Enter attribute name: Member
- Enter class name: (empty)
- Select object: DN
- Enter DN: "Users/ManagerGroup"

## 3.6.12 Clone By XPath Expression

Appends deep copies of a set of XML nodes selected by an XPath expression to a set of elements selected by another XPath expression.

### Fields

#### Source XPATH Expression

Specify the XPath 1.0 expression that returns a node set containing the nodes to be copied.

#### Destination XPATH Expression

Specify the XPath 1.0 expression that returns a node set containing the elements to which the copied nodes are to be appended.

## Example

The screenshot shows a configuration window for the operation 'clone by XPATH expressions'. It includes the following fields:

- Enter source XPATH expression: @\*
- Enter destination XPATH expression: ../modify[last()]

## 3.6.13 Clone Operation Attribute

Copies all occurrences of an attribute within the current operation to a different attribute within the current operation.

### Fields

#### Source Name

Specify the name of the attribute to be copied from.

#### Destination Name

Specify the name of the attribute to be copied to.

## Example

The example adds a User object to the appropriate group, Employee or Manager, based on Title. It also creates the group, if needed, and setup security equal to that group. The policy is Govern Groups for User Based on Title Attribute, and it is available for download from Novell's support Web site. For more information, see ["Downloadable Identity Manager Policies"](#) on page 36.

[Set local variables to test existence of groups and for placement](#)  
   [Create ManagersGroup, if needed](#)  
   [Create EmployeesGroup, if needed](#)  
   [If Title indicates Manager, add to ManagerGroup and set rights](#)

**Conditions**

- if class name equal "User"
- And**  if operation attribute 'Title' match ".\*manager.\*"

**Actions**

- set destination attribute value("Group Membership",Local Variable("manager-group-dn"))
- clone operation attribute("Group Membership","Security Equals")

[If Title does not indicate Manager, add to EmployeeGroup and set rights](#)

Do clone operation attribute
   
 Enter source name:\* Group Membership
   
 Enter destination name: Security Equals

The Clone Operation Attribute is taking the information from the Group Membership attribute and adding that to the Security Equals attribute so the values are the same.

### 3.6.14 Delete Destination Object

Deletes an object in the destination data store.

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

#### Example

Do delete destination object
   
 Select mode: add to current operation
   
 Select object: DN
   
 Enter DN:\* "Users/Fred Flintstone"

### 3.6.15 Delete Source Object

Deletes the object in the source data store.

## Fields

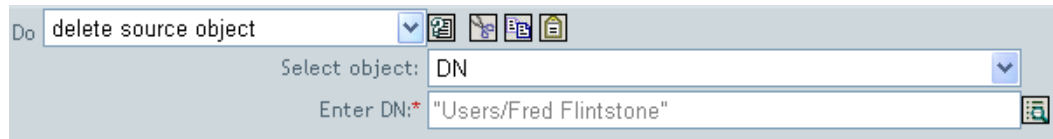
### Mode

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

### Example



## 3.6.16 Find Matching Object

Finds a match for the current object in the destination data store.

### Fields

#### Scope

Select the scope of the search. The scope might be an entry, a subordinates, or a subtree.

#### DN

Specify the DN that is the base of the search.

#### Match Attributes

Specify the attribute values to search for.

### Remarks

Find Matching Object is only valid when the current operation is an add.

The DN argument is required when scope is “entry”, and is optional otherwise. At least one match attribute is required when scope is “subtree” or “subordinates”.

The results are undefined if scope is entry and there are match attributes specified. If the destination data store is the connected application, then an association is added to the current operation for each successful match that is returned. No query is 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 the Identity Vault, 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 &#xFFFC;. If multiple results are returned, then the destination DN of the current operation is set to the single character &#xFFFD;.

## Example

The example matches on Users objects with the attributes CN and L. The location where the rule is searching starts at the Users container and adds the information stored in the OU attribute to the DN. The rule is from the predefined rules that come with Identity Manager 3.0. For more information, see [“Matching - By Attribute Value” on page 79](#).

The screenshot shows the Policy Builder interface for a rule named "Matching - by attribute value". It is divided into two main sections: "Conditions" and "Actions".

- Conditions:** A single condition is listed: "if class name equal 'User'".
- Actions:** A single action is listed: "find matching object(dn("[Enter base DN to start search]"),match("[Enter name of attribute to match on]"))".

Below the actions, there is a configuration panel for the "find matching object" action:

- Do:** find matching object
- Select scope:** subtree
- Enter DN:** "Users"+Attribute("OU")
- Enter match attributes:** CN,L

When you click on the Argument Builder icon, the Match Attribute Builder comes up. You specify the attribute you want to match on in the builder. This examples uses the CN and L attributes.

The screenshot shows the "Match Attributes" dialog box. It has a title bar "Match Attributes" and contains two rows of configuration:

Name	Value
<input type="checkbox"/> Name: * CN	Value from current object
<input type="checkbox"/> Name: * L	Value from current object

## 3.6.17 For Each

Repeats a set of actions for each node in a node set.

### Fields

#### Node Set

Specify the node set.

#### Action

Specify the actions to perform on each node in the node set.

### Remarks

The current node is a different value for each iteration of the actions, if a local variable is used.

If a node in the node set is an entitlement, then the for each implicitly performs an [“Implement Entitlement” on page 252](#) action.

## Example

Do for each

Enter node set:\* Added Entitlement("Group")

Enter action:\* do-add-dest-attr-value

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

**Actions**

Action List

Do add destination attribute value

Enter attribute name:\* Member

Enter class name:\* Group

Select mode: add to current operation

Select object: DN

Enter DN:\* Local Variable("current-node")

Enter value type: string

Enter tokens:\* Destination DN()

## 3.6.18 Generate Event

Sends a user-defined event to Novell Audit.

### 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 Metadirectory engine or driver to shut down.
log-alert	Events that require immediate attention.
log-critical	Events that can cause parts of the Metadirectory engine or driver to malfunction.
log-error	Events describing errors that can be handled by the Metadirectory engine or driver.
log-warning	Negative events not representing a problem.
log-notice	Events (positive or negative) an administrator can use to understand or improve use and operation.
log-info	Positive events of any importance.

Level	Description
log-debug	Events of relevance for support or engineers to debug the operation of the Metadirectory engine or driver.

## Strings

Specify 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 specifying a predefined format for the target. Predefined values for target-type are currently: <ul style="list-style-type: none"> <li>• 0 = None</li> <li>• 1 = Slash Notation</li> <li>• 2 = Dot Notation</li> <li>• 3 = LDAP Notation</li> </ul>
subTarget	The subcomponent of the target being acted upon.
text1	Text entered here is stored in the text1 event field.
text2	Text entered here is 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 is stored in the value3 event field.
data	Data entered here is stored in the blob event field.

## Remarks

The Novell Audit event structure contains a target, a subTarget, three strings (text1, text2, text3), two integers (value, value3), and a generic field (data). The text fields are limited to 256 bytes, and the data field can contain up to 3 KB of information, unless a larger data field is enabled in your environment.

## Example

The example has four rules that implements a placement policy for User objects based on the first character of the Surname attribute and generates both a trace message and a custom Novell Audit event. The Generate Event action is used to send Novell Audit an event. The policy name is Policy to Place by Surname and is available for download from Novell's support Web site. For more information [“Downloadable Identity Manager Policies” on page 36](#).

The screenshot displays a policy configuration window with the following elements:

- Setup Local Variables**: A checkbox and plus icon.
- Surname A-I: place in Users1**: A checkbox and plus icon.
  - Conditions**:
    - if class name equal "User"
    - And** if operation attribute 'Surname' match "[a-i].\*"
  - Actions**:
    - set operation destination DN(dn("Training\Users\Active\Users1"+"\"+Operation Attribute("CN")))
    - trace message(color="yellow",Local Variable("LVUsers1"))
    - generate event(id="1000",text1=Local Variable("LVUsers1"))
- Surname J-R: place in Users2**: A checkbox and plus icon.
- Surname S-Z: place in Users3**: A checkbox and plus icon.
- Do generate event**: A dropdown menu with a plus icon.
  - Enter ID: 1000
  - Select level: informational
  - Enter strings: text1

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

The screenshot shows the **Strings** dialog box with the following fields:

- Name:** text1
- String value:** Local Variable("LVUsers1")

Generate Event is creating an event with the ID 1000 and displaying the text that is generated by the local variable of LVUser1. The local variable LVUser1 is the string of User:Operation Attribute "cn" + " added to the "Training\Users\Active\Users1" container". The event will read User:jsmith added to the Training\Users\Active\Users1 container.

### 3.6.19 Implement Entitlement

Designates actions that implement an entitlement so that the status of those entitlements might be reported to the agent that granted or revoked the entitlement.

#### Fields

##### Node Set

Node set containing the entitlement being implemented by the specified actions.

##### Action

Actions that implement the specified entitlements.

## Example

The screenshot shows the 'Action List' window with the following configuration:

- Do: implement entitlement
- Enter node set: Removed Entitlement("Account")
- Enter action: do-add-dest-attr-value

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

The screenshot shows the 'Argument Actions Builder' window with the following configuration:

- Do: add destination attribute value
- Enter attribute name: Login Disabled
- Enter class name: User
- Select mode: add to current operation
- Select object: DN
- Enter DN: Local Variable("current-node")
- Enter value type: string
- Enter string: Destination DN()

## 3.6.20 Move Destination Object

Moves an object in the destination data store.

### Fields

#### Mode

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

#### Class Name

(Optional) Specify the class name of the object to be moved. Leave blank to use the class name from the current object.

#### Object to Move

Select the object to be moved. This object can be the current object, or can be 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.

## Example

The example contains a single rule which disables a user's account and moves them to a disabled container when the Description attribute indicates they are terminated. The policy is named Disable User Account and Move When Terminated, and it is available for download from Novell's support Web site. For more information, see ["Downloadable Identity Manager Policies" on page 36](#).

On Termination, disable user and move to Disabled container

**Conditions**

- if operation equal "modify"
- And** if class name equal "User"
- And** if operation attribute 'Description' match "^terminated.\*"

**Actions**

- set destination attribute value("Login Disabled",direct="true","True")
- move destination object(when="after",dn("Users\Disabled"))

---

Do move destination object

Select mode: add after current operation

Select object to move: Current object

Select container to move to: DN

Enter DN:\* "Users\Disabled"

The policy checks to see if it is a modify event on a User object and if the attribute Description contains the value of terminated. If that is the case, then it sets the attribute of Login Disabled to true and moves the object in to the User\Disabled container.

### 3.6.21 Move Source Object

Moves an object in the source data store.

#### Fields

##### Object to Move

Select the object to be moved. This object can be the current object, or can be 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.

#### Example

Do move source object

Select object to move: DN

Enter DN:\* "Users/Active/FredFlintstone"

Select container to move to: DN

Enter DN:\* "Users/InActive"

### 3.6.22 Reformat Operation Attribute

Reformats all values of an attribute within the current operation using a pattern.

## Fields

### Name

Specify the name of the attribute.

### Value Type

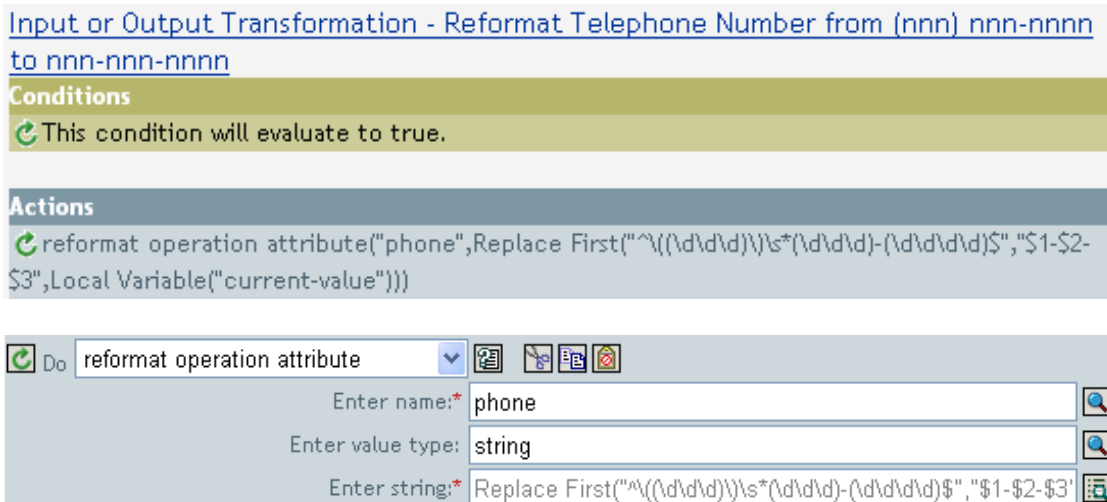
Specify the syntax of the new attribute value.

### Value

Specify a value to use as a pattern for the new format of the attribute values. If the original value is needed to constructed the new value, it must be obtained by referencing the local variable current-value.

## Example

The example reformats the telephone number. It changes it from (nnn)-nnn-nnnn to nnn-nnn-nnnn. The rule is from the predefined rules that come with Identity Manager 3.0. For more information, see [“Input or Output Transformation - Reformat Telephone Number from \(nnn\) nnn-nnnn to nnn-nnn-nnnn”](#) on page 205.



[Input or Output Transformation - Reformat Telephone Number from \(nnn\) nnn-nnnn to nnn-nnn-nnnn](#)

**Conditions**

🔄 This condition will evaluate to true.

**Actions**

🔄 reformat operation attribute("phone", Replace First("^\\(\\d\\d\\d)\\s\*(\\d\\d\\d)-(\\d\\d\\d\\d)\$", "\$1-\$2-\$3", Local Variable("current-value")))

🔄 Do reformat operation attribute

Enter name:\* phone

Enter value type: string

Enter string:\* Replace First("^\\(\\d\\d\\d)\\s\*(\\d\\d\\d)-(\\d\\d\\d\\d)\$", "\$1-\$2-\$3")

The action reformat operation attribute changes the format of the telephone number. The rule uses the Argument Builder and regular expressions to change how the information is displayed.

## 3.6.23 Remove Association

Sends a remove association command to the Identity Vault.

### 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



Specify the value of the association to be removed.

## Example



The example takes a delete operation and disables the User object instead. The transforms an event. The rule is from the predefined rules that come with Identity Manager 3.0. For more information, see [“Command Transformation - Publisher Delete to Disable”](#) on page 199.

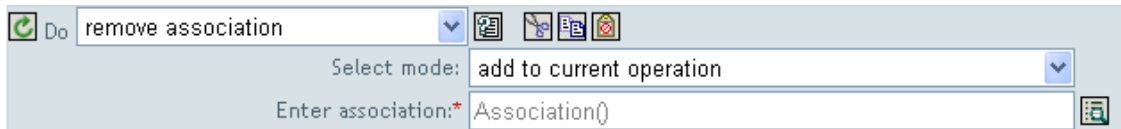
### Command Transformation - Publisher Delete to Disable

#### Conditions

-  if operation equal "delete"
- Or  if class name equal "User"

#### Actions

-  set destination attribute value("Login Disabled","true")
-  remove association(association(Association()))



Do remove association

Select mode: add to current operation

Enter association:\* Association()

When a delete operation occurs for a User object, value of the attribute Login Disabled is set to true and the association is removed from the object. The association is removed because the associated object in the connected application no longer exists.

## 3.6.24 Remove Destination Attribute Value

Removes an attribute value from an object in the destination data store.

### Fields

#### Attribute Name

Specify the name of the attribute.

#### Class Name

(Optional) Specify the class name of the target object. Leave blank to use the class name from the current object.

#### 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. This object can be the current object, or can be specified by a DN or an association.

#### Value Type

Specify the syntax of the new attribute value.

#### Value

Specify the value of the new attribute.



## Example

The screenshot shows a configuration dialog titled "Do remove destination attribute value". It contains the following fields:

- Enter attribute name\*: Member
- Enter class name: (empty)
- Select mode: add to current operation
- Select object: DN
- Enter DN\*: "Users/ManagerGroup"
- Enter value type: string
- Enter string\*: Destination DN()

### 3.6.25 Remove Source Attribute Value

Removes the specified value from the named attribute on an object in the source data store.

#### Fields

##### Attribute Name

Specify the name of the attribute.

##### Class Name

(Optional) Specify the class name of the target object. Leave blank to use the class name from the current object.

##### Object

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

##### Value Type

Specify the syntax of the attribute value to be removed.

##### Value

Specify the attribute value to be removed.

## Example

The screenshot shows a configuration dialog titled "Do remove source attribute value". It contains the following fields:

- Enter attribute name\*: Member
- Enter class name: (empty)
- Select object: DN
- Enter DN\*: "Users/ManagerGroup"
- Enter value type: string
- Enter string\*: Source DN()

### 3.6.26 Rename Destination Object

Renames an object in the destination data store.

#### Fields

##### Mode

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

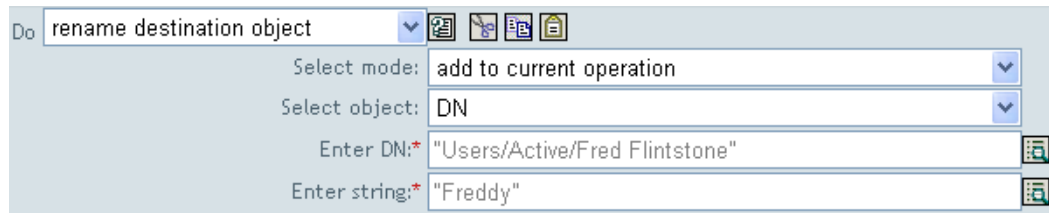
##### Object

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

##### String

Specify the new name of the object.

#### Example



The screenshot shows a configuration dialog for the 'rename destination object' operation. The title bar contains the text 'Do rename destination object' and several icons. Below the title bar, there are four input fields: 'Select mode:' with a dropdown menu set to 'add to current operation'; 'Select object:' with a dropdown menu set to 'DN'; 'Enter DN:\*' with a text input field containing '"Users/Active/Fred Flintstone"'; and 'Enter string:\*' with a text input field containing '"Freddy"'. Each of the last three fields has a search icon to its right.

### 3.6.27 Rename Operation Attribute

Renames all occurrences of an attribute within the current operation.

#### Fields

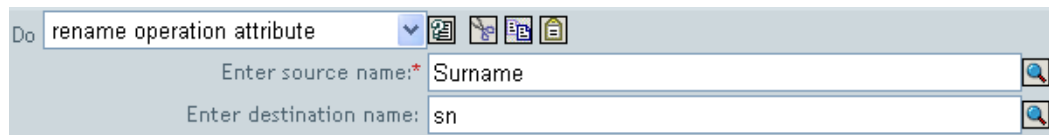
##### Source Name

Specify the original attribute name.

##### Destination Name

Specify the new attribute name.

#### Example



The screenshot shows a configuration dialog for the 'rename operation attribute' operation. The title bar contains the text 'Do rename operation attribute' and several icons. Below the title bar, there are two input fields: 'Enter source name:\*' with a text input field containing 'Surname' and a search icon to its right; and 'Enter destination name:' with a text input field containing 'sn' and a search icon to its right.

### 3.6.28 Rename Source Object

Renames an object in the source data store.

## Fields

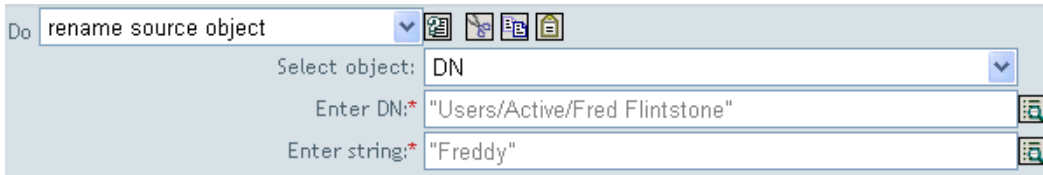
### Select Object

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

### String

Specify the new name of the object.

### Example



## 3.6.29 Send Email

Sends an e-mail notification.

### Fields

#### ID

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

#### Server

Specify the SMTP server name.

#### Password

(Optional) Specify SMTP server account password.

---

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

---

### Type

Select the e-mail message type.

### Strings

Specify the values containing 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.

String Name	Description
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.

### Example

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

Strings			
<input type="checkbox"/> Name:*	to	String tokens:*	"to_user1@company.com"
<input type="checkbox"/> Name:*	to	String tokens:*	"to_user2@company.com"
<input type="checkbox"/> Name:*	cc	String tokens:*	"cc_user@company.com"
<input type="checkbox"/> Name:*	bcc	String tokens:*	"bcc_user@company.com"
<input type="checkbox"/> Name:*	from	String tokens:*	"from_user@company.com"
<input type="checkbox"/> Name:*	subject	String tokens:*	"This is the e-mail subject"
<input type="checkbox"/> Name:*	message	String tokens:*	"This is the e-mail body"

### 3.6.30 Send Email from Template

Generates an e-mail notification using a template.

#### Fields

##### Notification DN

Specify the slash form DN of the SMTP notification configuration object.

##### Template DN

Specify the slash form DN of the e-mail template object.

##### Password

(Optional) Specify SMTP server account password.

---

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

---

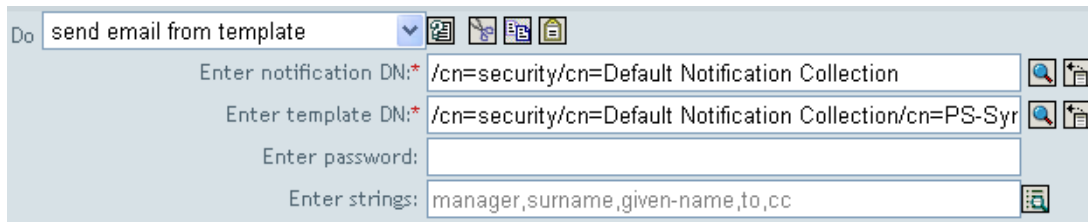
## Strings

Specify additional fields for the e-mail message. The following table contains reserved field names, 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.
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.

Each template might also define fields that can be replaced in the subject and body of the email message.

## Example



Do send email from template

Enter notification DN:\*/cn=security/cn=Default Notification Collection

Enter template DN:\*/cn=security/cn=Default Notification Collection/cn=PS-Syr

Enter password:

Enter strings: manager,surname,given-name,to,cc

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

Strings			
<input type="checkbox"/> Name:*	manager	String tokens:*	"Bill Jones"
<input type="checkbox"/> Name:*	surname	String tokens:*	"Smith"
<input type="checkbox"/> Name:*	given-name	String tokens:*	"Joe"
<input type="checkbox"/> Name:*	to	String tokens:*	"to_user@company.com"
<input type="checkbox"/> Name:*	cc	String tokens:*	"cc_user@company.com"

## 3.6.31 Set Default Attribute Value

Adds default values to the current operation (and optionally to the current object in the source data store) if no values for that attribute already exist. It is only valid when the current operation is add.

### Fields

#### Attribute Name

Specify the name of the default attribute.

#### Write Back

Select whether or not to also write back the default values to the source data store.

## Values

Specify the default values of the attribute.

## Example

The example sets the default value for the attribute company. You can set the value for an attribute of your choice. The rule is from the predefined rules that come with Identity Manager 3.0. For more information, see [“Creation - Set Default Attribute Value” on page 201](#).

The screenshot displays the configuration for the rule "Creation - Set Default Attribute Value". It is divided into several sections:

- Conditions:** A single condition is listed: "if class name equal 'User'".
- Actions:** A single action is listed: "set default attribute value('[Enter attribute name]',write-back='true','[Enter default attribute value]')".
- Action Configuration:** A detailed view of the "set default attribute value" action. It includes:
  - A dropdown menu set to "set default attribute value".
  - Fields for "Enter attribute name:\*" with the value "company".
  - A dropdown for "Write back:" set to "true".
  - A field for "Enter argument values:\*" with the value "Digital Airlines".
- Argument Values:** A section for defining argument values, showing:
  - A checkbox for "Type:\*" set to "string".
  - A field for "Enter string:\*" with the value "Digital Airlines Inc".

To build the value, the Argument Value List Builder is launched. See [“Argument Value List Builder” on page 192](#) for more information on the builder. You can set the value to what is needed. In this case, we used the Argument Builder and set the text to be the name of the company.

## 3.6.32 Set Destination Attribute Value

Adds a value to an attribute on an object in the destination data store, and removes all other values for that attribute.

### Fields

#### Attribute Name

Specify the name of the attribute.

#### Class Name

(Optional) Specify the class name of the target object in the destination data store. Leave blank to use the class name from the current object.

#### 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. This object can be the current object, or be specified by a DN or an association.

## Value Type

Select the syntax of the attribute value to set.

## Value

Specify the attribute values to set.

## Example

The example takes a delete operation and disables the User object instead. The rule is from the predefined rules that come with Identity Manager 3.0. For more information, see [“Command Transformation - Publisher Delete to Disable”](#) on page 199.

### Command Transformation - Publisher Delete to Disable

#### Conditions

if operation equal "delete"

Or if class name equal "User"

#### Actions

set destination attribute value("Login Disabled","true")

remove association(association(Association()))

The screenshot shows the configuration for the 'set destination attribute value' action. The dialog includes the following fields and options:

- Do:** set destination attribute value
- Enter attribute name:\*** Login Disabled
- Enter class name:** (empty)
- Select mode:** add to current operation
- Select object:** Current object
- Enter value type:** string
- Enter string:\*** "true"

The rule sets the value for the attribute of Login Disabled to true. The rule uses the Argument Builder to add the text of true for the value of the attribute. See [“Argument Builder”](#) on page 190 for more information about the builder.

## 3.6.33 Set Destination Password

Sets the password for the current object in the destination data store.

### 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. This object can be the current object, or be specified by an DN or an association.

### String

Specify the password to be set.

### Example

The example sets a default password for the User object that is created. The rule is from the predefined rules that come with Identity Manager 3.0. For more information, see [“Creation - Set Default Password” on page 202](#).

The screenshot displays the configuration for a rule named "Creation - Set Default Password". It is divided into two main sections: "Conditions" and "Actions".

- Conditions:** A single condition is listed: "if class name equal 'User'".
- Actions:** A single action is listed: "set destination password(Attribute('Given Name')+Attribute('Surname'))".

Below the actions list, there is a configuration panel for the selected action:

- Do:** set destination password
- Select mode:** add to current operation
- Enter string:** Attribute("Given Name")+Attribute("Surname")

When a User object is created, the password is set to the Given Name attribute plus the Surname attribute.

## 3.6.34 Set Local Variable

Sets a local variable.

### Fields

#### Variable Name

Specify the name of the new local variable.

#### Variable Type

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

### Example

The example adds a User object to the appropriate group, Employee or Manager, based on Title. It also creates the group, if needed, and setup security equal to that group. The policy name is Govern Groups for User Based on Title and it is available for download from Novell’s support Web site. For more information, see [“Downloadable Identity Manager Policies” on page 36](#).



**Set local variables to test existence of groups and for placement**

**Conditions**

- if class name equal "User"
- And**
- if operation equal "add"
- Or** if operation equal "modify"

**Actions**

- set local variable("manager-group-dn","Users\ManagersGroup")
- set local variable("manager-group-info",Destination Attribute("Object Class",dn(Local Variable ("manager-group-dn"))))
- set local variable("employee-group-dn","Users\EmployeesGroup")
- set local variable("employee-group-info",Destination Attribute("Object Class",dn(Local Variable ("employee-group-dn"))))

Do: set local variable

Enter variable name:\* manager-group-info

Select variable type: String

Enter string:\* Destination Attribute("Object Class",dn(Local Variable("ma

The local variable is set to the value that is in the User object’s destination attribute of Object Class plus the Local Variable of manager-group-info. The Argument Builder is used to construct the local variable. See [“Argument Builder” on page 190](#) for more information.

### 3.6.35 Set Operation Association

Sets the association value for the current operation.

#### Fields

##### Association

Provide the new association value.

#### Example

Do: set operation association

Enter association:\* Source Name()

### 3.6.36 Set Operation Class Name

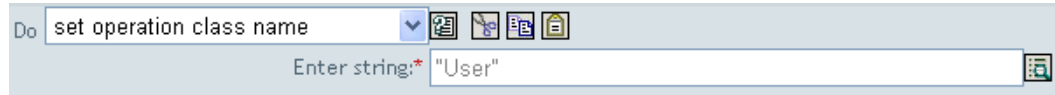
Sets the object class name for the current operation.

#### Fields

##### String

Specify the new class name.

## Example



Do set operation class name [dropdown icons]  
Enter string:\* "User" [icon]

### 3.6.37 Set Operation Destination DN

Sets the destination DN for the current operation.

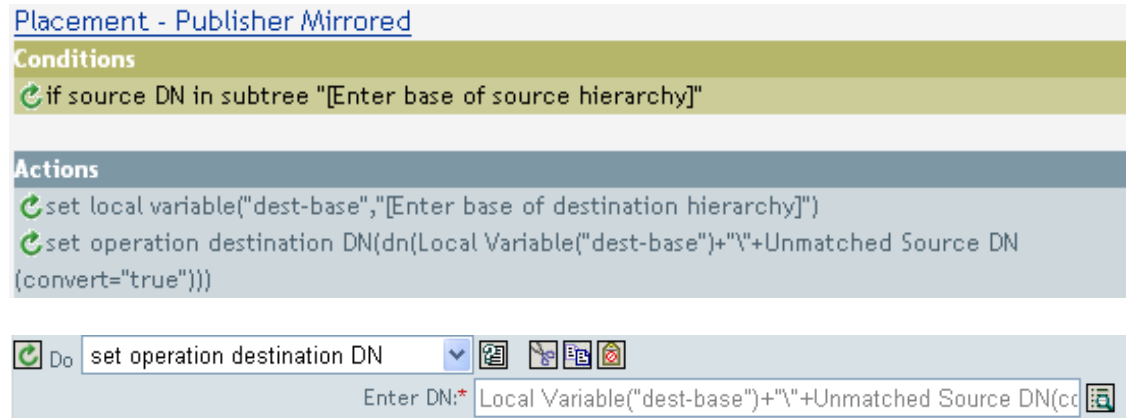
#### Fields

##### DN

Specify the new destination DN.

## Example

The example places the objects in the Identity Vault using the structure that is mirrored from the connected system. You need to define at what point the mirroring begins in the source and destination data stores. The rule is from the predefined rules that come with Identity Manager 3.0. For more information, see [“Creation - Set Default Attribute Value” on page 68](#).



Placement - Publisher Mirrored

**Conditions**

- if source DN in subtree "[Enter base of source hierarchy]"

**Actions**

- set local variable("dest-base","[Enter base of destination hierarchy]")
- set operation destination DN(dn(Local Variable("dest-base")+"\"+Unmatched Source DN (convert="true")))

Do set operation destination DN [dropdown icons]  
Enter DN:\* Local Variable("dest-base")+"\"+Unmatched Source DN(cc [icon]

The rule sets the operation destination DN to be the local variable of the destination base location plus the source DN.

### 3.6.38 Set Operation Property

Sets an operation property. An operation property is a named value that is stored within an operation. It is typically used to supply additional context that might be needed by the policy that handles the results of an operation.

#### Fields

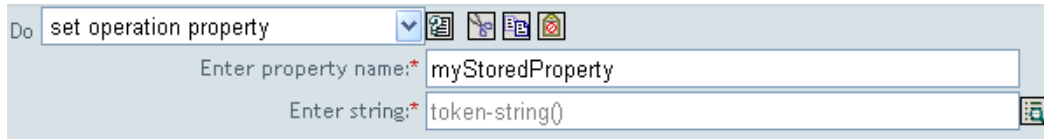
##### Property Name

Specify the name of the operation property.

##### String

Specify the name of the operation property.

### Example



Do set operation property

Enter property name:\* myStoredProperty

Enter string:\* token-string()

## 3.6.39 Set Operation Source DN

Sets the source DN for the current operation.

### Fields

#### DN

Specify the new source DN.

### Example



Do set operation source DN

Enter DN:\* "Novell\Users\'+Attribute("CN")

## 3.6.40 Set Operation Template DN

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

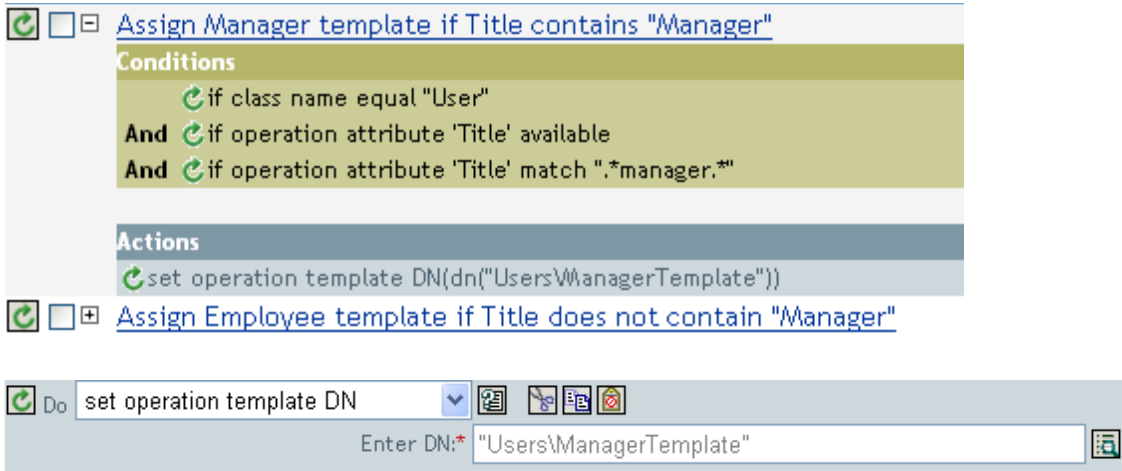
### Fields

#### DN

Specify the template DN.

### Example

The example applies the Manager template if the Title attribute contains the word Manager. The name of the policy is Policy: Assign Template to User Based on Tile, and it is available for download from Novell's support Web site. For more information, see ["Downloadable Identity Manager Policies"](#) on page 36.



The template Manager Template is applied to any User object the has the attribute of Title available and it contains the word manager somewhere in the title. The policy uses regular expressions to find all possible matches.

### 3.6.41 Set Source Attribute Value

Adds a value to an attribute on an object in the source data store, and removes all other values for that attribute.

#### Fields

##### Attribute Name

Specify the name of the attribute.

##### Class Name

(Optional) Specify the class name of the target object in the source data store. Leave blank to use the class name from the current object.

##### Object

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

##### Value Type

Select the syntax of the attribute value.

##### Value

Specify the attribute value to be set.

#### Example

The example detects when an e-mail address is changed and sets it back to what it was. The policy name is Policy: Reset Value of the E-mail Attribute, and it is available for download from Novell's support Web site. For more information, see ["Downloadable Identity Manager Policies" on page 36](#).

**Push back on email changing**

**Conditions**

- if class name equal "User"
- And if operation attribute 'Email' changing

**Actions**

- set source attribute value("Email",Destination Attribute("Internet EMail Address"))
- strip operation attribute("Email")

Do set source attribute value

Enter attribute value:\* Email

Enter class name:

Select object: Current object

Enter value type: string

Enter string:\* Destination Attribute("Internet EMail Address")

The action takes the value of the destination attribute Internet EMail Address and set the source attribute of Email to this same value.

### 3.6.42 Set Source Password

Sets the password for the current object in the source data store.

#### Fields

#### String

Specify the password to be set.

#### Example

Do set source password

Enter string:\* Attribute("Given Name")+Attribute("Surname")

### 3.6.43 Set XML Attribute

Sets an XML on a set of elements selected by an XPath expression.

#### Fields

#### Name

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

#### XPATH Expression

XPath 1.0 expression that returns a node set containing the elements on which the XML attribute should be set.

## String

Specify the value of the XML attribute.

### Example

The screenshot shows two instances of the 'set XML attribute' operation in a software interface. Each instance has three input fields: 'Enter name:', 'Enter XPATH expression:', and 'Enter string:'. The first instance has 'cert-id' in the name field, '.' in the XPATH field, and 'c:\lotus\domino\data\eng.id' in the string field. The second instance has 'cert-pwd' in the name field, '.' in the XPATH field, and 'certify2eng' in the string field. Each field has a search icon to its right.

## 3.6.44 Status

Generates a status notification.

### 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 immediately halt processing of the input document and schedules a retry of the event currently being processed.

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

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

### Example

The screenshot shows a single instance of the 'status' operation in a software interface. It has two input fields: 'Enter level:' and 'Message:'. The 'Enter level:' field contains 'warning' and the 'Message:' field contains 'Source DN()+: operation vetoed on out-of-scope object'. Each field has a search icon to its right.

## 3.6.45 Strip Operation Attribute

Strips all occurrences of an attribute from the current operation.

## Fields

### Name

Specify the name of the attribute to be stripped.

### Example

The example detects when an e-mail address is changed and sets it back to what it was. The policy name is Policy: Reset Value of the E-mail Attribute and it is available for download from Novell's support Web site. For more information, see [“Downloadable Identity Manager Policies” on page 36](#).

The screenshot shows a policy configuration window for "Push back on email changing". It includes a "Conditions" section with two items: "if class name equal 'User'" and "And if operation attribute 'Email' changing". The "Actions" section contains two items: "set source attribute value('Email',Destination Attribute('Internet EMail Address'))" and "strip operation attribute('Email')". Below this is a configuration window for the "strip operation attribute" action, where the name "Email" is entered in the "Enter name:" field.

The action strips the attribute of Email. The value that is kept is what was in the destination Email attribute.

## 3.6.46 Strip XPath

Strips nodes selected by an XPath 1.0 expression.

### Fields

#### XPATH Expression

Specify the XPath 1.0 expression that returns a node set containing the nodes to be stripped.

### Example

The screenshot shows the configuration window for the "strip XPath expression" action. The "Enter XPATH expression:" field contains the expression "\*[@attr-name='OU']".

## 3.6.47 Trace Message

Sends a message to DSTRACE.

### Fields

#### Level

Specify the trace level of the message. The default level is 0. The message only appears if the specified the trace level is less than or equal to the trace level configured in the driver.

For information on how to set the trace level on the driver, see “[Viewing Identity Manager Processes](#)” in the *Novell Identity Manager 3.0 Administration Guide*.

### Color

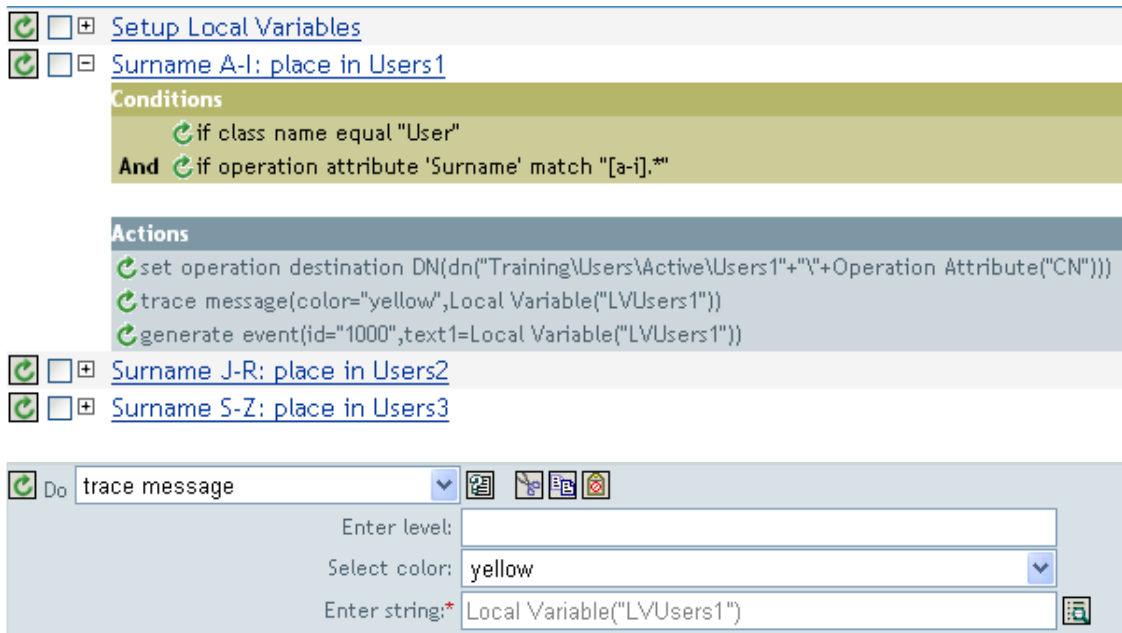
Select the color of the trace message.

### String

Specify the value of the trace message.

### Example

The example has four rules that implements a Placement policy for User objects based on the first character of the Surname attribute. It generates both a trace message and a custom Novell Audit event. The Trace Message action is used to send a trace message into DSTRACE. The policy name is Policy to Place by Surname and it is available for download from Novell’s support Web site. For more information “[Downloadable Identity Manager Policies](#)” on page 36.



The action sends a trace message to DSTRACE. The contents of the local variable is LVUsers1 and it shows up in yellow in DSTRACE.

## 3.6.48 Veto

Vetoes the current operation.

### Example

The example excludes all events that come from the specified subtree. The rule is from the predefined rules that come with Identity Manager 3.0. For more information, see “[Event Transformation - Scope Filtering - Exclude Subtrees](#)” on page 204 from the predefined rules.



## Event Transformation - Scope Filtering - Exclude subtree(s)

The screenshot shows a policy configuration window with two sections: **Conditions** and **Actions**. The **Conditions** section contains a single condition: "if source DN in subtree "[Enter a subtree to exclude]"". The **Actions** section contains a single action: "veto()". Below these sections is a toolbar with a dropdown menu showing "Do veto" and several icons for editing and saving.

The action vetoes all events that come from the specified subtree.

### 3.6.49 Veto if Operation Attribute Not Available

Conditionally cancels the current operation and ends processing of the current policy, based on the availability of an attribute in the current operation.

#### Fields

##### Name

Specify the name of the attribute.

#### Example

The example does not all User objects to be created unless the attributes Given Name, Surname, Title, Description, and Internet EMail Address are available. The policy name is Policy to Enforce the Presences of Attributes and it is available for download from Novell's support Web site. For more information, see ["Downloadable Identity Manager Policies" on page 36](#).

The screenshot shows a policy configuration window for a policy named "User required attributes: First/Last Name, Title, Description, Email". The **Conditions** section contains a single condition: "if class name equal "User"". The **Actions** section contains five actions: "veto if operation attribute not available("Given Name")", "veto if operation attribute not available("Surname")", "veto if operation attribute not available("Title")", "veto if operation attribute not available("Description")", and "veto if operation attribute not available("Internet EMail Address")". Below the actions is a toolbar with a dropdown menu showing "Do veto if operation attribute not avail:" and a search box containing "Given Name".

The actions vetoes the operation if the attributes of Given Name, Surname, Title, Description, and Internet Email Address are not available.

## 3.7 Noun Tokens

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

- [Section 3.7.1, “Added Entitlement,” on page 274](#)
- [Section 3.7.2, “Association,” on page 275](#)
- [Section 3.7.3, “Attribute,” on page 275](#)
- [Section 3.7.4, “Class Name,” on page 276](#)
- [Section 3.7.5, “Destination Attribute,” on page 276](#)
- [Section 3.7.6, “Destination DN,” on page 277](#)
- [Section 3.7.7, “Destination Name,” on page 278](#)
- [Section 3.7.8, “Entitlement,” on page 278](#)
- [Section 3.7.9, “Global Configuration Value,” on page 279](#)
- [Section 3.7.10, “Local Variable,” on page 279](#)
- [Section 3.7.11, “Named Password,” on page 280](#)
- [Section 3.7.12, “Operation,” on page 280](#)
- [Section 3.7.13, “Operation Attribute,” on page 280](#)
- [Section 3.7.14, “Operation Property,” on page 281](#)
- [Section 3.7.15, “Password,” on page 281](#)
- [Section 3.7.16, “Removed Attribute,” on page 282](#)
- [Section 3.7.17, “Removed Entitlements,” on page 282](#)
- [Section 3.7.18, “Source Attribute,” on page 282](#)
- [Section 3.7.19, “Source DN,” on page 283](#)
- [Section 3.7.20, “Source Name,” on page 283](#)
- [Section 3.7.21, “Text,” on page 283](#)
- [Section 3.7.22, “Unique Name,” on page 284](#)
- [Section 3.7.23, “Unmatched Source DN,” on page 286](#)
- [Section 3.7.24, “XPath,” on page 286](#)

### 3.7.1 Added Entitlement

Expands to the values of an entitlement granted in the current operation.

#### Fields

##### Name

Name of the entitlement.

#### Example

```
 Added Entitlement("manager")
```

## 3.7.2 Association

Expands to the association value from the current operation.


### Example


The example is from the predefined rules that come with Identity Manager 3.0. For more information on the predefined rule, see [“Command Transformation - Publisher Delete to Disable” on page 199](#).

The action of Remove Association uses the Association token to retrieve the value from the current operation. The rule removes the association from the User object so that any new events coming through do not affect the User object.


#### Command Transformation - Publisher Delete to Disable


##### Conditions

 if operation equal "delete"

Or  if class name equal "User"

##### Actions

 set destination attribute value("Login Disabled","true")

 remove association(association(Association()))

 Association()

## 3.7.3 Attribute

Expands to the value of an attribute from the current object in the current operation and in the source data store. It can be logically thought of as the union of the operation attribute token and the source attribute token. It does not include the removed values from a modify operation.

### Fields

#### Name

Specify the name of the attribute.


### Example

The example is from the predefined rules that come with Identity Manager 3.0. For more information, see [“Creation - Set Default Password” on page 202](#).

The action of Set Destination Password uses the attribute token to create the password. The password is made up of the Given Name attribute and the Surname attribute. When you are in the Argument Builder Editor, you browse and select the attribute you want to use.



## Creation - Set Default Password

### Conditions

 if class name equal "User"

### Actions

 set destination password(Attribute("Given Name")+Attribute("Surname"))

 Attribute("Given Name")  
 Attribute("Surname")

### Editor


Name: \*



## 3.7.4 Class Name

Expands to the object class name from the current operation.

### Example

 Class Name()

## 3.7.5 Destination Attribute

Expands to the specified attribute value of the current object, a DN, or association, in the destination data store.

### Fields

#### Class Name

(Optional) Specify the class name of the target object. Leave blank to use the class name from the current object.

#### Name

Name of the attribute.

### Example

The example is from the Govern Groups for User Based on Title policy and it is available for download from Novell's support Web site. For more information, see ["Downloadable Identity Manager Policies" on page 36](#).

The policy creates the Destination Attribute with the Argument Builder. The action of Set Local Variable contains the Destination Attribute token.

Set local variables to test existence of groups and for placement

**Conditions**

- if class name equal "User"
- And**
- if operation equal "add"
- Or** if operation equal "modify"

**Actions**

- set local variable("manager-group-dn","Users\ManagersGroup")
- set local variable("manager-group-info",Destination Attribute("Object Class",dn(Local Variable ("manager-group-dn"))))
- set local variable("employee-group-dn","Users\EmployeesGroup")
- set local variable("employee-group-info",Destination Attribute("Object Class",dn(Local Variable ("employee-group-dn"))))

Destination Attribute("Object Class", dn())

**Editor**

Name: \*

Class name:

Select object:

Enter DN: \*

You build the Destination Attribute through the Editor. In this example, the attribute of Object Class is set. DN is used to select the object. The value of DN is the Local Variable of manager-group-dn.

### 3.7.6 Destination DN

Expands to the destination DN specified in the current operation.

#### Fields

##### Convert

Select whether or not to convert the DN to the format used by the source data store.

##### Start

Specify the RDN index to start with:

- Index 0 is the root-most RDN
- Positive indexes are an offset from the root-most RDN
- Index -1 is the leaf-most segment
- Negative indexes are an offset from the leaf-most RDN towards the root-most RDN

##### Length

Specify the number of RDN to include. Negative numbers are interpreted as (total # of segments + length) + 1. For example, 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}, the entire DN is used; otherwise only the portion of the DN specified by start and length is used.

## Example

The example uses the Destination DN token to set the value for the local variable of target-container. The policy creates a department container for the User object if it does not exist. The policy is from the predefined rules that come with Identity Manager 3.0. For more information, see [“Command Transformation - Create Departmental Container - Part 1 and Part 2”](#) on page 197.

**Command Transformation - Create Departmental Container - Part 1**

**Conditions**

- if operation equal "add"

**Actions**

- set local variable("target-container",Destination DN(length="-2"))
- set local variable("does-target-exist",Destination Attribute("objectclass",class name="OrganizationalUnit",dn(Local Variable("target-container"))))

..... Destination DN(length="-2")

## 3.7.7 Destination Name

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

## Example

Destination Name()

## 3.7.8 Entitlement

Expands to the values of a granted entitlement from the current object.

## Fields

### Name

Name of the entitlement.

## Example

Entitlement("manager")

### 3.7.9 Global Configuration Value


Expands to the value of a global configuration variable.

#### Fields

##### Name

Name of the global configuration value.

#### Example

 Global Configuration Value("Fred")

### 3.7.10 Local Variable

Expands to the value of a local variable.

#### Fields

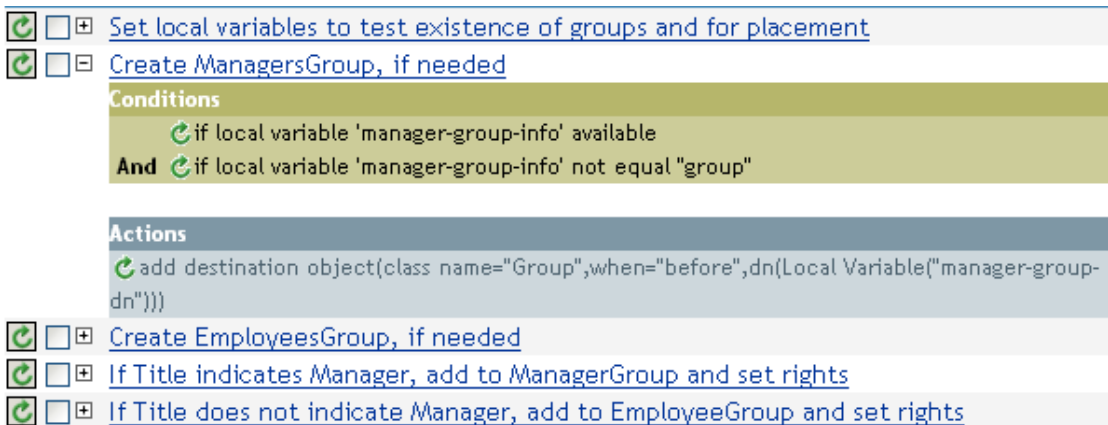
##### Name

Specify the name of the local variable.

#### Example


The example is from the Govern Groups for User Based on Title policy which is available for download from Novell's support Web site. For more information, see ["Downloadable Identity Manager Policies" on page 36](#).

The action Add Destination Object uses the Local Variable token.



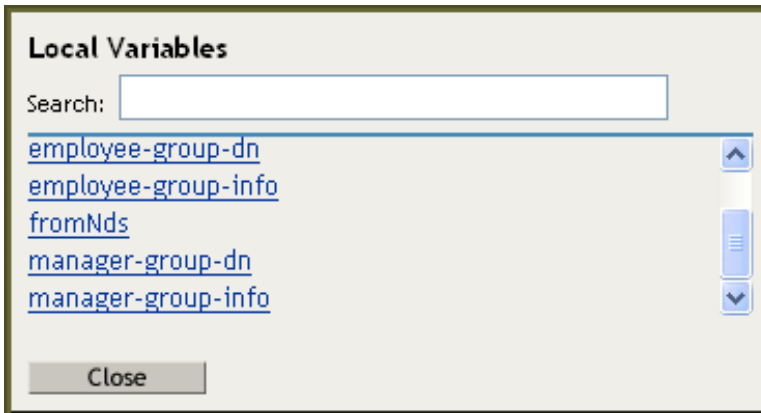
The screenshot shows a list of actions in a Policy Builder interface. The actions are:

- [Set local variables to test existence of groups and for placement](#)
- [Create ManagersGroup, if needed](#)
  - Conditions**
    - if local variable 'manager-group-info' available
    - And**  if local variable 'manager-group-info' not equal "group"
  - Actions**
    - add destination object(class name="Group",when="before",dn(Local Variable("manager-group-dn")))
- [Create EmployeesGroup, if needed](#)
- [If Title indicates Manager, add to ManagerGroup and set rights](#)
- [If Title does not indicate Manager, add to EmployeeGroup and set rights](#)

 Local Variable("manager-group-dn")

#### Editor

Variable name: \*



The Local Variable can only be used if the action Set Local Variable has been used previously in the policy. It sets the value that is stored in the Local Variable. In the Editor, you click the browse icon and all of the local variables that have been defined are listed. Select the correct local variable.

The value of the local variable is group-manager-dn. In the rule before this one, the Set Local Variable action defined group-manager-dn as DN of the manager's group Users\ManagersGroup.

### 3.7.11 Named Password


Expands to the named password from the driver.

#### Fields

##### Name

Name of the password.


#### Example

 Named Password("password")

### 3.7.12 Operation

Expands to the name of the current operation.

#### Example

 Operation()

### 3.7.13 Operation Attribute

Expands to the value of an attribute from the current operation. It does not include the removed values from a modify operation.

#### Fields

##### Name

Specify the name of the attribute.



## Example

The example has four rules that implements a Placement policy for User objects based on the first character of the Surname attribute. It generates both a trace message and a custom Novell Audit event. The policy name is Policy to Place by Surname, and it is available for download from Novell's support Web site. For more information "[Downloadable Identity Manager Policies](#)" on [page 36](#).

[Setup Local Variables](#)

[Surname A-I: place in Users1](#)

**Conditions**

- if class name equal "User"
- And**  if operation attribute 'Surname' match "[a-].\*\*"

**Actions**

- set operation destination DN(dn("Training\Users\Active\Users1"+"\"+Operation Attribute("CN")))
- trace message(color="yellow",Local Variable("LVUsers1"))
- generate event(id="1000",text1=Local Variable("LVUsers1"))

[Surname J-R: place in Users2](#)

[Surname S-Z: place in Users3](#)

"Training\Users\Active\Users1"  
"  
Operation Attribute("CN")

**Editor**

Name: \*

The action Set Operation Destination DN contains the Operation Attribute token. The Operation Attribute token sets the Destination DN to the CN attribute. The rule takes the context of Training\Users\Active\Users and adds a \ plus the value of the CN attribute.

### 3.7.14 Operation Property

Expands to the value of the specified operation property on the current operation.

#### Fields

##### Name

Specify the name of the operation property.


#### Example

Operation Property("myStoredProperty")

### 3.7.15 Password

Expands to the password specified in the current operation.

## Example

 Password()

### 3.7.16 Removed Attribute

Expands to the specified attribute value being removed in the current operation. It only applies to modify operation.

#### Fields

##### Name

Specify the name of the attribute.

#### Example

 Removed Attribute("OU")

### 3.7.17 Removed Entitlements


Expands to the values of the an entitlement revoked in the current operation.

#### Fields

##### Name

Specify the name of the entitlement.

#### Example

 Removed Entitlement("manager")

### 3.7.18 Source Attribute

Expands to the values of an attribute from an object in the source data store.

#### Fields

##### Class Name

(Optional) Specify the class name of the target object. Leave blank to use the class name from the current object.

##### Name

Name of the attribute.

#### Example

 Source Attribute("OU")

### 3.7.19 Source DN

Expands to the source DN from the current operation.

#### Fields

##### Convert

Select whether or not to convert the DN to the format used by the destination data store.

##### Start

Specify the RDN index to start with:

- Index 0 is the root-most RDN
- Positive indexes are an offset from the root-most RDN
- Index -1 is the leaf-most segment
- Negative indexes are an offset from the leaf-most RDN towards the root-most RDN


##### Length

Number of RDN's segments to include. Negative numbers are interpreted as (total # of segments + length) + 1. For example, 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}, the entire DN is used; otherwise only the portion of the DN specified by start and length is used.

#### Example

 Source DN()

### 3.7.20 Source Name

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

#### Example

 Source Name()

### 3.7.21 Text

Expands to the text.

#### Fields

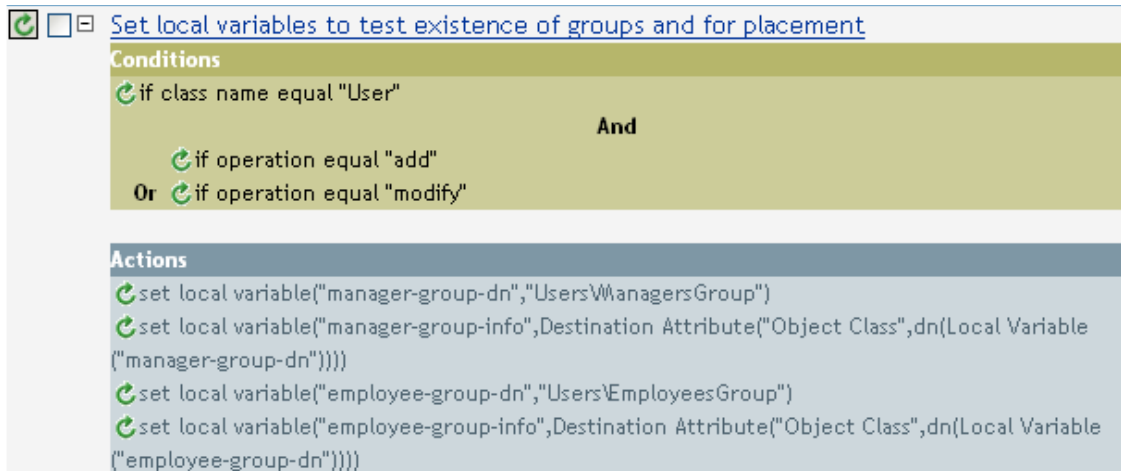
##### Text


Specify the text.

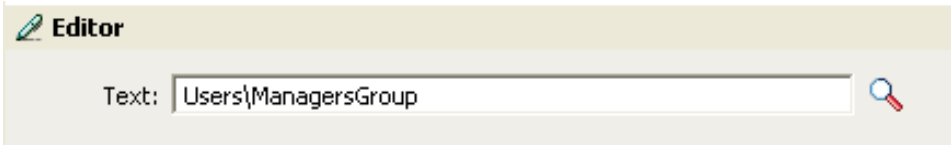
## Example

The example is from the Govern Groups for User Based on Title policy which is available for download from Novell's support Web site. For more information, see [“Downloadable Identity Manager Policies” on page 36](#).

The Text token is used in the action Set Location Variable to define the DN of the manager's group. The Text token can contain objects or plain text.



.....  "Users\ManagersGroup"



The Text token contains the DN for the manager's group. You can browse to the object you would like to use, or type in the information into the editor.

## 3.7.22 Unique Name

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

### Fields

#### Name

Specify the name of attribute to check for uniqueness.

#### Scope

Specify the scope in which to check uniqueness.

#### 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

Specify patterns to use to generate unique values by using the Argument Builder.

## Counter Start

Specify the a number to start counter used when needed to find a unique name.

## Digits

Specify the width in digits of counter; the 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 against the destination data store, using the supplied attribute name, scope, and search start. Each specified pattern is tried in order until a value is found that does not return any found objects.

If all of the specified patterns are exhausted, the final pattern has a counter appended to it and the pattern is tried repeatedly (increasing the counter each time) until the query does not return any instances.

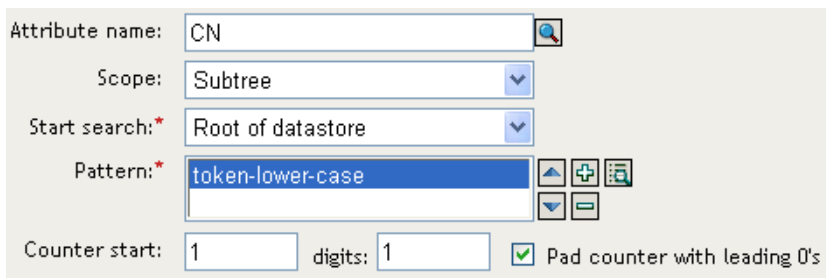
The counter can be set to start at a different number using the counter start field. The counter uses the number of digits specified by the digits field. If the number of digits is less than those specified, then the counter is right padded with zeros. When the number of digits exceeds those specified, then no unique name is generated and the enclosing rule returns an error status.


If the destination data store is the Identity Vault and name field is left blank, 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 connected application, then the name field is required.


## Example


 Unique Name("CN",scope="subtree",Lower Case())






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



Attribute name:  

Scope:  

Start search: \*  

Pattern: \*      
 

Counter start:  digits:   Pad counter with leading 0's

The following pattern was constructed to provide unique names:

```
Lower Case()
├── Substring()
│   ├── Attribute("Given Name")
│   └── +
│       └── Attribute("Surname")
```

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

### 3.7.23 Unmatched Source DN

Expands to the part 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.

#### Fields

##### Convert

Select whether or not to convert the DN format used by the destination data store.

#### Remarks

If there were no matches, the entire DN is used.

#### Example

The example is from the predefined rules that come with Identity Manager 3.0. For more information, see [“Matching - Subscriber Mirrored - LDAP Format” on page 208](#).

The action of Finding Matching Object uses the Unmatched Source DN token to build the matching information in LDAP format. It takes the unmatched portion of the source DN to make a match.

[Matching - Subscriber Mirrored - LDAP format](#)

**Conditions**

- if source DN in subtree "[Enter base of source hierarchy]"

**Actions**

- set local variable("dest-base","[Enter base of destination hierarchy]")
- find matching object(scope="entry",dn(Unmatched Source DN(convert="true")+","+Local Variable("dest-base")))

Unmatched Source DN(convert="true")  
"  
Local Variable("dest-base")

**Editor**

Convert to destination DN format:

### 3.7.24 XPath

Expands to results of evaluating an XPath 1.0 expression.

## Fields

### Expression

XPath 1.0 expression to evaluate.

### Example

```
⌘ XPATH("//*[@attr-name='OU']/value[starts-with(string(.),'xxx']")
```

## 3.8 Verb Tokens

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

- [Section 3.8.1, “Escape Destination DN,” on page 287](#)
- [Section 3.8.2, “Escape Source DN,” on page 288](#)
- [Section 3.8.3, “Lower Case,” on page 288](#)
- [Section 3.8.4, “Parse DN,” on page 289](#)
- [Section 3.8.5, “Replace All,” on page 290](#)
- [Section 3.8.6, “Replace First,” on page 291](#)
- [Section 3.8.7, “Substring,” on page 292](#)
- [Section 3.8.8, “Upper Case,” on page 293](#)

### 3.8.1 Escape Destination DN

Escapes a string according to the rules of the DN format of the destination data store.

#### Example

The example is from the predefined rules that come with Identity Manager 3.0. For more information, see [“Placement - Publisher Flat” on page 84](#).

The action of Set Operation Destination DN uses the Escape Destination DN token to build the destination DN of the User object.

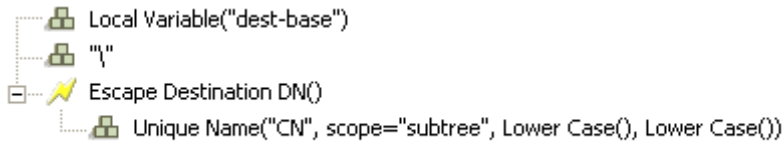
#### Placement - Publisher Flat

##### Conditions

```
⌘ if class name equal "User"
```

##### Actions

```
⌘ set local variable("dest-base", "[Enter DN of destination container]")  
⌘ set operation destination DN(dn(Local Variable("dest-base")+"\"+Escape Destination DN(Unique  
Name("CN",scope="subtree",Lower Case(Substring(length="1",Operation Attribute("Given Name"))  
+Operation Attribute("Surname")),Lower Case(Substring(length="2",Operation Attribute("Given  
Name"))+Operation Attribute("Surname"))))))
```

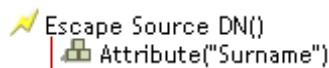


The Escape Destination DN token takes the value in Unique Name and sets it to the format for the destination DN.

### 3.8.2 Escape Source DN

Escapes a string according to the rules of the DN format of the source data store.

#### Example



### 3.8.3 Lower Case

Converts the characters in a string to lowercase.

#### Example

This example sets the e-mail address to be name@slartybartfast.com where the name equals the first character of the Given Name plus the Surname. The policy name is Policy: Create E-mail from Given Name and Surname and it is available for download at Novell’s support Web site. For more information, see [“Downloadable Identity Manager Policies” on page 36](#).

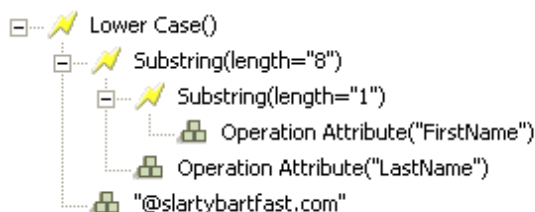
**Set email address: name@slartybartfast.com; name = (1 char of Given Name + Surname) <= 8 chars**

**Conditions**

- if class name equal "User"
- And if operation attribute 'Given Name' available
- And if operation attribute 'Surname' available

**Actions**

- strip operation attribute("Internet Email Address")
- set destination attribute value("Internet Email Address", Lower Case(Substring(length="8", Substring(length="1", Operation Attribute("FirstName")) + Operation Attribute("LastName")) + "@slartybartfast.com"))





The Lower Case token sets all of the information in the action Set Destination attribute value to lowercase.

### 3.8.4 Parse DN

Converts a DN to an alternate format.

#### Example

The example uses the Parse DN token to build the value the Add Destination Attribute Value action. The example is from the predefined rules that come with Identity Manager 3.0. For more information, see [“Command Transformation - Create Departmental Container - Part 1 and Part 2” on page 197](#).

The screenshot displays the configuration for the Parse DN token within a rule. The rule is titled "Command Transformation - Create Departmental Container - Part 2".

**Conditions:**

- if local variable 'does-target-exist' available
- And if local variable 'does-target-exist' equal ""

**Actions:**

- add destination object(class name="organizationalUnit",direct="true",dn(Local Variable("target-container")))
- add destination attribute value("ou",direct="true",dn(Local Variable("target-container")),Parse DN("dest-dn","dot",length="1",start="-1",Local Variable("target-container")))

The configuration for the Parse DN token is shown in a separate window:

- Parse DN("dest-dn", "dot", length="1", start="-1")
- Local Variable("target-container")

The **Editor** window shows the following settings:

- Start: -1
- Length: 1
- Source DN format: destination DN
- Destination DN format: dot

The Parse DN token is taking the information from the source DN and converting it to the dot notation. The information from the Parse DN is stored in the attribute value of OU.

#### Fields

##### Start

Specify the RDN index to start with:

- Index 0 is the root-most RDN
- Positive indexes are an offset from the root-most RDN
- Index -1 is the leaf-most segment
- Negative indexes are an offset from the leaf-most RDN towards the root-most RDN

**Length**

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

**Source DN Format**

Specifies the format used to parse the source DN.

**Destination DN Format**

Specify the format used to output the parsed DN.

**Source DN Delimiter**

Specify the custom source DN delimiter set if Source DN Format is set to custom.

**Destination DN Delimiter**

Specify the custom destination DN delimiter set if Destination DN Format is set to custom.

**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.

When specifying custom DN formats, the eight characters that make up the delimiter set are defined as follows:

1. Typed Name Boolean Flag: 0 means names are not typed, and 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, such as \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, the orientation of the name is root right, otherwise the orientation is root left.

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

### **3.8.5 Replace All**

Replaces all occurrences of a regular expression in a string.

## Fields

### Regular Expression

Specify the regular expression that matches the substring to be replaced.

### Replace With

Specify the replacement string.



### Remarks

For details on creating regular expressions, see:

- Sun's Java Web site (<http://java.sun.com/j2se/1.4/docs/api/java/util/regex/Pattern.html>)
- Sun's Java Web site ([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 options `CASE_INSENSITIVE`, `DOTALL`, and `UNICODE_CASE` are used but can be reversed by using the appropriate embedded escapes.

### Example

 Replace All("(.)", "\$1")  
 Destination DN()

## 3.8.6 Replace First

Replaces the first occurrence of a regular expression in a string.

### Fields

### Regular Expression

Specify the regular expression that matches the substring to replace.

### Replace With

Specify 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:

- Sun's Web site (<http://java.sun.com/j2se/1.4/docs/api/java/util/regex/Pattern.html>)
- Sun's Web site ([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.

## Example

The example reformats the telephone number (nnn)-nnn-nnnn to nnn-nnn-nnnn. The rule is from the predefined rules that come with Identity Manager 3.0. For more information, see [“Input or Output Transformation - Reformat Telephone Number from \(nnn\) nnn-nnnn to nnn-nnn-nnnn”](#) on page 205.

The Replace First token is used in the Reformat Operation Attribute action.

The screenshot shows the configuration for a transformation rule. At the top, the title is "Input or Output Transformation - Reformat Telephone Number from (nnn) nnn-nnnn to nnn-nnn-nnnn". Below the title, there are two main sections: "Conditions" and "Actions".

**Conditions:** A green bar with a refresh icon and the text "This condition will evaluate to true."

**Actions:** A blue bar with a refresh icon and the text "reformat operation attribute('phone', Replace First('^((\d\d\d)\s\*(\d\d\d)-(\d\d\d\d))\$', '\$1-\$2-\$3', Local Variable('current-value'))"

Below the actions, there is a detailed view of the "Replace First" action. It shows a lightning bolt icon, a plus sign, and the text "Replace First('^((\d\d\d)\s\*(\d\d\d)-(\d\d\d\d))\$', '\$1-\$2-\$3')". A small icon of a box with a plus sign is next to the text "Local Variable('current-value')".

**Editor:** A section with a pencil icon and the word "Editor". It contains two input fields: "Regular expression: \*" with the value "^((\d\d\d)\s\*(\d\d\d)-(\d\d\d\d))\$" and "Replace with:" with the value "\$1-\$2-\$3".

The regular expression of `^((\d\d\d)\s*(\d\d\d)-(\d\d\d\d))$` represents (nnn) nnn-nnnn and the regular expression of `$1-$2-$3` represents nnn. This rule transforms the format of the telephone number from (nnn) nnn-nnnn to nnn-nnn-nnnn.

## 3.8.7 Substring

Extracts a portion of a string.

### Fields

#### Start

Specify the starting character index:

- Index 0 is the first character.
- Positive indexes are an offset from the start of the string.
- Index -1 is the last character.
- Negative indexes are an offset from the last character towards the start of the string.

#### Length

Number of characters from the start to include in the substring. Negative numbers are interpreted as  $(\text{total \# of characters} + \text{length}) + 1$ . For example, for a string with 5 characters a length of  $-1 = (5 + (-1)) + 1 = 5$ ,  $-2 = (5 + (-2)) + 1 = 4$ , etc.

## Example

This example sets the e-mail address to be name@slartybartfast.com where the name equals the first character of the Given Name plus the Surname. The policy name is Policy: Create E-mail from Given Name and Surname and it is available for download at Novell's support Web site. For more information, see [“Downloadable Identity Manager Policies” on page 36](#).

**Push back on email changing**

**Conditions**

- if class name equal "User"
- And if operation attribute 'Email' changing

**Actions**

- set source attribute value("Email",Destination Attribute("Internet EMail Address"))
- strip operation attribute("Email")

Lower Case()  
Substring(length="8")  
Substring(length="1")  
Operation Attribute("FirstName")  
Operation Attribute("LastName")  
"@slartybartfast.com"

The Substring token is used twice in the action Set Destination Attribute Value. It takes the first character of the First Name attribute and adds eight characters of the Last Name attribute together to form one substring.

## 3.8.8 Upper Case

Converts the characters in a string to uppercase.

### Example

The example converts the first and last name attributes of the User object to uppercase. The policy name is Policy: Convert First/Last Name to Upper Case and it is available for download at Novell's support Web site. For more information, see [“Downloadable Identity Manager Policies” on page 36](#).

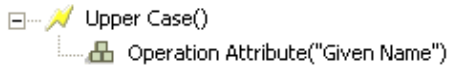
**Convert First/Last name to uppercase**

**Conditions**

- if class name equal "User"
- And if operation attribute 'Given Name' changing
- Or if operation attribute 'Surname' changing

**Actions**

- reformat operation attribute("Given Name",Upper Case(Operation Attribute("Given Name")))
- reformat operation attribute("Surname",Upper Case(Operation Attribute("Surname")))



## 3.9 Values

This section contains a list of common Policy Builder values.

### 3.9.1 Comparison Modes

Mode	Description
case	Character-by-character case sensitive comparison.
nocase	Character-by-character case insensitive comparison.
regex	<p>Regular expression match of entire string. Case insensitive by default, but may be changed by an escape in the expression.</p> <p>See <a href="http://java.sun.com/j2se/1.4/docs/api/java/util/regex/Pattern.html">Sun's Java Web site (http://java.sun.com/j2se/1.4/docs/api/java/util/regex/Pattern.html)</a> and <a href="http://java.sun.com/j2se/1.4/docs/api/java/util/regex/Matcher.html#matches()">Sun's Java Web site (http://java.sun.com/j2se/1.4/docs/api/java/util/regex/Matcher.html#matches())</a>.</p> <p>Note that pattern option CASE_INSENSITIVE, DOTALL, and UNICODE_CASE are used but can be reversed using the appropriate embedded escapes.</p>
src-dn	Compare using semantics appropriate to the DN format for the source data store.
dest-dn	Compare using semantics appropriate to the DN format for the destination data store.
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.

# Defining Policies using XSLT Style Sheets

Policies can be implemented as XSLT style sheets. XSLT is a standard language for transforming XML documents. The XSLT processor in the Metadirectory engine is compliant with the 16 November 1999 W3C recommendation. For the relevant specifications, see the following:

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

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

- [Section 4.1, “Managing XSLT Style Sheets in Designer,” on page 295](#)
- [Section 4.2, “Managing XSLT Style Sheets in iManager,” on page 297](#)
- [Section 4.3, “Starting with an Identity Transformation,” on page 298](#)
- [Section 4.4, “Using the Parameters that Identity Manager Passes,” on page 298](#)
- [Section 4.5, “Using Extension Functions,” on page 301](#)
- [Section 4.6, “Creating a Password Example: Creation Policy,” on page 302](#)
- [Section 4.7, “Creating an eDirectory User Example: Creation Policy,” on page 303](#)

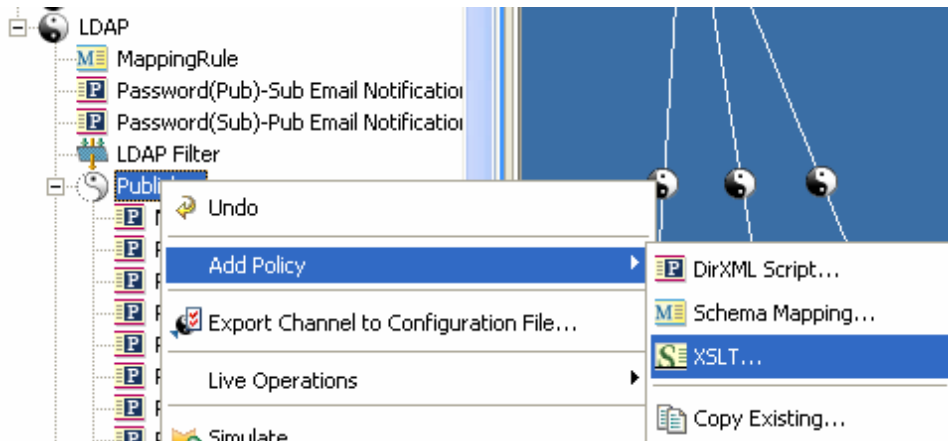
## 4.1 Managing XSLT Style Sheets in Designer

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

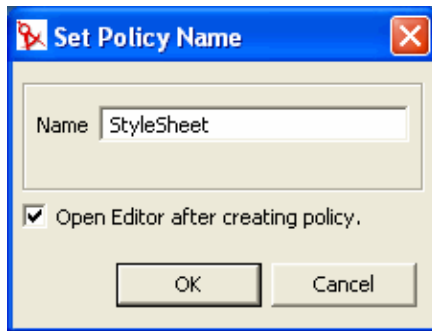
- [Section 4.1.1, “Adding an XSLT Policy in Designer,” on page 295](#)

### 4.1.1 Adding an XSLT Policy in Designer

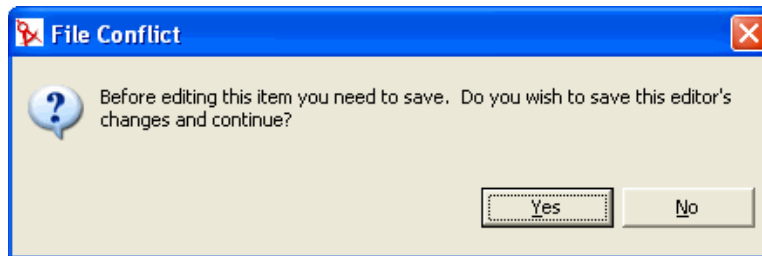
- 1 Open a project in Designer and select the Outline tab.
- 2 Select the driver and location where you want the style sheet.
- 3 Right-click and select *Add Policy >XSLT*.



- 4 Specify the name of the style sheet.
- 5 Select *Open Editor after creating policy*, then click *OK*.



- 6 Select *Yes* to save the project before editing the new policy.



- 7 Add the style sheet information below the line add your custom templates here.





```
<?xml version="1.0" encoding="UTF-8"?><xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">
  <xsl:param name="srcQueryProcessor"/>
  <xsl:param name="destQueryProcessor"/>
  <xsl:param name="srcCommandProcessor"/>
  <xsl:param name="destCommandProcessor"/>
  <xsl:param name="dnConverter"/>
  <xsl:param name="fromNds"/>
  <!-- identity transformation template -->
  <!-- in the absence of any other templates this will copy the input through unchanged -->
  <xsl:template match="node()|@*">
    <xsl:copy>
      <xsl:apply-templates select="@*|node()" />
    </xsl:copy>
  </xsl:template>
  <!-- add your custom templates here -->
</xsl:stylesheet>
```

8 Save the style sheet by selecting *File > Save*.

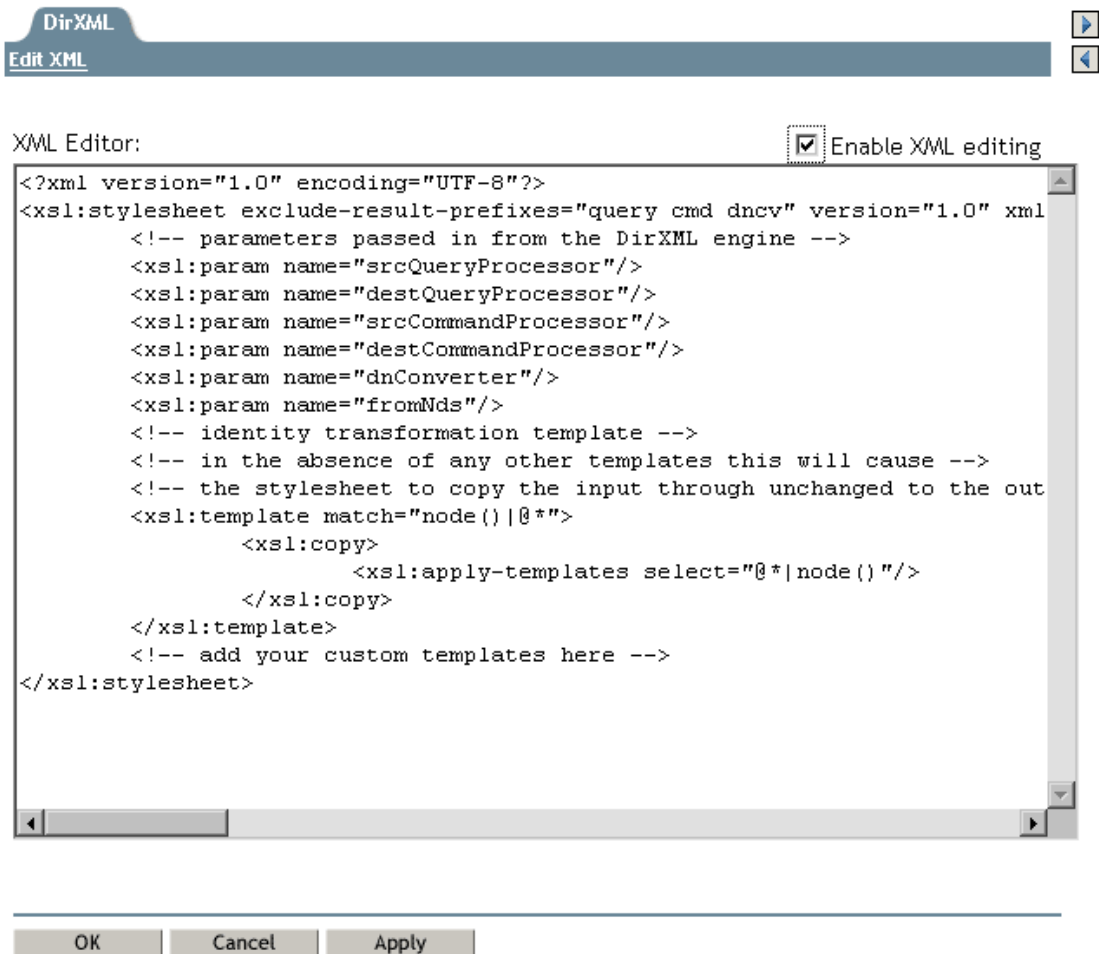
## 4.2 Managing XSLT Style Sheets in iManager

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

- [Section 4.2.1, “Adding an XSLT Policy in iManager,” on page 297](#)

### 4.2.1 Adding an XSLT Policy in iManager

- 1 Open the Identity Manager Driver Overview for the driver you want to manage.
- 2 Click the icon representing the policy you want to define.
- 3 Click *Insert*.
- 4 Provide a name for the new policy, select XSLT, then click *Enter*.
- 5 Define your XSLT policy, then click *OK*:



## 4.3 Starting with an Identity Transformation

When you create a new stylesheet in iManager or Designer, it is pre-populated with a stylesheet that implements the identity transformation. In the absence of additional templates, the identity transformation allows the input XML document to pass through the stylesheet unchanged. You usually implement policy by adding additional templates to act on just the XML that you want to be changed. If your stylesheet is being used to translate a document to or from an XML vocabulary that is different than XDS (such as the Input and Output Transformations for the SOAP and Delimited Text drivers) you may need to remove the identity template.

## 4.4 Using the Parameters that Identity Manager Passes

The Metadirectory engine passes the policy style sheets the following parameters that the style sheet can use.

- `srcQueryProcessor`—A Java object that implements the `XdsQueryProcessor` interface. This allows the style sheet to query the source datastore for more information.

- `destQueryProcessor`—A Java object that implements the `XdsQueryProcessor` interface. This allows the style sheet to query the destination datastore for more information.
- `srcCommandProcessor`—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`—A java object that implements the `XdsCommandProcessor` interface. This allows the style sheet to issue a command directly to send a command to the destination datastore.
- `dnConverter`—This is a java object that implements the `XdsCommandProcessor` interface. This allows the style sheet to convert Identity Vault object DNs from one format to another. For more information see [Interface DNConverter \(http://developer.novell.com/ndk/doc/dirxml/dirxmlbk/api/com/novell/nds/dirxml/driver/DNConverter.html\)](http://developer.novell.com/ndk/doc/dirxml/dirxmlbk/api/com/novell/nds/dirxml/driver/DNConverter.html).
- `fromNds`—This is a boolean value that is true if the source datastore is the Identity Vault and false if it is the connected application.

When you create a new stylesheet in iManager or Designer, it is pre-populated with a stylesheet that contains the declarations for these parameters.

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

- 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 references are added to the query.
- Responses from the application shim are in the form returned by the shim with no modification or schema mapping performed and no resolution of association references.
- Queries issued to eDirectory™ must be in the form expected by eDirectory. In other words schema names must be in the eDirectory namespace and the query must be XDS. Association references are not resolved.
- Responses from the application shim are 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 namespace 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"
```

When you create a new stylesheet in iManager or Designer, it is pre-populated with the namespace declaration. For more information about query processors see [Class XdsQueryProcessor \(http://developer.novell.com/ndk/doc/dirxml/dirxmlbk/api/com/novell/nds/dirxml/driver/XdsQueryProcessor.html\)](http://developer.novell.com/ndk/doc/dirxml/dirxmlbk/api/com/novell/nds/dirxml/driver/XdsQueryProcessor.html)

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 -->
<xsl:template name="query-object-name">
  <xsl:param name="object-name"/>
  <!-- build an xds query as a result tree fragment -->
  <xsl:variable name="query">
    <query>
      <search-class class-name="{ancestor-or-self:
        :add/@class-name}"/>
    </query>
    <!-- NOTE: depends on CN being the naming attribute -->
    <search-attr attr-name="CN">
      <value><xsl:value-of select="$object-name"/>
    </value>
    </search-attr>
    <!-- put an empty read attribute in so that we don't get -->
    <!-- the whole object back -->
    <read-attr/>
  </query>
</xsl:variable>
  <!-- query NDS -->
  <xsl:variable name="result" select="query:query($destQuery
    Processor,$query)"/>
  <!-- return an empty or non-empty result tree fragment -->
  <!-- depending on result of query -->
  <xsl:value-of select="$result//instance"/>
</xsl:template>

```

Here is another example.

```

<?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>

```

## 4.5 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 that 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>

```

## 4.6 Creating a Password Example: Creation Policy

The following style sheet can be used for a Creation policy. It creates a user, generates a password for the user from the user's Surname and CN attributes, and performs an identity transformation (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>
```

## 4.7 Creating an eDirectory User Example: Creation Policy

This style sheet can be used for a Creation policy. 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 Identity Manager so
things
  are pretty to look at -->
<xsl:strip-space elements="*" />
<xsl:preserve-space elements="value,component" />
<xsl:output method="xml" indent="yes" />

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

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

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

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

      <!-- call a template to construct the object name and place the
```

```

result in a variable -->
    <xsl:variable name="object-name">
        <xsl:call-template name="create-object-name"/>
    </xsl:variable>

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

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

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

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

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

<!-- get-dn-prefix places the part of the passed dn that precedes the
-->
<!-- last 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>

<!-- 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: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>
```

# Managing Filters

The Filter Editor allows you to manage the filter. In the Filter Editor, you define how each class and attribute is to be handled by the Publisher and Subscriber channels.

This section covers the following filter-related topics:

- [Section 5.1, “Filter Tasks in Designer,” on page 309](#)
- [Section 5.2, “Filter Tasks in iManager,” on page 321](#)


## 5.1 Filter Tasks in Designer

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

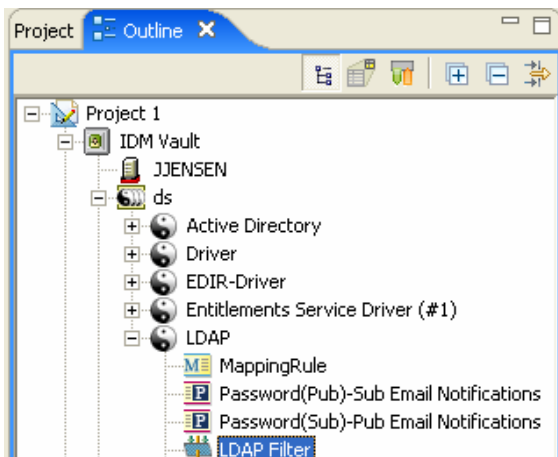
- [Section 5.1.1, “Accessing the Filter Editor,” on page 309](#)
- [Section 5.1.2, “Editing the Filter,” on page 310](#)
- [Section 5.1.3, “Testing Filters,” on page 315](#)
- [Section 5.1.4, “Viewing the Filter XML Source,” on page 319](#)

### 5.1.1 Accessing the Filter Editor

The Filter Editor allows you to edit the filter. There are two different ways to access the filter. To access the Filter Editor from within a project:


- 1 In an open project, click the Outline tab.
- 2 Click the Model Outline icon. 
- 3 Select the driver you want to manage the filter for, then click the plus sign to the right.
- 4 Double-click the Filter icon and to launch the Filter Editor.

**Figure 5-1** Outline Access

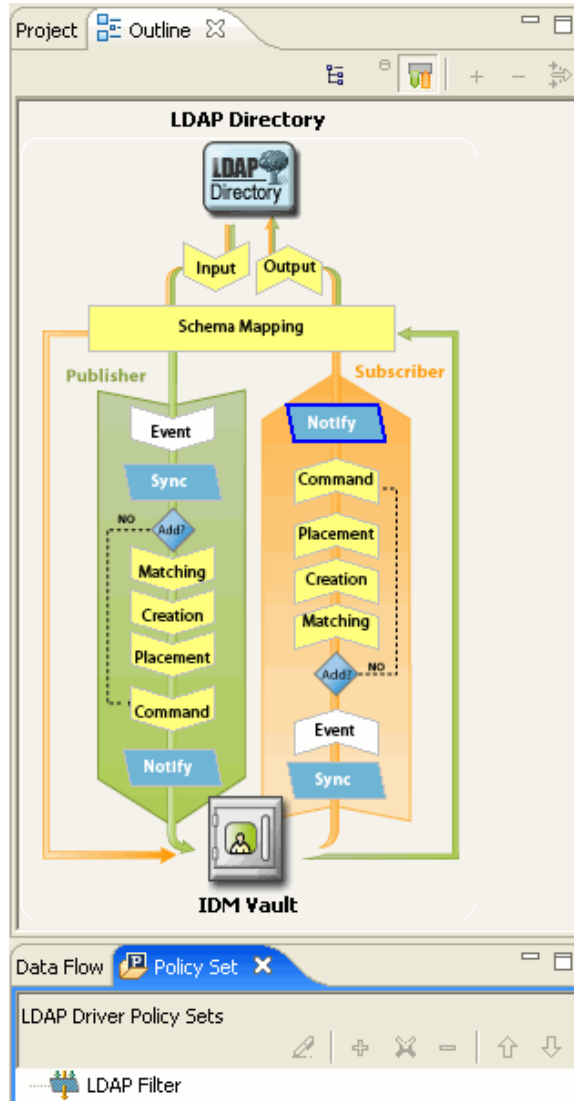


To access the Filter Editor through the Policy Flow:

- 1 In Designer, open a project, then click the Outline tab.

- 2 Select the Policy Flow icon. 
  - 3 Select the filter, which is represented by the Sync or Notify icons.
  - 4 Double-click the filter as it appears in the Policy Set Manager below the Policy Flow to launch the Filter Editor.
- Or
- Double-click the Sync or Notify icons to launch the Filter Editor.

**Figure 5-2** Policy Flow Access



## 5.1.2 Editing the Filter

The Filter Editor allows you to create and edit the filter. To display a context menu, right-click an item.

## Removing or Adding Classes and Attributes


By removing or adding classes and attributes, you determine what objects synchronize between the connected data store and the Identity Vault.

### Removing a Class or Attribute


If you do not want a class or an attribute to synchronize, the best practice is to remove the class or the attribute completely from the filter. There are two different ways to add or remove attributes and classes from the filter:

- Right-click the class or attribute you want to remove, then select *Delete*.
- Select the class or attribute you want to remove, then click the *Delete* icon in the upper-right corner.

### Adding a Class


- 1 Right-click in the Filter Editor, then click *Add class*.  
Or  
Click the class icon  in the upper-right corner
- 2 Browse and select the class you want to add, then click *OK*.
- 3 Change the options to synchronize the information.
- 4 To save the changes, click *File > Save*.

### Adding an Attribute

- 1 Right-click in the Filter Editor, then click *Add attribute*.  
Or  
Click the attribute icon  in the upper-right corner.
- 2 Browse and select the attribute you want to add, then click *OK*.
- 3 Change the options to synchronize the information.
- 4 To save the changes, click *File > Save*.

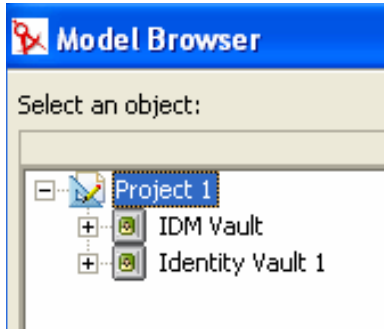
## Copying an Existing Filter

You can copy an existing filter from another driver and use it in the driver you are currently working with.


- 1 Click the *Copy an existing filter icon*   
Or  
Right-click in the Filter Editor, then click *Copy an existing filter*.
- 2 Browse to and select the filter object you want to copy, then click *OK*.

If you have more than one Identity Vault in your project, you can copy filters from the other Identity Vaults. When you are browsing to select the other object, you can browse to the other Identity Vault and use a filter stored there.

**Figure 5-3** Multiple Identity Vaults




## Refreshing the Application Schema

If you have modified the schema in the connected application, these changes need to be reflected in the Filter. To make the new schema available, click the *Refresh application schema* icon  in the toolbar.

When you create a new class or attribute mapping, you can see the new schema in the drop-down list for the connected application.

## Setting Default Values for Attributes

You can define the default values for new attributes when they are added to the Filter.

- 1 Click the *Set default values for new attributes* icon  in the upper-right corner.
- 2 Select the options you want new attributes to have, then click *OK*.

## Modifying the Filter

The Filter Editor gives you the option of modifying how information is synchronized between the Identity Vault and the connected system. The Filter has different options for classes and attributes.

### Class Options

Options	Definitions
Publisher	<ul style="list-style-type: none"> <li>• Synchronize - Allows the class to synchronize from the connected system into the Identity Vault.</li> <li>• Ignore - Does not synchronize the class from the connected system into the Identity Vault.</li> </ul>
Subscriber	<ul style="list-style-type: none"> <li>• Synchronize - Allows the class to synchronize from the Identity Vault into the connected system.</li> <li>• Ignore - Does not synchronize the class from the Identity Vault into the connected system.</li> </ul>
Create Home Directory	<ul style="list-style-type: none"> <li>• Yes - Automatically creates home directories.</li> <li>• No - Does not create home directories.</li> </ul>



---

**Options****Definitions**

---

Track Member of Template

- Yes - Determines whether or not the Publisher channel maintains the Member of Template attribute when it creates objects from a template.
  - No - Does not track the Member of Template attribute.
- 

## Attribute Options

---

**Options****Definitions**

---

Publisher

- Synchronize - Changes to this object are reported and automatically synchronized.
- Ignore - Changes to this object are not reported nor automatically synchronized.
- Notify - Changes to this object are reported, but not automatically synchronized.
- Reset - Resets the object value to the value specified by the opposite channel. (You can set this value on either the Publisher or Subscriber channel, not both.)

Subscriber

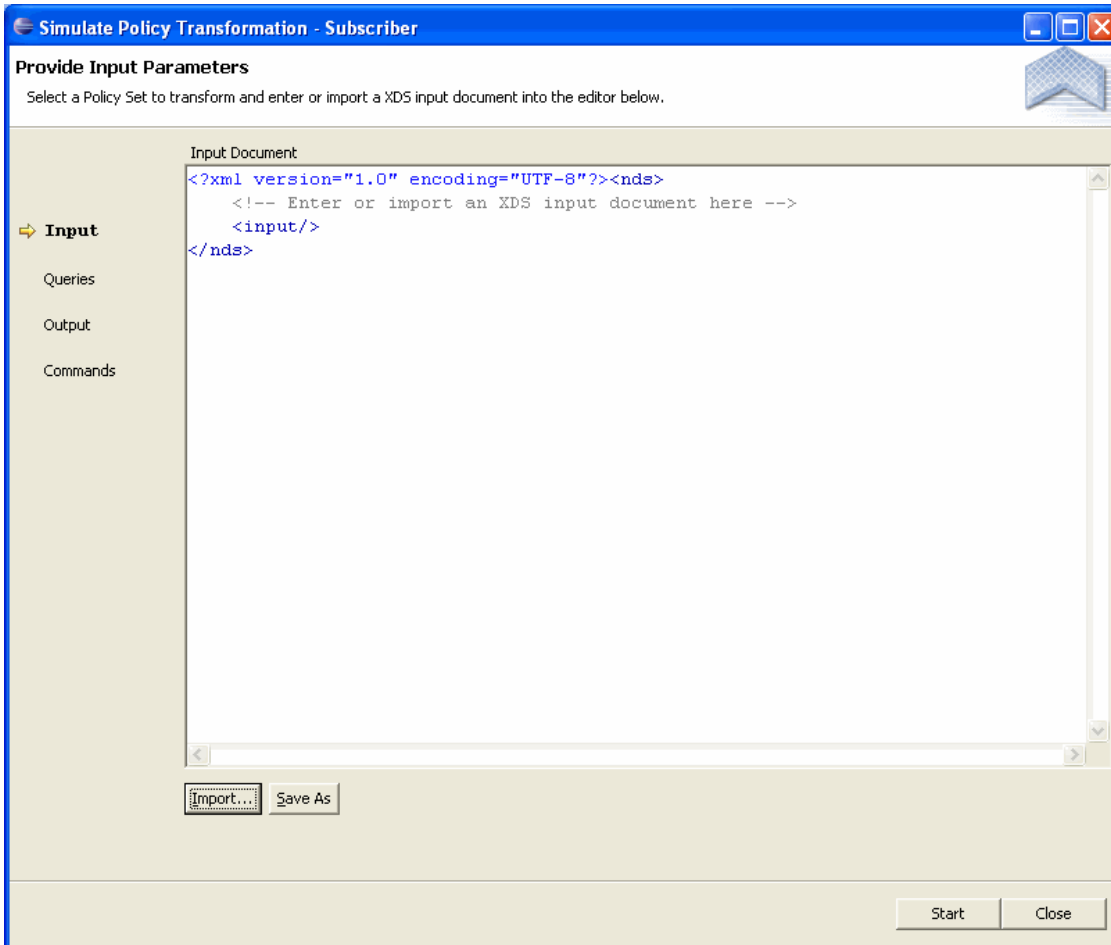
- Synchronize - Changes to this object are reported and automatically synchronized.
  - Ignore - Changes to this object are not reported nor automatically synchronized.
  - Notify - Changes to this object are reported, but not automatically synchronized.
  - Reset - Resets the object value to the value specified by the opposite channel. (You can set this value on either the Publisher or Subscriber channel, not both.)
-

Options	Definitions
Merge Authority	<ul style="list-style-type: none"> <li data-bbox="863 213 1376 298">• Default Behavior - If an attribute is not being synchronized in either channel, no merging occurs.</li> </ul> <p data-bbox="887 324 1376 556">If an attribute is being synchronized in one channel and not the other, then all existing values on the destination for that channel are removed and replaced with the values from the source for that channel. If the source has multiple values and the destination can only accommodate a single value, then only one of the values is used on the destination side.</p> <p data-bbox="887 582 1376 794">If an attribute is being synchronized in both channels and both sides can accommodate only a single value, the connected application acquires the Identity Vault values unless there is no value in the Identity Vault. If this is the case, the Identity Vault acquires the values from the connected application (if any).</p> <p data-bbox="887 820 1376 1012">If an attribute is being synchronized in both channels and only one side can accommodate multiple values, the single-valued side's value is added to the multi-valued side if it is not already there. If there is no value on the single side, you can choose the value to add to the single side.</p> <p data-bbox="887 1038 1198 1064">This is always valid behavior.</p> <ul style="list-style-type: none"> <li data-bbox="863 1090 1376 1197">• Identity Vault - Behaves the same way as the default behavior if the attribute is being synchronized on the Subscriber channel and not on the Publisher channel.</li> </ul> <p data-bbox="887 1223 1376 1268">This is valid behavior when synchronizing on the Subscriber channel.</p> <ul style="list-style-type: none"> <li data-bbox="863 1294 1376 1401">• Application - Behaves the same as the default behavior if the attribute is being synchronized on the Publisher channel and not on the Subscriber channel.</li> </ul> <p data-bbox="887 1427 1376 1471">This is valid behavior when synchronizing on the Publisher channel.</p> <ul style="list-style-type: none"> <li data-bbox="863 1497 1376 1542">• None - No merging occurs regardless of synchronization.</li> </ul>
Optimize Modification to Identity Manager	<ul style="list-style-type: none"> <li data-bbox="863 1560 1376 1645">• Yes - Changes to this attribute are examined on the Publisher channel to determine the minimal change made in the Identity Vault.</li> <li data-bbox="863 1671 1376 1687">• No - Changes are not examined.</li> </ul>

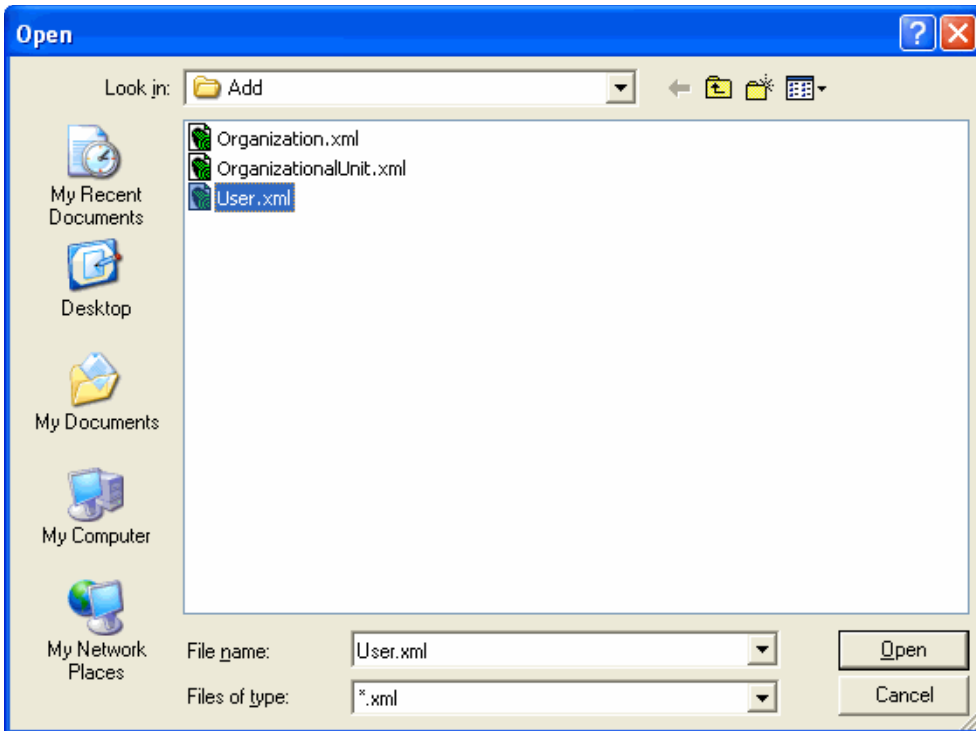
## 5.1.3 Testing Filters

Designer comes with a new tool called the Policy Simulator. It allows you to test your policies and filters before deploying them. You can launch the Policy Simulator through the Filter Editor to test your Filter after you have modified it. Follow the steps listed below to access the Policy Simulator and to test the Filter:

- 1 To access the Policy Simulator, click the Launch Policy Simulator icon  in the toolbar.

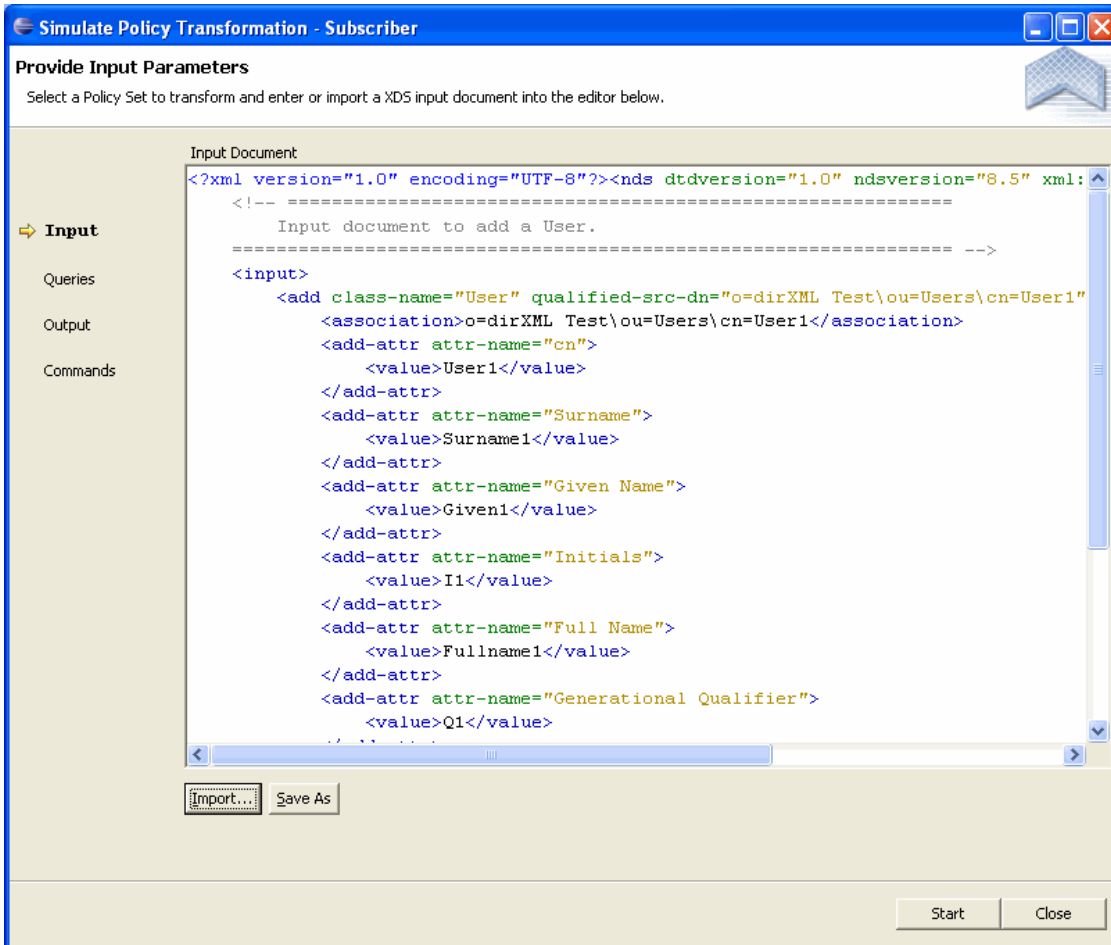


- 2 Select Import to browse to a file that simulates an event, then click Open. This example uses the `\simulation\add\User.xml` file, which simulates an Add event for a user object.

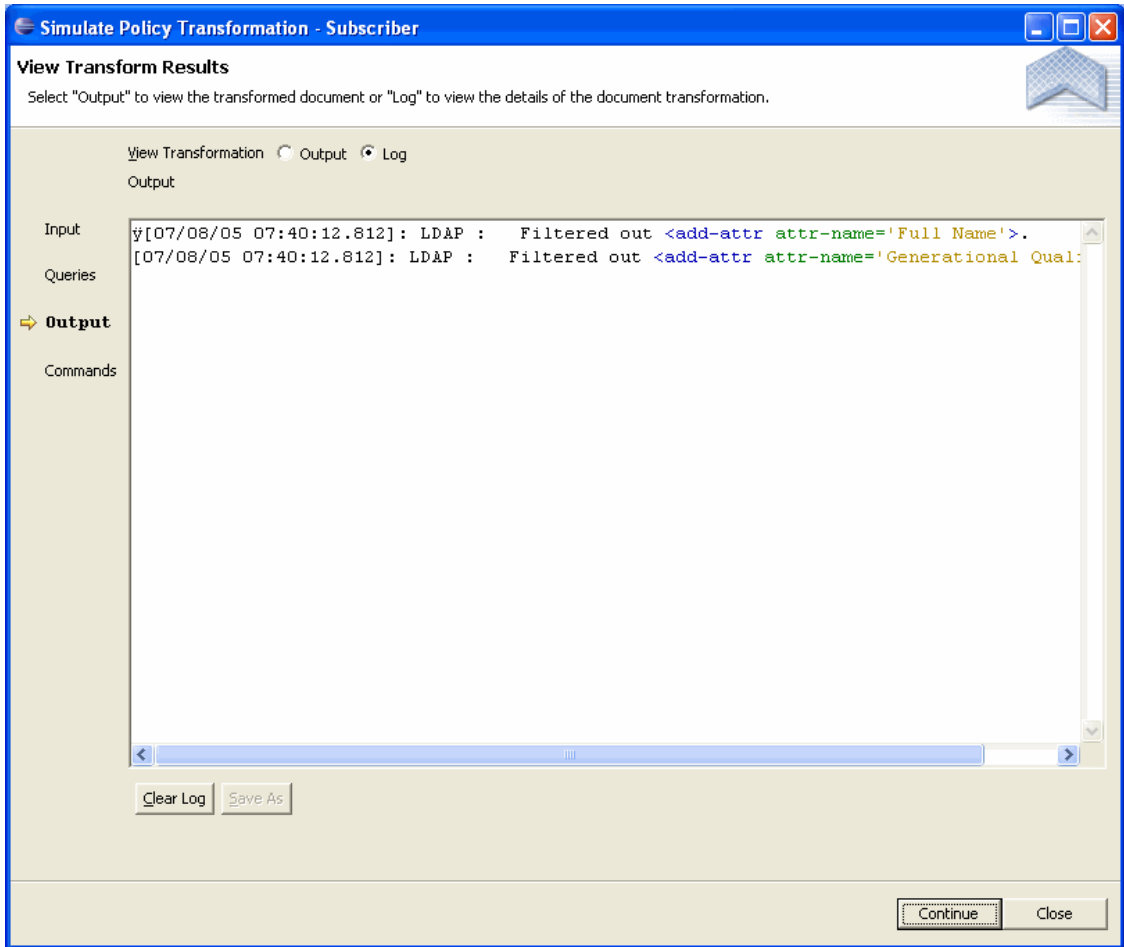


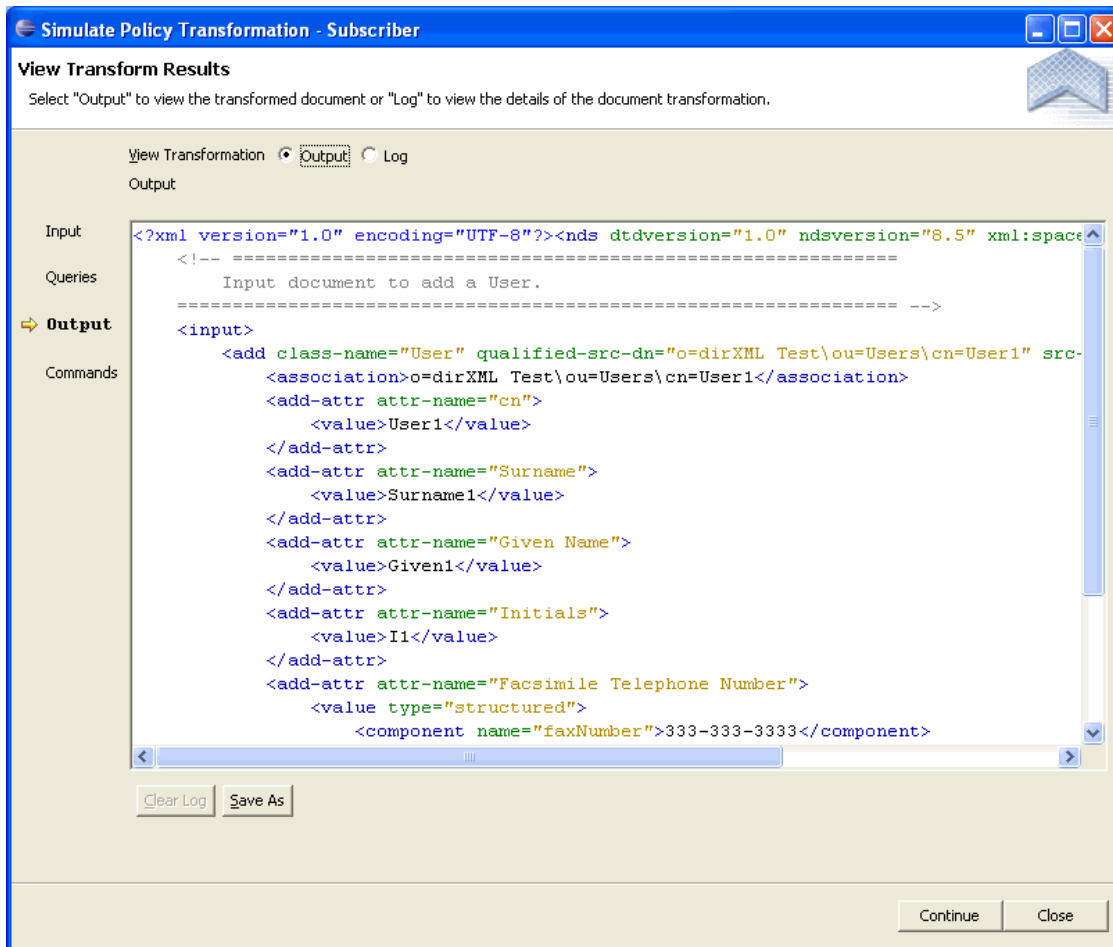
The Policy Simulator displays the input document of the user Add event.

- 3 Click Start to begin the simulation.



The Policy Simulator displays the log of the Add event as well as the output document. With the radio button set to Log, you see the results of the Add event as you would through DSTRACE. With the radio button set to Output, the Policy Simulator displays the output document.





- 4 Click Continue to select a different input document and see the results of that event.
- 5 When you have finished testing the filter, click Close to close the Policy Simulator.

---

**NOTE:** You can edit the input and output documents. If you want to keep the changes, click the Save As icon.

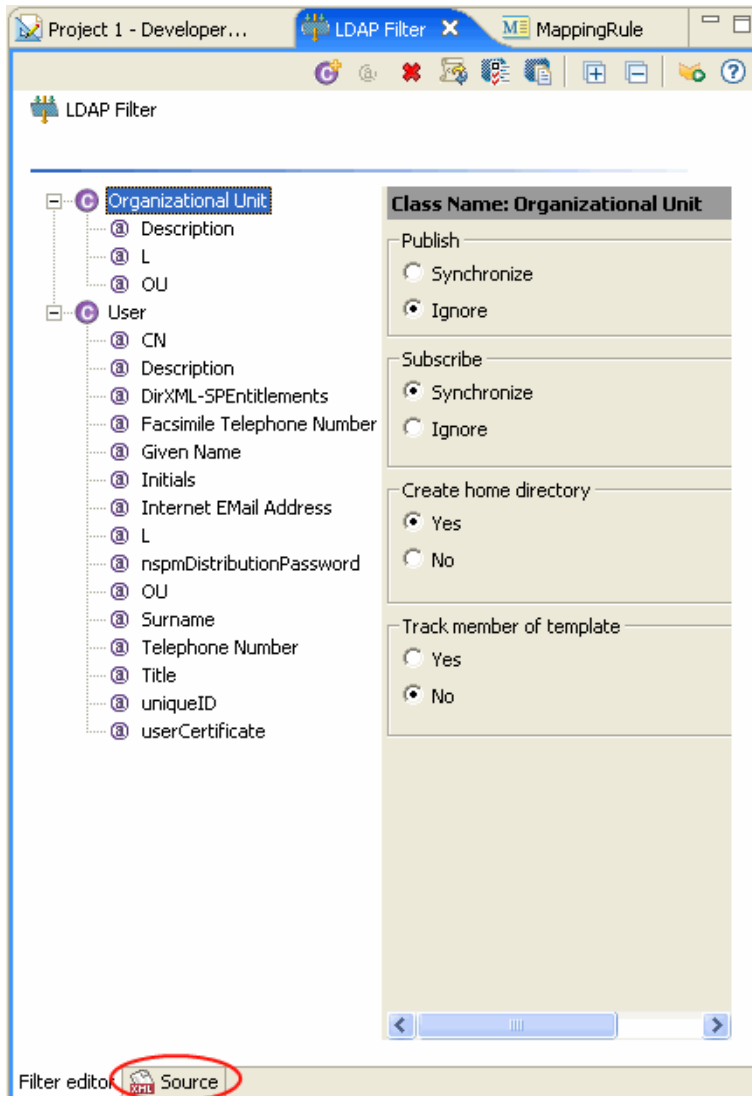
---

### 5.1.4 Viewing the Filter XML Source

The Source view enables you to view and edit the XML by using an XML editor or text editor. The default editor that is loaded is associated to .xml file types. If a default editor can't be found, the system text editor is loaded. The functionality and operations of the Source view are based on the editor that loads.

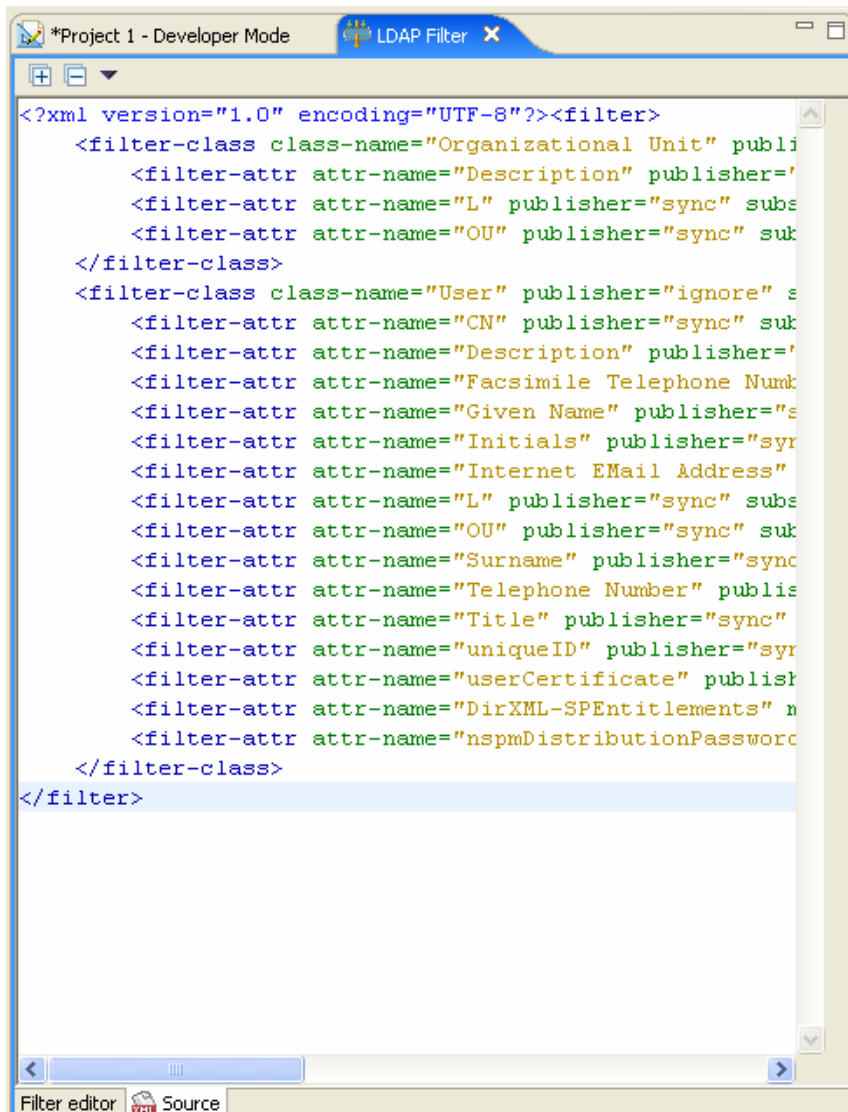
For more information about the XML source see [“Editing the DirXML Script” on page 97](#).

To open the Source view, select Source, at the bottom of the Filter Editor’s workspace.



You can edit the XML through the XML editor. You can make changes here as well as through the GUI interface.





To choose a different XML editor for your source view:

- 1 From the Main menu, click *Window > Preferences*.
- 2 Click *General > Editor > File Associations*.
- 3 Select *\*.xml* from the list of file types.
- 4 Select the editor you want (for example, Novell XML Editor) in the Associated editors pane. (If the editor you want isn't in the list, you can click *Add*, then add it to the list.)
- 5 Click *OK*.
- 6 Close and reopen the Filter Editor. The default editor should be loaded in the Source view.

## 5.2 Filter Tasks in iManager

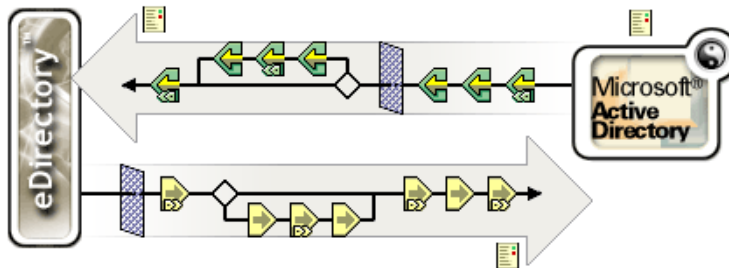
This section contains instructions on performing common filter-related tasks in iManager:

- [Section 5.2.1, “Accessing the Filter,” on page 322](#)
- [Section 5.2.2, “Editing the Filter,” on page 322](#)

## 5.2.1 Accessing the Filter

- 1 In iManager, expand the *Identity Manager Role*, then click *Identity Manager Overview*.
- 2 Select *Search entire tree* or *Search in container*, then click *Search*.
- 3 Click the driver for which you want to access the filter. The Identity Manager Driver Overview opens:

**Figure 5-4** Driver Overview



- 4 Click the Filter icon on the Publisher or Subscriber channel. It is the same object.



## 5.2.2 Editing the Filter

The Filter Editor gives you the options of editing how information is synchronized between the Identity Vault and the connected system. Here is a list of most common tasks when editing the filter:

- [“Removing a Class or an Attribute from the Filter” on page 322](#)
- [“Adding a Class” on page 322](#)
- [“Adding an Attribute” on page 322](#)
- [“Copying a Filter” on page 323](#)
- [“Setting a Template” on page 323](#)

### Removing a Class or an Attribute from the Filter

- 1 Select the class or attribute, then click *Delete*.

### Adding a Class

- 1 Click *Add Class*.
- 2 Change the options to synchronize the information.
- 3 Click *Apply*.

### Adding an Attribute

- 1 Click *Add Attribute*.
- 2 Change the option to synchronize the information.
- 3 Click *Apply*.

## Copying a Filter

Allows you to copy the filter from an existing driver into the driver you are currently working on.

- 1 Click *Copy Filter From*.
- 2 Browse to the driver you want to copy the filter from, then click *OK*.

## Setting a Template

Allows you to set the default values for an attribute you add to the filter.

- 1 Click *Set Template*.
- 2 Select options you would like new attributes to have, then click *OK*.

You can change the values of the attributes after they have been created.

## Class Options

---

Options	Definitions
Publisher	<ul style="list-style-type: none"><li>• Synchronize - Allows the class to synchronize from the connected system into Identity Vault.</li><li>• Ignore - Does not synchronize the class from the connected system into the Identity Vault.</li></ul>
Subscriber	<ul style="list-style-type: none"><li>• Synchronize - Allows the class to synchronize from Identity Vault into the connected system.</li><li>• Ignore - Does not synchronize the class from the Identity Vault into the connected system.</li></ul>
Create Home Directory	<ul style="list-style-type: none"><li>• Yes - Automatically creates home directories.</li><li>• No - Does not allow for the creation of home directories.</li></ul>
Track Member of Template	<ul style="list-style-type: none"><li>• Yes - Determines whether or not the Publisher channel maintains the Member of Template attribute when it creates objects from a template.</li><li>• No - Does not track the Member of Template attribute.</li></ul>

---

## Attribute Options

---

Options	Definitions
Publisher	<ul style="list-style-type: none"><li>• Synchronize - Changes to this object are reported and automatically synchronized.</li><li>• Ignore - Changes to this object are not reported and they are not automatically synchronized.</li><li>• Notify - Changes to this object are reported, but not automatically synchronized.</li><li>• Reset - Resets the object value to the value specified by the opposite channel. (You can set this value on either the Publisher or Subscriber channel, not both.)</li></ul>
Subscriber	<ul style="list-style-type: none"><li>• Synchronize - Changes to this object are reported and automatically synchronized.</li><li>• Ignore - Changes to this object are not reported and are not automatically synchronized.</li><li>• Notify - Changes to this object are reported, but not automatically synchronized.</li><li>• Reset - Resets the object value to the value specified by the opposite channel. (You can set this value on either the Publisher or Subscriber channel, not both.)</li></ul>

---

Options	Definitions
Merge Authority	<ul style="list-style-type: none"> <li>• Default Behavior - If an attribute is not being synchronized in either channel, no merging occurs.</li> </ul> <p>If an attribute is being synchronized in one channel and not the other, then all existing values on the destination for that channel are removed and replaced with the values from the source for that channel. If the source has multiple values and the destination can only accommodate a single value, then only one of the values is used on the destination side.</p> <p>If an attribute is being synchronized in both channels and both sides can accommodate only a single value, the connected application acquires the Identity Vault values unless there is no value in the Identity Vault. If this is the case, the Identity Vault acquires the values from the connected application (if any).</p> <p>If an attribute is being synchronized in both channels and only one side can accommodate multiple values, the single-valued side's value is added to the multi-valued side if it is not already there. If there is no value on the single side, you can choose the value to add to the single side.</p> <p>This is always valid behavior.</p> <ul style="list-style-type: none"> <li>• Identity Vault - Behaves the same way as the default behavior if the attribute is being synchronized on the Subscriber channel and not on the Publisher channel.</li> </ul> <p>This is valid behavior when synchronizing on the Subscriber channel.</p> <ul style="list-style-type: none"> <li>• Application - Behaves the same as the default behavior if the attribute is being synchronized on the Publisher channel and not on the Subscriber channel.</li> </ul> <p>This is valid behavior when synchronizing on the Publisher channel.</p> <ul style="list-style-type: none"> <li>• None - No merging occurs regardless of synchronization.</li> </ul>
Optimize Modification to Identity Manager	<ul style="list-style-type: none"> <li>• Yes - Changes to this attribute are examined on the Publisher channel to determine the minimal change made in the Identity Vault.</li> <li>• No - Changes are not examined.</li> </ul>



# Managing Schema Mapping Policies

Schema Mapping policies map class names and attribute names between the Identity Vault namespace and the application namespace. The same schema mapping policy is applied in both directions. All documents that are passed in either direction on either channel between the Metadirectory engine and the application shim are passed through the Schema Mapping policy.

There is one Schema Mapping policy per driver.

This section covers the following filter-related topics:

- [Section 6.1, “Schema Mapping Policy Tasks in Designer,” on page 327](#)
- [Section 6.2, “Schema Mapping Policy Tasks in iManager,” on page 338](#)

## 6.1 Schema Mapping Policy Tasks in Designer


This section contains instructions on performing common tasks related to Schema Mapping policies in Designer:

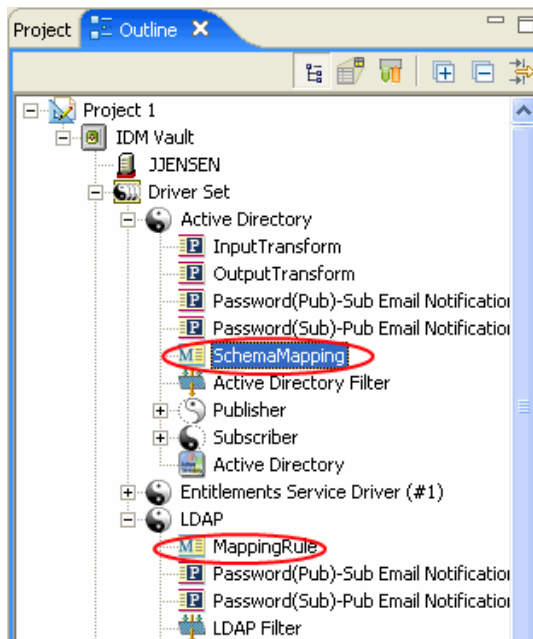
- [Section 6.1.1, “Accessing the Schema Map Editor,” on page 327](#)
- [Section 6.1.2, “Editing a Schema Mapping Policy,” on page 329](#)
- [Section 6.1.3, “Testing Schema Mapping Policies,” on page 331](#)
- [Section 6.1.4, “Viewing the Schema Mapping Policy XML Source,” on page 336](#)

### 6.1.1 Accessing the Schema Map Editor


The Schema Map Editor allows you to edit the schema mapping policies. There are two different ways to access the Schema Map Editor in Designer.

To access the Schema Map Editor through the Model Outline:

- 1 In an open project, click the Outline tab.
- 2 Click the Model Outline icon. 
- 3 Select the driver you want to manage the schema mapping policy on, and click the plus sign to the right.
- 4 Double-click the Schema Map icon to launch the Schema Map Editor.



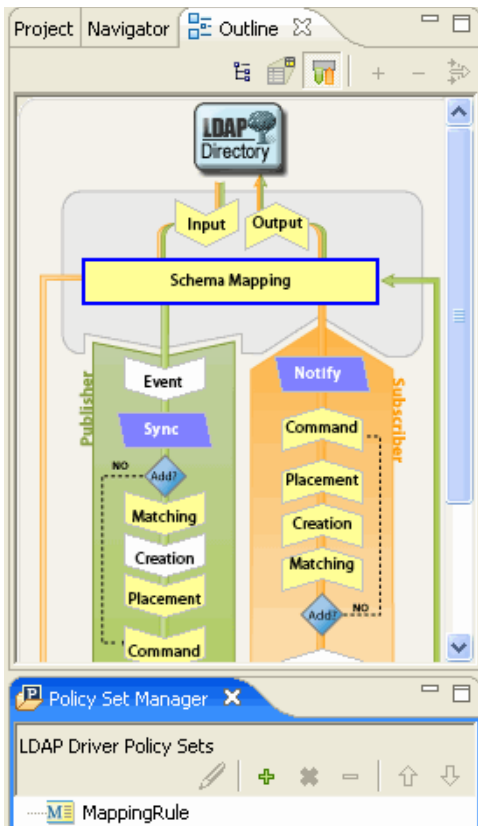
To access the Schema Map Editor through the Policy Flow:

- 1 In an open project, click the Outline tab.
- 2 Click the Policy Flow icon. 
- 3 Double-click the Schema Mapping policy as it appears in the Policy Set Manager below the Policy Flow to launch the Schema Map Editor.

Or

Double-click the Schema Mapping icon to launch the Schema Map Editor.






## 6.1.2 Editing a Schema Mapping Policy

The Schema Map Editor allows you to create and edit schema mapping policies. To display a context menu, right-click an item.

- “Removing Classes and Attributes” on page 329
- “Adding a Class” on page 330
- “Adding a Attribute” on page 330
- “Refreshing the Application Schema” on page 330
- “Editing Items” on page 331
- “Sorting Items” on page 331

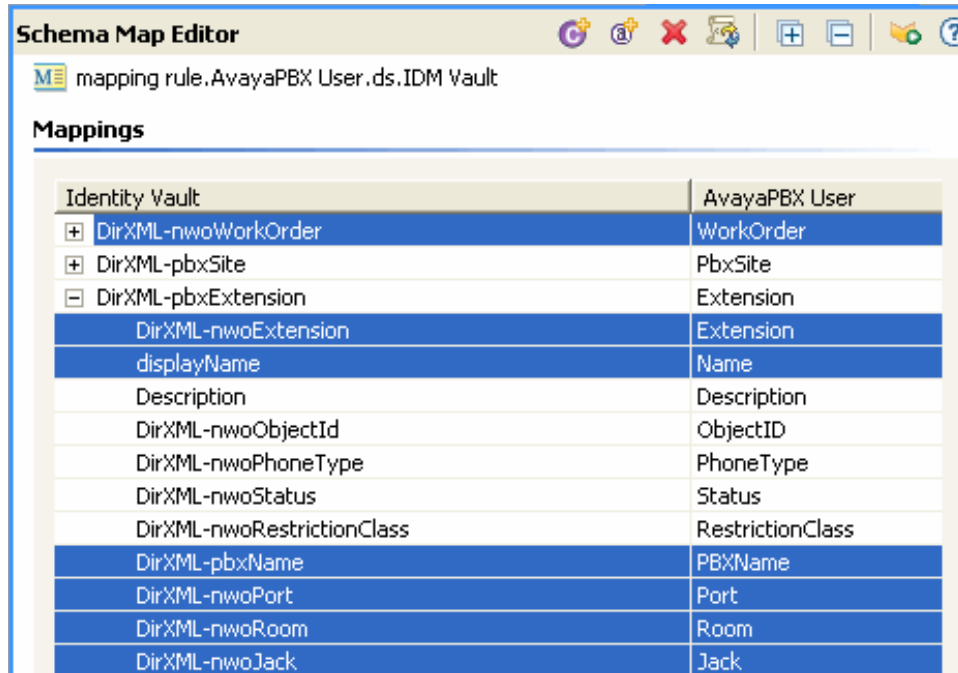
### Removing Classes and Attributes

If you do not want a class or an attribute to be mapped to a class or attribute in the connected system, the best practice is to completely remove the class or the attribute from the schema mapping policy. There are two different ways to add or remove attributes and classes from the schema mapping policy:


- Select the class or attribute you want to remove, then right-click in the pane and click *Delete*.
- Select the class or attribute you want to remove, then click the *Delete* icon  in the upper-right corner.
- Select the class or attribute you want to remove, then press the Delete key.

You can select multiple classes or attributes to delete at the same time.


- 1 Press ctrl and select each item with the mouse.
- 2 Press the Delete key and the items are deleted.




### Adding a Class

- 1 Right-click in the Schema Map Editor, then click *Add class mapping*.
- Or
- Select the *Add class mapping* icon  in the upper-right corner.
- 2 From the drop-down list for the Identity Vault, select the class you want to add.
  - 3 From the drop-down list for the connected system, select the class you want to add.
  - 4 To save the changes, click *File > Save*.

### Adding a Attribute

- 1 Right click in the Schema Map Editor, then click *Add attribute mapping*.
- 2 Or select the *Add attribute mapping* icon  in the upper-right corner.
- 3 From the drop-down list for the Identity Vault, select the attribute you want to add.
- 4 From the drop-down list for the connected system, select the attribute you want to add.
- 5 To save the changes, click *File > Save*.

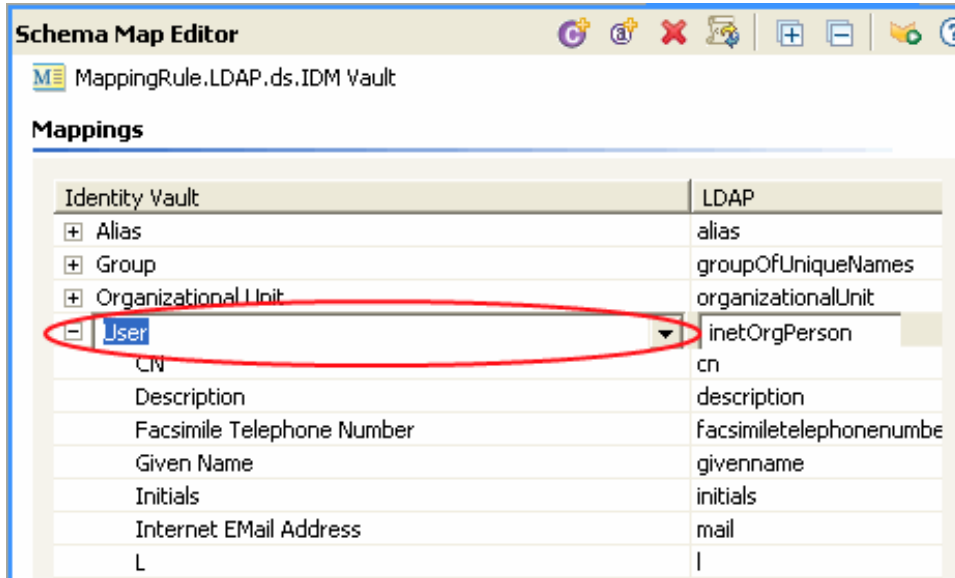
### Refreshing the Application Schema

If you have modified the schema in the connected application, these changes need to be reflected in the Schema Mapping policy. To make the new schema available, click the *Refresh application schema* icon  in the toolbar.

When you create a new class or attribute mapping, you can see the new schema in the drop-down list for the connected application.

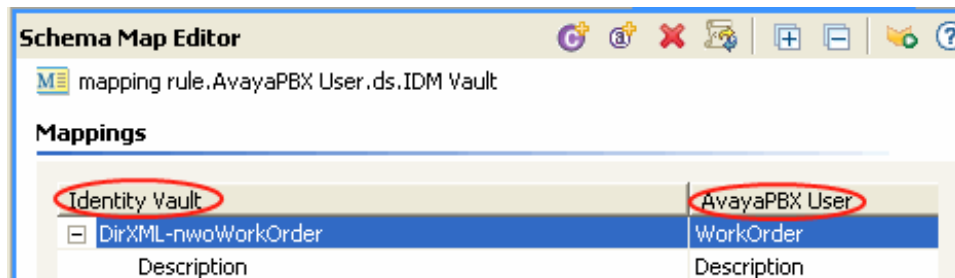
## Editing Items

To edit a mapping, double-click the selected row. An in-place editor appears allowing you to edit the mapping.



## Sorting Items


The Schema editor allows you to sort the items in ascending order based on either Identity Manager or the connected system. To sort, click the header of either column.

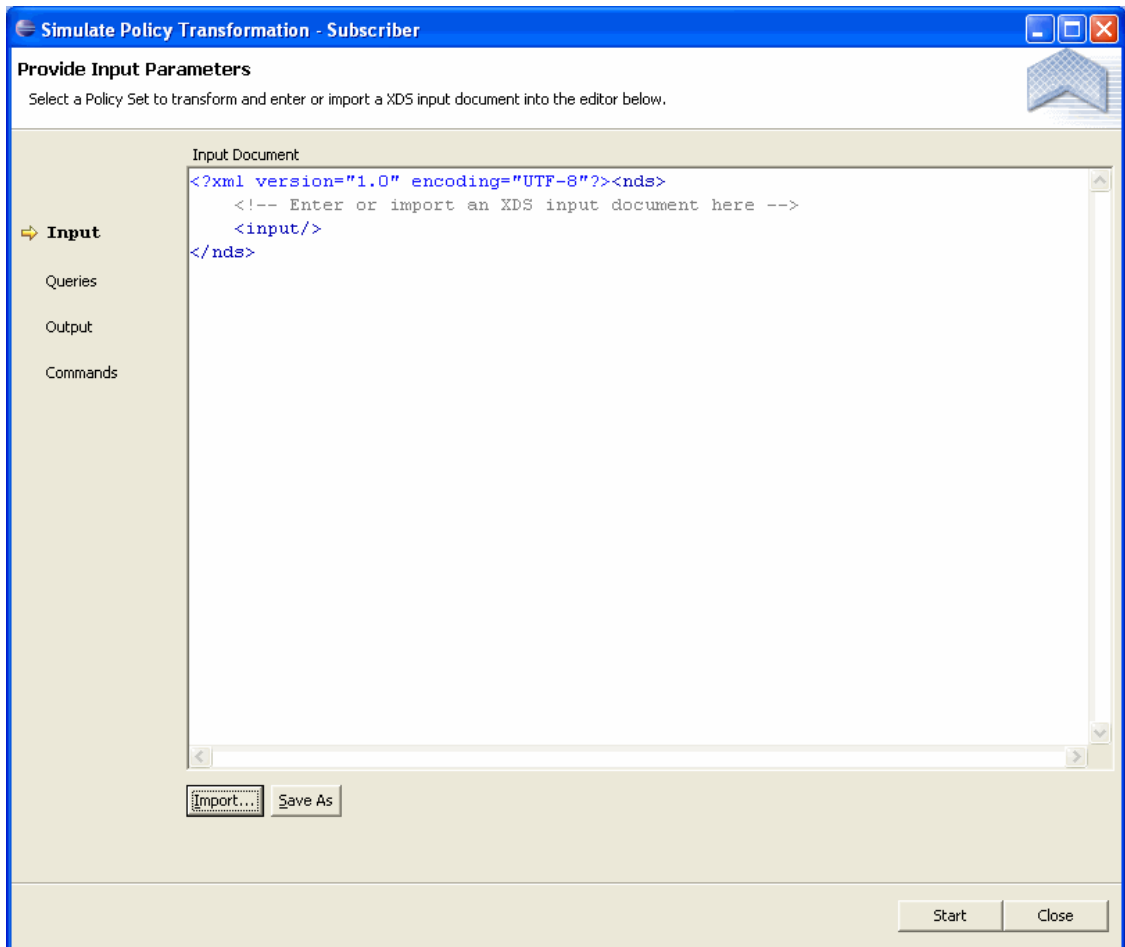


## 6.1.3 Testing Schema Mapping Policies

Designer comes with a new tool called the Policy Simulator. It allows you to test your policies before deploying them. You can launch the Policy Simulator through the Schema Mapping Editor to test your policy after you have modified it.

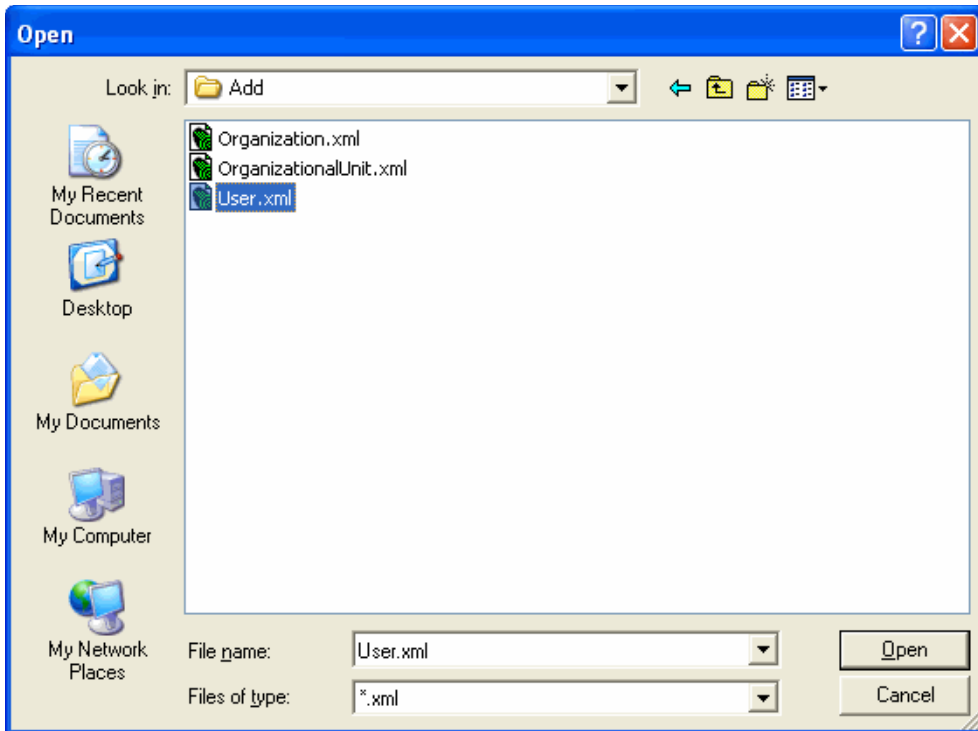
To access the Policy Simulator and test the Schema Mapping policy:

- 1 To access the Policy Simulator, click the Launch Policy Simulator icon  in the toolbar.



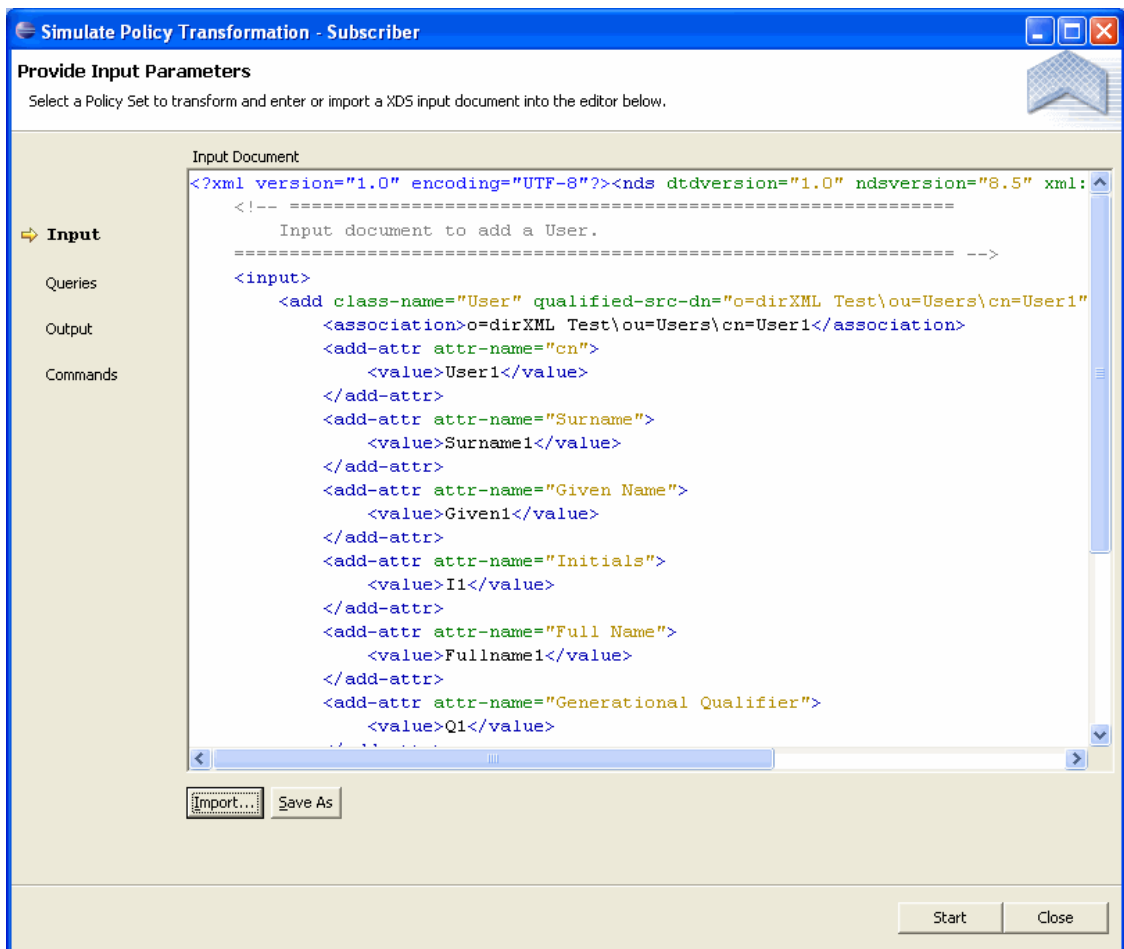
2 Select *Import* to browse to a file that simulates an event, then click *Open*.

This example uses the `\simulation\add\user.xml` file, which simulates an Add event of a user object.

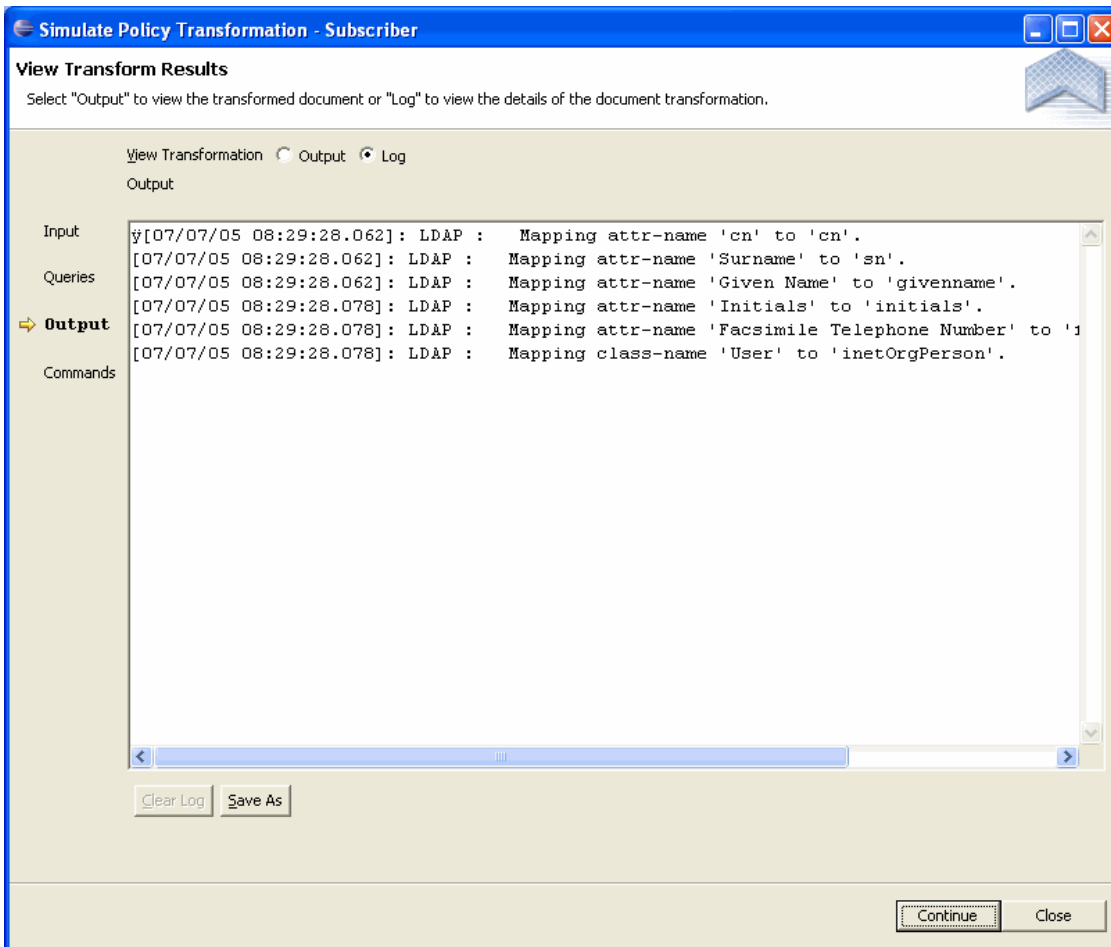


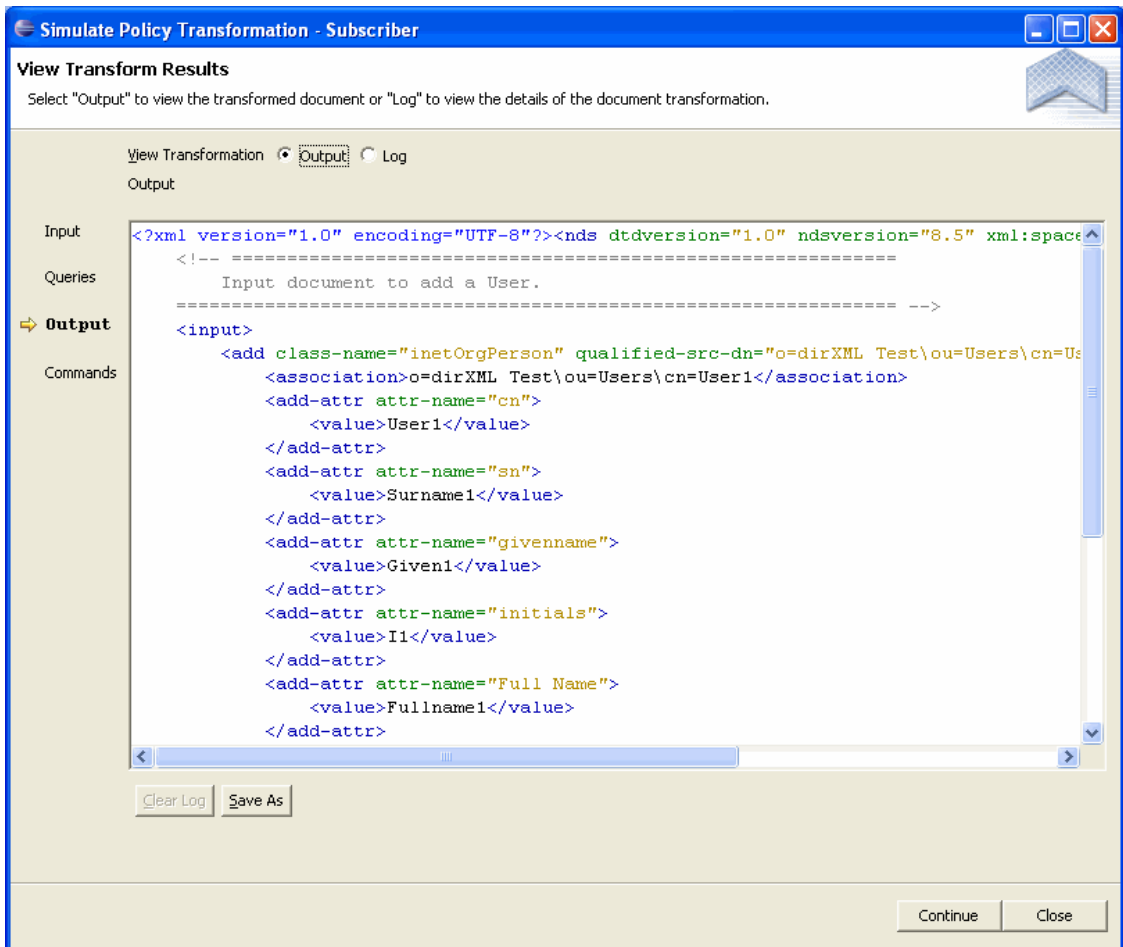
The Policy Simulator displays the input document of the user Add event.

- 3 Click *Start* to begin the simulation.



The Policy Simulator displays the log of the Add event as well as the output document. With the radio button set to Log, you see the results of the Add event as you would through DSTRACE. With the radio button set to Output, the Policy Simulator displays the output document.





- 4 Click *Continue* to select a different input document and see the results of that event.
- 5 When you have finished testing the Schema Mapping Policy, click *Close* to close the Policy Simulator.

You can edit the input and output documents. If you want to keep the changes, click the *Save As* icon.

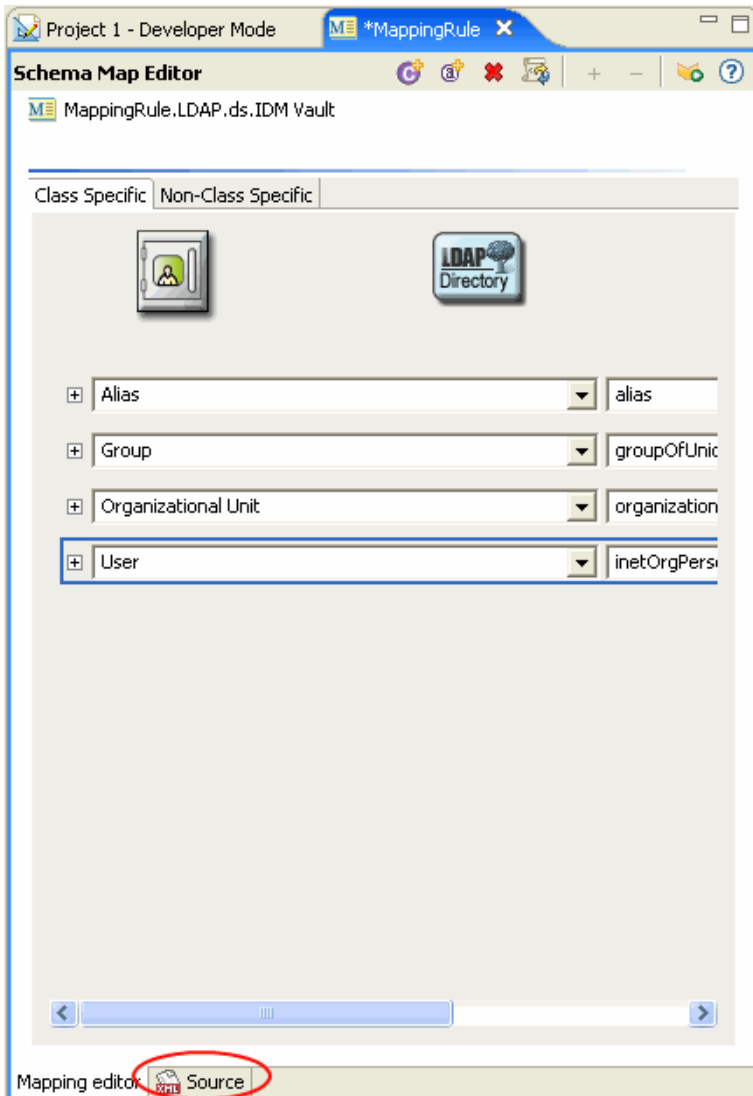
### 6.1.4 Viewing the Schema Mapping Policy XML Source

The Source view enables you to view and edit the XML by using an XML editor or text editor. The default editor that is loaded is associated to .xml file types. If a default editor can't be found, the system text editor is loaded. The functionality and operations of the Source view are based on the editor that loads.

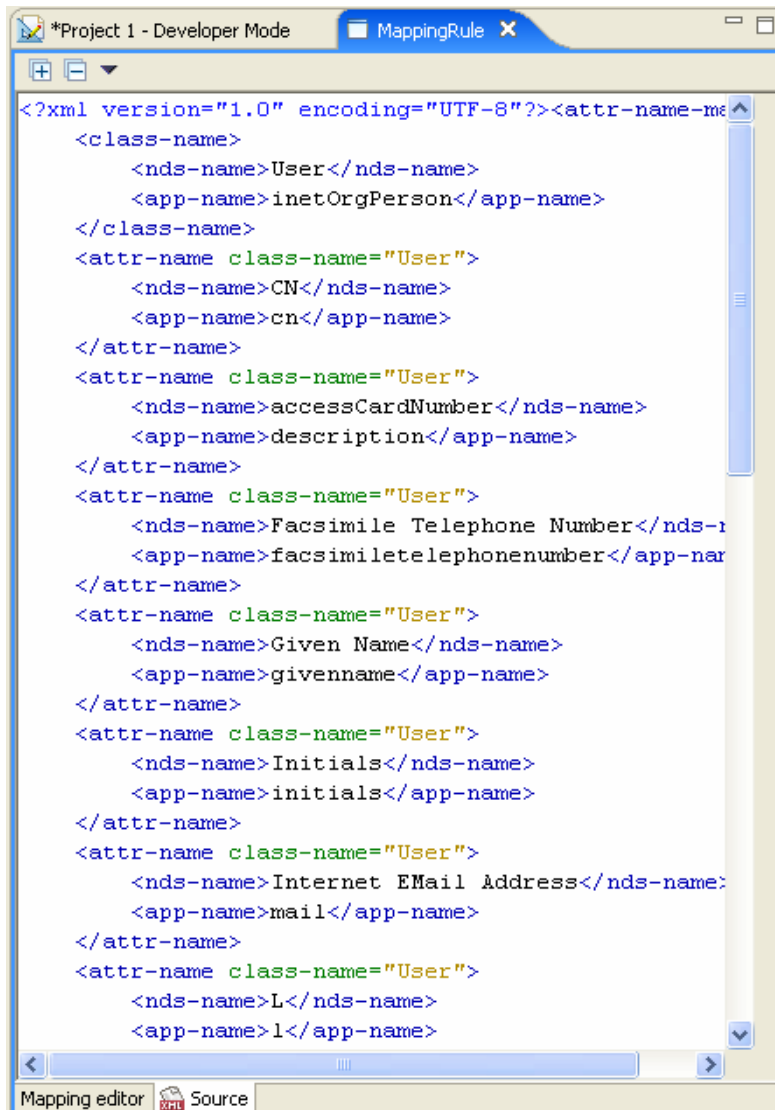
For more information about the XML source see [“Editing the DirXML Script” on page 97](#).

To open the Source view, select Source at the bottom of the Schema Map Editor's workspace.





You can edit the XML through the XML editor. You can make changes here as well as through the GUI interface.



To choose a different XML editor for your source view:

- 1 From the Main menu, click *Window > Preferences*.
- 2 Click *General > Editor > File Associations*.
- 3 Select *\*.xml* from the list of file types.
- 4 Select the editor you want (for example, Novell XML Editor) in the Associated Editors pane. (If the editor you want isn't in the list, you can click Add, then add it to the list.)
- 5 Click *OK*.
- 6 Close and reopen the Schema Map Editor. The default editor should be loaded in the Source view.

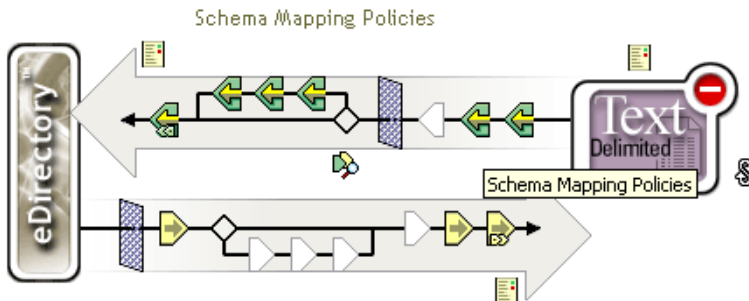
## 6.2 Schema Mapping Policy Tasks in iManager

This section contains instructions on performing common tasks related to Schema Mapping policies in iManager:

- Accessing Schema Mapping Policies
- Editing Schema Mapping Policies

## 6.2.1 Accessing Schema Mapping Policies

- 1 In iManager, expand the *Identity Management Role*, then click *Identity Manager Overview*.
- 2 Select *Search entire tree* or *Search in container* for a Driver set, then click *Search*.
- 3 Click the driver you want to manage the Schema Mapping Policy. The Identity Manager Driver Overview page opens.



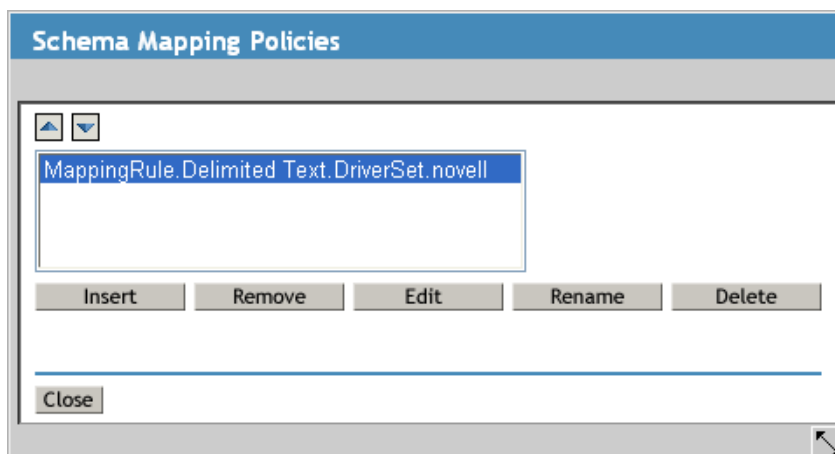
- 4 Click the Schema Mapping Policy.
- 5 Click *Edit*.

## 6.2.2 Editing the Schema Mapping Policy

There are two different parts to editing a Schema Mapping policy. First, you edit the placement of the policies in the policy set. Second, you edit the policy itself through the Schema Map Editor.

### Placement of the Policies

When you click on the Schema Mapping Policy, it brings up a window with options.

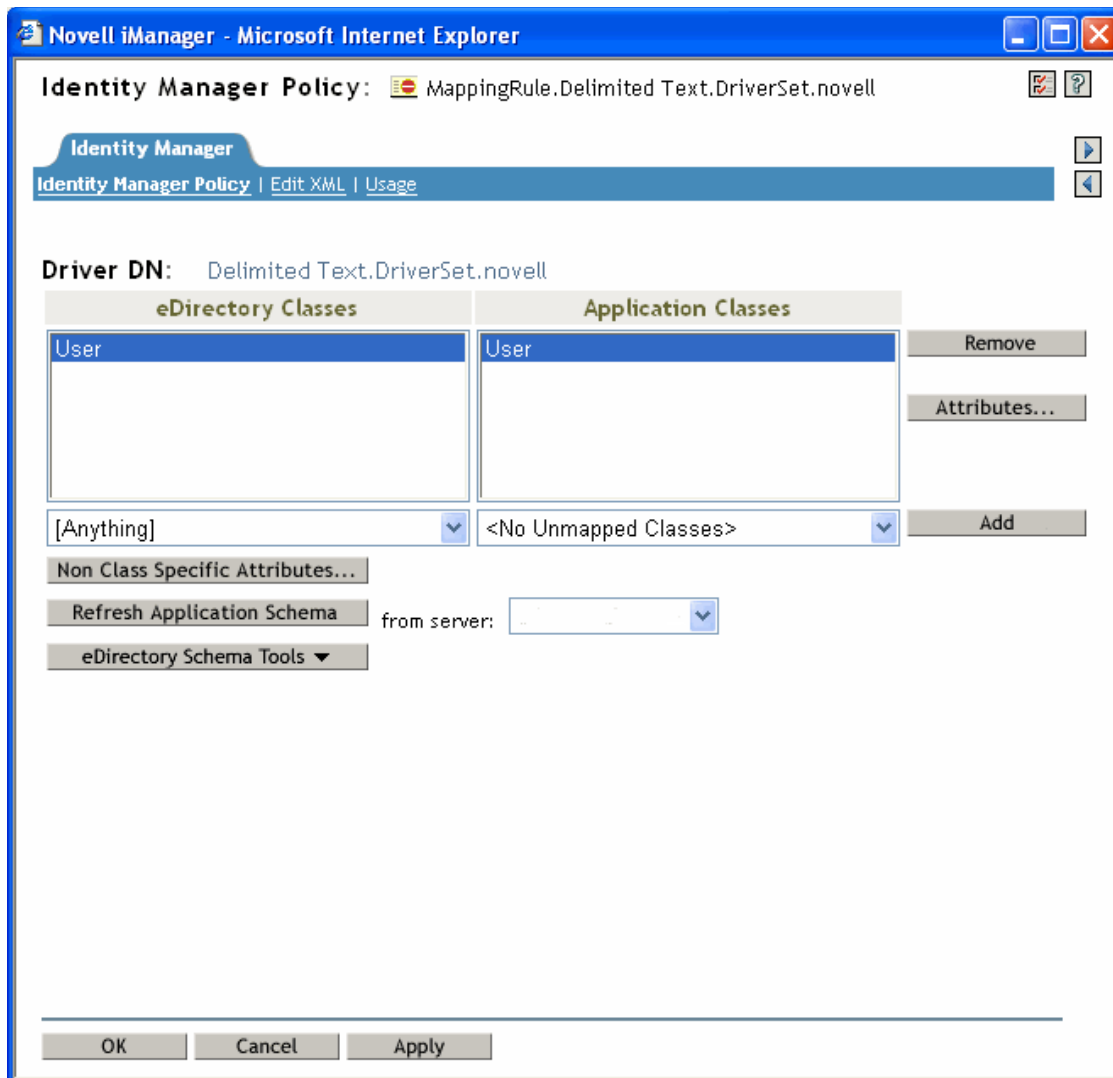


These options allow you to position the policy you are currently working with. The following table explains each of the options.

<b>Option</b>	<b>Description</b>
Move Policy Up	Moves the selected policy up if there is more than one policy.
Move Policy Down	Moves the selected policy down if there is more than one policy.
Insert	Inserts a new or an existing policy into the policies listed.
Remove	Removes the selected policy without deleting the policy from the policy set.
Edit	Launches the Schema Map Editor.
Rename	Renames the selected policy.
Delete	Deletes the selected policy.

### **Schema Map Editor**

The Schema Map Editor is a complete graphical interface for creating and managing the schema mapping policies. The Schema Map Editor creates a policy by using XML.



The Schema Map Editor has three tabs:

- “Identity Manager Policy” on page 341
- “Edit XML” on page 342
- “Usage” on page 342

## Identity Manager Policy

Contains the most information and is where you edit the policy through the GUI interface. You can do the following tasks in the Schema Map Editor:

---

### Removing Classes and Attributes

Select the class or attribute you would like to remove, then click Remove.

### Adding Classes

Select the eDirectory class from the drop-down list and then select the Application class from the drop-down list. With the items selected, click Add, then click *Apply* to save the change.

---

---

Adding Attributes	Select the class of the attribute you want to add, then click Attribute. Select the eDirectory attribute from the drop-down list and then select the Application attribute from the drop-down list. With the items selected, click Add, then click OK to save the changes.
Listing Non Specific Class Attributes	If there are attributes that are not associated with a class, click the Non-specific Class Attributes icon and all of these attributes are listed.
Refreshing Application Schema	If the schema has changed for the application, click the Refresh Application Schema icon. The wizard contacts the Connected System server to retrieve the new schema. After the schema has been updated, the schema is listed in the drop-down lists.
Using eDirectory Schema Tools	<ul style="list-style-type: none"> <li>• Add Attribute - Adds an existing attribute to the selected class.</li> <li>• Create Attribute - Creates a new attribute.</li> <li>• Create Class - Creates a new class.</li> <li>• Delete Attribute - Deletes the selected attribute.</li> <li>• Delete Class - Deletes the selected class.</li> <li>• Refresh eDirectory Schema - After making changes to the eDirectory schema, click Refresh eDirectory Schema and the drop-down lists are updated with the new information.</li> </ul>

---

**WARNING:** Do not delete any classes or attributes that are being used in the Identity Vault. It can cause objects to become unknown.

---

## Edit XML

Clicking *Enable XML editing* allows you to edit the DirXML Script policy. Make the changes you desire to the DirXML Script, then click *Apply* to save the changes.

## Usage

Shows you a list of the drivers that are currently referencing this policy. The list only refers to policies in this policy's driver set. If this policy is referenced from a different driver set, those references do not appear here.